# БАКАЛАВРСЬКА РОБОТА

**БР.КІ–00.00.00.000 ПЗ**

**Група КІ-22-2**

**Андрій Гуменюк**

**2026**

Івано-Франківський національний технічний університет нафти і газу
Факультет інформаційних технологій
Кафедра комп'ютерних систем і мереж

**Гуменюк Андрій Любомирович**

УДК 004.4

# БАКАЛАВРСЬКА РОБОТА

## Розробка web-сервісу динамічного планування підсумкової атестації аспірантів на основі Python, PostgreSQL та Docker

Комп'ютерна інженерія
(назва освітньої програми)

123 - Комп'ютерна інженерія
(шифр і назва спеціальності)

Робота містить результати власних досліджень, використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело:

Здобувач освітнього ступеня *Гуменюк А.Р.*
(підпис, прізвище та ініціали здобувача)

Науковий керівник *Гуменюк Т.В., доцент*
(підпис, прізвище, ім'я, по батькові, науковий ступінь, вчене звання керівника)

**Допущено до захисту**

Завідувач кафедри КСМ

д.т.н., професор Мельничук С. І.
(посада) (підпис) (дата) (прізвище та ініціали)

**м. Івано-Франківськ – 2026**

**Івано-Франківський національний технічний університет нафти і газу**

Факультет інформаційних технологій

Кафедра комп'ютерних систем і мереж

Освітній рівень бакалавр

Спеціальність 123 Комп’ютерна інженерія

**ЗАТВЕРДЖУЮ**

Завідувач кафедри

Мельничук С. І.

" 21 " *квітня* 2026 року

З А В Д А Н Н Я

# НА БАКАЛАВРСЬКУ РОБОТУ СТУДЕНТОВІ

Гуменюку Андрію Любомировичу

(прізвище, ім’я, по батькові)

1. Тема роботи *Розробка web-сервісу динамічного планування підсумкової атестації аспірантів на основі Python, PostgreSQL та Docker*

керівник роботи Гуменюк Тарас Володимирович, доцент

затверджені наказом закладу вищої освіти від *21.04.2026 №180/7*

2. Термін подання студентом роботи *11 червня* 2026 року

3. Вихідні дані до роботи *Методичні вказівки, технічна література*

4. Зміст пояснювальної записки: 1 Аналіз предметної області та постановка задачі. 2 Проєктування web-додатка супроводу атестації аспірантів і планування захистів. 3 Реалізація web-додатка супроводу атестації аспірантів.

5. Перелік графічного матеріалу (з точним зазначенням обов’язкових креслень)

6. Консультанти розділів роботи

| Розділ | Прізвище, ініціали та посада консультанта | Підпис, дата | |
|---|---|---|---|
| | | Завдання видав | Завдання прийняв |
| | | | |
| | | | |

7. Дата видачі завдання – 02 лютого 2026 року

## КАЛЕНДАРНИЙ ПЛАН

| № з/п | Назва етапів бакалаврської роботи | Термін виконання етапів роботи | Примітка |
|---|---|---|---|
| 1 | Пошук інформативних аналогів, теоретична підготовка до виконання бакалаврської роботи | 06.02.2026 – 15.03.2026 | виконав |
| 2 | Написання коду | 16.03.2026 – 20.04.2026 | виконав |
| 3 | Розробка діаграм UML | 23.04.2026 – 25.04.2026 | виконав |
| 4 | Оформлення пояснювальної записки | 22.05.2026 – 30.05.2026 | виконав |

**Студент** _____ Гуменюк А.Л.
(підпис) (прізвище та ініціали)

**Керівник роботи** _____ Гуменюк Т.В.
(підпис) (прізвище та ініціали)

# АНОТАЦІЯ

У бакалаврській роботі розроблено web-додаток для супроводу підсумкової атестації аспірантів. Система призначена для імпорту CSV-файлів ЄДЕБО, ведення карток аспірантів, формування нормативних етапів атестації, роботи зі складом разових спеціалізованих вчених рад, планування графіка захистів і динамічного перепланування після фіксації фактичних подій.

У роботі проаналізовано нормативні вимоги до підготовки та проведення захисту дисертації доктора філософії, спроєктовано модель бази даних, рольову модель доступу, REST API та алгоритм планування. Алгоритм враховує нормативні строки, дату завершення підготовки аспіранта, тривалість захисту, аудиторії, склад ради та неможливість одночасної участі членів рад у кількох захистах.

Програмну реалізацію виконано з використанням FastAPI, PostgreSQL, SQLAlchemy, HTML/CSS/JavaScript і Docker Compose. Проведено функціональне, інтеграційне та експериментальне тестування системи. Результати підтвердили працездатність розробленого web-сервісу.

Ключові слова: аспірант, атестація, разова спеціалізована вчена рада, ЄДЕБО, графік захистів, нормативний контроль, FastAPI, PostgreSQL, Docker Compose.

# ANNOTATION

The bachelor's thesis presents the development of a web application for supporting the final attestation of PhD students. The system is intended for importing CSV files exported from EDEBO, maintaining PhD student records, generating regulatory attestation stages, managing one-time specialized academic councils, planning dissertation defense schedules and dynamically recalculating them after actual events are recorded.

The thesis analyzes regulatory requirements for preparing and conducting a PhD dissertation defense, designs a database model, a role-based access model, a REST API and a scheduling algorithm. The algorithm takes into account regulatory deadlines, the end date of the PhD student's training period, defense duration, available rooms, council composition and conflicts involving council members.

The software implementation is based on FastAPI, PostgreSQL, SQLAlchemy, HTML/CSS/JavaScript and Docker Compose. Functional, integration and experimental testing was carried out. The results confirmed the operability of the developed web service.

Keywords: PhD student, attestation, specialized academic council, EDEBO, defense schedule, regulatory control, FastAPI, PostgreSQL, Docker Compose.

**ЗМІСТ**

# ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

API - Application Programming Interface, програмний інтерфейс застосунку.

CSV - Comma-Separated Values, текстовий формат подання табличних даних.

ER - Entity-Relationship, модель «сутність-зв'язок».

HTML - HyperText Markup Language, мова розмітки web-сторінок.

HTTP - HyperText Transfer Protocol, протокол передавання даних у web-середовищі.

JWT - JSON Web Token, формат токена для передавання даних автентифікації та авторизації.

MRV - Minimum Remaining Values, евристика вибору змінної з мінімальною кількістю допустимих варіантів.

ORM - Object-Relational Mapping, об'єктно-реляційне відображення.

PDF/A - формат довгострокового архівного зберігання PDF-документів.

REST - Representational State Transfer, архітектурний стиль побудови web API.

SQL - Structured Query Language, мова структурованих запитів.

UI - User Interface, інтерфейс користувача.

ЄДЕБО - Єдина державна електронна база з питань освіти.

ЗВО - заклад вищої освіти.

ІФНТУНГ - Івано-Франківський національний технічний університет нафти і газу.

КЕП - кваліфікований електронний підпис.

КМУ - Кабінет Міністрів України.

МОН - Міністерство освіти і науки України.

НАQA.Svr - інформаційна система Національного агентства із забезпечення якості вищої освіти для обліку разових рад і захистів.

ОНП - освітньо-наукова програма.

РСВР - разова спеціалізована вчена рада.

СКБД - система керування базами даних.

УкрІНТЕІ - Український інститут науково-технічної експертизи та інформації.

# ВСТУП

Підсумкова атестація аспірантів є регламентованим адміністративно-науковим процесом, який охоплює підготовку дисертації, формування разової спеціалізованої вченої ради, оприлюднення матеріалів, призначення дати захисту, проведення засідання ради та оформлення результатів. Особливістю цього процесу є поєднання значної кількості нормативних строків, відповідальних осіб, документів, рецензій, відгуків, рішень і публічних матеріалів. Порушення хоча б одного строку або неузгодженість складу ради може призвести до затримки захисту чи необхідності повторного коригування графіка [1; 2].

Актуальність обраної теми полягає в тому, що традиційне ведення даних щодо атестації аспірантів у таблицях, календарях, електронному листуванні та окремих документах не забезпечує цілісного контролю процедури. Табличні файли зручні для первинного обліку, однак вони не перевіряють нормативні обмеження, не фіксують причин конфліктів, не зберігають повну історію перепланування та не гарантують відсутність накладання аудиторій або членів разових рад. Окремі календарі дають змогу бачити події, але не пов’язують їх із етапами проходження дисертаційної роботи, складом ради, документами та датою завершення підготовки аспіранта.

Аналіз сучасного стану конкретного інженерного завдання показує, що універсальні інструменти спільної роботи, календарного планування та бронювання подій розв’язують лише окремі допоміжні задачі: спільне редагування таблиць, узгодження часу зустрічі або відображення календарних подій [3-7]. Водночас вони не враховують нормативні етапи атестації аспірантів, не контролюють граничну дату захисту, не перевіряють склад разової спеціалізованої вченої ради, не працюють із даними ЄДЕБО та не забезпечують автоматичного перепланування після фіксації фактичної події. Тому інженерна проблема полягає у створенні спеціалізованого web-сервісу, адаптованого до нормативного процесу закладу вищої освіти.

Окремі питання автоматизованого планування, формування розкладів і розв'язання задач розподілу ресурсів розглядалися у працях українських і зарубіжних дослідників. Зокрема, у роботі І. Ю. Юрчака та Т. Р. Москович досліджено застосування генетичних алгоритмів в автоматизованій системі розподілу навчального навантаження [19]. Зарубіжні автори розглядають задачі планування захистів дисертацій як окремий клас задач розкладу з багатьма обмеженнями, зокрема із застосуванням генетичних алгоритмів, моделей цілочислового програмування, евристичних методів і підходів до динамічного перепланування [20-24]. Водночас наявні підходи переважно зосереджені на математичному формуванні розкладу і недостатньо враховують специфіку українського нормативного процесу атестації аспірантів, імпорт даних ЄДЕБО, формування разових спеціалізованих вчених рад та підготовку публічних матеріалів.

Додатковою проблемою є використання даних, експортованих з ЄДЕБО. Такі файли потребують перевірки структури, кодування, обов'язкових полів, дат і дублікатів. Без автоматизованої обробки зростає ризик помилкового перенесення даних, випадкового перезапису опорної дати завершення підготовки або використання неповної інформації для формування графіка. Тому доцільним є створення спеціалізованого web-сервісу, який об'єднує імпорт даних, нормативний контроль, ведення картки аспіранта, роботу з разовою спеціалізованою вченою радою та формування графіка захистів..

О б ' є к т  д о с л і д ж е н н я – процес організації та планування підсумкової атестації аспірантів.

П р е д м е т  д о с л і д ж е н н я – моделі, алгоритм і web-сервіс динамічного формування та перепланування графіка захистів дисертацій аспірантів з урахуванням нормативних строків, складу разових спеціалізованих вчених рад, доступності учасників і аудиторій. Об'єкт і предмет співвідносяться як загальне і часткове: у межах загального процесу атестації виділено саме ту його частину, що пов'язана з інформаційним, алгоритмічним і програмним забезпеченням планування захистів..

Метою бакалаврської роботи є розроблення працездатного web-сервісу для імпорту даних ЄДЕБО, ведення карток аспірантів, контролю нормативних етапів атестації, формування складу разових спеціалізованих вчених рад, планування та динамічного перепланування графіка захистів. Таке формулювання мети безпосередньо пов'язане з темою роботи і спрямоване на отримання конкретного інженерного результату.

Для досягнення поставленої мети необхідно виконати такі завдання:

1. Проаналізувати нормативні вимоги до підготовки та проведення атестації аспірантів.

2. Визначити недоліки ручного ведення таблиць, документів, листування та календарів.

3. Сформулювати функціональні та нефункціональні вимоги до web-сервісу.

4. Спроєктувати рольову модель доступу для адміністратора, оператора, аспіранта, наукового керівника та члена разової ради.

5. Розробити модель бази даних для збереження аспірантів, етапів, рад, документів, аудиторій, налаштувань і версій графіка.

6. Розробити алгоритм формування графіка захистів із перевіркою нормативних, часових і ресурсних обмежень.

7. Реалізувати динамічне перепланування зі збереженням зафіксованих подій та діагностуванням причин неможливості побудови графіка.

8. Реалізувати web-інтерфейс для імпорту ЄДЕБО, роботи з карткою аспіранта, складом РСВР, календарем і налаштуваннями.

9. Забезпечити контейнерне розгортання web-сервісу та бази даних PostgreSQL засобами Docker Compose [18].

10. Провести функціональне, інтеграційне та експериментальне тестування розробленої системи.

Для розв'язання поставлених завдань використано такі методи: аналіз нормативних документів - для визначення етапів атестації, строків і обмежень процесу; об'єктно-орієнтоване проєктування - для побудови структури web-

додатка та розподілу відповідальності між модулями; реляційне моделювання - для проєктування бази даних аспірантів, етапів, рад, документів і графіків; алгоритмічне моделювання - для формалізації задачі планування захистів; пошук із поверненням, MRV-упорядкування та branch-and-bound - для побудови допустимого графіка з урахуванням обмежень; функціональне, інтеграційне та експериментальне тестування - для перевірки працездатності web-сервісу й оцінювання ефективності алгоритму.

Практичне значення отриманих результатів полягає у створенні web-додатка, який може бути використаний відділом аспірантури і докторантури як демонстраційна основа для автоматизації супроводу підсумкової атестації аспірантів. Результати роботи можуть застосовуватися для імпорту даних ЄДЕБО, ведення карток аспірантів, контролю нормативних строків, формування складу разових спеціалізованих вчених рад, планування графіка захистів, динамічного перепланування після фіксації фактичних подій і підготовки публічних інформаційних матеріалів.

# 1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАДАЧІ

## 1.1 Особливості організації підсумкової атестації аспірантів

Організація завершального етапу підготовки здобувача ступеня доктора філософії регламентується двома взаємопов’язаними нормативними актами. Порядок, затверджений постановою Кабінету Міністрів України № 261, визначає підготовку аспіранта та дії, що передують утворенню разової спеціалізованої вченої ради. Порядок, затверджений постановою № 44, регулює утворення такої ради, проведення публічного захисту та оформлення його результатів [1; 2]. Отже, підсумкову атестацію доцільно розглядати не як окрему подію, а як послідовність нормативно залежних етапів.

Нормативний строк підготовки здобувача в аспірантурі становить чотири роки. Протягом цього періоду аспірант виконує індивідуальний навчальний план та індивідуальний план наукової роботи, а також двічі на рік звітує про результати їх виконання. Завершення підготовки передбачає виконання освітньої і наукової складових освітньо-наукової програми, підготовку дисертації, опублікування наукових результатів та проходження атестації у формі публічного захисту [1]. Таким чином, готовність аспіранта до атестації залежить не лише від стану дисертації, а й від виконання сукупності пов’язаних вимог.

Початковою контрольною точкою завершального етапу є визначення стану готовності дисертації науковим керівником. Не пізніше ніж за дев’ять місяців до завершення нормативного строку навчання здобувач повинен отримати довідку про виконання освітньо-наукової програми та висновок наукового керівника. Після цього він звертається до структурного підрозділу із заявою про надання висновку щодо наукової новизни, теоретичного і практичного значення результатів дисертації. До заяви додаються дисертація, наукові публікації, довідка про виконання освітньо-наукової програми та висновок керівника [1].

Публічна презентація наукових результатів має бути проведена не пізніше ніж через місяць після надходження заяви. За результатами презентації структурний підрозділ готує висновок, який повинен бути наданий здобувачеві протягом двох тижнів. У разі позитивного висновку здобувач має не більше двох тижнів для подання заяви про утворення разової спеціалізованої вченої ради [1]. Ці етапи утворюють послідовний ланцюг, у якому дата завершення попередньої дії визначає допустимий строк наступної.

Після отримання заяви вчена рада закладу повинна не пізніше ніж протягом двох місяців утворити разову раду у складі п’яти осіб. До її складу входять голова, рецензенти та офіційні опоненти. Рішення про утворення ради вводиться в дію наказом керівника закладу протягом п’яти робочих днів [2]. Кандидатури членів ради повинні відповідати вимогам щодо наукової компетентності, місця роботи, наявності публікацій і відсутності конфлікту інтересів. Одна особа може брати участь не більше ніж у восьми захистах протягом календарного року [2]. Тому формування складу ради є не лише кадровою, але й обмежувальною задачею, яка потребує перевірки допустимості кожної кандидатури.

Протягом п’яти робочих днів після видання наказу заклад повинен оприлюднити дисертацію, висновок про наукову новизну та інформацію про склад разової ради на офіційному вебсайті, а також внести відповідні відомості до інформаційної системи Національного агентства із забезпечення якості вищої освіти [2]. Отже, дата видання наказу є базовою для розрахунку строків оприлюднення та подальшої підготовки рецензій і відгуків.

Рецензенти й офіційні опоненти повинні надати рецензії та відгуки протягом 45 календарних днів із дня оприлюднення інформації про утворення ради. Після надходження останнього документа рада протягом трьох робочих днів призначає дату, час і місце захисту. Захист може відбутися не раніше ніж через два та не пізніше ніж через чотири тижні після надходження останньої рецензії або відгуку [2]. Через це дата захисту не може бути визначена незалежно від фактичного завершення рецензування.

Після проведення захисту заклад протягом трьох робочих днів оприлюднює рішення разової ради та відеозапис засідання на своєму вебсайті, а протягом п’яти робочих днів вносить результати захисту до інформаційної системи. Наказ про видачу диплома доктора філософії видається не раніше ніж через 15 і не пізніше ніж через 30 календарних днів із дня захисту [2]. Таким чином, процедура не завершується проведенням засідання, а охоплює також післязахисне оприлюднення та оформлення результатів.

Узагальнену послідовність підготовки та проведення атестації наведено на рисунку 1.1.


Рисунок 1.1 – Процес підготовки та проведення атестації: від завершення ОНП до видання диплома

Аналіз регламенту показує, що строки процедури мають різну природу: частина обчислюється в календарних днях, частина — у робочих днях, а окремі етапи задаються в тижнях або місяцях. Крім того, постановою № 44 установлено, що період дії воєнного стану та три місяці після його припинення або скасування не враховуються під час обчислення визначених цим Порядком строків [2]. Тому

використання простого додавання фіксованої кількості днів не забезпечує коректного планування.

З погляду автоматизації процедура підсумкової атестації є процесом із часовими, ресурсними та логічними обмеженнями. До часових обмежень належать граничні строки виконання дій. Логічні обмеження визначають порядок етапів: наприклад, утворення ради можливе після отримання позитивного висновку, а призначення захисту — після надходження рецензій і відгуків. Ресурсні обмеження пов’язані з доступністю членів ради, відповідністю їхньої кваліфікації та відсутністю конфлікту інтересів [1; 2].

Перенесення одного з етапів не може оброблятися як механічне зміщення всього графіка. Зміна дати публічної презентації впливає на строк отримання висновку та подання заяви, тоді як затримка останньої рецензії змінює допустимий інтервал проведення захисту. Водночас уже виконані етапи повинні залишатися зафіксованими. Отже, динамічне перепланування має здійснюватися лише для залежних незавершених подій із повторною перевіркою нормативних обмежень.

Для інформаційного супроводу процедури необхідно зберігати не тільки планові дати, а й фактичні дати виконання, статуси етапів, відповідальних осіб і документи, що підтверджують завершення дії. Доцільно також фіксувати нормативну підставу та правило розрахунку кожного строку. Такий підхід дає змогу відрізняти обов’язкові строки від внутрішніх рекомендованих дат і формувати попередження про можливе порушення регламенту.

Отже, особливістю організації підсумкової атестації аспірантів є поєднання послідовного документообігу, нормативно встановлених часових інтервалів, участі декількох категорій користувачів і необхідності оперативного коригування графіка. Зазначені особливості обґрунтовують розроблення спеціалізованого web-сервісу, у якому графік формується як система взаємопов’язаних етапів, а зміна вихідних даних супроводжується автоматичним перерахунком залежних дат і перевіркою їх відповідності вимогам постанов № 261 та № 44.

## 1.2 Аналіз існуючих засобів планування розкладів

Для автоматизації підсумкової атестації необхідно розв’язати дві взаємопов’язані задачі: сформувати послідовність нормативних етапів із визначеними строками та узгодити дату захисту з доступністю членів разової спеціалізованої вченої ради. Тому придатність наявних програмних засобів оцінено за підтримкою залежностей між етапами, обмежень щодо учасників, динамічного перепланування, імпорту даних з ЄДЕБО, контролю документів і розмежування прав доступу.

Найпростішим засобом ведення графіків є електронні таблиці. Їхніми перевагами є доступність, можливість колективного редагування та збереження історії версій [3]. У таблиці можна сформувати перелік аспірантів, установити планові дати та використати формули для розрахунку окремих строків. Такий підхід не потребує розроблення спеціального програмного забезпечення та придатний для невеликої кількості записів.

Водночас таблиця не відображає процес атестації як систему залежних об’єктів. Зміна дати одного етапу потребує перевірки формул і ручного контролю наступних подій. Помилка у формулі, сортуванні або копіюванні клітинки може призвести до некоректного графіка. Також ускладнюються розмежування доступу до персональних даних, прив’язування документів до конкретних етапів і ведення контрольованого журналу дій.

Аналіз наданих тестових експортів з ЄДЕБО показав, що кожний файл містить 66 полів. Серед них наявні ідентифікатор картки, статус навчання, дати початку та завершення підготовки, структурний підрозділ, спеціальність, освітня програма, курс та ознака поновлення для захисту дисертації. Значна частина полів не потрібна для планування атестації. Тому пряме відкриття CSV-файлу в електронній таблиці не розв’язує задачі відбору, перевірки та контрольованого оновлення необхідних даних.

Google Calendar дає змогу визначати доступні часові інтервали, створювати сторінки бронювання, встановлювати мінімальний строк попереднього запису,

додавати перерви між подіями та перевіряти календарі для уникнення конфліктів [4]. Його доцільно використовувати для відображення погодженої дати презентації або захисту, надсилання запрошень і нагадувань учасникам.

Однак календарна подія не містить предметної моделі атестації. Google Calendar не перевіряє строки утворення ради, підготовки відгуків та оприлюднення матеріалів, установлені постановами № 261 і № 44 [1; 2]. Перенесення події також не спричиняє автоматичного перерахунку всіх нормативно залежних етапів.

Microsoft Bookings підтримує персональні й колективні сторінки бронювання, налаштування послуг, складу працівників, робочих годин і доступності. Сервіс інтегрований з Outlook та Microsoft Teams і може автоматично надсилати повідомлення учасникам [5]. Це робить його придатним для організації окремих зустрічей, зокрема попереднього обговорення дисертації або консультації.

Основним об’єктом Microsoft Bookings залишається зустріч між користувачем і одним або кількома працівниками. Сервіс не враховує кваліфікаційні вимоги до членів разової ради, конфлікти інтересів, обмеження щодо кількості захистів та залежність дати захисту від надходження останньої рецензії або відгуку. Отже, він може доповнювати спеціалізовану систему, але не замінює її.

Calendly також забезпечує синхронізацію з календарями, налаштування доступності, часових буферів, типів подій, нагадувань та колективного планування [6]. Його перевагою є зручне узгодження зустрічей із зовнішніми учасниками. Проте логіка сервісу орієнтована на бронювання окремих подій, а не на супровід багатоступеневої нормативної процедури.

Для пошуку спільного часу групи користувачів може застосовуватися Doodle. Організатор створює опитування із запропонованими датами, після чого учасники позначають прийнятні варіанти [7]. Такий механізм корисний для збирання інформації про доступність голови, рецензентів та опонентів. Водночас результат опитування потребує ручного перенесення до основного графіка.

Doodle не перевіряє нормативну допустимість вибраної дати та не формує пов’язані етапи атестації.

Окрему групу становлять бібліотеки оптимізаційного планування. Google OR-Tools підтримує моделювання задач, у яких ресурси призначаються часовим інтервалам з урахуванням обов’язкових обмежень і бажаних критеріїв. У документації наведено формування розкладу з перевіркою зайнятості учасників, розподілом навантаження та максимізацією кількості задоволених побажань [8]. Наявність інтерфейсу для Python дає змогу інтегрувати OR-Tools у серверну частину розроблюваного web-сервісу.

Timefold Solver також призначений для задач обмежувального планування. Він розділяє обмеження на жорсткі, порушення яких робить план недопустимим, і м’які, що визначають якість отриманого розкладу [9]. Такий підхід відповідає предметній області: нормативні строки та відсутність одночасних засідань є жорсткими обмеженнями, тоді як рівномірність розподілу засідань або бажаний час їх проведення можуть розглядатися як м’які. Однак Timefold Solver орієнтований на екосистему Java, тому його застосування у проєкті на основі Python потребувало б окремого сервісу або зміни технологічного стеку [10].

Порівняння розглянутих засобів наведено в таблиці 1.1.

Результати порівняння показують, що календарні сервіси ефективно розв’язують задачу вибору часу, але не забезпечують супроводу всієї процедури атестації. Електронні таблиці придатні для початкового обліку, проте не гарантують цілісності даних і коректного перерахунку залежних строків. Оптимізаційні бібліотеки дають змогу реалізувати складну логіку планування, але потребують розроблення бази даних, web-інтерфейсу, авторизації, імпорту та документообігу.

**Таблиця 1.1 – Порівняння засобів планування**

| Засіб | Основна перевага | Обмеження для поставленої задачі | Доцільне застосування |
|---|---|---|---|
| Excel, Google Sheets | Просте ведення переліків і спільне редагування | Відсутність надійної предметної моделі та автоматичного перепланування | Попередній облік і перевірка даних |
| Google Calendar | Календарі, нагадування та контроль зайнятості | Не підтримує нормативну послідовність етапів | Відображення погоджених подій |
| Microsoft Bookings | Бронювання часу з урахуванням доступності працівників | Не перевіряє склад ради та нормативні строки | Організація окремих зустрічей |
| Calendly | Зручне планування та інтеграція із зовнішніми календарями | Орієнтація на окремі зустрічі | Збирання доступності учасників |
| Doodle | Узгодження часу групою користувачів | Ручне перенесення результату до графіка | Попереднє опитування членів ради |
| Google OR-Tools | Підтримка складних обмежень та Python | Не є готовим web-застосунком | Реалізація алгоритмічного модуля |
| Timefold Solver | Розподіл обмежень за рівнями пріоритету | Потребує інтеграції з Java-стеком | Альтернативний оптимізаційний засіб |

Отже, використання одного з наявних засобів не забезпечує повної відповідності вимогам предметної області. Обґрунтованим є створення спеціалізованого web-сервісу, який поєднуватиме імпорт даних з ЄДЕБО, централізоване збереження відомостей, нормативну модель етапів, контроль доступності учасників і динамічний перерахунок графіка. Календарні системи доцільно використовувати як зовнішні засоби інформування, а оптимізаційні бібліотеки — як можливу основу або засіб порівняння для алгоритму планування.

### 1.3 Вимоги до web-сервісу динамічного планування

Вимоги до web-сервісу сформовано на підставі нормативної послідовності підготовки та проведення захисту, визначеної постановами Кабінету Міністрів України № 261 і № 44 [1; 2], результатів аналізу наявних засобів планування та

структури наданих CSV-експортів з ЄДЕБО [11]. Основним призначенням системи є централізований супровід аспірантів на завершальному етапі підготовки та формування допустимого графіка захистів.

Надані CSV-файли містять по 66 полів і використовують крапку з комою як роздільник. Для планування необхідна лише частина відомостей: ідентифікатор картки, статус навчання, ПІБ здобувача, дати початку та завершення підготовки, структурний підрозділ, спеціальність, освітня програма, курс і ознака поновлення для захисту дисертації [11]. Поля, що містять дату народження, громадянство, РНОКПП та реквізити документів, не повинні імпортуватися, оскільки вони не потрібні для визначеної мети обробки. Такий підхід відповідає принципу пропорційності складу персональних даних установленій меті їх обробки [14].

Імпорт має виконуватися з файлу, сформованого засобами ЄДЕБО, без прямого підключення web-сервісу до державної системи. Перед збереженням система повинна перевіряти формат файлу, роздільник, кодування, наявність обов’язкових стовпців, коректність дат і унікальність ідентифікаторів. Потрібно підтримати файли у кодуваннях Windows-1251 та UTF-8, оскільки надані експорти не містять маркера кодування [11].

Повторний імпорт не повинен створювати дублікати. Ідентифікацію запису доцільно здійснювати за полем «ID картки», а результатом імпорту має бути звіт із кількістю доданих, оновлених, пропущених і помилкових записів. Перед остаточним збереженням оператор повинен переглянути виявлені зміни та підтвердити їх.

Основні функціональні вимоги наведено в таблиці 1.2.

Для кожного аспіранта система повинна формувати індивідуальний план, що охоплює отримання довідки про виконання освітньо-наукової програми, висновку наукового керівника, проведення публічної презентації, надання висновку про наукову новизну, подання заяви про утворення разової ради, формування її складу, оприлюднення матеріалів, отримання рецензій і відгуків, призначення та проведення захисту, а також оформлення його результатів [1; 2].

**Таблиця 1.2 – Функціональні вимоги до web-сервісу**

| Код | Вимога |
|---|---|
| FR-01 | Імпорт даних про аспірантів із CSV-файлу ЄДЕБО з перевіркою структури, кодування, типів даних і дублікатів [11] |
| FR-02 | Створення та оновлення картки аспіранта за унікальним ідентифікатором ЄДЕБО |
| FR-03 | Формування для аспіранта послідовності нормативних етапів підготовки до захисту [1; 2] |
| FR-04 | Розрахунок строків у календарних і робочих днях, тижнях та місяцях відповідно до нормативного правила [1; 2] |
| FR-05 | Зберігання планової і фактичної дат, статусу, відповідальної особи, коментарів та документів для кожного етапу |
| FR-06 | Ведення відомостей про склад разової ради та перевірка обмежень щодо її членів [2] |
| FR-07 | Реєстрація доступності членів ради, часових інтервалів і приміщень |
| FR-08 | Автоматизоване формування графіка захистів із перевіркою часових, ресурсних і нормативних обмежень |
| FR-09 | Динамічне перепланування незавершених подій зі збереженням зафіксованих етапів |
| FR-10 | Формування діагностичного повідомлення, якщо допустимий графік побудувати неможливо |
| FR-11 | Підготовка проєкту сторінки разової ради, оголошення та календарного представлення графіка [2] |
| FR-12 | Розмежування доступу, журналювання змін та збереження версій опублікованого графіка |

Нормативні правила не доцільно жорстко записувати в програмному коді. Для кожного типу етапу потрібно зберігати базову подію, величину строку, одиницю обчислення, напрям відліку та посилання на нормативну підставу. Правило також повинно мати період дії. Це дасть змогу змінювати алгоритм розрахунку після оновлення законодавства без редагування історичних графіків.

Етап може мати статус «заплановано», «виконується», «завершено», «прострочено» або «потребує корекції». Перехід до стану «завершено» повинен передбачати зазначення фактичної дати та, коли це необхідно, додавання підтверджувального документа. Після завершення етапу його планова та фактична дати не можуть автоматично змінюватися під час перепланування.

Формування графіка захистів має враховувати такі жорсткі обмеження:

— дата захисту перебуває в нормативно допустимому інтервалі [2];

— усі члени відповідної разової ради доступні у вибраний час;

— аспірант, член ради або приміщення не можуть одночасно брати участь у двох захистах;

— до складу ради входить установлена кількість осіб із визначеними ролями [2];

— опубліковані або вручну зафіксовані події не змінюються автоматично;

— захист не призначається до завершення обов’язкових попередніх етапів.

До м’яких обмежень належать бажаний час учасників, рівномірний розподіл захистів між днями, зменшення кількості змін порівняно з попередньою версією графіка та уникнення надмірного навантаження членів рад. Порушення м’якого обмеження допускається, якщо це необхідно для побудови нормативно коректного графіка.

У разі зміни доступності учасника, приміщення або допустимого періоду система повинна повторно сформувати лише залежну частину графіка. Раніше опубліковані події зберігаються за можливості, а їх перенесення отримує більший штраф під час оцінювання варіантів. Якщо допустимого рішення немає, користувачеві потрібно повідомити причину: відсутність спільного часу, конфлікт приміщень, порушення строку або недостатню кількість доступних інтервалів.

У системі доцільно передбачити такі ролі:

— адміністратор керує обліковими записами, ролями, довідниками та нормативними правилами;

— оператор відділу аспірантури імпортує дані, веде етапи, формує та публікує графік;

— аспірант переглядає власний план, строки, коментарі та доступні документи;

— науковий керівник переглядає інформацію про закріплених аспірантів;

— член разової ради зазначає доступність і переглядає призначені засідання.

Вхід користувачів доцільно реалізувати через корпоративні облікові записи Google Workspace із застосуванням OpenID Connect. Сервер повинен перевіряти ідентифікаційний токен, підтвердження електронної адреси та належність адреси до дозволеного домену університету [16]. Адміністративні ролі призначаються вручну й не можуть визначатися лише за доменом електронної пошти.

Інформація в системі має бути поділена на внутрішню та публічну. Внутрішня частина може містити контактні відомості, службові коментарі, доступність учасників і документи. До публічної частини передаються лише відомості, які підлягають оприлюдненню відповідно до постанови № 44 [2]. Безпосередня публікація на сайті закладу або в інформаційній системі НАЗЯВО не належить до базової версії системи; web-сервіс формує перевірений набір матеріалів для подальшого розміщення.

Нефункціональні вимоги (табл.1.13) визначено з урахуванням моделі якості ISO/IEC 25010:2023 [12] і рекомендацій OWASP ASVS щодо перевірки засобів безпеки web-застосунків [13].

Цілісність інформації повинна забезпечуватися не тільки прикладною логікою, але й обмеженнями бази даних. Унікальний ідентифікатор картки запобігатиме дублюванню аспірантів, зовнішні ключі забезпечуватимуть зв’язок між аспірантом, етапами, документами та графіком, а перевірочні обмеження не допускатимуть некоректних статусів і часових інтервалів [17].

Отже, web-сервіс повинен поєднати контрольований імпорт даних з ЄДЕБО, нормативне планування етапів, формування розкладу захистів, динамічне перепланування та рольовий доступ. Визначені вимоги обмежують предметну область функціями, які безпосередньо необхідні для інформаційного супроводу підсумкової атестації, і не передбачають дублювання функцій ЄДЕБО, електронного документообігу або офіційного сайту закладу.

**Таблиця 1.3 – Нефункціональні вимоги**

| Категорія | Вимога |
|---|---|
| Продуктивність | Типова операція перегляду виконується не довше ніж за 2 с, імпорт 1000 записів — не довше ніж за 10 с |
| Надійність | Помилка імпорту або планування не повинна призводити до часткового збереження даних |
| Безпека | Автентифікація, рольовий доступ, перевірка вхідних даних, захист завантаження файлів і журналювання критичних дій [13–15] |
| Цілісність | Унікальні, зовнішні та перевірочні обмеження PostgreSQL повинні запобігати суперечливим записам [17] |
| Зручність | Україномовний адаптивний інтерфейс, фільтрування аспірантів і візуальне позначення прострочених етапів |
| Супроводжуваність | Розділення імпорту, нормативних правил, планувальника та інтерфейсу на окремі модулі |
| Переносимість | Розгортання web-сервісу та PostgreSQL у контейнерах за допомогою Docker Compose [18] |
| Відновлення | Регулярне резервне копіювання бази даних і перевірка можливості її відновлення |

## 1.4 Аналіз розв’язання задачі в наукових працях

Задача, що розв’язується у бакалаврській роботі, поєднує формування календарного плану підготовки аспіранта та складання розкладу захистів. Перша частина визначається нормативною послідовністю етапів, а друга передбачає призначення аспірантів, членів разових рад і приміщень на часові інтервали. У наукових працях найбільше досліджено саме задачу складання розкладу захистів, тоді як інформаційний супровід усього регламентованого процесу висвітлений значно менше.

У роботі І. Юрчак і Т. Москович досліджено автоматизований розподіл навчального навантаження із застосуванням модифікованого генетичного алгоритму. Автори формалізували часові інтервали, доступність викладачів, типи занять і приміщень, а якість розкладу оцінювали за допомогою системи штрафів. Розроблений алгоритм інтегровано у web-сервіс, а його параметри винесено до конфігураційних файлів [19].

Практично корисною для цього проєкту є запропонована авторами модель поділу вимог на обов’язкові умови та критерії якості. Недопущення одночасної участі особи у двох подіях або подвійного використання приміщення має бути

обов’язковою умовою. Побажання учасників щодо часу проведення захисту можуть враховуватися як критерії покращення графіка. Водночас генетичний алгоритм є стохастичним, тому потребує налаштування розміру популяції, операторів відбору, схрещування, мутації та умови завершення пошуку [19].

Т. Т. Б. Хюінь, К. Д. Фам і Д. Д. Фам розглянули задачу формування розкладу захистів магістерських робіт із декількома критеріями. У моделі враховувалися склад комісій, часові інтервали, доступність викладачів і розподіл навантаження. Автори обґрунтували належність задачі до класу NP-складних і запропонували генетичний алгоритм, перевірений на даних Ханойського університету науки і технологій [20].

Перевагою такого підходу є можливість отримувати прийнятні рішення для великого простору варіантів без повного перебору. Недоліком є відсутність гарантії знаходження оптимального рішення та залежність результату від параметрів алгоритму. Для web-сервісу, у якому користувач повинен отримувати відтворюваний результат і зрозуміле пояснення неможливості побудови графіка, ця властивість генетичного алгоритму створює додаткові труднощі.

М. Баттістутта та співавтори сформували узагальнену модель задачі планування захистів і порівняли три методи її розв’язання: цілочисельне програмування, програмування в обмеженнях і локальний пошук. Модель передбачала формування комісій та призначення кандидатів на захисні сесії. Експериментальне дослідження проводилося на реальних і штучно сформованих наборах даних [21].

За результатами експериментів цілочисельне програмування продемонструвало найкращі середні показники, локальний пошук посів друге місце, а програмування в обмеженнях — третє. Водночас автори не визначили методу, який був би найкращим для всіх наборів даних [21]. Це підтверджує, що вибір алгоритму має залежати від розміру задачі, структури обмежень і вимог до часу формування результату.

Цілочисельна модель забезпечує точне математичне подання залежностей і дає змогу оцінити якість знайденого рішення. Проте зі збільшенням кількості

аспірантів, членів рад, приміщень і часових інтервалів кількість змінних та обмежень істотно зростає. Крім того, зміна нормативного правила може вимагати модифікації математичної моделі.

У дослідженні Ж. Алмейди та співавторів запропоновано багатокритеріальну модель змішаного цілочисельного лінійного програмування для планування захистів. У ній кожному захисту призначається комісія, дата, час і приміщення. Автори врахували не один узагальнений показник, а декілька цілей, пов’язаних із якістю складу комісій, компактністю розкладу, зручністю учасників і справедливістю розподілу навантаження [22].

Багатокритеріальний підхід дає змогу отримати множину недомінованих рішень і надати відповідальній особі можливість вибору між альтернативами. Однак для прикладного web-сервісу такий підхід ускладнює взаємодію з користувачем: оператор повинен розуміти значення кожного критерію та наслідки компромісу між ними. Тому в розроблюваній системі доцільно спочатку забезпечити виконання всіх жорстких обмежень, а потім ранжувати допустимі графіки за зваженою штрафною оцінкою.

А. Дімітсас і К. Гогос запропонували точну модель задачі планування захистів на основі програмування в обмеженнях. Автори виявили симетричні варіанти розкладу, які фактично мають однакову якість, але збільшують простір пошуку. Додавання обмежень для усунення трьох типів симетрій дало змогу швидше отримувати оптимальні або близькі до оптимальних рішення та поліпшити найкращі результати для відкритих тестових наборів [23].

Для розроблюваного web-сервісу важливою є не тільки якість початкового графіка, але й можливість його коригування. Необхідність перепланування може виникнути через зміну доступності члена ради, перенесення попереднього етапу, затримку рецензії або недоступність приміщення. Повне формування нового графіка без урахування попереднього стану може спричинити зміну великої кількості вже погоджених подій.

Р. Ларсен і М. Пранзо запропонували загальну модель динамічного перепланування, що складається з контролера та детермінованого

планувальника. Контролер відстежує зміни вхідних даних і визначає момент повторного запуску алгоритму, тоді як планувальник формує нове рішення для актуального стану системи. Експерименти показали, що такий поділ дає змогу застосовувати статичний алгоритм у динамічному середовищі з невеликими додатковими часовими витратами [24].

Цей підхід відповідає архітектурі розроблюваного web-сервісу. Серверна частина фіксує зміни статусів, строків і доступності учасників, після чого передає актуальні дані модулю планування. При цьому опубліковані або вручну зафіксовані події залишаються незмінними, а перенесення інших подій отримує штраф. Завдяки цьому новий графік повинен не лише залишатися допустимим, але й мінімально відрізнятися від попереднього.

Порівняння проаналізованих наукових підходів наведено в таблиці 1.4.

**Таблиця 1.4 – Порівняння методів формування розкладу**

| Метод | Переваги | Обмеження | Придатність для проєкту |
|---|---|---|---|
| Генетичний алгоритм | Працює з великим простором рішень і декількома критеріями [19; 20] | Стохастичний результат, необхідність налаштування параметрів | Доцільний для великих наборів даних або порівняльного експерименту |
| Цілочисельне програмування | Формальна модель, можливість підтвердження оптимальності [21; 22] | Зростання обчислювальної складності та розміру моделі | Придатне для перевірки результатів на невеликих наборах |
| Програмування в обмеженнях | Природне подання часових і ресурсних обмежень [21; 23] | Ефективність залежить від структури моделі та евристик пошуку | Добре відповідає предметній області |
| Локальний пошук | Швидке покращення наявного рішення [21] | Можливе потрапляння в локальний оптимум | Доцільний як допоміжний етап оптимізації |
| Динамічне перепланування | Реагування на зміни та повторне використання статичного планувальника [24] | Потребує правила запуску та оцінки стабільності | Необхідне для коригування графіка після змін |

На відміну від розкладів занять, у задачі атестації аспірантів кількість одночасно запланованих захистів у межах одного закладу зазвичай є порівняно

невеликою. Водночас кожен захист має значну кількість жорстких обмежень. За таких умов доцільним є використання детермінованого алгоритму пошуку з поверненням, відсіканням неперспективних гілок і пріоритетним опрацюванням аспірантів із найменшою кількістю доступних варіантів.

Вибраний підхід повинен забезпечити:

— відтворюваність результату за однакових вхідних даних;

— раннє виявлення недопустимих варіантів;

— збереження зафіксованих подій;

— урахування штрафу за зміну попереднього графіка;

— формування пояснення, якщо повний графік побудувати неможливо;

— можливість перевірки алгоритму модульними тестами.

Проведений аналіз також показав, що більшість наукових праць розглядає призначення захистів на часові інтервали та формування складу комісій [20–23]. У них не враховано українську нормативну послідовність підготовки докторів філософії, імпорт відомостей з ЄДЕБО, контроль строків постанов № 261 і № 44, роботу з документами та рольовий доступ користувачів.

Отже, наукові праці містять достатньо опрацьовані методи формування розкладу захистів, але не пропонують готового рішення для повного супроводу атестації аспірантів відповідно до законодавства України. У розроблюваному web-сервісі доцільно поєднати детермінований алгоритм обмежувального пошуку з механізмом динамічного перепланування. Жорсткі обмеження визначатимуть допустимість графіка, а штрафи за небажаний час і зміну попередніх призначень — його якість та стабільність.

## 1.5 Постановка задачі

За результатами аналізу предметної області встановлено, що наявні календарні сервіси, електронні таблиці та окремі документи не забезпечують комплексного супроводу підсумкової атестації аспірантів. Наукові методи

розв'язання задач розкладу дають змогу враховувати часові та ресурсні обмеження, однак не охоплюють імпорт даних з ЄДЕБО, нормативну послідовність етапів, рольову взаємодію користувачів і підготовку матеріалів про разові спеціалізовані вчені ради [19-24]. Тому поставлено задачу розроблення спеціалізованого web-сервісу динамічного планування та нормативного супроводу підсумкової атестації аспірантів.

Метою розроблення є автоматизація формування індивідуальних планів атестації та графіків захистів аспірантів з урахуванням вимог постанов Кабінету Міністрів України № 261 і № 44, дати завершення підготовки аспіранта, складу разової спеціалізованої вченої ради, доступності учасників, аудиторій і параметрів планування [1; 2].

Об'єктом автоматизації є процес інформаційного супроводу аспіранта від імпорту його облікових даних до формування графіка захисту, фіксації результатів проходження етапів і підготовки матеріалів для оприлюднення.

Предметом розроблення є модель даних, алгоритм і web-сервіс динамічного формування та перепланування графіка захистів дисертацій аспірантів.

Вхідними даними системи є:

- CSV-файл, експортований з ЄДЕБО;
- нормативні правила та строки виконання етапів;
- відомості про аспірантів і дисертаційні дослідження;
- дата завершення підготовки аспіранта як опорна дата нормативного контролю;
- склад разових спеціалізованих вчених рад;
- доступність членів рад;
- перелік часових інтервалів, аудиторій і посилань на онлайн-трансляцію;
- операційні налаштування планування: тривалість засідання, стартові години, горизонт планування та денні ліміти участі членів рад;
- раніше сформований, погоджений або опублікований графік;
- фактичні дати виконання етапів.

Надані експорти з ЄДЕБО містять 66 стовпців, проте для роботи web-сервісу необхідно імпортувати лише дані, пов'язані з ідентифікацією здобувача та його освітньою траєкторією. До них належать ідентифікатор картки, ПІБ, статус навчання, дати початку й завершення підготовки, структурний підрозділ, спеціальність, освітня програма, курс та ознака поновлення для захисту дисертації [11]. Персональні дані, які не потрібні для планування, не повинні зберігатися в системі [14].

Результатом роботи web-сервісу мають бути:

- перевірені й оновлені картки аспірантів;
- індивідуальні плани проходження етапів атестації;
- допустимий графік захистів для одного аспіранта або групи аспірантів;
- повідомлення про конфлікти, порушення строків і причини неможливості формування графіка;
- оновлена версія графіка після зміни вихідних даних або фіксації фактичної події;
- проєкт інформаційної сторінки разової спеціалізованої вченої ради;
- публічне представлення утворених РСВР і матеріалів захисту;
- журнал змін, результати виконання імпорту та протоколи тестування.

Задачу формування розкладу можна подати як задачу призначення. Нехай $C$ - множина аспірантів, T - множина доступних часових інтервалів, $R$ - множина аудиторій, а $M$ - множина членів разових рад. Для кожного аспіранта c визначено склад ради $M_c$, допустиму множину інтервалів $A_c$ та бажану множину інтервалів $P_c$. Якщо оператор формує графік для одного аспіранта, множина C містить один елемент, однак перевірка часових, ресурсних і нормативних обмежень залишається такою самою.

Ключовою нормативною умовою є дотримання граничної дати захисту. Захист має бути призначений не пізніше ніж за 15 днів до завершення підготовки аспіранта. Якщо дата завершення підготовки дорівнює Dend, то верхня межа допустимого періоду захисту визначається як Dend - 15 днів [1; 2].

Тривалість засідання разової спеціалізованої вченої ради в одній аудиторії приймається не меншою ніж 180 хвилин. Тому під час перевірки конфліктів необхідно враховувати не тільки час початку захисту, а весь зайнятий інтервал. Одна аудиторія не може бути використана двома захистами в періодах, що перетинаються.

Призначення допускається лише тоді, коли часовий інтервал належить нормативно допустимому періоду, аудиторія є активною, а всі члени відповідної ради доступні. Член разової ради не може одночасно брати участь у декількох захистах. Додатково перевіряється відповідність складу РСВР: можливий склад із двома рецензентами і двома офіційними опонентами або з одним рецензентом і трьома офіційними опонентами. Голова та рецензенти мають належати до ІФНТУНГ, а офіційні опоненти не можуть належати до ІФНТУНГ і не повинні представляти один і той самий заклад [2].

Під час перепланування погоджені, опубліковані або вручну зафіксовані призначення повинні залишатися незмінними. Для інших подій необхідно мінімізувати відхилення нового графіка від попередньої версії. Цільову функцію можна подати як суму штрафів за зміну попереднього призначення, невраховані побажання учасників і нерівномірне навантаження. Найбільшу вагу доцільно надати зміні погоджених призначень, оскільки стабільність графіка є важливою вимогою динамічного перепланування [24].

Для розв'язання поставленої задачі необхідно виконати такі етапи:

- розробити модуль імпорту CSV-файлів ЄДЕБО з перевіркою структури, кодування, значень і дублікатів.

- спроєктувати базу даних для збереження аспірантів, користувачів, етапів, нормативних правил, складів разових спеціалізованих вчених рад, доступності учасників, документів, аудиторій, операційних налаштувань і версій графіка.

- реалізувати рольову модель доступу для адміністратора, оператора відділу аспірантури, аспіранта, наукового керівника та члена разової ради.

- реалізувати формування індивідуального плану атестації відповідно до нормативних строків [1; 2].

- розробити алгоритм формування розкладу захистів із перевіркою жорстких і м'яких обмежень.

- реалізувати динамічне перепланування зі збереженням зафіксованих подій та мінімізацією кількості змін.

- забезпечити діагностування причин, через які допустимий графік неможливо сформувати.

- розробити web-інтерфейс для перегляду аспірантів, етапів, календаря, конфліктів, результатів імпорту, складу РСВР і системних налаштувань.

- реалізувати підготовку проєктів інформаційних матеріалів і публічної сторінки про разову спеціалізовану вчену раду.

- забезпечити контейнерне розгортання web-сервісу та PostgreSQL із використанням Docker Compose [18].

- провести функціональне, інтеграційне та алгоритмічне тестування розробленої системи.

- алгоритм планування доцільно реалізувати як детермінований пошук із поверненням та відсіканням неперспективних варіантів. Першими повинні опрацьовуватися аспіранти з найменшою кількістю допустимих інтервалів. Така евристика дає змогу раніше виявляти конфлікти та зменшувати кількість досліджуваних варіантів. Підхід відповідає обмежувальній природі задачі та забезпечує однаковий результат за однакових вхідних даних [21; 23].

Критеріями правильності роботи системи визначено:

- відсутність дублікатів після повторного імпорту;

- правильне обчислення нормативних строків і граничної дати захисту Dend - 15 днів;

- відсутність одночасних призначень для членів рад і аудиторій;

- коректна перевірка складу разової спеціалізованої вченої ради;

- урахування тривалості засідання та операційних налаштувань планування;

- збереження зафіксованих подій під час перепланування;

- дотримання рольових обмежень доступу;

- відтворюваність результату планування;

- формування пояснення за відсутності допустимого рішення;

- збереження історії змін графіка та працездатність публічного представлення РСВР.

Таким чином, задача бакалаврської роботи полягає не лише у формуванні календаря захистів, а в розробленні цілісного web-сервісу, який перетворює дані з ЄДЕБО, нормативні правила, склад разових рад, відомості про аудиторії та доступність учасників на нормативно коректний і стабільний графік. Система повинна підтримувати весь цикл роботи з графіком: від перевірки вхідних даних до перепланування, діагностування конфліктів і підготовки матеріалів для оприлюднення.

# 2 ПРОЄКТУВАННЯ WEB-ДОДАТКА СУПРОВОДУ АТЕСТАЦІЇ АСПІРАНТІВ І ПЛАНУВАННЯ ЗАХИСТІВ

## 2.1 Сценарії використання, архітектура та рольова модель доступу

Проєктування web-додатка виконано відповідно до функціональних вимог, сформульованих у пункті 1.5, та фактичного процесу супроводу атестації аспіранта від імпорту даних ЄДЕБО до підготовки матеріалів про разову спеціалізовану вчену раду. Система повинна підтримувати не лише календар захистів, а повний службовий контур: картку аспіранта, нормативні етапи, склад ради, аудиторію, документи, версії графіка, публікацію та аудит дій [1; 2].

Основним користувацьким сценарієм є робота оператора відділу аспірантури і докторантури. Оператор завантажує CSV-файл ЄДЕБО, перевіряє результат імпорту, відкриває картку конкретного аспіранта, уточнює дату завершення підготовки, формує або коригує індивідуальний план, заповнює склад РСВР, вибирає або дозволяє автоматично вибрати аудиторію захисту та створює версію графіка. Після настання фактичної події оператор фіксує її дату, а система перераховує залежні етапи й перевіряє, чи не порушено нормативні строки.

Для системи передбачено такі ролі: адміністратор, оператор відділу аспірантури, аспірант, науковий керівник і член разової ради. Адміністратор керує користувачами, ролями та службовими налаштуваннями. Оператор працює з імпортом, картками, етапами, радами, графіками й публічними матеріалами. Аспірант переглядає власну траєкторію атестації, науковий керівник контролює пов'язані з ним етапи, а член ради має доступ лише до засідань і матеріалів відповідної ради. Діаграму варіантів використання наведено на рисунку 2.1.

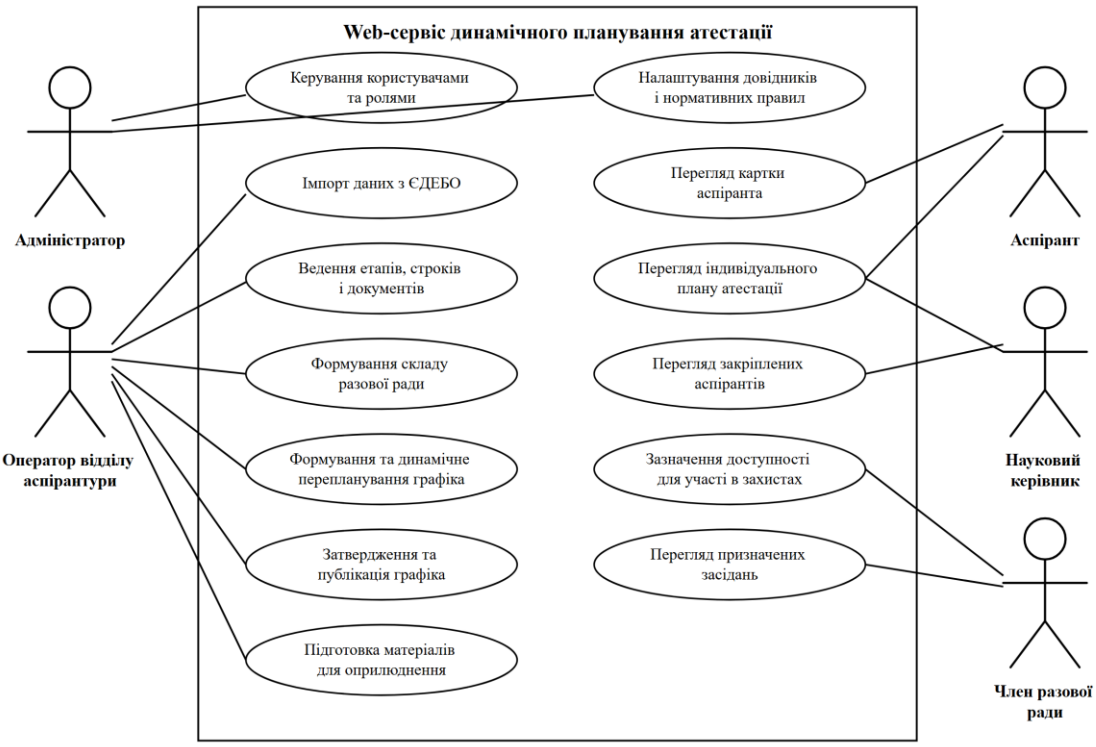


Рисунок 2.1 – Діаграма варіантів використання web-додатка

Як видно з рисунка 2.1, найширший набір операцій належить адміністратору й оператору, оскільки саме ці ролі відповідають за службову достовірність даних. Інші ролі отримують обмежений доступ до власних або пов'язаних із ними записів, що відповідає принципу мінімально необхідних повноважень [14].

Архітектуру реалізовано як модульний моноліт: усі функціональні частини працюють в одному FastAPI-застосунку, але поділені на маршрути, сервіси, репозиторії, ORM-моделі, шаблони та статичні файли. Такий підхід є доцільним для бакалаврського проєкту, оскільки зменшує інфраструктурну складність, але залишає можливість окремо тестувати імпорт, нормативний контроль, планувальник, авторизацію та публікацію [25].


Рисунок 2.2 – Архітектура web-додатка

Клієнтська частина складається з HTML-сторінок, CSS-оформлення та JavaScript-логіки, які взаємодіють із REST API. Серверна частина FastAPI обробляє запити, перевіряє JWT-токен і роль користувача, викликає предметні сервіси та зберігає дані в PostgreSQL через SQLAlchemy. Web-сервіс і база даних розгортаються в окремих контейнерах Docker Compose [18; 25; 26].

Таблиця 2.1 – Основні компоненти web-додатка

| Компонент | Призначення | Основні вхідні дані | Результат |
|---|---|---|---|
| Authentication service | Автентифікація, JWT-токени, перевірка ролей і корпоративної пошти | Google/OIDC-дані, роль користувача | Авторизований користувач або відмова |
| Import service | Перевірка та застосування CSV-експорту ЄДЕБО | CSV-файл, обов'язкові колонки | Звіт імпорту, додані й оновлені картки |
| Attestation service | Формування 20 етапів, строків і нормативних ризиків | Картка аспіранта, фактичні дати | Індивідуальний план атестації |
| Council service | Ведення складу РСВР і перевірка ролей учасників | Голова, рецензенти, опоненти, установи | Перевірений склад ради |

Продовження табл. 2.1

| Компонент | Призначення | Основні вхідні дані | Результат |
|---|---|---|---|
| Schedule service | Побудова та перепланування графіка захистів | Аспіранти, слоти, аудиторії, ради, попередній графік | Версія графіка або діагностика конфліктів |
| Settings service | Редагування операційних параметрів планування | Аудиторії, тривалість, стартові години, ліміти | Актуальні налаштування системи |
| Publication service | Підготовка відкритих матеріалів РСВР | Дані ради, захисту, документи, посилання | Публічна сторінка або службовий проєкт |
| Audit service | Фіксація критичних змін | Користувач, операція, сутність | Запис журналу аудиту |

## 2.2 Проєктування моделі даних та структури бази даних

Модель даних побудовано навколо аспіранта як центральної сутності. Саме з аспірантом пов'язані дата завершення підготовки, індивідуальні етапи, документи, склад разової ради, публічні матеріали та події графіка. Для зберігання даних використовується PostgreSQL, а відображення таблиць на об'єкти Python виконується засобами SQLAlchemy [17; 26].

Під час проєктування враховано вимогу мінімізації персональних даних: із CSV-файлів ЄДЕБО до системи переносяться лише ті поля, які потрібні для ідентифікації аспіранта, формування плану та контролю строків. Відомості, що не використовуються в алгоритмі планування або службовому супроводі, не зберігаються в базі [11; 14].

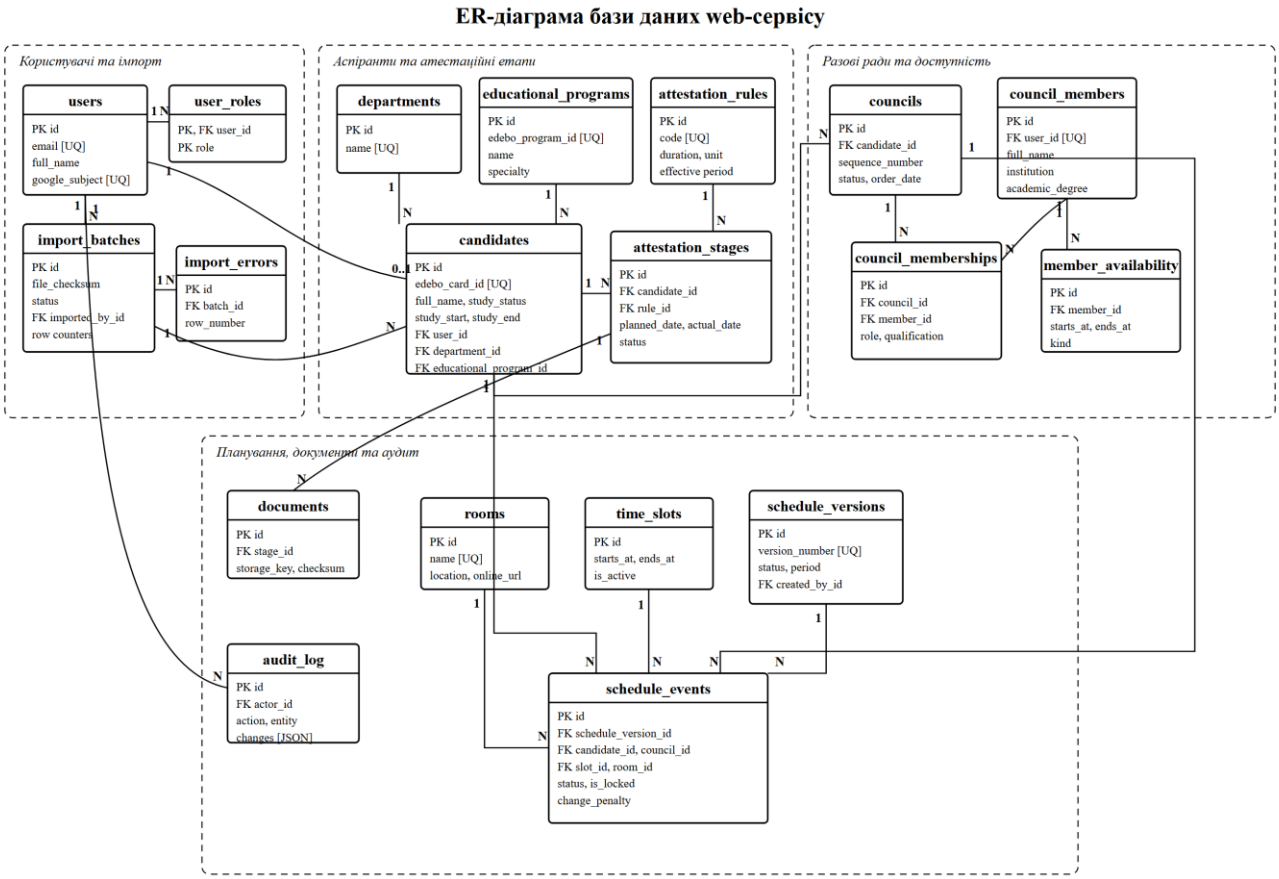


Рисунок 2.3 – ER-діаграма бази даних web-додатка

Схему бази даних наведено на рисунку 2.3. Уточнена модель містить 20 основних таблиць. Порівняно з попереднім варіантом до моделі додано сутність AppSetting, оскільки частина параметрів не повинна бути зашита в коді: тривалість захисту, стартові години, горизонт планування, максимальне денне навантаження члена ради та перелік активних аудиторій редагуються через службове меню.

Таблиця candidates містить унікальний ідентифікатор картки ЄДЕБО, ПІБ, статус навчання, дати початку та завершення підготовки, форму навчання, курс, групу та ознаку поновлення для захисту. Дата завершення підготовки є опорною: повторний імпорт не повинен змінювати її без явної дії оператора, оскільки від неї залежить гранична дата захисту.

Таблиці attestation_rules і attestation_stages забезпечують формування індивідуального плану. Правило містить код, базову подію, напрям обчислення, тривалість, одиницю вимірювання, нормативне посилання та період чинності.

Етап зберігає планову дату, фактичну дату, граничну дату, статус, відповідального та зв'язок із документами.

Таблиця 2.2 – Основні групи таблиць бази даних

| **Група** | **Таблиці** | **Призначення** |
|---|---|---|
| Користувачі | users, user_roles, audit_log | Облікові записи, ролі та журнал критичних дій. |
| Імпорт | import_batches, import_errors | Пакети CSV-імпорту, контрольні суми, статистика та помилки перевірки. |
| Аспіранти | candidates, departments, educational_programs | Картка аспіранта, підрозділ, освітньо-наукова програма, дата завершення підготовки. |
| Етапи | attestation_rules, attestation_stages, documents | Нормативні правила, планові й фактичні дати, документи та посилання. |
| РСВР | councils, council_members, council_memberships, member_availability | Склад разової ради, ролі, установи, доступність і публічні реквізити. |
| Графік | rooms, time_slots, schedule_versions, schedule_events | Аудиторії, часові слоти, версії графіка та події захистів. |
| Налаштування | app_settings | Операційні параметри, які змінюються адміністратором без редагування коду. |

Сутності councils, council_members і council_memberships дають змогу зберігати склад РСВР. У моделі передбачено два допустимі варіанти складу: голова, два рецензенти і два офіційні опоненти або голова, один рецензент і три офіційні опоненти. Голова та рецензенти повинні належати до ІФНТУНГ, офіційні опоненти не можуть належати до ІФНТУНГ і не повинні бути з одного закладу [2].

Таблиці rooms, time_slots, schedule_versions і schedule_events забезпечують версіонування графіка. Подія графіка пов'язує аспіранта, раду, часовий слот і аудиторію. Для чернеткових записів допускається видалення, якщо вони створені помилково. Для погоджених або опублікованих графіків доцільно

використовувати статуси approved, published або cancelled, щоб зберігати історію прийнятих рішень.

AppSetting зберігає параметри, які безпосередньо впливають на планувальник: тривалість засідання РСВР, стартові години, горизонт планування, перелік активних аудиторій і максимальну кількість захистів члена ради протягом дня. Така таблиця усуває суперечність між проєктом і реалізацією, де ці параметри вже редагуються на сторінці налаштувань.

### 2.3 Математична модель задачі планування

Задача планування полягає у виборі дати, часу й аудиторії захисту для одного обраного аспіранта або для множини аспірантів з урахуванням нормативного вікна, складу РСВР, доступності учасників і вже зафіксованих подій. Модель має підтримувати як індивідуальне формування графіка в картці аспіранта, так і групове планування календаря захистів.

Нехай C - множина аспірантів, T - множина часових слотів, R - множина аудиторій, M - множина членів рад, Mc - склад ради аспіранта c, Ac - множина нормативно допустимих слотів для аспіранта c, а Pc - множина бажаних слотів. Якщо оператор працює з одним аспірантом, множина C містить один елемент, але всі обмеження залишаються тими самими.

Ключовою нормативною умовою є верхня межа вікна захисту: захист має бути призначений не пізніше ніж за 15 днів до дати завершення підготовки аспіранта. Якщо дата завершення підготовки дорівнює Dend, то гранична дата захисту визначається як Ddef_max = Dend - 15 днів. Слот, що виходить за цю межу, не включається до множини Ac [1; 2].

Тривалість засідання РСВР приймається як налаштовуваний параметр L. У демонстраційній конфігурації L = 180 хвилин, оскільки захист дисертації в одній аудиторії зазвичай триває не менше трьох годин. Під час перевірки конфліктів порівнюється не лише момент початку, а весь інтервал [start, start + L].

Бінарна змінна xctr дорівнює 1, якщо аспіранту c призначено слот t та аудиторію r, і 0 в іншому випадку. Для кожного аспіранта має бути сформовано одне призначення, якщо множина допустимих слотів не порожня. Якщо Ac порожня, система не повинна створювати некоректний графік, а має повернути діагностичну причину.

Таблиця 2.3 – Жорсткі обмеження задачі планування

| **Позначення** | **Зміст обмеження** |
|---|---|
| H1 | Для кожного аспіранта формується не більше одного активного призначення захисту. |
| H2 | Слот належить нормативному вікну, зокрема дата захисту не пізніше Dend - 15 днів. |
| H3 | Тривалість засідання становить не менше 180 хвилин або інше значення, задане в налаштуваннях. |
| H4 | Одна аудиторія не може бути використана двома захистами в інтервалах, що перетинаються. |
| H5 | Член РСВР не може одночасно брати участь у двох захистах. |
| H6 | Склад ради відповідає одному з допустимих варіантів 2+2 або 1+3 для рецензентів і опонентів. |
| H7 | Голова й рецензенти належать до ІФНТУНГ, опоненти є зовнішніми та представляють різні заклади. |
| H8 | Зафіксовані події не переносяться під час автоматичного перепланування. |

М'які обмеження не визначають допустимість графіка, але впливають на вибір найкращого варіанта. До них належать мінімізація кількості змін під час перепланування, перевага раніше узгоджених слотів, рівномірніше денне навантаження членів рад і використання пріоритетних аудиторій. Ваги цих критеріїв можуть бути винесені до системних налаштувань.


Рисунок 2.4 – Математична модель формування графіка захистів

Як видно з рисунка 2.4, формування графіка починається з нормативної фільтрації слотів, після чого перевіряються аудиторії, склад ради, доступність учасників і фіксовані події. Якщо всі жорсткі обмеження виконуються, алгоритм оцінює м'які критерії й обирає варіант із найменшою штрафною оцінкою.

## 2.4 Алгоритм формування та динамічного перепланування графіка

Для розв'язання задачі використовується детермінований пошук із поверненням, MRV-евристикою та відсіканням неперспективних варіантів. Алгоритм може працювати у двох режимах: сформувати графік для конкретного аспіранта або перебудувати графік для множини аспірантів. Обидва режими використовують однакові перевірки, тому індивідуальне планування не суперечить груповому календарю.

На вхід алгоритму передаються: перелік аспірантів, склад рад, активні аудиторії, стартові години, тривалість засідання, горизонт планування, попередня версія графіка, зафіксовані події, заблоковані події та доступність

членів рад. Частина цих значень береться з AppSetting, тому адміністратор може змінити їх через web-інтерфейс без редагування програмного коду.

Перед пошуком алгоритм обчислює нормативне вікно кожного аспіранта. Верхня межа визначається датою завершення підготовки мінус 15 днів. Нижня межа залежить від фактичного стану попередніх етапів, зокрема утворення ради, оприлюднення матеріалів, надходження рецензій і відгуків. Якщо оператор фіксує фактичну дату події, залежні етапи та допустиме вікно перераховуються автоматично.

Алгоритм 2.1 – Формування та перепланування графіка захистів

Вхід: аспіранти C, слоти T, аудиторії R, ради M, налаштування P, попередній графік S0

Вихід: версія графіка S або INFEASIBLE із поясненням

1. Завантажити активні аудиторії, стартові години, тривалість і ліміти з AppSetting.

2. Для кожного аспіранта обчислити нормативне вікно захисту.

3. Перевірити склад РСВР і відкинути аспірантів із некоректним складом ради.

4. Виділити зафіксовані події та перевірити їх на конфлікти.

5. Для кожного незапланованого аспіранта сформувати допустимі пари слот-аудиторія.

6. Якщо допустимих пар немає, повернути діагностичне повідомлення.

7. Упорядкувати аспірантів за мінімальною кількістю варіантів.

8. Послідовно вибирати варіанти, перевіряючи аудиторії, членів рад, тривалість і денні ліміти.

9. Для кожного допустимого варіанта нарахувати штраф за зміну попереднього призначення.

10. Зберегти найкращий повний графік як нову версію або повернути INFEASIBLE.

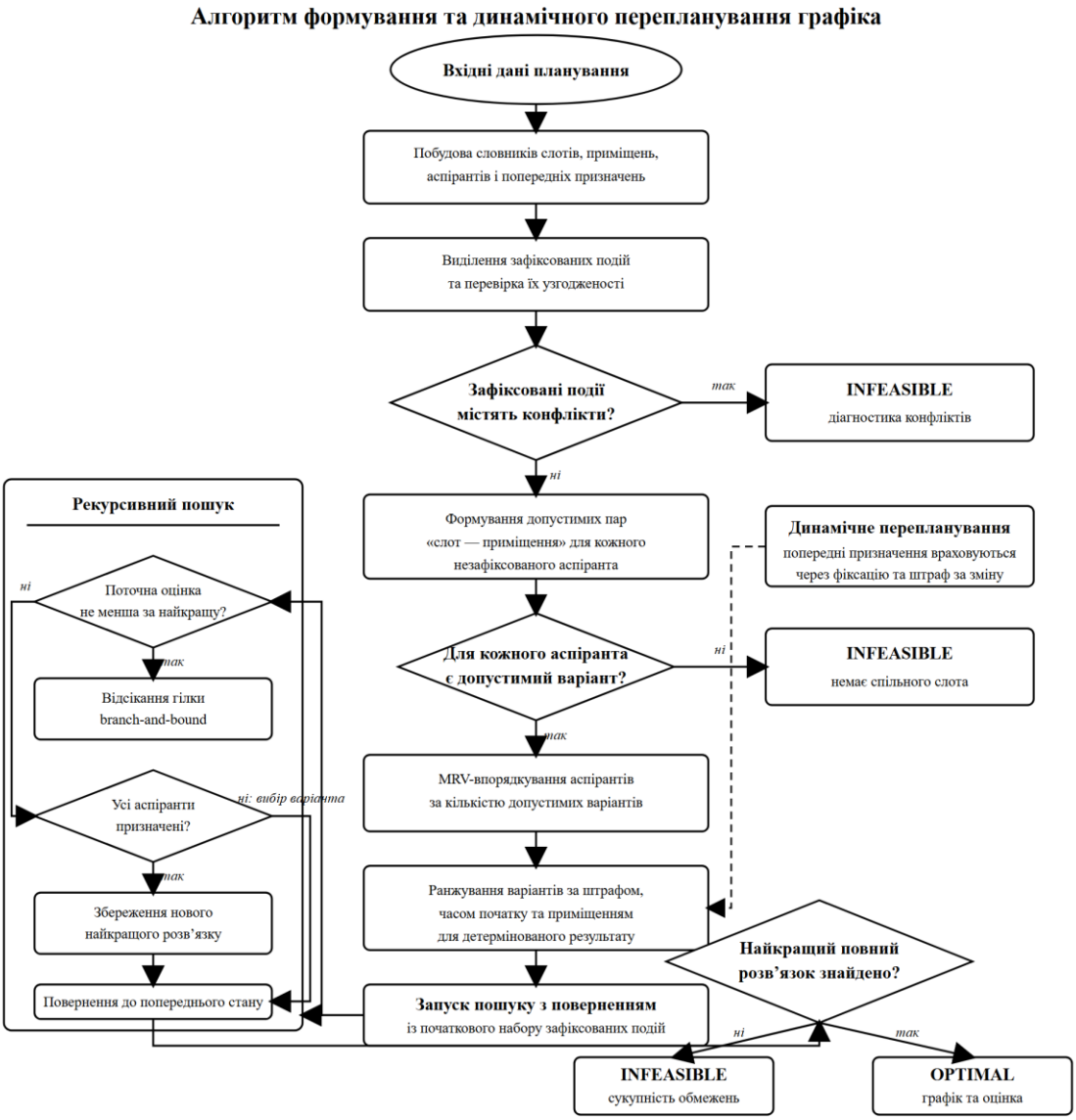


Рисунок 2.5 – Алгоритм формування та динамічного перепланування графіка

Динамічне перепланування виконується повторним запуском того самого алгоритму з актуальними даними та попередньою версією графіка. Зафіксовані події залишаються незмінними, заблоковані вилучаються з доступних варіантів, а зміна незафіксованих подій отримує штраф. Такий підхід забезпечує стабільність графіка й водночас дозволяє реагувати на нові фактичні дати, зміну доступності або додавання аудиторій.

Якщо допустимий графік сформувати неможливо, система повинна повернути не загальну помилку, а конкретну причину: вихід за нормативне вікно, відсутність активної аудиторії, накладання засідань у приміщенні, одночасна участь члена ради, некоректний склад РСВР або конфлікт із зафіксованою

подією. Ця вимога безпосередньо пов'язана з практичною роботою оператора, який має знати, які дані потрібно уточнити.

### 2.5 Проєктування web-інтерфейсу та операційних налаштувань

Оскільки система призначена для щоденної роботи відділу аспірантури і докторантури, інтерфейс спроєктовано як адміністративну панель українською мовою. Основні розділи відповідають реальному порядку роботи: панель, імпорт ЄДЕБО, аспіранти, картка аспіранта, контроль строків, календар захистів, налаштування та документація API.

У картці аспіранта зосереджено ключову предметну логіку. Саме тут оператор бачить дату завершення підготовки, допустиме вікно захисту, 20 етапів атестації, фактичні дати, документи, склад РСВР, реквізити ради, публічні посилання та блок формування графіка. Така структура зручніша за окреме розкидання даних по різних сторінках, оскільки всі критичні рішення ухвалюються щодо конкретного аспіранта.

Таблиця 2.4 – Основні екрани web-інтерфейсу

| **Екран** | **Основні функції** | **Пов'язані модулі** |
|---|---|---|
| Панель | Статистика аспірантів, етапів, рад, графіків і найближчих ризиків | Attestation, Schedule |
| Імпорт ЄДЕБО | Перевірка CSV, звіт помилок, застосування імпорту | Import |
| Аспіранти | Пошук, відкриття картки, видалення помилково створених записів | Candidate |
| Картка аспіранта | Етапи, РСВР, документи, аудиторія, індивідуальне планування | Attestation, Council, Schedule |
| Контроль строків | Перелік ризиків, прострочених і критичних етапів | Attestation |
| Календар захистів | Версії графіків, конфлікти, погодження, публікація, видалення чернеток | Schedule |
| Налаштування | Аудиторії, тривалість, стартові години, горизонт планування, денні ліміти | Settings |
| Публічна сторінка | Коротка й повна інформація про утворені РСВР | Publication |

Окремої уваги потребує сторінка налаштувань. Вона повинна містити не лише довідковий текст, а робочі елементи для керування параметрами: додавання або вимкнення аудиторій, редагування місткості й посилання на трансляцію, зміна тривалості засідання РСВР, стартових годин, горизонту планування та максимального навантаження члена ради на день. Саме ці параметри використовуються планувальником під час побудови графіка.

На головній відкритій сторінці доцільно показувати інформацію про утворені разові спеціалізовані вчені ради. Короткий блок містить номер ради, ПІБ здобувача, спеціальність, дату й аудиторію захисту. Повна інформація відкривається через випадаючий блок і містить склад ради, реквізити наказу, посилання на дисертацію, рецензії, відгуки, трансляцію та інші матеріали, передбачені порядком оприлюднення [2].

### 2.6 Проєктування API, безпеки та публікації

Взаємодію клієнтської частини з прикладною логікою передбачено через REST API з префіксом /api/v1. FastAPI автоматично формує OpenAPI-специфікацію, що спрощує тестування маршрутів і перевірку структури запитів [25; 30]. Дані передаються у форматі JSON, а CSV-файли й документи - через multipart/form-data.

Автентифікація у продуктивному сценарії має виконуватися через корпоративну поштову адресу Google Workspace із використанням OAuth 2.0 та OpenID Connect. Сервер перевіряє підпис і термін дії токена, значення iss, aud, sub, підтвердження електронної адреси та домен організації. Для доступу до прикладного API використовується JWT-токен, який передається в заголовку Authorization [16; 31; 32].

Авторизація поєднує рольову модель і перевірку належності об'єкта. Наприклад, аспірант може переглядати власну траєкторію, науковий керівник - записи пов'язаних здобувачів, член ради - матеріали засідань, до яких його

залучено, а оператор - службові картки, етапи, ради та графіки. Зміна ролей і системних налаштувань лишається функцією адміністратора.

Таблиця 2.5 – Основні групи маршрутів API

| **Група маршрутів** | **Призначення** | **Доступ** |
|---|---|---|
| Авторизація | Демо-токен, Google OAuth/OIDC, перевірка ролей | Усі користувачі, адміністратор |
| Імпорт | Перевірка та застосування CSV-файлів ЄДЕБО | Адміністратор, оператор |
| Аспіранти | Список, картка, редагування, видалення помилкових записів | Адміністратор, оператор |
| Етапи | Фіксація фактичних дат, документи, перерахунок плану | Адміністратор, оператор, пов'язані ролі |
| Ради | Склад РСВР, перевірка ролей, службові матеріали | Адміністратор, оператор |
| Графіки | Створення, перегляд, перепланування, погодження, публікація, видалення чернеток | Адміністратор, оператор |
| Налаштування | Аудиторії, тривалість, стартові години, ліміти | Адміністратор |
| Публікація | Відкрита сторінка РСВР та оприлюднені матеріали | Публічний доступ |

Видалення в API розмежовується за станом сутності. Помилково імпортованого аспіранта або чернеткову версію графіка можна видалити разом із пов'язаними чернетковими даними. Натомість погоджені й опубліковані матеріали мають не фізично знищуватися, а переводитися в стан скасування або архіву, щоб зберегти простежуваність процедури.

Завантаження файлів потребує окремого контролю: перевірки розширення, MIME-типу, розміру, контрольної суми та структури. Початкова назва файлу не повинна використовуватися як шлях зберігання. Метадані документа зберігаються в PostgreSQL, а сам файл - у захищеному сховищі або постійному томі контейнера [29].

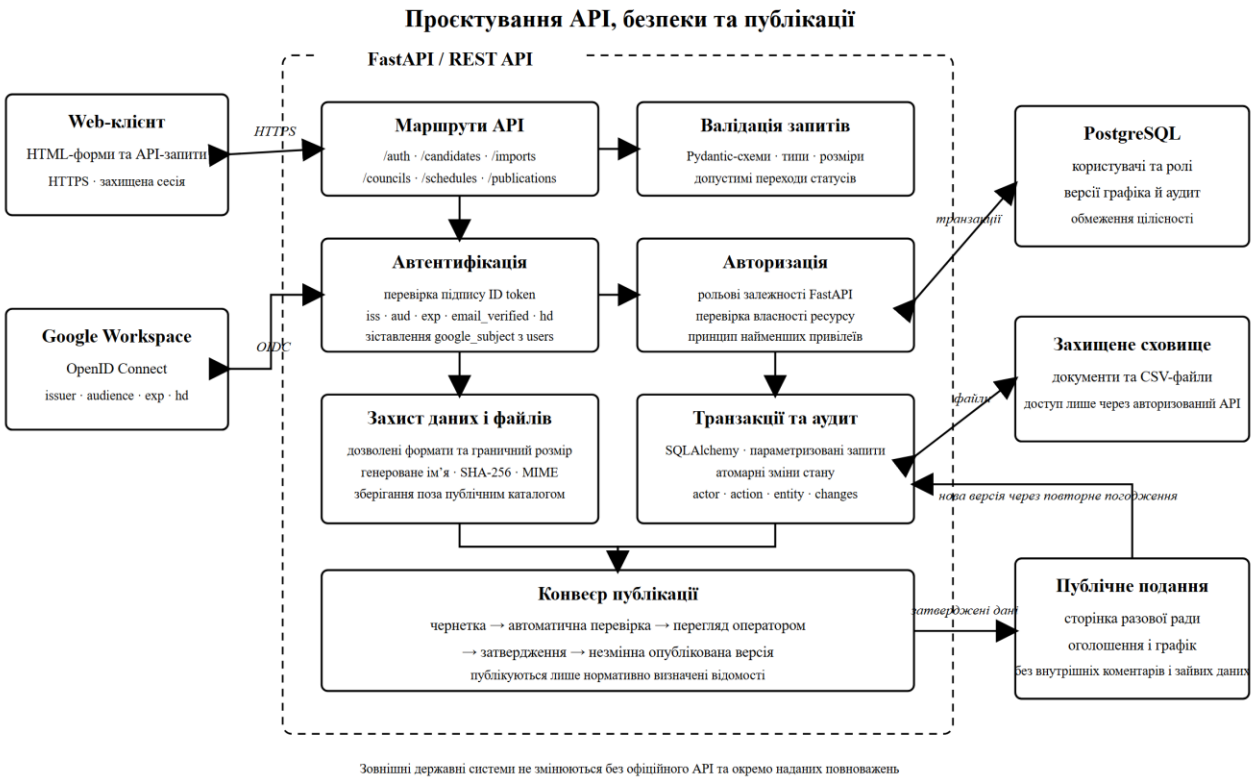


Рисунок 2.6 – API, безпека та публікація web-додатка

Як видно з рисунка 2.6, запит проходить послідовні етапи: автентифікацію, перевірку ролі, валідацію даних, виконання бізнес-операції, транзакційне збереження в PostgreSQL та запис у журнал аудиту. Публічне представлення формується лише з перевірених і погоджених даних, щоб уникнути передчасного оприлюднення службових або неповних матеріалів [2; 14].

Контейнерне розгортання передбачає два основні сервіси: app і db. Контейнер app містить FastAPI-застосунок, статичні файли, шаблони та міграції Alembic; контейнер db містить PostgreSQL. Docker Compose описує мережу, том бази даних, healthcheck і порядок запуску, що забезпечує відтворюваність середовища розробки та демонстраційного розгортання [18].

Отже, уточнений проєкт розділу 2 відповідає реалізації, описаній у розділі 3: база даних містить операційні налаштування, планувальник враховує індивідуальне формування графіка, 15-денне нормативне обмеження, тривалість засідання, аудиторії та склад РСВР, а API підтримує фактичні операції web-

додатка, включаючи видалення чернеткових записів і публікацію інформації про разові ради.

**Висновки до розділу 2**

У другому розділі виконано проєктування web-сервісу супроводу атестації аспірантів і планування захистів дисертацій. На основі вимог, сформульованих у першому розділі, визначено функціональну структуру системи, основні ролі користувачів, інформаційні потоки та склад програмних модулів.

Обґрунтовано архітектуру web-додатка, у якій клієнтська частина забезпечує взаємодію користувача із системою, серверна частина на FastAPI реалізує бізнес-логіку, а база даних PostgreSQL використовується для збереження аспірантів, складів разових спеціалізованих вчених рад, етапів атестації, версій графіків, документів, налаштувань і результатів перевірок. Використання Docker Compose передбачено для відтворюваного розгортання сервісу разом із базою даних.

Спроєктовано модель даних, яка охоплює ключові сутності предметної області: аспірантів, користувачів, ролі доступу, нормативні правила, етапи атестації, членів разових рад, аудиторії, події графіка, документи та версії розкладу. Така структура дає змогу не лише зберігати довідкові й операційні дані, а й підтримувати контроль строків, виявлення конфліктів та повторне формування графіка після зміни вихідних умов.

Розроблено логіку рольового доступу для адміністратора, оператора відділу аспірантури, аспіранта, наукового керівника та члена разової ради. Це забезпечує розмежування прав на імпорт даних, редагування карток аспірантів, формування складу ради, перегляд індивідуальних етапів, погодження або публікацію графіків і роботу з інформаційними матеріалами.

У розділі також визначено підхід до проєктування API, безпеки та публікації сервісу. Передбачено використання REST API для взаємодії між інтерфейсом і серверною логікою, корпоративної авторизації через поштові облікові записи, перевірки доступу за ролями та контейнерного розгортання.

Запропонована структура API створює основу для реалізації імпорту CSV-файлів ЄДЕБО, формування індивідуальних планів атестації, планування захистів, діагностування конфліктів і підготовки інформаційних повідомлень про РСВР.

Отже, у другому розділі сформовано проєктну основу для реалізації програмного продукту: визначено архітектуру, модель даних, ролі користувачів, основні сценарії роботи, склад API та принципи розгортання. Результати проєктування безпосередньо використовуються у третьому розділі під час програмної реалізації web-сервісу, алгоритму планування та інтерфейсу користувача.

# 3 РЕАЛІЗАЦІЯ WEB-ДОДАТКА СУПРОВОДУ АТЕСТАЦІЇ АСПІРАНТІВ

## 3.1 Загальна характеристика реалізації

У третьому розділі розглянуто практичну реалізацію web-додатка, призначеного для супроводу атестації аспірантів і планування захистів дисертацій доктора філософії. Реалізація виконана як web-сервіс із серверною частиною FastAPI, базою даних PostgreSQL, клієнтським інтерфейсом на HTML/CSS/JavaScript та контейнерним розгортанням у Docker Compose. Головна ідея реалізації полягає не в простому веденні списку аспірантів, а в контролі нормативних строків і автоматичному перерахунку плану після фіксації фактичної події відділом аспірантури.

Нормативну основу прикладної логіки становлять Порядок підготовки здобувачів вищої освіти ступеня доктора філософії та доктора наук, затверджений постановою Кабінету Міністрів України № 261, і Порядок присудження ступеня доктора філософії, затверджений постановою Кабінету Міністрів України № 44 [1; 2]. Саме тому опорною датою у системі є дата завершення підготовки аспіранта: вона визначає граничне вікно захисту, не повинна неявно змінюватися повторним CSV-імпортом і використовується для виявлення ризику порушення строків.

У реалізованому додатку кожна функція з технічного завдання прив'язана до окремого модуля або групи модулів. Узагальнену відповідність подано в таблиці 3.1.

**Таблиця 3.1 - Відповідність поставлених задач реалізованим модулям**

| Задача | Реалізація в додатку | Файли / модулі |
|---|---|---|
| Імпорт CSV ЄДЕБО | Перевірка кодування, структури, дат, дублікатів і чутливих полів; окремі режими перевірки та застосування імпорту. | app/services/edebo_import.py; app/repositories/imports.py |
| База даних | Збереження аспірантів, етапів, рад, учасників, доступності, документів, аудиторій і версій графіка. | app/models/entities.py; alembic/versions/*.py |
| Рольовий доступ | JWT-токени, демо-роль адміністратора, ролі admin/operator/candidate/supervisor/council_member. | app/security.py; app/services/google_auth.py |
| Індивідуальний план | 20 етапів атестації з плановими, фактичними та граничними датами; перерахунок після фіксації факту. | app/services/attestation.py |
| Графік захистів | Побудова версій графіка з урахуванням аудиторій, складів рад, тривалості засідання та нормативного вікна. | scheduler.py; app/services/scheduling.py |
| Динамічне перепланування | Фіксовані події залишаються без змін, заблоковані вилучаються, кількість змін мінімізується. | app/services/attestation.py; scheduler.py |
| Діагностика | Повідомлення про неможливість сформувати графік або про конфлікти аудиторій і членів рад. | diagnose_schedule_conflicts(); DynamicScheduler |
| Web-інтерфейс | Панель, імпорт, список аспірантів, картка, контроль строків, календар, налаштування, публічна сторінка РСВР. | app/templates/*.html; app/static/app.js; app/static/styles.css |
| Матеріали РСВР | Формування тексту службового повідомлення та публічних реквізитів ради. | build_council_material(); public_home.html |
| Docker | Два сервіси app/db, healthcheck, мережа, volume PostgreSQL. | Dockerfile; docker-compose.yml |
| Тестування | Модульні, інтеграційні та алгоритмічні тести; benchmark планувальника. | tests/*.py; run_benchmarks.py |

## 3.2 Структура web-додатка та розподіл відповідальності модулів

Архітектура реалізована за принципом розділення клієнтського інтерфейсу, HTTP API, сервісної бізнес-логіки, моделей бази даних і контейнерної інфраструктури. Це дозволяє змінювати правила планування або інтерфейс без переписування всієї системи. На рисунку 3.1 наведено деталізовану структуру web-додатка з основними файлами, які мають бути включені до додатків пояснювальної записки.

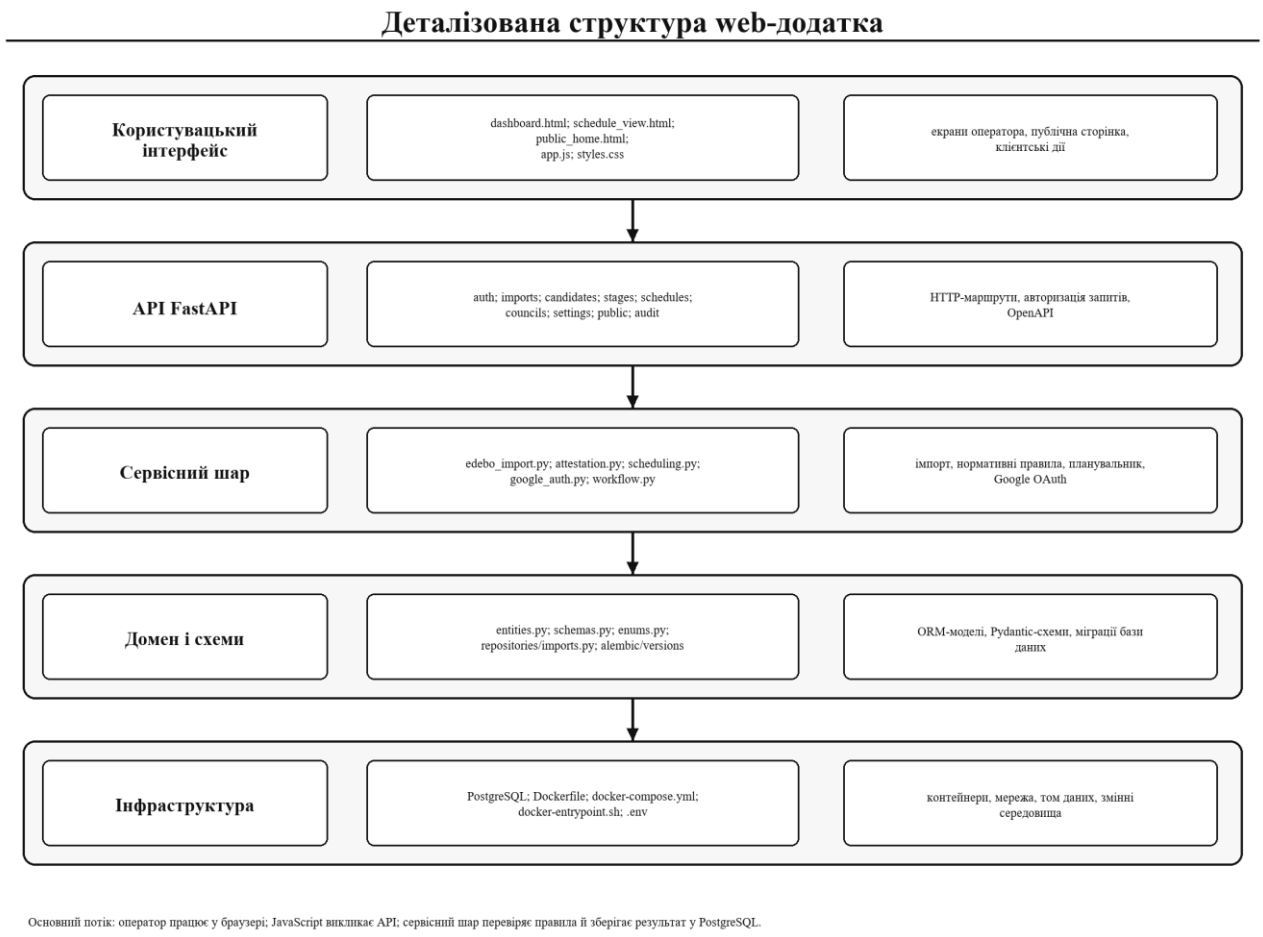


Рисунок 3.1 - Деталізована структура web-додатка

Файл app/api/routes.py виконує роль вхідної точки прикладного API: він містить маршрути авторизації, імпорту, роботи з аспірантами, етапами, графіками, радами, налаштуваннями та публічними сторінками. Сервіс app/services/attestation.py містить основну предметну логіку: формування 20 етапів атестації, нормативний контроль, роботу зі складом разової ради, створення графіка з бази даних і публікацію результатів. Сервіс app/services/edebo_import.py відповідає за розбір CSV-файлів, а scheduler.py містить незалежний алгоритм пошуку допустимого розкладу. Використання FastAPI забезпечує одночасне створення API та OpenAPI-документації [3].

Таблиця 3.2 - Структура програмних модулів реалізованого додатка

| Компонент | Призначення | Ключові файли |
|---|---|---|
| UI | Робоча панель оператора, публічна сторінка РСВР, перегляд графіка, валідація дій на клієнті. | app/templates/dashboard.html; public_home.html; schedule_view.html; app/static/app.js; styles.css |
| API | Маршрути REST, авторизація запитів, перетворення помилок у HTTP-відповіді. | app/api/routes.py |
| Імпорт | Розпізнавання кодування та роздільника, перевірка обов'язкових полів, підготовка результату імпорту. | app/services/edebo_import.py; app/repositories/imports.py |
| Атестація | Етапи, нормативні дати, картка аспіранта, РСВР, документи, аудит. | app/services/attestation.py |
| Планувальник | Пошук допустимих слотів, мінімізація змін, діагностика конфліктів. | scheduler.py; app/services/scheduling.py |
| Авторизація | JWT, ролі, Google OAuth для корпоративної пошти. | app/security.py; app/services/google_auth.py |
| Дані | ORM-моделі, перелічення, міграції, підключення PostgreSQL. | app/models/entities.py; app/models/enums.py; alembic/versions |

### 3.3 Контейнерне середовище розгортання

Для відтворюваного запуску сервіс розгортається у двох контейнерах: app і db. Контейнер app збирається з Dockerfile на базі python:3.13-slim, встановлює залежності з requirements.txt, працює від непривілейованого користувача appuser і запускає docker-entrypoint.sh. Контейнер db використовує образ postgres:17-alpine, зберігає дані у volume pg_data і перевіряє готовність через pg_isready. Така схема відповідає призначенню Docker Compose як засобу опису багатоконтейнерного застосунку в одному YAML-файлі [9].


Рисунок 3.2 - Схема контейнерного розгортання web-сервісу

Під час старту app очікує готовності PostgreSQL, після чого виконує міграції Alembic і запускає Uvicorn. Наявність healthcheck для app і db потрібна не лише для Docker Desktop, а й для подальшого розгортання в Coolify або іншому контейнерному середовищі: платформа може автоматично відстежувати, чи доступний endpoint /health, і перезапустити сервіс у разі відмови. Для FastAPI така контейнеризація є типовим способом підготовки web-сервісу до продуктивного запуску [11].

### 3.4 Структура бази даних

Схема PostgreSQL побудована навколо сутності Candidate, оскільки саме аспірант є носієм опорної дати, індивідуального плану, документів, складу разової ради та подій графіка. Для роботи з базою використано SQLAlchemy ORM, а зміни структури фіксуються міграціями Alembic [5; 6]. Узагальнену ER-структуру наведено на рисунку 3.3.

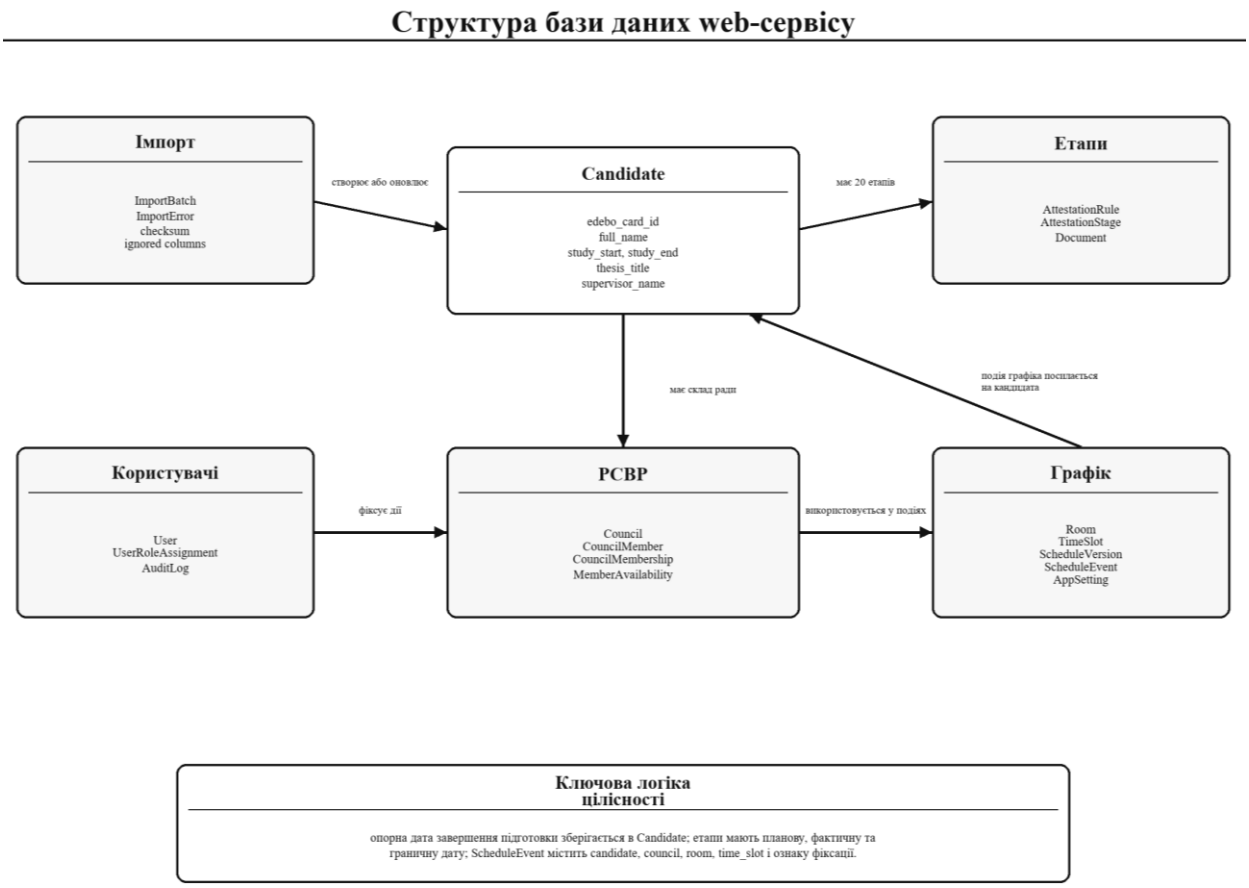


Рисунок 3.3 - Структура бази даних web-сервісу

**Таблиця 3.3 - Основні сутності бази даних**

| Група | Сутності | Призначення |
|---|---|---|
| Користувачі | User, UserRoleAssignment, AuditLog | Збереження облікових записів, ролей і журналу критичних дій. |
| Імпорт | ImportBatch, ImportError | Фіксація пакетів CSV-імпорту, помилок і статистики оброблення. |
| Аспіранти | Candidate, Department, EducationalProgram | Профіль аспіранта, освітньо-наукова програма, підрозділ, дата завершення підготовки. |
| Етапи | AttestationRule, AttestationStage, Document | Нормативні правила, планові/фактичні/граничні дати, зареєстровані документи. |
| РСВР | Council, CouncilMember, CouncilMembership, MemberAvailability | Склад разової ради, ролі учасників, доступність членів ради. |
| Графік | Room, TimeSlot, ScheduleVersion, ScheduleEvent, AppSetting | Аудиторії, слоти, версії графіка, події захистів і операційні налаштування. |

У моделі даних передбачено не лише довідкові поля, а й обмеження, важливі для алгоритму. ScheduleEvent пов'язує аспіранта, раду, аудиторію і

часовий слот. CouncilMembership задає роль учасника у конкретній раді. MemberAvailability дає змогу розширити алгоритм з урахуванням індивідуальної доступності. AppSetting зберігає операційні параметри, які редагуються через сторінку налаштувань: тривалість засідання, стартові години, горизонт планування і максимальне навантаження члена ради на день.

### 3.5 Реалізація модуля імпорту CSV-файлів ЄДЕБО

CSV-файли ЄДЕБО використовуються як початкове джерело даних про аспірантів. Реалізований імпорт не довіряє файлу автоматично: спочатку виконується перевірка, і лише після цього оператор застосовує зміни до бази. Такий поділ важливий для службової системи, оскільки помилка у даті завершення підготовки може призвести до неправильного нормативного графіка. Алгоритм модуля імпорту наведено на рисунку 3.4.

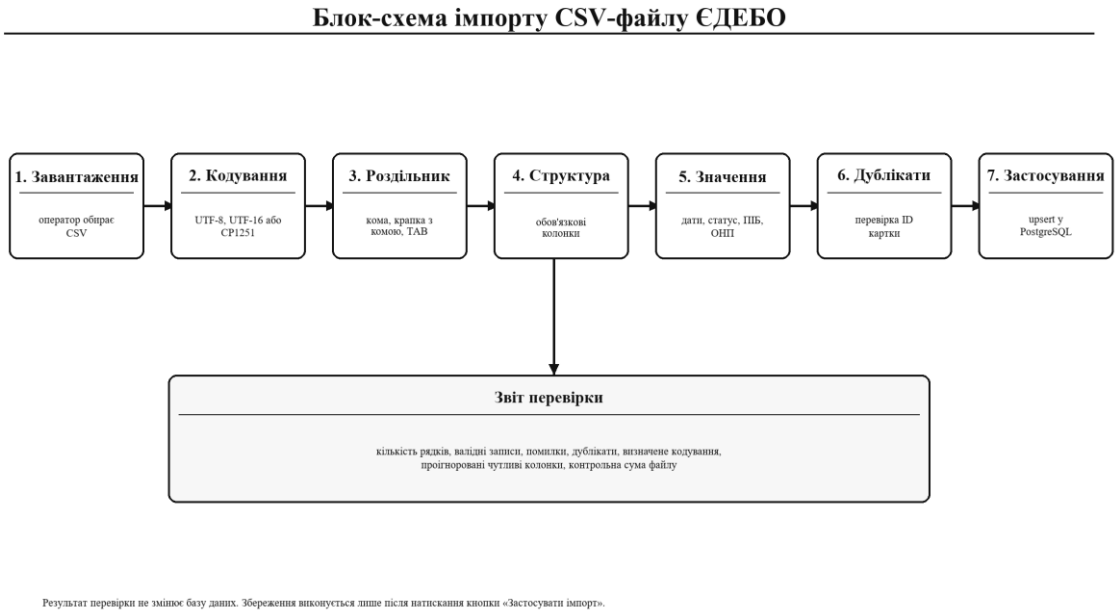


Рисунок 3.4 - Блок-схема імпорту CSV-файлу ЄДЕБО

Модуль parse_edebo_csv визначає кодування UTF-8, UTF-16 або CP1251, розпізнає роздільник, перевіряє наявність обов'язкових колонок, нормалізує значення дат і формує список зауважень. Для CSV-даних використано штатний підхід до табличного обміну, описаний у документації Python [4]. Повторний

імпорт не перезаписує опорну дату завершення підготовки без явної зміни в картці аспіранта; це захищає нормативний контур від випадкової втрати ключової дати.

Екран результату імпорту з поточного web-додатка наведено на рисунку 3.5.

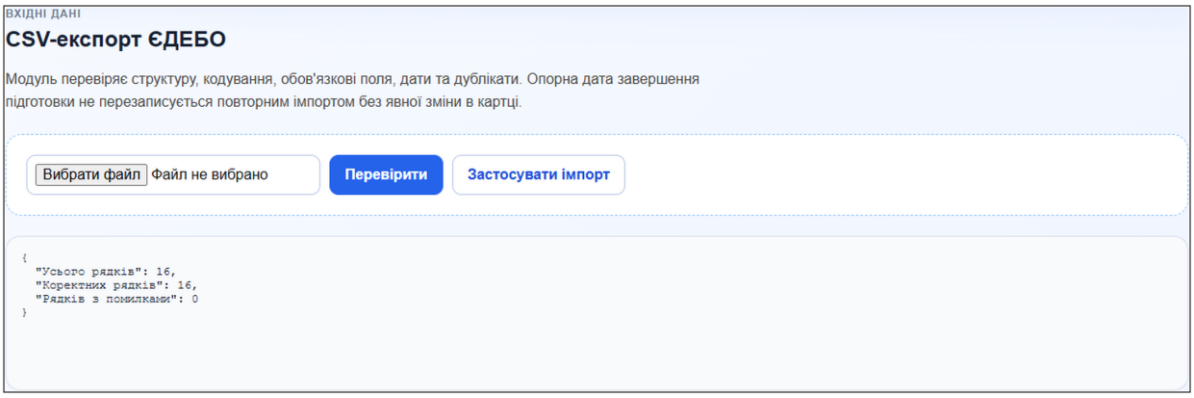


Рисунок 3.5 - Екран перевірки CSV-імпорту ЄДЕБО

### 3.6 Формування індивідуального плану атестації

Індивідуальний план атестації формується для кожного аспіранта окремо. У системі реалізовано 20 етапів, що відповідають фактичному процесу від оформлення дисертації до внесення рішення ради, відеозапису та матеріалів до відкритого доступу. Додаток не дублює теоретичні відомості про ці етапи, а використовує їх як прикладні контрольні точки, для яких потрібні планова, фактична та гранична дата.

Фактичну дату події фіксує відділ аспірантури. Після фіксації система перераховує залежні майбутні етапи і повторно перевіряє нормативний стан. Цей процес показано на рисунку 3.6.

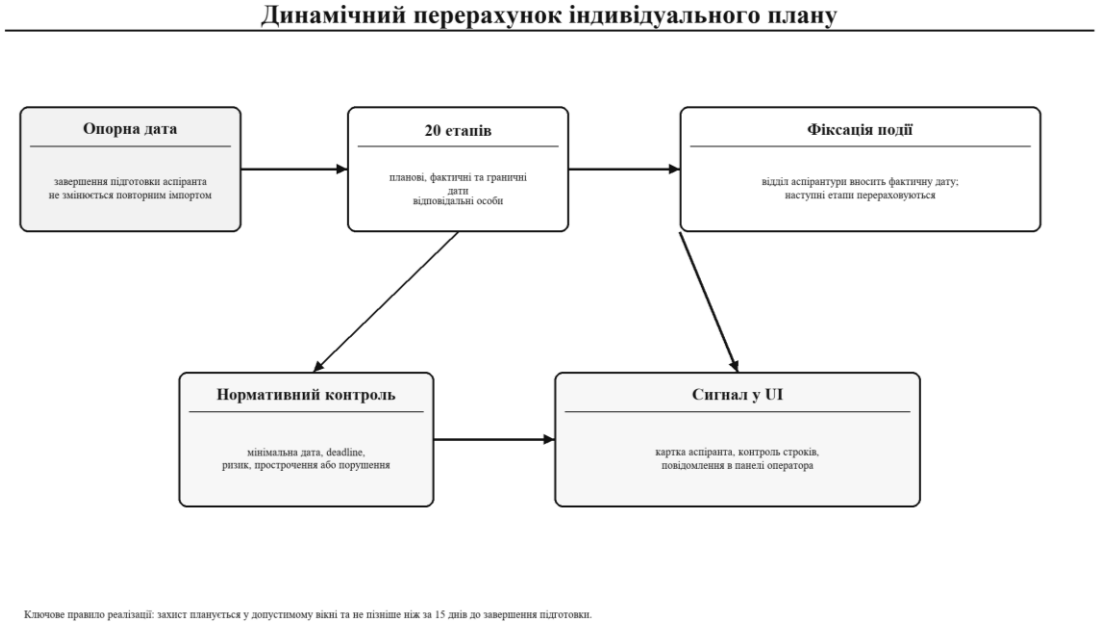


Рисунок 3.6 - Динамічний перерахунок індивідуального плану

**Таблиця 3.4 - Реалізація етапів проходження дисертаційної роботи**

| № | Етап у системі | Реалізаційна дія |
|---|---|---|
| 1 | Оформлення дисертації та перевірка публікацій | Створюється початковий етап із відповідальними «здобувач, керівник» і контрольним повідомленням. |
| 2 | Визначення готовності дисертації | Фіксується фактична дата висновку керівника; наступні строки перераховуються. |
| 3 | Довідка про виконання ОНП і висновок керівника | Реєструється службовий документ у картці аспіранта. |
| 4 | Заява до структурного підрозділу | Етап може отримати документ-заяву або службовий ключ. |
| 5 | Публічна презентація та обговорення | Дата події використовується як тригер для подальшого висновку. |
| 6 | Висновок про наукову новизну | Контролюється наявність документа та його посилання. |
| 7 | Формування пропозицій складу РСВР | Активується редактор складу ради й перевірка ролей. |
| 8 | КЕП на остаточний PDF/A | Реєструється посилання або службовий ключ електронного підпису. |
| 9 | Заява до вченої ради | Контролюється подання обов'язкового пакета документів. |
| 10 | Утворення ради рішенням вченої ради | Змінюється статус ради та реквізити рішення. |
| 11 | Наказ керівника закладу | Заповнюються номер і дата наказу; дані йдуть на публічну сторінку. |

**Продовження табл. 3.4**

| № | Етап у системі | Реалізаційна дія |
|---|---|---|
| 12 | Оприлюднення матеріалів | Публічні URL зберігаються в council.public_details. |
| 13 | Перевірка МОН | Етап контролюється як зовнішня подія з очікуваною датою. |
| 14 | Подання рецензій і відгуків | Рецензії та відгуки додаються до публічних матеріалів. |
| 15 | Призначення дати, часу і місця захисту | Створюється ScheduleEvent із аудиторією та часовим слотом. |
| 16 | Публічний захист | Захист триває не менше 180 хвилин за поточним налаштуванням. |
| 17 | Голосування та рішення ради | Фіксується рішення РСВР і службовий документ. |
| 18 | Оприлюднення рішення і відеозапису | URL відеозапису та рішення відображаються на публічній сторінці. |
| 19 | Наказ про видачу диплома | Контроль завершального адміністративного етапу. |
| 20 | Додавання копій до бібліотеки та справи | Фінальний етап відкритого доступу і завершення траєкторії. |

У картці аспіранта оператор бачить дату завершення підготовки, допустиме вікно захисту, відсоток готовності, ризики та всі 20 етапів. Для здобувача з датою завершення підготовки 30.09.2027 система формує крайню дату захисту 15.09.2027, тобто не пізніше ніж за 15 днів до завершення підготовки. Фрагмент картки наведено на рисунку 3.7.

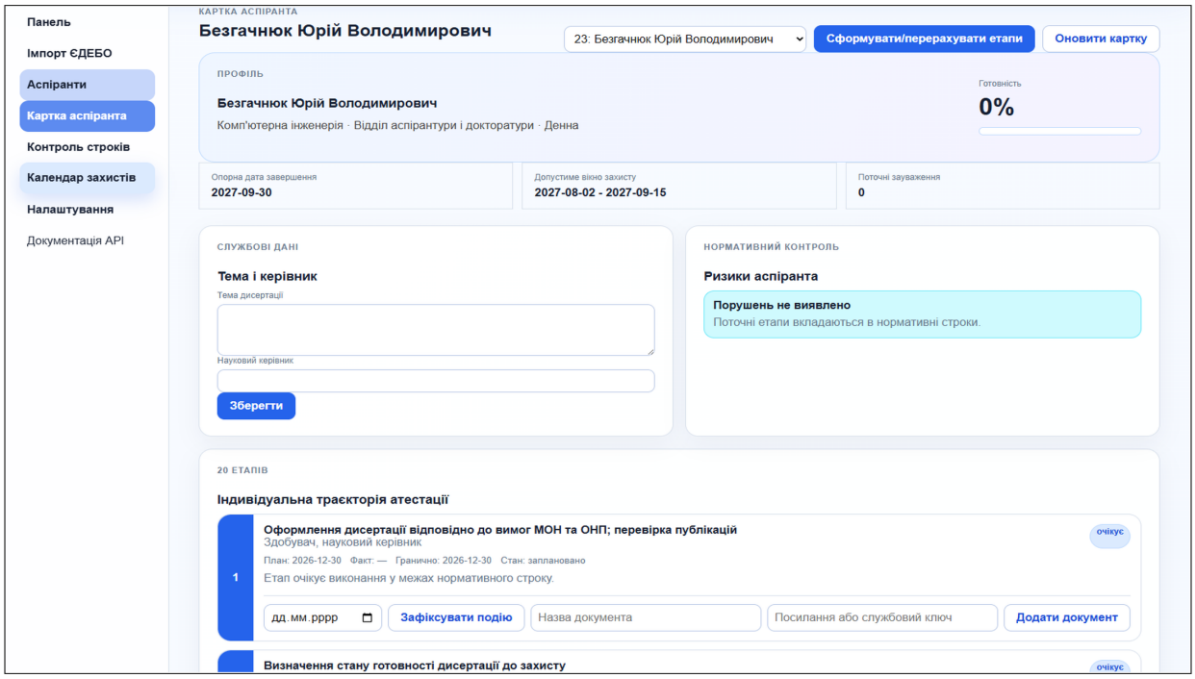


Рисунок 3.7 - Картка аспіранта з індивідуальною траєкторією атестації

### 3.7 Реалізація роботи з разовою спеціалізованою вченою радою

Окремий блок картки аспіранта призначений для роботи зі складом разової спеціалізованої вченої ради. Реалізація враховує два нормативно допустимі варіанти складу: голова, два рецензенти і два офіційні опоненти або голова, один рецензент і три офіційні опоненти [2]. Для кожного учасника зберігаються ПІБ, установа, науковий ступінь, посада, спеціальність, ознака кваліфікації, відсутність конфлікту інтересів і посилання на профіль.

Валідація складу ради реалізує предметні обмеження: голова і рецензенти мають бути з ІФНТУНГ, опоненти не можуть бути з ІФНТУНГ і не можуть належати до одного закладу. Ці обмеження потрібні не лише для коректності складу ради, а й для алгоритму графіка: система перевіряє, чи не працює один і той самий член ради у двох захистах одночасно. Екран блоку РСВР наведено на рисунку 3.8.

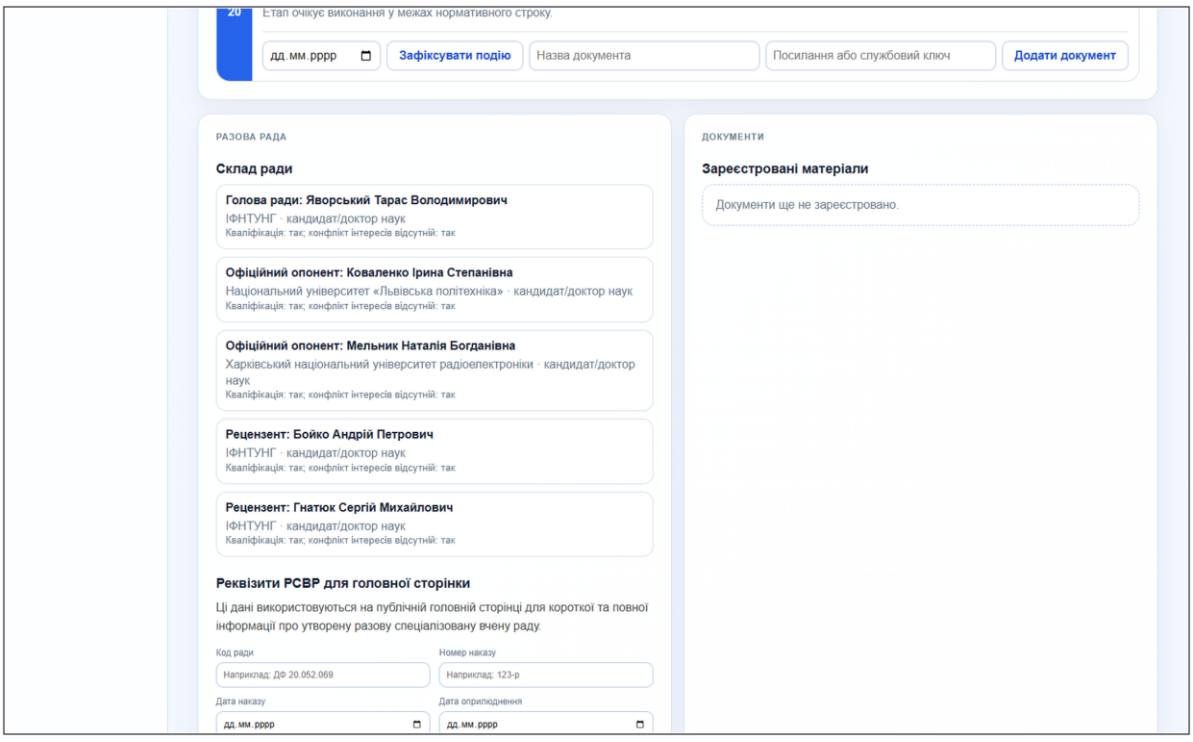


Рисунок 3.8 - Блок редагування складу та реквізитів РСВР у картці аспіранта

### 3.8 Формування графіка захистів і динамічне перепланування

Найважливішою функцією додатка є формування дати, часу й аудиторії захисту без порушення нормативних строків. Планування виконується за жорсткими та м'якими обмеженнями. До жорстких належать нормативне вікно захисту, тривалість засідання не менше 180 хвилин, неперетинання аудиторій, неперетинання членів рад і обов'язкове збереження зафіксованих подій. До м'яких належать мінімізація кількості змін під час перепланування та перевага вже узгоджених слотів.


Рисунок 3.9 - Алгоритм формування та перепланування графіка захистів

**Таблиця 3.5 - Обмеження алгоритму планування**

| Тип | Обмеження | Реалізація |
|---|---|---|
| Жорстке | Захист має бути у допустимому нормативному вікні. | _candidate_defense_schedule_window(); _slot_matches_defense_window() |
| Жорстке | Одна аудиторія не може містити два захисти в один час. | DynamicScheduler перевіряє пару room_id/time_slot. |
| Жорстке | Член ради не може бути залучений до двох одночасних захистів. | Перевірка council_members для кожного candidate. |
| Жорстке | Фіксовані події не змінюються під час перепланування. | locked_event_ids і прапорець locked у ScheduleEvent. |
| М'яке | Бажано мінімізувати кількість змінених призначень. | Штраф за відхилення від previous assignments. |
| М'яке | Обмеження кількості захистів члена ради на день. | default_max_member_defenses_per_day у налаштуваннях. |

Після настання події оператор фіксує фактичну дату в картці аспіранта. Якщо ця дата змінює доступне вікно майбутніх подій, система може сформувати нову версію графіка. Послідовність взаємодії під час такого сценарію наведено на рисунку 3.10.

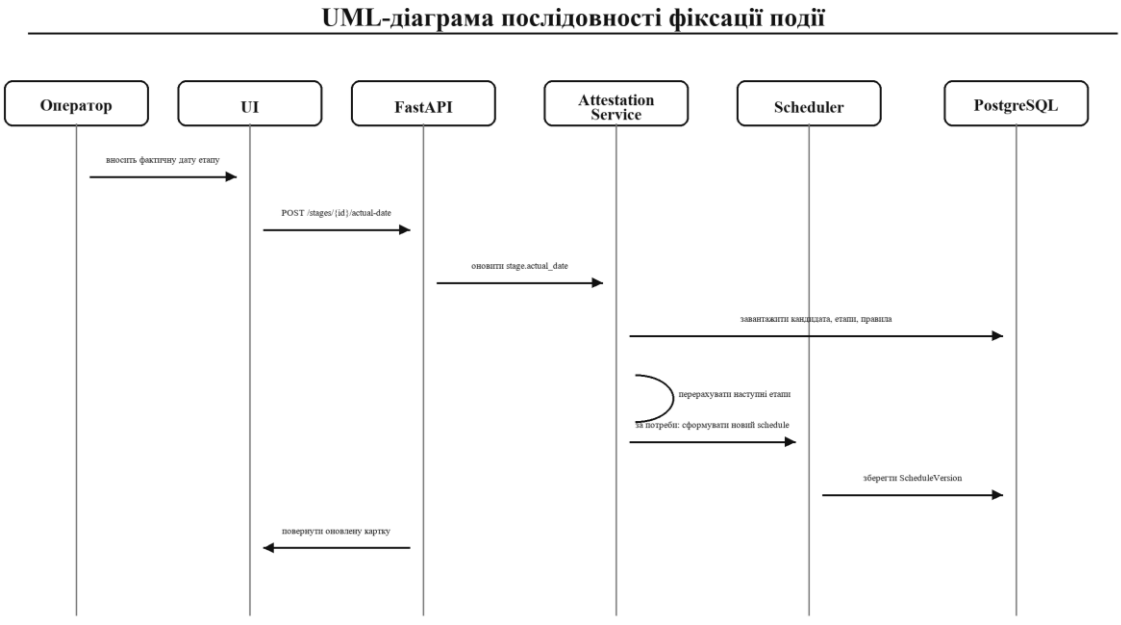


Рисунок 3.10 - UML-діаграма послідовності фіксації події та перепланування

Усі варіанти графіка зберігаються як окремі ScheduleVersion. Це дає змогу порівнювати чернетки, погоджувати або публікувати обрану версію, а також видаляти помилкові версії без втрати історії інших графіків. Екран перегляду сформованої версії наведено на рисунку 3.11.

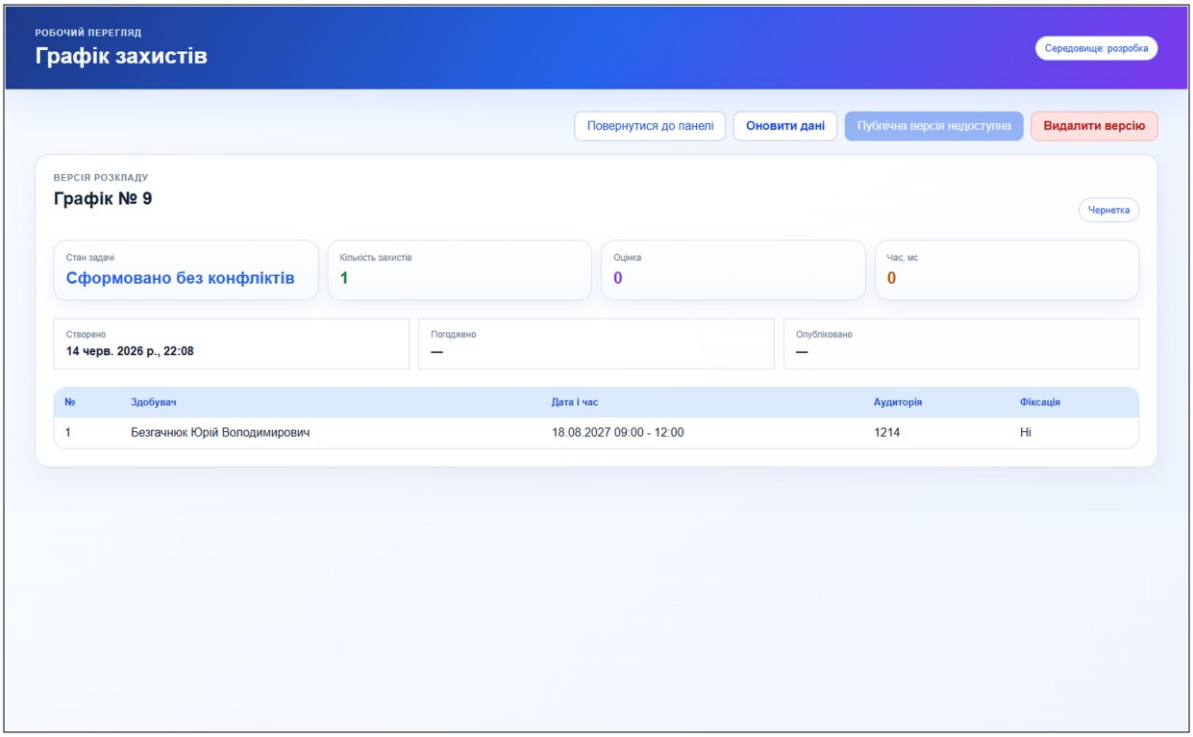


Рисунок 3.11 - Перегляд сформованої версії графіка захистів

## 3.9 Web-інтерфейс користувача

Інтерфейс поділено на робочі розділи, які відповідають реальному сценарію оператора відділу аспірантури: панель, імпорт ЄДЕБО, список аспірантів, картка аспіранта, контроль строків, календар захистів, налаштування та OpenAPI-документація. Панель показує кількість аспірантів, етапів, сформованих рад і версій графіка, а також найближчі ризики. Її поточний вигляд наведено на рисунку 3.12.

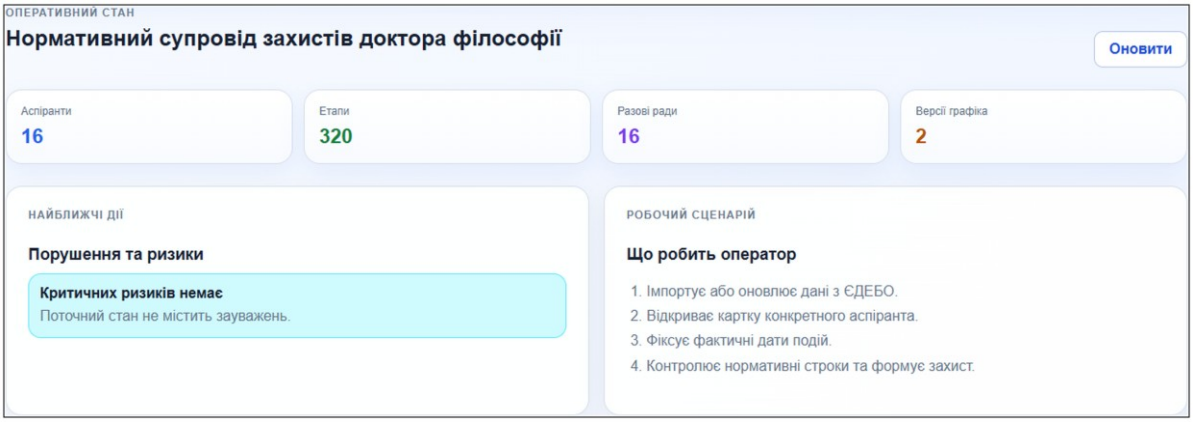


Рисунок 3.12 - Головна панель робочого інтерфейсу

Список аспірантів дає змогу швидко знайти картку, відкрити її або видалити аспіранта з бази разом із пов'язаними етапами, документами, радою і подіями графіка. Календарний розділ відображає версії графіків, меню дій, перевірку конфліктів, погодження, публікацію та видалення версій. Для зручності дії над графіком реалізовано як випадаюче меню, щоб кнопки не перекривали одна одну.

Сторінка налаштувань перетворена на службовий центр керування параметрами планування: тут можна змінити тривалість засідання РСВР, горизонт планування, стартові години захистів, максимальне навантаження члена ради на день, додати або вимкнути аудиторії, зазначити їх місткість і посилання на трансляцію. Екран налаштувань наведено на рисунку 3.13.

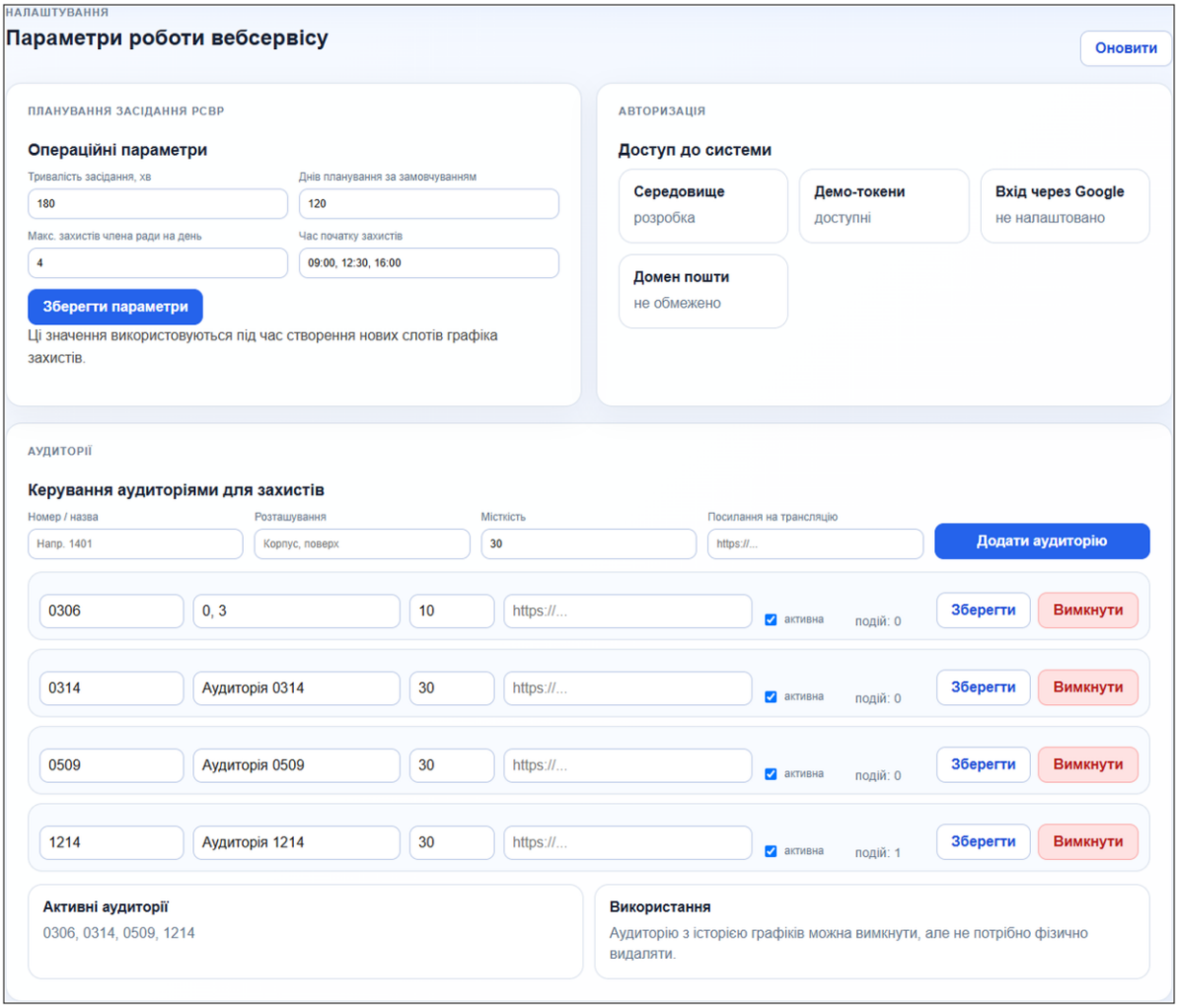


Рисунок 3.13 - Сторінка операційних налаштувань web-сервісу

### 3.10 API, авторизація та рольова модель

Серверна частина надає REST API для всіх основних дій. Робочий інтерфейс використовує ті самі маршрути, що й зовнішній клієнт, тому функціональність можна перевіряти через OpenAPI-сторінку /docs. Доступ до службових маршрутів захищено JWT-токеном, який передається в HTTP-заголовку Authorization. Формат JWT відповідає стандарту RFC 7519 [8].

Для демонстраційного режиму передбачено отримання токена адміністратора. Для продуктивного сценарію додано модуль Google OAuth: користувач авторизується через корпоративну поштову адресу на базі Google

Workspace, після чого система зіставляє його з роллю в базі. Реалізація враховує типовий web server OAuth flow, описаний у документації Google [10].

**Таблиця 3.6 - Основні групи API**

| Група API | Маршрути | Призначення |
|---|---|---|
| Авторизація | /api/v1/auth/demo-token; /api/v1/auth/google/* | Демо-токен і корпоративний вхід через Google. |
| Імпорт | /api/v1/imports/validate; /api/v1/imports/apply | Перевірка та застосування CSV-файлів ЄДЕБО. |
| Аспіранти | /api/v1/candidates; /api/v1/candidates/{id} | Список, картка, редагування, видалення аспіранта. |
| Етапи | /api/v1/stages/{id}/actual-date; /documents | Фіксація фактичних дат і додавання документів. |
| Графіки | /api/v1/schedules/* | Створення, перегляд, погодження, публікація, перепланування, видалення. |
| Ради | /api/v1/candidates/{id}/council; /api/v1/councils/{id}/materials | Склад РСВР, реквізити, службові матеріали. |
| Налаштування | /api/v1/settings/* | Аудиторії, тривалість, стартові години, ліміти планування. |
| Публікація | /; /public/schedules/{id}; /api/v1/public/schedules/{id} | Відкриті сторінки для оприлюднених матеріалів. |

**Таблиця 3.7 - Рольова модель доступу**

| Роль | Доступ у системі |
|---|---|
| Адміністратор | Повний доступ до імпорту, налаштувань, аспірантів, рад, графіків і аудиту. |
| Оператор відділу аспірантури | Ведення карток, фіксація дат, формування графіків, підготовка матеріалів РСВР. |
| Аспірант | Перегляд власної траєкторії, документів і статусів. |
| Науковий керівник | Перегляд етапів, пов'язаних зі здобувачем, і контроль готовності. |
| Член разової ради | Перегляд пов'язаних засідань і матеріалів ради. |

## 3.11 Публікація інформації про РСВР

Публічна головна сторінка web-сервісу відображає інформацію про утворені разові спеціалізовані вчені ради. Коротка інформація подається в картці, а повний склад ради, реквізити наказу, посилання на УкрІНТЕІ, НРАТ, NAQA, Zoom, дисертацію, рецензії, відгуки, відеозапис і рішення відкриваються через розкривний блок. Ці дані редагуються у картці аспіранта, але

відображаються окремо на публічній сторінці, що розділяє службову та відкриту частини системи.

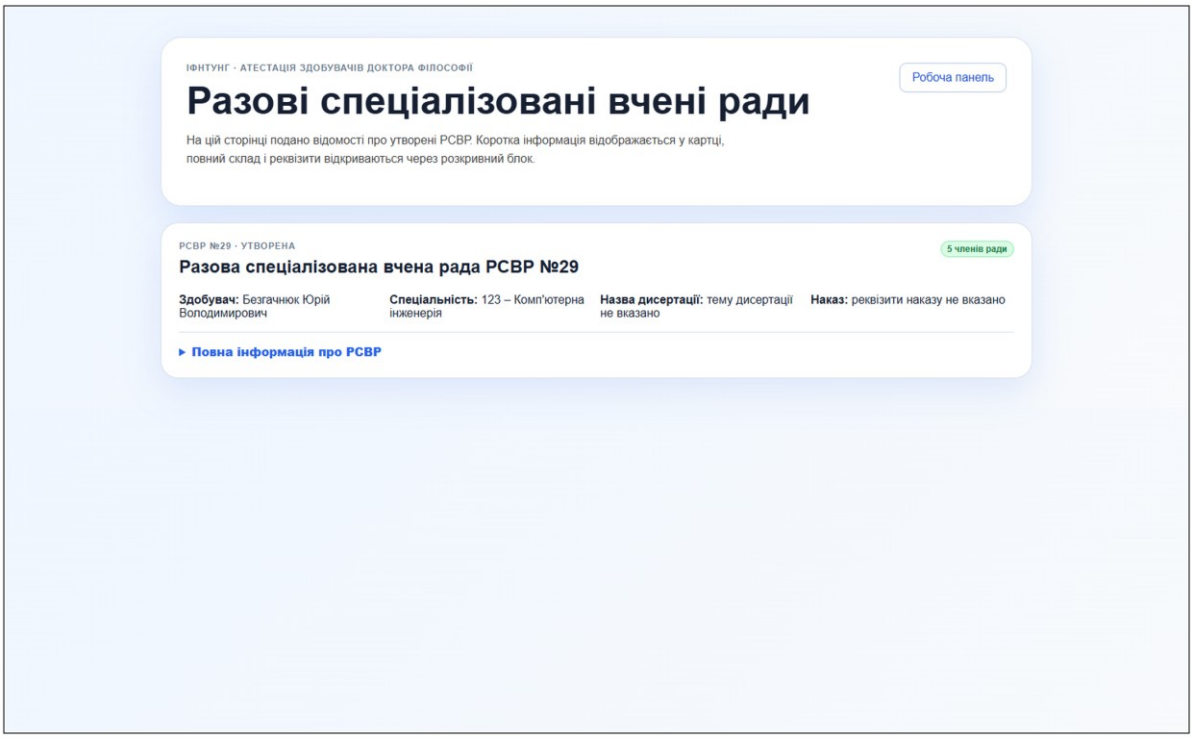


Рисунок 3.14 - Публічна сторінка з інформацією про утворені РСВР

### 3.12 Тестування та оцінка працездатності

Перевірка реалізації виконувалася на трьох рівнях: модульному, інтеграційному та алгоритмічному. Модульні тести охоплюють CSV-імпорт, моделі, планувальник і допоміжні функції. Інтеграційні тести перевіряють маршрути API, авторизацію, створення графіка, публікацію та роботу з нормативним контролем. Алгоритмічні тести перевіряють здатність планувальника знайти допустиме рішення, зберегти фіксовані події і повідомити причину неможливості розкладу.

**Таблиця 3.8 - Результати тестування та експериментального оцінювання**

| Перевірка | Результат |
|---|---|
| Автоматичні тести | 39 тестів виконано успішно; 1 попередження StarletteDeprecationWarning не впливає на функціональність. |
| CSV-імпорт | Перевірено розпізнавання кодування, обов'язкових полів, дублікатів і помилкових дат. |
| Нормативний контроль | Перевірено формування етапів, виявлення ризиків і правильний deadline захисту за 15 днів до завершення підготовки. |
| Планування 8 аспірантів | OPTIMAL, score = 0, median = 0,911 мс, p95 = 1,311 мс. |
| Планування 12 аспірантів | OPTIMAL, score = 0, median = 2,022 мс, p95 = 2,586 мс. |
| Планування 16 аспірантів | OPTIMAL, score = 0, median = 3,749 мс, p95 = 4,262 мс. |
| Перепланування | 3 зафіксовані події, changed_assignments = 0, median = 1,548 мс. |

Отримані результати показують, що для демонстраційного набору з 16 аспірантів алгоритм працює з великим запасом швидкодії. Важливішим за абсолютний час є те, що алгоритм повертає діагностику і не приховує неможливість сформувати допустимий графік. Для службового web-додатка це критично: оператор має бачити причину конфлікту, а не лише факт помилки.

Повний код розробленого програмного прототипу знаходиться у додатках А , Б та В.

**Висновки до розділу 3**

У розділі реалізовано повний функціональний контур web-додатка супроводу атестації аспірантів: імпорт CSV-файлів ЄДЕБО, ведення картки аспіранта, формування 20 етапів атестації, нормативний контроль, роботу зі складом разової ради, планування і перепланування захистів, публікацію матеріалів РСВР, рольовий доступ і контейнерне розгортання. Основна вимога системи - недопущення порушення нормативних дат - забезпечується через незмінну опорну дату завершення підготовки, розрахунок граничного вікна захисту, повторний перерахунок після фактичних подій та діагностику конфліктів.

Розроблений web-сервіс може використовуватися як демонстраційна основа для впровадження у відділі аспірантури і докторантури. Подальше розширення доцільно спрямувати на інтеграцію з календарями, автоматичне надсилання повідомлень, прикріплення файлів до захищеного сховища та повну продуктивну авторизацію через корпоративний домен Google Workspace.

# ВИСНОВКИ

У бакалаврській роботі розроблено web-сервіс супроводу підсумкової атестації аспірантів, призначений для імпорту даних ЄДЕБО, ведення карток аспірантів, контролю нормативних етапів, формування складу разових спеціалізованих вчених рад і планування графіка захистів.

У першому розділі проаналізовано предметну область, нормативну основу процесу атестації аспірантів і недоліки ручного ведення даних у таблицях, документах, листуванні та календарях. Обґрунтовано потребу в спеціалізованому web-сервісі, який забезпечує не лише календарне планування, а й нормативний контроль усієї процедури.

У другому розділі спроєктовано архітектуру web-додатка, рольову модель доступу, структуру бази даних, математичну модель задачі планування, алгоритм формування графіка, API, механізми безпеки та публікації матеріалів. У моделі враховано опорну дату завершення підготовки аспіранта, обмеження щодо захисту не пізніше ніж за 15 днів до цієї дати, тривалість засідання РСВР, доступність аудиторій і неможливість одночасної участі члена ради у кількох захистах.

У третьому розділі реалізовано web-додаток на базі FastAPI, PostgreSQL, HTML/CSS/JavaScript і Docker Compose. Реалізовано модуль імпорту CSV-файлів ЄДЕБО, картку аспіранта, 20 етапів атестації, роботу зі складом РСВР, формування графіка захисту для обраного аспіранта, динамічне перепланування, сторінку налаштувань, публічне представлення інформації про ради та рольову авторизацію.

Розроблена база даних охоплює аспірантів, користувачів, ролі, імпорт, нормативні етапи, документи, ради, членів рад, доступність, аудиторії, часові слоти, версії графіка, події графіка, аудит і операційні налаштування. Це забезпечує збереження історії змін і контроль цілісності даних.

Алгоритм планування реалізовано як детермінований пошук із поверненням, MRV-упорядкуванням і branch-and-bound-відсіканням. Він перевіряє жорсткі обмеження, формує допустимий графік або повертає діагностичне повідомлення про причини неможливості планування.

Контейнерне розгортання реалізовано за допомогою Docker Compose. До складу розгортання входять web-сервіс FastAPI та база даних PostgreSQL. Такий підхід забезпечує відтворюваність середовища розробки й демонстраційного запуску.

Проведено тестування системи. Автоматизовані тести підтвердили працездатність модулів імпорту, API, нормативного контролю, роботи з радами, графіками, авторизацією та моделями даних. Експериментальне оцінювання планувальника показало коректне формування графіка для демонстраційних наборів аспірантів і стабільне перепланування зі збереженням зафіксованих подій.

Отже, поставлену мету досягнуто: розроблено працездатний web-сервіс, який автоматизує ключові етапи супроводу атестації аспірантів, зменшує ризик порушення нормативних строків і забезпечує контрольоване формування графіка захистів.

# ПЕРЕЛІК ПОСИЛАНЬ НА ДЖЕРЕЛА


1. Про затвердження Порядку підготовки здобувачів вищої освіти ступеня доктора філософії та доктора наук у закладах вищої освіти (наукових установах) : постанова Кабінету Міністрів України від 23.03.2016 № 261 : станом на 17.04.2025. Законодавство України. URL: https://zakon.rada.gov.ua/laws/show/261-2016-%D0%BF (дата звернення: 14.06.2026).

2. Про затвердження Порядку присудження ступеня доктора філософії та скасування рішення разової спеціалізованої вченої ради закладу вищої освіти, наукової установи про присудження ступеня доктора філософії : постанова Кабінету Міністрів України від 12.01.2022 № 44 : станом на 08.05.2024. Законодавство України. URL: https://zakon.rada.gov.ua/laws/show/44-2022-%D0%BF (дата звернення: 14.06.2026).

3. Collaborate on Excel workbooks at the same time with co-authoring. Microsoft Support. URL: https://support.microsoft.com/en-us/excel/get-started-collaborate-on-excel-workbooks-at-the-same-time-with-co-authoring (дата звернення: 14.06.2026).

4. Create an appointment schedule. Google Calendar Help. URL: https://support.google.com/calendar/answer/10729749 (дата звернення: 14.06.2026).

5. Microsoft Bookings. Microsoft Learn. 02.04.2025. URL: https://learn.microsoft.com/en-us/microsoft-365/bookings/bookings-overview (дата звернення: 14.06.2026).

6. Calendly features. Calendly. URL: https://calendly.com/features (дата звернення: 14.06.2026).

7. How do I create a group poll? Doodle Help Center. 20.03.2026. URL: https://help.doodle.com/en/articles/9457353-how-do-i-create-a-group-poll (дата звернення: 14.06.2026).

8. Employee scheduling. Google for Developers. 28.08.2024. URL: https://developers.google.com/optimization/scheduling/employee_scheduling (дата звернення: 14.06.2026).

9. Constraints and score: Overview. Timefold Solver Documentation. URL: https://docs.timefold.ai/timefold-solver/latest/constraints-and-score/overview (дата звернення: 14.06.2026).

10. Introduction. Timefold Solver Documentation. URL: https://docs.timefold.ai/timefold-solver/latest/introduction (дата звернення: 14.06.2026).

11. Єдина державна електронна база з питань освіти. Експорт здобувачів : непубліковані набори даних у форматі CSV. 2026.

12. ISO/IEC 25010:2023. Systems and software engineering — Systems and software Quality Requirements and Evaluation (SQuaRE) — Product quality model. International Organization for Standardization. 2023. URL: https://www.iso.org/standard/78176.html (дата звернення: 14.06.2026).

13. OWASP Application Security Verification Standard. OWASP Foundation. URL: https://owasp.org/www-project-application-security-verification-standard/ (дата звернення: 14.06.2026).

14. Про захист персональних даних : Закон України від 01.06.2010 № 2297-VI. Законодавство України. URL: https://zakon.rada.gov.ua/laws/show/2297-17 (дата звернення: 14.06.2026).

15. Про захист інформації в інформаційно-комунікаційних системах : Закон України від 05.07.1994 № 80/94-ВР. Законодавство України. URL: https://zakon.rada.gov.ua/laws/show/80/94-%D0%B2%D1%80 (дата звернення: 14.06.2026).

16. OpenID Connect. Google for Developers. URL: https://developers.google.com/identity/openid-connect/openid-connect (дата звернення: 14.06.2026).

17. Constraints. PostgreSQL Documentation. URL: https://www.postgresql.org/docs/current/ddl-constraints.html (дата звернення: 14.06.2026).

18. Docker Compose. Docker Documentation. URL: https://docs.docker.com/compose/ (дата звернення: 14.06.2026).

19. Юрчак І. Ю., Москович Т. Р. Застосування генетичних алгоритмів в автоматизованій системі розподілу навчального навантаження. Вісник Національного університету «Львівська політехніка». Серія: Комп'ютерні системи та мережі. 2018. № 905. С. 149–157. URL: https://science.lpnu.ua/sites/default/files/journal-paper/2019/sep/18668/182073vis905ksmzvichniy-149-157.pdf (дата звернення: 14.06.2026).

20. Huynh T. T. B., Pham Q. D., Pham D. D. Genetic algorithm for solving the master thesis timetabling problem with multiple objectives. 2012 Fourth International Conference on Knowledge and Systems Engineering. 2012. P. 74–79. DOI: https://doi.org/10.1109/TAAI.2012.50.

21. Battistutta M., Ceschia S., De Cesco F., Di Gaspero L., Schaerf A. Modelling and solving the thesis defense timetabling problem. Journal of the Operational Research Society. 2019. Vol. 70, No. 7. P. 1039–1050. DOI: https://doi.org/10.1080/01605682.2018.1495870.

22. Almeida J., Santos D., Figueira J. R., Francisco A. P. A multi-objective mixed integer linear programming model for thesis defence scheduling. European Journal of Operational Research. 2024. Vol. 312, No. 1. P. 92–116. DOI: https://doi.org/10.1016/j.ejor.2023.06.031.

23. Dimitsas A., Gogos C. Finding near optimal solutions to the thesis defense timetabling problem by exploiting symmetries. SN Operations Research Forum. 2024. Vol. 5, No. 3. Article 54. DOI: https://doi.org/10.1007/s43069-024-00346-4.

24. Larsen R., Pranzo M. A framework for dynamic rescheduling problems. International Journal of Production Research. 2019. Vol. 57, No. 1. P. 16–33. DOI: https://doi.org/10.1080/00207543.2018.1456700.

25. FastAPI documentation. FastAPI. URL: https://fastapi.tiangolo.com/ (дата звернення: 15.05.2026).

26. SQLAlchemy documentation. SQLAlchemy. URL: https://docs.sqlalchemy.org/ (дата звернення: 15.05.2026).

27. Jinja documentation. Pallets. URL: https://jinja.palletsprojects.com/ (дата звернення: 15.05.2026).

28. Alembic documentation. Alembic. URL: https://alembic.sqlalchemy.org/ (дата звернення: 15.05.2026).

29. File upload cheat sheet. OWASP Cheat Sheet Series. URL: https://cheatsheetseries.owasp.org/cheatsheets/File_Upload_Cheat_Sheet.html (дата звернення: 15.05.2026).

30. OpenAPI specification. OpenAPI Initiative. URL: https://spec.openapis.org/oas/latest.html (дата звернення: 15.05.2026).

31. Using OAuth 2.0 for web server applications. Google for Developers. URL: https://developers.google.com/identity/protocols/oauth2/web-server (дата звернення: 15.05.2026).

32. Jones M., Bradley J., Sakimura N. JSON Web Token (JWT) : RFC 7519. Internet Engineering Task Force. 2015. URL: https://datatracker.ietf.org/doc/html/rfc7519.

33. csv — CSV File Reading and Writing. Python documentation. Python Software Foundation. 2026. URL: https://docs.python.org/3/library/csv.html (дата звернення: 15.05.2026).

34. PostgreSQL 17 documentation. PostgreSQL Global Development Group. 2026. URL: https://www.postgresql.org/docs/17/ (дата звернення: 15.05.2026).

35. FastAPI in Containers — Docker. FastAPI. URL: https://fastapi.tiangolo.com/deployment/docker/ (дата звернення: 15.05.2026).

# ДОДАТКИ

# ДОДАТОК А

# ДОДАТКИ А

У додатках наведено програмний код і конфігураційні файли web-додатка супроводу атестації аспірантів відповідно до переліку, поданого в таблиці 3.9 розділу 3. Код подано у вигляді лістингів зі збереженням назв файлів, структури модулів і нумерації рядків.

З метою безпеки до додатків включено лише приклад файлу змінних середовища .env.example. Реальний файл .env, який може містити службові параметри або ключі, до додатків не вноситься.

**Таблиця А.1 - Перелік додатків**

| Додаток | Назва | Лістинги |
|---|---|---|
| А.1 | Маршрути API, схеми запитів і відповідей | А.1.1 - app/main.py; А.1.2 - app/api/routes.py; А.1.3 - app/schemas.py |
| А.2 | ORM-моделі та міграції структури бази даних | А.2.1 - app/models/entities.py; А.2.2 - app/models/enums.py; А.2.3 - alembic/versions/572c3715004e_initial_schema.py; А.2.4 - alembic/versions/9c1d2e3f4a5b_app_settings.py; А.2.5 - alembic/versions/b7d8e9f0a1b2_council_public_details.py; А.2.6 - alembic/versions/c8d9e0f1a2b3_council_member_public_fields.py |
| А.3 | Модуль перевірки та застосування CSV-імпорту ЄДЕБО | А.3.1 - app/services/edebo_import.py; А.3.2 - app/repositories/imports.py |
| А.4 | Формування індивідуального плану, нормативний контроль і РСВР | А.4.1 - app/services/attestation.py |
| А.5 | Алгоритм формування графіка та динамічного перепланування | А.5.1 - scheduler.py; А.5.2 - app/services/scheduling.py |
| А.6 | Авторизація, ролі та корпоративна пошта | А.6.1 - app/security.py; А.6.2 - app/services/google_auth.py |
| Б.1 | Код інтерфейсу користувача та публічних сторінок | Б.1.1 - app/templates/dashboard.html; Б.1.2 - app/templates/public_home.html; Б.1.3 - app/templates/public_schedule.html; Б.1.4 - app/templates/schedule_view.html; Б.1.5 - app/static/app.js; Б.1.6 - app/static/styles.css |
| В.1 | Контейнерне розгортання FastAPI-сервісу та PostgreSQL | В.1.1 - Dockerfile; В.1.2 - docker-compose.yml; В.1.3 - docker-entrypoint.sh; В.1.4 - .env.example |

## ДОДАТОК А.1

### Маршрути API, схеми запитів і відповідей

У додатку наведено основні файли, що підключають FastAPI-додаток, визначають HTTP-маршрути, структури запитів і відповідей, а також зв'язок зі статичними файлами та шаблонами.

### Лістинг А.1.1 - Точка входу FastAPI-додатка (app/main.py)

```
 1 | from pathlib import Path
 2 |
 3 | from fastapi import FastAPI
 4 | from fastapi.responses import RedirectResponse
 5 | from fastapi.staticfiles import StaticFiles
 6 |
 7 | from app.api.routes import router
 8 |
 9 |
10 | app = FastAPI(
11 |     title="PhD Attestation Planner",
12 |     version="0.2.0",
13 |     description=(
14 |         "API web-сервісу формування, погодження та публікації "
15 |         "розкладу атестації аспірантів."
16 |     ),
17 | )
18 | app.mount(
19 |     "/static",
20 |     StaticFiles(directory=Path(__file__).with_name("static")),
21 |     name="static",
22 | )
23 | app.include_router(router)
24 |
25 |
26 | @app.get("/", include_in_schema=False)
27 | async def root() -> RedirectResponse:
28 |     return RedirectResponse(url="/ui")
29 |
30 |
31 | @app.get("/health", tags=["system"])
32 | async def health() -> dict[str, str]:
33 |     return {"status": "ok"}
```

### Лістинг А.1.2 - Маршрути API та сторінок web-додатка (app/api/routes.py)

```
 1 | from __future__ import annotations
 2 |
 3 | import json
 4 | import secrets
 5 | from pathlib import Path
 6 |
 7 | from fastapi import (
 8 |     APIRouter,
 9 |     Depends,
10 |     File,
```

```
11 |     HTTPException,
12 |     Request,
13 |     UploadFile,
14 |     status,
15 | )
16 | from fastapi.responses import HTMLResponse, RedirectResponse
17 | from fastapi.templating import Jinja2Templates
18 | from sqlalchemy.exc import SQLAlchemyError
19 | from sqlalchemy.ext.asyncio import AsyncSession
20 |
21 | from app.config import settings
22 | from app.db import get_session
23 | from app.models.enums import UserRole
24 | from app.repositories.imports import apply_candidate_import
25 | from app.schemas import (
26 |     AuditLogResponse,
27 |     CandidateDetailResponse,
28 |     CandidateListResponse,
29 |     CandidateUpdatePayload,
30 |     ConflictListResponse,
31 |     CouncilCompositionPayload,
32 |     CouncilMetadataPayload,
33 |     CouncilListResponse,
34 |     CouncilMaterialResponse,
35 |     DeleteResponse,
36 |     DemoScheduleRequest,
37 |     DemoTokenRequest,
38 |     GeneratePlansResponse,
39 |     ImportApplyResponse,
40 |     ImportValidationResponse,
41 |     NormativeControlResponse,
42 |     OperationalSettingsUpdateRequest,
43 |     ReplanScheduleRequest,
44 |     RoomAdminSummary,
45 |     RoomCreateRequest,
46 |     RoomUpdateRequest,
47 |     ScheduleGenerateRequest,
48 |     ScheduleListResponse,
49 |     ScheduleSolveRequest,
50 |     ScheduleVersionResponse,
51 |     StageEventPayload,
52 |     StageEventResponse,
53 |     StageDocumentPayload,
54 |     StageDocumentResponse,
55 |     StageListResponse,
56 |     SystemSettingsResponse,
57 |     TokenResponse,
58 |     WorkspaceSummaryResponse,
59 | )
60 | from app.security import Principal, create_access_token, require_roles
61 | from app.services.attestation import (
62 |     approve_db_schedule,
63 |     build_council_material,
64 |     cancel_db_schedule,
65 |     create_schedule_from_database,
66 |     create_room_setting,
67 |     delete_candidate,
68 |     delete_schedule_version,
69 |     diagnose_schedule_conflicts,
70 |     deactivate_room_setting,
71 |     generate_individual_plans,
72 |     get_candidate_detail,
73 |     get_db_schedule,
```

```
    get_system_settings,
    get_workspace_summary,
    list_audit_log,
    list_candidates,
    list_councils,
    list_normative_control,
    list_public_council_cards,
    list_schedules,
    list_stages,
    publish_db_schedule,
    record_stage_actual_date,
    register_stage_document,
    replan_schedule,
    update_operational_settings,
    update_candidate_council,
    update_candidate_council_metadata,
    update_candidate_profile,
    update_room_setting,
)
from app.services.edebo_import import parse_edebo_csv
from app.services.google_auth import (
    GoogleAuthError,
    build_google_authorization_url,
    exchange_code_for_identity,
    issue_local_token_for_google_identity,
)
from app.services.scheduling import build_demo_request
from app.services.workflow import (
    InvalidScheduleTransitionError,
    ScheduleNotFoundError,
    schedule_workflow,
)


router = APIRouter()
templates = Jinja2Templates(
    directory=str(Path(__file__).resolve().parents[1] / "templates")
)

operator_access = require_roles(UserRole.ADMIN, UserRole.OPERATOR)
admin_access = require_roles(UserRole.ADMIN)
authenticated_access = require_roles(*tuple(UserRole))


@router.post(
    "/api/v1/auth/demo-token",
    response_model=TokenResponse,
    tags=["authentication"],
)
async def demo_token(payload: DemoTokenRequest) -> TokenResponse:
    if settings.app_environment.casefold() == "production":
        raise HTTPException(
            status_code=status.HTTP_404_NOT_FOUND,
            detail="Маршрут недоступний.",
        )
    token, expires_in = create_access_token(
        subject=f"demo-{payload.role.value}",
        roles={payload.role},
        email=f"{payload.role.value}@example.test",
    )
    return TokenResponse(access_token=token, expires_in=expires_in)


```

```
@router.get(
    "/api/v1/auth/google/login",
    tags=["authentication"],
    include_in_schema=False,
)
async def google_login() -> RedirectResponse:
    state = secrets.token_urlsafe(32)
    try:
        url = build_google_authorization_url(state)
    except GoogleAuthError as exc:
        raise HTTPException(
            status_code=status.HTTP_503_SERVICE_UNAVAILABLE,
            detail=str(exc),
        ) from exc
    response = RedirectResponse(url=url)
    response.set_cookie(
        "google_oauth_state",
        state,
        max_age=600,
        httponly=True,
        secure=settings.app_environment.casefold() == "production",
        samesite="lax",
    )
    return response


@router.get(
    "/api/v1/auth/google/callback",
    response_class=HTMLResponse,
    tags=["authentication"],
    include_in_schema=False,
)
async def google_callback(
    request: Request,
    code: str | None = None,
    state: str | None = None,
    error: str | None = None,
    session: AsyncSession = Depends(get_session),
) -> HTMLResponse:
    if error:
        raise HTTPException(
            status_code=status.HTTP_400_BAD_REQUEST,
            detail=f"Google OAuth повернув помилку: {error}.",
        )
    expected_state = request.cookies.get("google_oauth_state")
    if not code or not state or not expected_state or state !=
expected_state:
        raise HTTPException(
            status_code=status.HTTP_400_BAD_REQUEST,
            detail="Некоректний або прострочений OAuth state.",
        )
    try:
        identity = exchange_code_for_identity(code)
        token, expires_in, roles = await
issue_local_token_for_google_identity(
            session,
            identity,
        )
    except GoogleAuthError as exc:
        raise HTTPException(
            status_code=status.HTTP_401_UNAUTHORIZED,
            detail=str(exc),
        ) from exc
```

```
198 |
199 |     html = f"""
200 |     <!doctype html>
201 |     <html lang="uk">
202 |     <meta charset="utf-8">
203 |     <title>Авторизація виконана</title>
204 |     <body>
205 |       <p>Авторизацію через корпоративну пошту виконано. Повертаємося до
системи...</p>
206 |       <script>
207 |         localStorage.setItem("attestationAccessToken",
{json.dumps(token)});
208 |         localStorage.setItem("attestationRoles",
{json.dumps(sorted(role.value for role in roles))});
209 |         localStorage.setItem("attestationTokenExpiresIn",
{json.dumps(expires_in)});
210 |         window.location.replace("/ui");
211 |       </script>
212 |     </body>
213 |     </html>
214 |     """
215 |     response = HTMLResponse(html)
216 |     response.delete_cookie("google_oauth_state")
217 |     return response
218 |
219 |
220 | @router.post(
221 |     "/api/v1/imports/validate",
222 |     response_model=ImportValidationResponse,
223 |     tags=["imports"],
224 | )
225 | async def validate_import(
226 |     file: UploadFile = File(...),
227 |     _: Principal = Depends(operator_access),
228 | ) -> ImportValidationResponse:
229 |     file_name, data = await _read_csv_upload(file)
230 |     try:
231 |         return parse_edebo_csv(file_name, data).report
232 |     except ValueError as exc:
233 |         raise HTTPException(
234 |             status_code=status.HTTP_422_UNPROCESSABLE_CONTENT,
235 |             detail=str(exc),
236 |         ) from exc
237 |
238 |
239 | @router.get(
240 |     "/api/v1/workspace",
241 |     response_model=WorkspaceSummaryResponse,
242 |     tags=["workspace"],
243 | )
244 | async def workspace_summary(
245 |     _: Principal = Depends(authenticated_access),
246 |     session: AsyncSession = Depends(get_session),
247 | ) -> WorkspaceSummaryResponse:
248 |     return await get_workspace_summary(session)
249 |
250 |
251 | @router.get(
252 |     "/api/v1/settings",
253 |     response_model=SystemSettingsResponse,
254 |     tags=["settings"],
255 | )
256 | async def system_settings(
```

```
257 |     _: Principal = Depends(operator_access),
258 |     session: AsyncSession = Depends(get_session),
259 | ) -> SystemSettingsResponse:
260 |     return await get_system_settings(session)
261 |
262 |
263 | @router.patch(
264 |     "/api/v1/settings/operational",
265 |     response_model=SystemSettingsResponse,
266 |     tags=["settings"],
267 | )
268 | async def patch_operational_settings(
269 |     payload: OperationalSettingsUpdateRequest,
270 |     principal: Principal = Depends(operator_access),
271 |     session: AsyncSession = Depends(get_session),
272 | ) -> SystemSettingsResponse:
273 |     try:
274 |         return await update_operational_settings(session, principal,
payload)
275 |     except ValueError as exc:
276 |         raise HTTPException(
277 |             status_code=status.HTTP_422_UNPROCESSABLE_CONTENT,
278 |             detail=str(exc),
279 |         ) from exc
280 |
281 |
282 | @router.post(
283 |     "/api/v1/settings/rooms",
284 |     response_model=RoomAdminSummary,
285 |     status_code=status.HTTP_201_CREATED,
286 |     tags=["settings"],
287 | )
288 | async def create_settings_room(
289 |     payload: RoomCreateRequest,
290 |     principal: Principal = Depends(operator_access),
291 |     session: AsyncSession = Depends(get_session),
292 | ) -> RoomAdminSummary:
293 |     try:
294 |         return await create_room_setting(session, principal, payload)
295 |     except ValueError as exc:
296 |         raise HTTPException(
297 |             status_code=status.HTTP_422_UNPROCESSABLE_CONTENT,
298 |             detail=str(exc),
299 |         ) from exc
300 |
301 |
302 | @router.patch(
303 |     "/api/v1/settings/rooms/{room_id}",
304 |     response_model=RoomAdminSummary,
305 |     tags=["settings"],
306 | )
307 | async def patch_settings_room(
308 |     room_id: int,
309 |     payload: RoomUpdateRequest,
310 |     principal: Principal = Depends(operator_access),
311 |     session: AsyncSession = Depends(get_session),
312 | ) -> RoomAdminSummary:
313 |     try:
314 |         return await update_room_setting(session, principal, room_id,
payload)
315 |     except LookupError as exc:
316 |         raise HTTPException(
317 |             status_code=status.HTTP_404_NOT_FOUND,
```

```
318 |             detail="Аудиторію не знайдено.",
319 |         ) from exc
320 |     except ValueError as exc:
321 |         raise HTTPException(
322 |             status_code=status.HTTP_422_UNPROCESSABLE_CONTENT,
323 |             detail=str(exc),
324 |         ) from exc
325 |
326 |
327 | @router.delete(
328 |     "/api/v1/settings/rooms/{room_id}",
329 |     response_model=RoomAdminSummary,
330 |     tags=["settings"],
331 | )
332 | async def deactivate_settings_room(
333 |     room_id: int,
334 |     principal: Principal = Depends(operator_access),
335 |     session: AsyncSession = Depends(get_session),
336 | ) -> RoomAdminSummary:
337 |     try:
338 |         return await deactivate_room_setting(session, principal, room_id)
339 |     except LookupError as exc:
340 |         raise HTTPException(
341 |             status_code=status.HTTP_404_NOT_FOUND,
342 |             detail="Аудиторію не знайдено.",
343 |         ) from exc
344 |
345 |
346 | @router.get(
347 |     "/api/v1/candidates",
348 |     response_model=CandidateListResponse,
349 |     tags=["candidates"],
350 | )
351 | async def candidates(
352 |     _: Principal = Depends(authenticated_access),
353 |     session: AsyncSession = Depends(get_session),
354 | ) -> CandidateListResponse:
355 |     return await list_candidates(session)
356 |
357 |
358 | @router.get(
359 |     "/api/v1/candidates/{candidate_id}",
360 |     response_model=CandidateDetailResponse,
361 |     tags=["candidates"],
362 | )
363 | async def candidate_detail(
364 |     candidate_id: int,
365 |     _: Principal = Depends(authenticated_access),
366 |     session: AsyncSession = Depends(get_session),
367 | ) -> CandidateDetailResponse:
368 |     try:
369 |         return await get_candidate_detail(session, candidate_id)
370 |     except LookupError as exc:
371 |         raise HTTPException(
372 |             status_code=status.HTTP_404_NOT_FOUND,
373 |             detail="Аспіранта не знайдено.",
374 |         ) from exc
375 |
376 |
377 | @router.patch(
378 |     "/api/v1/candidates/{candidate_id}",
379 |     response_model=CandidateDetailResponse,
380 |     tags=["candidates"],
```

```
381 | )
382 | async def update_candidate(
383 |     candidate_id: int,
384 |     payload: CandidateUpdatePayload,
385 |     principal: Principal = Depends(operator_access),
386 |     session: AsyncSession = Depends(get_session),
387 | ) -> CandidateDetailResponse:
388 |     try:
389 |         return await update_candidate_profile(
390 |             session,
391 |             principal,
392 |             candidate_id,
393 |             payload,
394 |         )
395 |     except LookupError as exc:
396 |         raise HTTPException(
397 |             status_code=status.HTTP_404_NOT_FOUND,
398 |             detail="Аспіранта не знайдено.",
399 |         ) from exc
400 |
401 |
402 | @router.delete(
403 |     "/api/v1/candidates/{candidate_id}",
404 |     response_model=DeleteResponse,
405 |     tags=["candidates"],
406 | )
407 | async def remove_candidate(
408 |     candidate_id: int,
409 |     principal: Principal = Depends(operator_access),
410 |     session: AsyncSession = Depends(get_session),
411 | ) -> DeleteResponse:
412 |     try:
413 |         return await delete_candidate(session, principal, candidate_id)
414 |     except LookupError as exc:
415 |         raise HTTPException(
416 |             status_code=status.HTTP_404_NOT_FOUND,
417 |             detail="Аспіранта не знайдено.",
418 |         ) from exc
419 |
420 |
421 | @router.put(
422 |     "/api/v1/candidates/{candidate_id}/council",
423 |     response_model=CandidateDetailResponse,
424 |     tags=["councils"],
425 | )
426 | async def replace_candidate_council(
427 |     candidate_id: int,
428 |     payload: CouncilCompositionPayload,
429 |     principal: Principal = Depends(operator_access),
430 |     session: AsyncSession = Depends(get_session),
431 | ) -> CandidateDetailResponse:
432 |     try:
433 |         return await update_candidate_council(session, principal,
candidate_id, payload)
434 |     except LookupError as exc:
435 |         raise HTTPException(
436 |             status_code=status.HTTP_404_NOT_FOUND,
437 |             detail="Аспіранта не знайдено.",
438 |         ) from exc
439 |     except ValueError as exc:
440 |         raise HTTPException(
441 |             status_code=status.HTTP_422_UNPROCESSABLE_CONTENT,
442 |             detail=str(exc),
```

```
443 |         ) from exc
444 |
445 |
446 | @router.patch(
447 |     "/api/v1/candidates/{candidate_id}/council/metadata",
448 |     response_model=CandidateDetailResponse,
449 |     tags=["councils"],
450 | )
451 | async def update_council_metadata(
452 |     candidate_id: int,
453 |     payload: CouncilMetadataPayload,
454 |     principal: Principal = Depends(operator_access),
455 |     session: AsyncSession = Depends(get_session),
456 | ) -> CandidateDetailResponse:
457 |     try:
458 |         return await update_candidate_council_metadata(
459 |             session,
460 |             principal,
461 |             candidate_id,
462 |             payload,
463 |         )
464 |     except LookupError as exc:
465 |         raise HTTPException(
466 |             status_code=status.HTTP_404_NOT_FOUND,
467 |             detail="Аспіранта не знайдено.",
468 |         ) from exc
469 |
470 |
471 | @router.get(
472 |     "/api/v1/stages",
473 |     response_model=StageListResponse,
474 |     tags=["attestation"],
475 | )
476 | async def stages(
477 |     _: Principal = Depends(authenticated_access),
478 |     session: AsyncSession = Depends(get_session),
479 | ) -> StageListResponse:
480 |     return await list_stages(session)
481 |
482 |
483 | @router.get(
484 |     "/api/v1/attestation/normative-control",
485 |     response_model=NormativeControlResponse,
486 |     tags=["attestation"],
487 | )
488 | async def normative_control(
489 |     _: Principal = Depends(authenticated_access),
490 |     session: AsyncSession = Depends(get_session),
491 | ) -> NormativeControlResponse:
492 |     return await list_normative_control(session)
493 |
494 |
495 | @router.post(
496 |     "/api/v1/attestation/plans/generate",
497 |     response_model=GeneratePlansResponse,
498 |     tags=["attestation"],
499 | )
500 | async def generate_plans(
501 |     principal: Principal = Depends(operator_access),
502 |     session: AsyncSession = Depends(get_session),
503 | ) -> GeneratePlansResponse:
504 |     return await generate_individual_plans(session, principal)
505 |
```

```
506 |
507 | @router.post(
508 |     "/api/v1/stages/{stage_id}/actual-date",
509 |     response_model=StageEventResponse,
510 |     tags=["attestation"],
511 | )
512 | async def fix_stage_actual_date(
513 |     stage_id: int,
514 |     payload: StageEventPayload,
515 |     principal: Principal = Depends(operator_access),
516 |     session: AsyncSession = Depends(get_session),
517 | ) -> StageEventResponse:
518 |     try:
519 |         return await record_stage_actual_date(session, principal,
stage_id, payload)
520 |     except LookupError as exc:
521 |         raise HTTPException(
522 |             status_code=status.HTTP_404_NOT_FOUND,
523 |             detail="Етап атестації не знайдено.",
524 |         ) from exc
525 |     except ValueError as exc:
526 |         raise HTTPException(
527 |             status_code=status.HTTP_422_UNPROCESSABLE_CONTENT,
528 |             detail=str(exc),
529 |         ) from exc
530 |
531 |
532 | @router.post(
533 |     "/api/v1/stages/{stage_id}/documents",
534 |     response_model=StageDocumentResponse,
535 |     tags=["documents"],
536 | )
537 | async def register_document(
538 |     stage_id: int,
539 |     payload: StageDocumentPayload,
540 |     principal: Principal = Depends(operator_access),
541 |     session: AsyncSession = Depends(get_session),
542 | ) -> StageDocumentResponse:
543 |     try:
544 |         return await register_stage_document(session, principal, stage_id,
payload)
545 |     except LookupError as exc:
546 |         raise HTTPException(
547 |             status_code=status.HTTP_404_NOT_FOUND,
548 |             detail="Етап атестації не знайдено.",
549 |         ) from exc
550 |
551 |
552 | @router.post(
553 |     "/api/v1/imports/apply",
554 |     response_model=ImportApplyResponse,
555 |     tags=["imports"],
556 | )
557 | async def apply_import(
558 |     file: UploadFile = File(...),
559 |     principal: Principal = Depends(operator_access),
560 |     session: AsyncSession = Depends(get_session),
561 | ) -> ImportApplyResponse:
562 |     file_name, data = await _read_csv_upload(file)
563 |     try:
564 |         parsed = parse_edebo_csv(file_name, data)
565 |         return await apply_candidate_import(session, parsed, principal)
566 |     except ValueError as exc:
```

```
567 |         raise HTTPException(
568 |             status_code=status.HTTP_422_UNPROCESSABLE_CONTENT,
569 |             detail=str(exc),
570 |         ) from exc
571 |
572 |
573 | @router.post(
574 |     "/api/v1/schedules",
575 |     response_model=ScheduleVersionResponse,
576 |     status_code=status.HTTP_201_CREATED,
577 |     tags=["schedules"],
578 | )
579 | async def create_schedule(
580 |     payload: ScheduleSolveRequest,
581 |     _: Principal = Depends(operator_access),
582 | ) -> ScheduleVersionResponse:
583 |     try:
584 |         return schedule_workflow.create(payload)
585 |     except ValueError as exc:
586 |         raise HTTPException(
587 |             status_code=status.HTTP_422_UNPROCESSABLE_CONTENT,
588 |             detail=str(exc),
589 |         ) from exc
590 |
591 |
592 | @router.get(
593 |     "/api/v1/schedules",
594 |     response_model=ScheduleListResponse,
595 |     tags=["schedules"],
596 | )
597 | async def schedules(
598 |     _: Principal = Depends(authenticated_access),
599 |     session: AsyncSession = Depends(get_session),
600 | ) -> ScheduleListResponse:
601 |     return await list_schedules(session)
602 |
603 |
604 | @router.post(
605 |     "/api/v1/schedules/generate",
606 |     response_model=ScheduleVersionResponse,
607 |     status_code=status.HTTP_201_CREATED,
608 |     tags=["schedules"],
609 | )
610 | async def generate_schedule_from_db(
611 |     payload: ScheduleGenerateRequest,
612 |     principal: Principal = Depends(operator_access),
613 |     session: AsyncSession = Depends(get_session),
614 | ) -> ScheduleVersionResponse:
615 |     try:
616 |         return await create_schedule_from_database(session, principal,
payload)
617 |     except ValueError as exc:
618 |         raise HTTPException(
619 |             status_code=status.HTTP_422_UNPROCESSABLE_CONTENT,
620 |             detail=str(exc),
621 |         ) from exc
622 |
623 |
624 | @router.post(
625 |     "/api/v1/schedules/demo",
626 |     response_model=ScheduleVersionResponse,
627 |     status_code=status.HTTP_201_CREATED,
628 |     tags=["schedules"],
```

```
629 | )
630 | async def create_demo_schedule(
631 |     payload: DemoScheduleRequest,
632 |     _: Principal = Depends(operator_access),
633 | ) -> ScheduleVersionResponse:
634 |     if settings.app_environment.casefold() == "production":
635 |         raise HTTPException(
636 |             status_code=status.HTTP_404_NOT_FOUND,
637 |             detail="Маршрут недоступний.",
638 |         )
639 |     return
schedule_workflow.create(build_demo_request(payload.candidate_count))
640 |
641 |
642 | @router.get(
643 |     "/api/v1/schedules/{schedule_id}",
644 |     response_model=ScheduleVersionResponse,
645 |     tags=["schedules"],
646 | )
647 | async def get_schedule(
648 |     schedule_id: str,
649 |     _: Principal = Depends(authenticated_access),
650 |     session: AsyncSession = Depends(get_session),
651 | ) -> ScheduleVersionResponse:
652 |     if schedule_id.isdigit():
653 |         try:
654 |             return await get_db_schedule(session, schedule_id)
655 |         except LookupError as exc:
656 |             raise _not_found() from exc
657 |     return _get_schedule(schedule_id)
658 |
659 |
660 | @router.post(
661 |     "/api/v1/schedules/{schedule_id}/approve",
662 |     response_model=ScheduleVersionResponse,
663 |     tags=["schedules"],
664 | )
665 | async def approve_schedule(
666 |     schedule_id: str,
667 |     principal: Principal = Depends(operator_access),
668 |     session: AsyncSession = Depends(get_session),
669 | ) -> ScheduleVersionResponse:
670 |     if schedule_id.isdigit():
671 |         try:
672 |             return await approve_db_schedule(session, principal,
schedule_id)
673 |         except LookupError as exc:
674 |             raise _not_found() from exc
675 |         except ValueError as exc:
676 |             raise _transition_error(str(exc)) from exc
677 |     try:
678 |         return schedule_workflow.approve(schedule_id)
679 |     except ScheduleNotFoundError as exc:
680 |         raise _not_found() from exc
681 |     except InvalidScheduleTransitionError as exc:
682 |         raise _transition_error(str(exc)) from exc
683 |
684 |
685 | @router.post(
686 |     "/api/v1/schedules/{schedule_id}/publish",
687 |     response_model=ScheduleVersionResponse,
688 |     tags=["schedules"],
689 | )
```

```
690 | async def publish_schedule(
691 |     schedule_id: str,
692 |     principal: Principal = Depends(admin_access),
693 |     session: AsyncSession = Depends(get_session),
694 | ) -> ScheduleVersionResponse:
695 |     if schedule_id.isdigit():
696 |         try:
697 |             return await publish_db_schedule(session, principal,
schedule_id)
698 |         except LookupError as exc:
699 |             raise _not_found() from exc
700 |         except ValueError as exc:
701 |             raise _transition_error(str(exc)) from exc
702 |     try:
703 |         return schedule_workflow.publish(schedule_id)
704 |     except ScheduleNotFoundError as exc:
705 |         raise _not_found() from exc
706 |     except InvalidScheduleTransitionError as exc:
707 |         raise _transition_error(str(exc)) from exc
708 |
709 |
710 | @router.post(
711 |     "/api/v1/schedules/{schedule_id}/replan",
712 |     response_model=ScheduleVersionResponse,
713 |     status_code=status.HTTP_201_CREATED,
714 |     tags=["schedules"],
715 | )
716 | async def replan_schedule_from_db(
717 |     schedule_id: str,
718 |     payload: ReplanScheduleRequest,
719 |     principal: Principal = Depends(operator_access),
720 |     session: AsyncSession = Depends(get_session),
721 | ) -> ScheduleVersionResponse:
722 |     try:
723 |         return await replan_schedule(session, principal, schedule_id,
payload)
724 |     except LookupError as exc:
725 |         raise _not_found() from exc
726 |     except ValueError as exc:
727 |         raise HTTPException(
728 |             status_code=status.HTTP_422_UNPROCESSABLE_CONTENT,
729 |             detail=str(exc),
730 |         ) from exc
731 |
732 |
733 | @router.post(
734 |     "/api/v1/schedules/{schedule_id}/cancel",
735 |     response_model=ScheduleVersionResponse,
736 |     tags=["schedules"],
737 | )
738 | async def cancel_schedule(
739 |     schedule_id: str,
740 |     principal: Principal = Depends(operator_access),
741 |     session: AsyncSession = Depends(get_session),
742 | ) -> ScheduleVersionResponse:
743 |     try:
744 |         return await cancel_db_schedule(session, principal, schedule_id)
745 |     except LookupError as exc:
746 |         raise _not_found() from exc
747 |     except ValueError as exc:
748 |         raise _transition_error(str(exc)) from exc
749 |
750 |
```

```
@router.delete(
    "/api/v1/schedules/{schedule_id}",
    response_model=DeleteResponse,
    tags=["schedules"],
)
async def remove_schedule(
    schedule_id: str,
    principal: Principal = Depends(operator_access),
    session: AsyncSession = Depends(get_session),
) -> DeleteResponse:
    if schedule_id.isdigit():
        try:
            return await delete_schedule_version(session, principal,
schedule_id)
        except LookupError as exc:
            raise _not_found() from exc
    try:
        record = schedule_workflow.delete(schedule_id)
    except ScheduleNotFoundError as exc:
        raise _not_found() from exc
    return DeleteResponse(
        entity_type="schedule_versions",
        id=schedule_id,
        related_counts={"schedule_events":
len(record.result.assignments)},
    )


@router.get(
    "/api/v1/schedules/{schedule_id}/conflicts",
    response_model=ConflictListResponse,
    tags=["schedules"],
)
async def schedule_conflicts(
    schedule_id: str,
    _: Principal = Depends(authenticated_access),
    session: AsyncSession = Depends(get_session),
) -> ConflictListResponse:
    try:
        return await diagnose_schedule_conflicts(session, schedule_id)
    except LookupError as exc:
        raise _not_found() from exc


@router.get(
    "/api/v1/public/schedules/{schedule_id}",
    response_model=ScheduleVersionResponse,
    tags=["publication"],
)
async def public_schedule(
    schedule_id: str,
    session: AsyncSession = Depends(get_session),
) -> ScheduleVersionResponse:
    if schedule_id.isdigit():
        try:
            return await get_db_schedule(session, schedule_id,
public_only=True)
        except LookupError as exc:
            raise _not_found() from exc
    try:
        return schedule_workflow.get_public(schedule_id)
    except ScheduleNotFoundError as exc:
        raise _not_found() from exc
```

```
811 |
812 |
813 | @router.get(
814 |     "/api/v1/councils",
815 |     response_model=CouncilListResponse,
816 |     tags=["councils"],
817 | )
818 | async def councils(
819 |     _: Principal = Depends(authenticated_access),
820 |     session: AsyncSession = Depends(get_session),
821 | ) -> CouncilListResponse:
822 |     return await list_councils(session)
823 |
824 |
825 | @router.get(
826 |     "/api/v1/councils/{council_id}/materials",
827 |     response_model=CouncilMaterialResponse,
828 |     tags=["councils"],
829 | )
830 | async def council_materials(
831 |     council_id: int,
832 |     _: Principal = Depends(operator_access),
833 |     session: AsyncSession = Depends(get_session),
834 | ) -> CouncilMaterialResponse:
835 |     try:
836 |         return await build_council_material(session, council_id)
837 |     except LookupError as exc:
838 |         raise HTTPException(
839 |             status_code=status.HTTP_404_NOT_FOUND,
840 |             detail="Разову раду не знайдено.",
841 |         ) from exc
842 |
843 |
844 | @router.get(
845 |     "/api/v1/audit-log",
846 |     response_model=AuditLogResponse,
847 |     tags=["audit"],
848 | )
849 | async def audit_log(
850 |     _: Principal = Depends(admin_access),
851 |     session: AsyncSession = Depends(get_session),
852 | ) -> AuditLogResponse:
853 |     return await list_audit_log(session)
854 |
855 |
856 | @router.get("/", response_class=HTMLResponse, include_in_schema=False)
857 | async def public_home(
858 |     request: Request,
859 |     session: AsyncSession = Depends(get_session),
860 | ) -> HTMLResponse:
861 |     load_error = False
862 |     try:
863 |         councils = await list_public_council_cards(session)
864 |     except (OSError, SQLAlchemyError):
865 |         councils = []
866 |         load_error = True
867 |     return templates.TemplateResponse(
868 |         request=request,
869 |         name="public_home.html",
870 |         context={
871 |             "app_environment": settings.app_environment,
872 |             "councils": councils,
873 |             "council_load_error": load_error,
```

```
874 |         },
875 |     )
876 |
877 |
878 | @router.get("/ui", response_class=HTMLResponse, include_in_schema=False)
879 | async def dashboard(request: Request) -> HTMLResponse:
880 |     import_report = None
881 |     if (
882 |         settings.app_environment.casefold() != "production"
883 |         and request.query_params.get("capture") == "import"
884 |     ):
885 |         report_path = (
886 |             Path(__file__).resolve().parents[2]
887 |             / "docs"
888 |             / "edebo_import_validation_report.json"
889 |         )
890 |         if report_path.exists():
891 |             import_report =
json.loads(report_path.read_text(encoding="utf-8"))
892 |     return templates.TemplateResponse(
893 |         request=request,
894 |         name="dashboard.html",
895 |         context={
896 |             "app_environment": settings.app_environment,
897 |             "capture_import_report": import_report,
898 |         },
899 |     )
900 |
901 |
902 | @router.get(
903 |     "/ui/schedules/{schedule_id}",
904 |     response_class=HTMLResponse,
905 |     include_in_schema=False,
906 | )
907 | async def schedule_view_page(
908 |     request: Request,
909 |     schedule_id: str,
910 | ) -> HTMLResponse:
911 |     return templates.TemplateResponse(
912 |         request=request,
913 |         name="schedule_view.html",
914 |         context={
915 |             "schedule_id": schedule_id,
916 |             "app_environment": settings.app_environment,
917 |         },
918 |     )
919 |
920 |
921 | @router.get(
922 |     "/public/schedules/{schedule_id}",
923 |     response_class=HTMLResponse,
924 |     include_in_schema=False,
925 | )
926 | async def public_schedule_page(
927 |     request: Request,
928 |     schedule_id: str,
929 |     session: AsyncSession = Depends(get_session),
930 | ) -> HTMLResponse:
931 |     if schedule_id.isdigit():
932 |         try:
933 |             schedule = await get_db_schedule(session, schedule_id,
public_only=True)
934 |         except LookupError as exc:
```

```
            raise _not_found() from exc
    else:
        try:
            schedule = schedule_workflow.get_public(schedule_id)
        except ScheduleNotFoundError as exc:
            raise _not_found() from exc
    return templates.TemplateResponse(
        request=request,
        name="public_schedule.html",
        context={"schedule": schedule},
    )


def _get_schedule(schedule_id: str) -> ScheduleVersionResponse:
    try:
        return schedule_workflow.get(schedule_id)
    except ScheduleNotFoundError as exc:
        raise _not_found() from exc


def _not_found() -> HTTPException:
    return HTTPException(
        status_code=status.HTTP_404_NOT_FOUND,
        detail="Версію розкладу не знайдено.",
    )


def _transition_error(message: str) -> HTTPException:
    return HTTPException(
        status_code=status.HTTP_409_CONFLICT,
        detail=message,
    )


async def _read_csv_upload(file: UploadFile) -> tuple[str, bytes]:
    if not file.filename or not file.filename.casefold().endswith(".csv"):
        raise HTTPException(
            status_code=status.HTTP_415_UNSUPPORTED_MEDIA_TYPE,
            detail="Підтримується лише CSV-файл.",
        )
    data = await file.read(settings.max_upload_bytes + 1)
    if len(data) > settings.max_upload_bytes:
        raise HTTPException(
            status_code=status.HTTP_413_REQUEST_ENTITY_TOO_LARGE,
            detail="Розмір файлу перевищує встановлене обмеження.",
        )
    return file.filename, data
```

## Лістинг А.1.3 - Pydantic-схеми запитів і відповідей (app/schemas.py)

```
from __future__ import annotations

from datetime import date, datetime
from enum import StrEnum
from typing import Any

from pydantic import BaseModel, ConfigDict, Field

from app.models.enums import CouncilRole, UserRole

```

```
 12 | class SlotPayload(BaseModel):
 13 |     id: str = Field(min_length=1, max_length=64)
 14 |     starts_at: datetime
 15 |     ends_at: datetime | None = None
 16 |
 17 |
 18 | class RoomPayload(BaseModel):
 19 |     id: str = Field(min_length=1, max_length=64)
 20 |     name: str = Field(min_length=1, max_length=255)
 21 |
 22 |
 23 | class CandidatePayload(BaseModel):
 24 |     id: str = Field(min_length=1, max_length=64)
 25 |     name: str = Field(min_length=1, max_length=255)
 26 |     council_members: tuple[str, ...] = Field(min_length=1)
 27 |     allowed_slots: frozenset[str] = Field(min_length=1)
 28 |     preferred_slots: frozenset[str] = frozenset()
 29 |     allowed_rooms: frozenset[str] = frozenset()
 30 |
 31 |
 32 | class AssignmentPayload(BaseModel):
 33 |     event_id: int | None = None
 34 |     candidate_id: str
 35 |     slot_id: str
 36 |     room_id: str
 37 |     locked: bool = False
 38 |     candidate_name: str | None = None
 39 |     slot_label: str | None = None
 40 |     room_name: str | None = None
 41 |
 42 |
 43 | class ScheduleSolveRequest(BaseModel):
 44 |     slots: list[SlotPayload] = Field(min_length=1)
 45 |     rooms: list[RoomPayload] = Field(min_length=1)
 46 |     member_availability: dict[str, set[str]]
 47 |     candidates: list[CandidatePayload] = Field(min_length=1,
max_length=100)
 48 |     previous_assignments: list[AssignmentPayload] = []
 49 |     preference_penalty: int = Field(default=10, ge=0, le=10_000)
 50 |     change_penalty: int = Field(default=30, ge=0, le=10_000)
 51 |     max_defenses_per_member_per_day: int = Field(default=4, ge=1, le=20)
 52 |
 53 |
 54 | class ScheduleSolveResponse(BaseModel):
 55 |     status: str
 56 |     assignments: list[AssignmentPayload]
 57 |     score: int
 58 |     explored_nodes: int
 59 |     elapsed_ms: float
 60 |     diagnostics: list[str]
 61 |
 62 |
 63 | class ScheduleState(StrEnum):
 64 |     DRAFT = "draft"
 65 |     APPROVED = "approved"
 66 |     PUBLISHED = "published"
 67 |     CANCELLED = "cancelled"
 68 |
 69 |
 70 | class ScheduleVersionResponse(BaseModel):
 71 |     model_config = ConfigDict(from_attributes=True)
 72 |
 73 |     id: str
```

```
 74 |     state: ScheduleState
 75 |     created_at: datetime
 76 |     approved_at: datetime | None = None
 77 |     published_at: datetime | None = None
 78 |     result: ScheduleSolveResponse
 79 |
 80 |
 81 | class DemoScheduleRequest(BaseModel):
 82 |     candidate_count: int = Field(default=8, ge=1, le=30)
 83 |
 84 |
 85 | class DemoTokenRequest(BaseModel):
 86 |     role: UserRole = UserRole.OPERATOR
 87 |
 88 |
 89 | class TokenResponse(BaseModel):
 90 |     access_token: str
 91 |     token_type: str = "bearer"
 92 |     expires_in: int
 93 |
 94 |
 95 | class DeleteResponse(BaseModel):
 96 |     entity_type: str
 97 |     id: str
 98 |     deleted: bool = True
 99 |     related_counts: dict[str, int] = Field(default_factory=dict)
100 |
101 |
102 | class ImportIssue(BaseModel):
103 |     row_number: int
104 |     field_name: str | None = None
105 |     message: str
106 |
107 |
108 | class CandidatePreview(BaseModel):
109 |     edebo_card_id: str
110 |     full_name: str
111 |     study_status: str
112 |     study_start: str | None
113 |     study_end: str | None
114 |     educational_program: str
115 |     restored_for_defense: bool
116 |
117 |
118 | class ImportValidationResponse(BaseModel):
119 |     file_name: str
120 |     checksum_sha256: str
121 |     detected_encoding: str
122 |     delimiter: str
123 |     column_count: int
124 |     total_rows: int
125 |     valid_rows: int
126 |     error_rows: int
127 |     duplicate_ids: int
128 |     mapped_columns: dict[str, str]
129 |     ignored_sensitive_columns: list[str]
130 |     issues: list[ImportIssue]
131 |     preview: list[CandidatePreview]
132 |
133 |
134 | class ImportApplyResponse(BaseModel):
135 |     batch_id: int
136 |     total_rows: int
```

```
137 |     inserted_rows: int
138 |     updated_rows: int
139 |     error_rows: int
140 |
141 |
142 | class CandidateSummary(BaseModel):
143 |     id: int
144 |     edebo_card_id: str
145 |     full_name: str
146 |     study_status: str
147 |     study_start: date | None = None
148 |     study_end: date | None = None
149 |     educational_program: str | None = None
150 |     department: str | None = None
151 |     stage_count: int = 0
152 |     council_status: str | None = None
153 |
154 |
155 | class CandidateListResponse(BaseModel):
156 |     items: list[CandidateSummary]
157 |
158 |
159 | class CandidateUpdatePayload(BaseModel):
160 |     thesis_title: str | None = Field(default=None, max_length=2000)
161 |     supervisor_name: str | None = Field(default=None, max_length=255)
162 |
163 |
164 | class DocumentSummary(BaseModel):
165 |     id: int
166 |     stage_id: int
167 |     stage_title: str
168 |     original_name: str
169 |     content_type: str
170 |     size_bytes: int
171 |     storage_key: str
172 |     created_at: datetime | None = None
173 |
174 |
175 | class StageDocumentPayload(BaseModel):
176 |     original_name: str = Field(min_length=1, max_length=255)
177 |     external_url: str | None = Field(default=None, max_length=500)
178 |     content_type: str | None = Field(default=None, max_length=255)
179 |
180 |
181 | class StageDocumentResponse(BaseModel):
182 |     document: DocumentSummary
183 |
184 |
185 | class CouncilMemberSummary(BaseModel):
186 |     id: int
187 |     full_name: str
188 |     role: str
189 |     institution: str
190 |     academic_degree: str | None = None
191 |     specialty: str | None = None
192 |     position: str | None = None
193 |     profile_url: str | None = None
194 |     is_external: bool = False
195 |     qualification_confirmed: bool = False
196 |     conflict_of_interest_absent: bool = False
197 |
198 |
199 | class CouncilMemberInput(BaseModel):
```

```
200 |     role: CouncilRole
201 |     full_name: str = Field(min_length=1, max_length=255)
202 |     email: str | None = Field(default=None, max_length=255)
203 |     institution: str = Field(default="ІФНТУНГ", min_length=1,
max_length=500)
204 |     academic_degree: str | None = Field(
205 |         default="кандидат/доктор наук",
206 |         max_length=255,
207 |     )
208 |     specialty: str | None = Field(default="Комп'ютерна інженерія",
max_length=500)
209 |     position: str | None = Field(default=None, max_length=1000)
210 |     profile_url: str | None = Field(default=None, max_length=1000)
211 |     is_external: bool = False
212 |     qualification_confirmed: bool = True
213 |     conflict_of_interest_absent: bool = True
214 |
215 |
216 | class CouncilCompositionPayload(BaseModel):
217 |     members: list[CouncilMemberInput] = Field(min_length=5, max_length=5)
218 |
219 |
220 | class PublicReviewPayload(BaseModel):
221 |     member_name: str = Field(min_length=1, max_length=255)
222 |     kind: str = Field(default="Рецензія", max_length=50)
223 |     material_url: str | None = Field(default=None, max_length=1000)
224 |     signature_url: str | None = Field(default=None, max_length=1000)
225 |
226 |
227 | class CouncilMetadataPayload(BaseModel):
228 |     order_number: str | None = Field(default=None, max_length=100)
229 |     order_date: date | None = None
230 |     publication_date: date | None = None
231 |     council_code: str | None = Field(default=None, max_length=100)
232 |     order_url: str | None = Field(default=None, max_length=1000)
233 |     announcement_url: str | None = Field(default=None, max_length=1000)
234 |     knowledge_field: str | None = Field(default=None, max_length=500)
235 |     specialty: str | None = Field(default=None, max_length=500)
236 |     ukrintei_card_number: str | None = Field(default=None, max_length=100)
237 |     ukrintei_card_url: str | None = Field(default=None, max_length=1000)
238 |     nrat_url: str | None = Field(default=None, max_length=1000)
239 |     naqa_id: str | None = Field(default=None, max_length=100)
240 |     naqa_url: str | None = Field(default=None, max_length=1000)
241 |     defense_datetime: datetime | None = None
242 |     defense_room: str | None = Field(default=None, max_length=255)
243 |     defense_platform: str | None = Field(default=None, max_length=255)
244 |     conference_url: str | None = Field(default=None, max_length=1000)
245 |     conference_id: str | None = Field(default=None, max_length=100)
246 |     conference_passcode: str | None = Field(default=None, max_length=100)
247 |     livestream_url: str | None = Field(default=None, max_length=1000)
248 |     dissertation_url: str | None = Field(default=None, max_length=1000)
249 |     dissertation_signature_url: str | None = Field(default=None,
max_length=1000)
250 |     novelty_conclusion_url: str | None = Field(default=None,
max_length=1000)
251 |     video_url: str | None = Field(default=None, max_length=1000)
252 |     decision_url: str | None = Field(default=None, max_length=1000)
253 |     video_stamp_note: str | None = Field(default=None, max_length=1000)
254 |     reviews: list[PublicReviewPayload] = Field(default_factory=list,
max_length=12)
255 |
256 |
257 | class CandidateScheduleSummary(BaseModel):
```

```
258 |     schedule_id: str
259 |     state: str
260 |     event_id: int
261 |     starts_at: datetime
262 |     ends_at: datetime
263 |     room_name: str
264 |     locked: bool = False
265 |
266 |
267 | class CandidateDetailResponse(BaseModel):
268 |     candidate: CandidateSummary
269 |     thesis_title: str | None = None
270 |     supervisor_name: str | None = None
271 |     study_form: str | None = None
272 |     course: str | None = None
273 |     group_name: str | None = None
274 |     restored_for_defense: bool = False
275 |     progress_percent: int = 0
276 |     stage_counts: dict[str, int]
277 |     primary_council_id: int | None = None
278 |     council_status: str | None = None
279 |     council_order_number: str | None = None
280 |     council_order_date: date | None = None
281 |     council_publication_date: date | None = None
282 |     council_public_details: dict[str, Any] = Field(default_factory=dict)
283 |     defense_window_start: date | None = None
284 |     defense_window_end: date | None = None
285 |     stages: list["StageSummary"]
286 |     issues: list["NormativeIssue"]
287 |     council_members: list[CouncilMemberSummary]
288 |     documents: list[DocumentSummary]
289 |     schedules: list[CandidateScheduleSummary]
290 |
291 |
292 | class StageSummary(BaseModel):
293 |     id: int
294 |     candidate_id: int
295 |     candidate_name: str
296 |     code: str
297 |     title: str
298 |     responsible: str | None = None
299 |     planned_date: date | None = None
300 |     actual_date: date | None = None
301 |     status: str
302 |     source_reference: str | None = None
303 |     normative_status: str = "pending"
304 |     normative_message: str | None = None
305 |     deadline_date: date | None = None
306 |
307 |
308 | class StageListResponse(BaseModel):
309 |     items: list[StageSummary]
310 |
311 |
312 | class NormativeIssue(BaseModel):
313 |     candidate_id: int
314 |     candidate_name: str
315 |     stage_id: int
316 |     code: str
317 |     title: str
318 |     severity: str
319 |     message: str
320 |     planned_date: date | None = None
```

```
    actual_date: date | None = None
    deadline_date: date | None = None


class NormativeControlResponse(BaseModel):
    items: list[NormativeIssue]


class StageEventPayload(BaseModel):
    actual_date: date
    comment: str | None = Field(default=None, max_length=1000)


class StageEventResponse(BaseModel):
    stage: StageSummary
    recalculated_stages: int
    issues: list[NormativeIssue]


class WorkspaceSummaryResponse(BaseModel):
    candidates: int
    stages: int
    councils: int
    schedules: int
    published_schedules: int
    rooms: list[str]


class RoomAdminSummary(BaseModel):
    id: int
    name: str
    location: str | None = None
    online_url: str | None = None
    capacity: int
    is_active: bool
    schedule_event_count: int = 0


class RoomCreateRequest(BaseModel):
    name: str = Field(min_length=1, max_length=255)
    location: str | None = Field(default=None, max_length=500)
    online_url: str | None = Field(default=None, max_length=1000)
    capacity: int = Field(default=30, ge=1, le=500)
    is_active: bool = True


class RoomUpdateRequest(BaseModel):
    name: str | None = Field(default=None, min_length=1, max_length=255)
    location: str | None = Field(default=None, max_length=500)
    online_url: str | None = Field(default=None, max_length=1000)
    capacity: int | None = Field(default=None, ge=1, le=500)
    is_active: bool | None = None


class OperationalSettingsResponse(BaseModel):
    defense_duration_minutes: int
    default_planning_days: int
    default_max_member_defenses_per_day: int
    defense_start_times: list[str]


class OperationalSettingsUpdateRequest(BaseModel):
```

```
383 |     defense_duration_minutes: int | None = Field(default=None, ge=180,
le=480)
384 |     default_planning_days: int | None = Field(default=None, ge=1, le=120)
385 |     default_max_member_defenses_per_day: int | None = Field(
386 |         default=None,
387 |         ge=1,
388 |         le=20,
389 |     )
390 |     defense_start_times: list[str] | None = Field(default=None,
min_length=1, max_length=8)
391 |
392 |
393 | class SystemSettingsResponse(BaseModel):
394 |     operational: OperationalSettingsResponse
395 |     rooms: list[RoomAdminSummary]
396 |     active_rooms: list[str]
397 |     app_environment: str
398 |     demo_tokens_enabled: bool
399 |     google_allowed_domain: str | None = None
400 |     google_oauth_configured: bool = False
401 |
402 |
403 | class GeneratePlansResponse(BaseModel):
404 |     candidates_processed: int
405 |     stages_created: int
406 |     rules_created: int
407 |
408 |
409 | class ScheduleGenerateRequest(BaseModel):
410 |     candidate_id: int | None = Field(default=None, ge=1)
411 |     candidate_limit: int = Field(default=20, ge=1, le=100)
412 |     start_date: date | None = None
413 |     room_name: str | None = Field(default=None, max_length=255)
414 |     day_count: int | None = Field(default=None, ge=1, le=120)
415 |     locked_event_ids: list[int] = Field(default_factory=list)
416 |     blocked_event_ids: list[int] = Field(default_factory=list)
417 |     max_defenses_per_member_per_day: int | None = Field(default=None,
ge=1, le=20)
418 |
419 |
420 | class ReplanScheduleRequest(BaseModel):
421 |     blocked_event_ids: list[int] = Field(default_factory=list)
422 |     locked_event_ids: list[int] = Field(default_factory=list)
423 |     room_name: str | None = Field(default=None, max_length=255)
424 |     day_count: int | None = Field(default=None, ge=1, le=120)
425 |     max_defenses_per_member_per_day: int | None = Field(default=None,
ge=1, le=20)
426 |
427 |
428 | class ScheduleSummary(BaseModel):
429 |     id: str
430 |     version_number: int
431 |     state: ScheduleState
432 |     created_at: datetime
433 |     published_at: datetime | None = None
434 |     score: int | None = None
435 |     event_count: int = 0
436 |
437 |
438 | class ScheduleListResponse(BaseModel):
439 |     items: list[ScheduleSummary]
440 |
441 |
```

```
442 | class ConflictSummary(BaseModel):
443 |     severity: str
444 |     message: str
445 |
446 |
447 | class ConflictListResponse(BaseModel):
448 |     items: list[ConflictSummary]
449 |
450 |
451 | class CouncilSummary(BaseModel):
452 |     id: int
453 |     candidate_id: int
454 |     candidate_name: str
455 |     status: str
456 |     member_count: int
457 |     order_number: str | None = None
458 |     order_date: date | None = None
459 |
460 |
461 | class CouncilListResponse(BaseModel):
462 |     items: list[CouncilSummary]
463 |
464 |
465 | class CouncilMaterialResponse(BaseModel):
466 |     council_id: int
467 |     candidate_name: str
468 |     title: str
469 |     body: str
470 |
471 |
472 | class AuditLogEntry(BaseModel):
473 |     id: int
474 |     action: str
475 |     entity_type: str
476 |     entity_id: str | None = None
477 |     occurred_at: datetime
478 |
479 |
480 | class AuditLogResponse(BaseModel):
481 |     items: list[AuditLogEntry]
```

## ДОДАТОК А.2

### ORM-моделі та міграції структури бази даних

У додатку подано моделі сутностей PostgreSQL, перелічення доменної моделі та міграції Alembic, що формують структуру бази даних web-сервісу.

### Лістинг А.2.1 - ORM-моделі бази даних (app/models/entities.py)

```
  1 | from __future__ import annotations
  2 |
  3 | from datetime import date, datetime
  4 | from typing import Any
  5 |
  6 | from sqlalchemy import (
  7 |     BigInteger,
  8 |     Boolean,
```

```
  9 |     CheckConstraint,
 10 |     Date,
 11 |     DateTime,
 12 |     Enum,
 13 |     ForeignKey,
 14 |     Index,
 15 |     Integer,
 16 |     JSON,
 17 |     String,
 18 |     Text,
 19 |     UniqueConstraint,
 20 |     func,
 21 | )
 22 | from sqlalchemy.orm import Mapped, mapped_column, relationship
 23 |
 24 | from app.models.base import Base, TimestampMixin
 25 | from app.models.enums import (
 26 |     AvailabilityKind,
 27 |     CouncilRole,
 28 |     CouncilStatus,
 29 |     ImportStatus,
 30 |     RuleDirection,
 31 |     RuleUnit,
 32 |     ScheduleEventStatus,
 33 |     ScheduleStatus,
 34 |     StageStatus,
 35 |     UserRole,
 36 | )
 37 |
 38 |
 39 | def enum_type(enum_class: type, name: str, length: int = 32) -> Enum:
 40 |     return Enum(
 41 |         enum_class,
 42 |         name=name,
 43 |         native_enum=False,
 44 |         create_constraint=True,
 45 |         validate_strings=True,
 46 |         values_callable=lambda members: [member.value for member in
members],
 47 |         length=length,
 48 |     )
 49 |
 50 |
 51 | class User(TimestampMixin, Base):
 52 |     __tablename__ = "users"
 53 |
 54 |     id: Mapped[int] = mapped_column(primary_key=True)
 55 |     email: Mapped[str] = mapped_column(String(255), unique=True,
index=True)
 56 |     full_name: Mapped[str] = mapped_column(String(255))
 57 |     google_subject: Mapped[str | None] = mapped_column(
 58 |         String(255),
 59 |         unique=True,
 60 |         nullable=True,
 61 |     )
 62 |     is_active: Mapped[bool] = mapped_column(Boolean, default=True,
nullable=False)
 63 |
 64 |     roles: Mapped[list["UserRoleAssignment"]] = relationship(
 65 |         back_populates="user",
 66 |         cascade="all, delete-orphan",
 67 |     )
```

```
 68 |     candidate: Mapped["Candidate | None"] =
relationship(back_populates="user")
 69 |     council_member: Mapped["CouncilMember | None"] = relationship(
 70 |         back_populates="user"
 71 |     )
 72 |
 73 |
 74 | class UserRoleAssignment(Base):
 75 |     __tablename__ = "user_roles"
 76 |
 77 |     user_id: Mapped[int] = mapped_column(
 78 |         ForeignKey("users.id", ondelete="CASCADE"),
 79 |         primary_key=True,
 80 |     )
 81 |     role: Mapped[UserRole] = mapped_column(
 82 |         enum_type(UserRole, "user_role", 24),
 83 |         primary_key=True,
 84 |     )
 85 |
 86 |     user: Mapped[User] = relationship(back_populates="roles")
 87 |
 88 |
 89 | class Department(TimestampMixin, Base):
 90 |     __tablename__ = "departments"
 91 |
 92 |     id: Mapped[int] = mapped_column(primary_key=True)
 93 |     name: Mapped[str] = mapped_column(String(500), unique=True)
 94 |
 95 |     candidates: Mapped[list["Candidate"]] =
relationship(back_populates="department")
 96 |
 97 |
 98 | class EducationalProgram(TimestampMixin, Base):
 99 |     __tablename__ = "educational_programs"
100 |     __table_args__ = (
101 |         UniqueConstraint(
102 |             "edebo_program_id",
103 |             name="uq_educational_programs_edebo_program_id",
104 |         ),
105 |     )
106 |
107 |     id: Mapped[int] = mapped_column(primary_key=True)
108 |     edebo_program_id: Mapped[str | None] = mapped_column(String(64),
nullable=True)
109 |     name: Mapped[str] = mapped_column(String(500))
110 |     specialty: Mapped[str] = mapped_column(String(500))
111 |     degree_level: Mapped[str | None] = mapped_column(String(255),
nullable=True)
112 |
113 |     candidates: Mapped[list["Candidate"]] = relationship(
114 |         back_populates="educational_program"
115 |     )
116 |
117 |
118 | class ImportBatch(TimestampMixin, Base):
119 |     __tablename__ = "import_batches"
120 |     __table_args__ = (
121 |         CheckConstraint(
122 |             "total_rows >= 0 AND inserted_rows >= 0 AND updated_rows >= 0
"
123 |             "AND skipped_rows >= 0 AND error_rows >= 0",
124 |             name="non_negative_counters",
125 |         ),
```

```
126 |     )
127 |
128 |     id: Mapped[int] = mapped_column(primary_key=True)
129 |     file_name: Mapped[str] = mapped_column(String(255))
130 |     file_checksum: Mapped[str] = mapped_column(String(64), index=True)
131 |     detected_encoding: Mapped[str] = mapped_column(String(32))
132 |     status: Mapped[ImportStatus] = mapped_column(
133 |         enum_type(ImportStatus, "import_status", 16),
134 |         default=ImportStatus.PENDING,
135 |     )
136 |     total_rows: Mapped[int] = mapped_column(Integer, default=0)
137 |     inserted_rows: Mapped[int] = mapped_column(Integer, default=0)
138 |     updated_rows: Mapped[int] = mapped_column(Integer, default=0)
139 |     skipped_rows: Mapped[int] = mapped_column(Integer, default=0)
140 |     error_rows: Mapped[int] = mapped_column(Integer, default=0)
141 |     imported_by_id: Mapped[int] = mapped_column(ForeignKey("users.id"))
142 |     applied_at: Mapped[datetime | None] = mapped_column(
143 |         DateTime(timezone=True),
144 |         nullable=True,
145 |     )
146 |
147 |     imported_by: Mapped[User] = relationship()
148 |     errors: Mapped[list["ImportError"]] = relationship(
149 |         back_populates="batch",
150 |         cascade="all, delete-orphan",
151 |     )
152 |     candidates: Mapped[list["Candidate"]] =
relationship(back_populates="source_import")
153 |
154 |
155 | class ImportError(Base):
156 |     __tablename__ = "import_errors"
157 |     __table_args__ = (
158 |         UniqueConstraint(
159 |             "batch_id",
160 |             "row_number",
161 |             "field_name",
162 |             name="uq_import_errors_row_field",
163 |         ),
164 |         CheckConstraint("row_number > 0", name="positive_row_number"),
165 |     )
166 |
167 |     id: Mapped[int] = mapped_column(primary_key=True)
168 |     batch_id: Mapped[int] = mapped_column(
169 |         ForeignKey("import_batches.id", ondelete="CASCADE"),
170 |         index=True,
171 |     )
172 |     row_number: Mapped[int] = mapped_column(Integer)
173 |     field_name: Mapped[str | None] = mapped_column(String(255),
nullable=True)
174 |     message: Mapped[str] = mapped_column(Text)
175 |     raw_value: Mapped[str | None] = mapped_column(Text, nullable=True)
176 |
177 |     batch: Mapped[ImportBatch] = relationship(back_populates="errors")
178 |
179 |
180 | class Candidate(TimestampMixin, Base):
181 |     __tablename__ = "candidates"
182 |     __table_args__ = (
183 |         CheckConstraint(
184 |             "study_end IS NULL OR study_start IS NULL OR study_end >=
study_start",
185 |             name="valid_study_period",
```

```
186 |         ),
187 |     )
188 |
189 |     id: Mapped[int] = mapped_column(primary_key=True)
190 |     edebo_card_id: Mapped[str] = mapped_column(String(64), unique=True,
index=True)
191 |     full_name: Mapped[str] = mapped_column(String(255), index=True)
192 |     study_status: Mapped[str] = mapped_column(String(255))
193 |     study_start: Mapped[date | None] = mapped_column(Date, nullable=True)
194 |     study_end: Mapped[date | None] = mapped_column(Date, nullable=True,
index=True)
195 |     study_form: Mapped[str | None] = mapped_column(String(255),
nullable=True)
196 |     course: Mapped[str | None] = mapped_column(String(32), nullable=True)
197 |     group_name: Mapped[str | None] = mapped_column(String(255),
nullable=True)
198 |     restored_for_defense: Mapped[bool] = mapped_column(
199 |         Boolean,
200 |         default=False,
201 |         nullable=False,
202 |     )
203 |     source_updated_at: Mapped[datetime | None] = mapped_column(
204 |         DateTime(timezone=True),
205 |         nullable=True,
206 |     )
207 |     thesis_title: Mapped[str | None] = mapped_column(Text, nullable=True)
208 |     supervisor_name: Mapped[str | None] = mapped_column(String(255),
nullable=True)
209 |     user_id: Mapped[int | None] = mapped_column(
210 |         ForeignKey("users.id", ondelete="SET NULL"),
211 |         unique=True,
212 |         nullable=True,
213 |     )
214 |     department_id: Mapped[int | None] = mapped_column(
215 |         ForeignKey("departments.id", ondelete="SET NULL"),
216 |         nullable=True,
217 |     )
218 |     educational_program_id: Mapped[int | None] = mapped_column(
219 |         ForeignKey("educational_programs.id", ondelete="SET NULL"),
220 |         nullable=True,
221 |     )
222 |     source_import_id: Mapped[int | None] = mapped_column(
223 |         ForeignKey("import_batches.id", ondelete="SET NULL"),
224 |         nullable=True,
225 |     )
226 |
227 |     user: Mapped[User | None] = relationship(back_populates="candidate")
228 |     department: Mapped[Department | None] =
relationship(back_populates="candidates")
229 |     educational_program: Mapped[EducationalProgram | None] = relationship(
230 |         back_populates="candidates"
231 |     )
232 |     source_import: Mapped[ImportBatch | None] = relationship(
233 |         back_populates="candidates"
234 |     )
235 |     stages: Mapped[list["AttestationStage"]] = relationship(
236 |         back_populates="candidate",
237 |         cascade="all, delete-orphan",
238 |     )
239 |     councils: Mapped[list["Council"]] =
relationship(back_populates="candidate")
240 |
241 |
```

```
class AttestationRule(TimestampMixin, Base):
    __tablename__ = "attestation_rules"
    __table_args__ = (
        CheckConstraint("duration_value >= 0",
name="non_negative_duration"),
        CheckConstraint(
            "effective_to IS NULL OR effective_to >= effective_from",
            name="valid_effective_period",
        ),
    )

    id: Mapped[int] = mapped_column(primary_key=True)
    code: Mapped[str] = mapped_column(String(64), unique=True)
    title: Mapped[str] = mapped_column(String(500))
    trigger_code: Mapped[str] = mapped_column(String(64))
    duration_value: Mapped[int] = mapped_column(Integer)
    unit: Mapped[RuleUnit] = mapped_column(
        enum_type(RuleUnit, "rule_unit", 24)
    )
    direction: Mapped[RuleDirection] = mapped_column(
        enum_type(RuleDirection, "rule_direction", 12)
    )
    source_reference: Mapped[str] = mapped_column(String(500))
    effective_from: Mapped[date] = mapped_column(Date)
    effective_to: Mapped[date | None] = mapped_column(Date, nullable=True)
    is_active: Mapped[bool] = mapped_column(Boolean, default=True)

    stages: Mapped[list["AttestationStage"]] =
relationship(back_populates="rule")


class AttestationStage(TimestampMixin, Base):
    __tablename__ = "attestation_stages"
    __table_args__ = (
        UniqueConstraint(
            "candidate_id",
            "code",
            name="uq_attestation_stages_candidate_code",
        ),
        CheckConstraint(
            "actual_date IS NULL OR status IN ('completed',
'needs_revision')",
            name="actual_date_matches_status",
        ),
    )

    id: Mapped[int] = mapped_column(primary_key=True)
    candidate_id: Mapped[int] = mapped_column(
        ForeignKey("candidates.id", ondelete="CASCADE"),
        index=True,
    )
    rule_id: Mapped[int | None] = mapped_column(
        ForeignKey("attestation_rules.id", ondelete="SET NULL"),
        nullable=True,
    )
    code: Mapped[str] = mapped_column(String(64))
    title: Mapped[str] = mapped_column(String(500))
    planned_date: Mapped[date | None] = mapped_column(Date, nullable=True,
index=True)
    actual_date: Mapped[date | None] = mapped_column(Date, nullable=True)
    status: Mapped[StageStatus] = mapped_column(
        enum_type(StageStatus, "stage_status", 24),
        default=StageStatus.PLANNED,
```

```
301 |         index=True,
302 |     )
303 |     responsible_user_id: Mapped[int | None] = mapped_column(
304 |         ForeignKey("users.id", ondelete="SET NULL"),
305 |         nullable=True,
306 |     )
307 |     comment: Mapped[str | None] = mapped_column(Text, nullable=True)
308 |
309 |     candidate: Mapped[Candidate] = relationship(back_populates="stages")
310 |     rule: Mapped[AttestationRule | None] =
relationship(back_populates="stages")
311 |     responsible_user: Mapped[User | None] = relationship()
312 |     documents: Mapped[list["Document"]] = relationship(
313 |         back_populates="stage",
314 |         cascade="all, delete-orphan",
315 |     )
316 |
317 |
318 | class Document(TimestampMixin, Base):
319 |     __tablename__ = "documents"
320 |
321 |     id: Mapped[int] = mapped_column(primary_key=True)
322 |     stage_id: Mapped[int] = mapped_column(
323 |         ForeignKey("attestation_stages.id", ondelete="CASCADE"),
324 |         index=True,
325 |     )
326 |     uploaded_by_id: Mapped[int] = mapped_column(
327 |         ForeignKey("users.id", ondelete="RESTRICT")
328 |     )
329 |     original_name: Mapped[str] = mapped_column(String(255))
330 |     storage_key: Mapped[str] = mapped_column(String(500), unique=True)
331 |     content_type: Mapped[str] = mapped_column(String(255))
332 |     size_bytes: Mapped[int] = mapped_column(BigInteger)
333 |     checksum_sha256: Mapped[str] = mapped_column(String(64), index=True)
334 |
335 |     stage: Mapped[AttestationStage] =
relationship(back_populates="documents")
336 |     uploaded_by: Mapped[User] = relationship()
337 |
338 |     __table_args__ = (
339 |         CheckConstraint("size_bytes >= 0", name="non_negative_size"),
340 |     )
341 |
342 |
343 | class Council(TimestampMixin, Base):
344 |     __tablename__ = "councils"
345 |     __table_args__ = (
346 |         UniqueConstraint(
347 |             "candidate_id",
348 |             "sequence_number",
349 |             name="uq_councils_candidate_sequence",
350 |         ),
351 |         CheckConstraint("sequence_number > 0",
name="positive_sequence_number"),
352 |     )
353 |
354 |     id: Mapped[int] = mapped_column(primary_key=True)
355 |     candidate_id: Mapped[int] = mapped_column(
356 |         ForeignKey("candidates.id", ondelete="CASCADE"),
357 |         index=True,
358 |     )
359 |     sequence_number: Mapped[int] = mapped_column(Integer, default=1)
360 |     status: Mapped[CouncilStatus] = mapped_column(
```

```
361 |         enum_type(CouncilStatus, "council_status", 16),
362 |         default=CouncilStatus.DRAFT,
363 |     )
364 |     order_number: Mapped[str | None] = mapped_column(String(100),
nullable=True)
365 |     order_date: Mapped[date | None] = mapped_column(Date, nullable=True)
366 |     publication_date: Mapped[date | None] = mapped_column(Date,
nullable=True)
367 |     public_details: Mapped[dict[str, Any]] = mapped_column(
368 |         JSON,
369 |         default=dict,
370 |         nullable=False,
371 |     )
372 |
373 |     candidate: Mapped[Candidate] = relationship(back_populates="councils")
374 |     memberships: Mapped[list["CouncilMembership"]] = relationship(
375 |         back_populates="council",
376 |         cascade="all, delete-orphan",
377 |     )
378 |     schedule_events: Mapped[list["ScheduleEvent"]] = relationship(
379 |         back_populates="council"
380 |     )
381 |
382 |
383 | class CouncilMember(TimestampMixin, Base):
384 |     __tablename__ = "council_members"
385 |
386 |     id: Mapped[int] = mapped_column(primary_key=True)
387 |     user_id: Mapped[int | None] = mapped_column(
388 |         ForeignKey("users.id", ondelete="SET NULL"),
389 |         unique=True,
390 |         nullable=True,
391 |     )
392 |     full_name: Mapped[str] = mapped_column(String(255), index=True)
393 |     email: Mapped[str | None] = mapped_column(String(255), nullable=True)
394 |     institution: Mapped[str] = mapped_column(String(500))
395 |     academic_degree: Mapped[str | None] = mapped_column(String(255),
nullable=True)
396 |     specialty: Mapped[str | None] = mapped_column(String(500),
nullable=True)
397 |     position: Mapped[str | None] = mapped_column(String(1000),
nullable=True)
398 |     profile_url: Mapped[str | None] = mapped_column(String(1000),
nullable=True)
399 |
400 |     user: Mapped[User | None] =
relationship(back_populates="council_member")
401 |     memberships: Mapped[list["CouncilMembership"]] = relationship(
402 |         back_populates="member"
403 |     )
404 |     availability: Mapped[list["MemberAvailability"]] = relationship(
405 |         back_populates="member",
406 |         cascade="all, delete-orphan",
407 |     )
408 |
409 |
410 | class CouncilMembership(Base):
411 |     __tablename__ = "council_memberships"
412 |     __table_args__ = (
413 |         UniqueConstraint(
414 |             "council_id",
415 |             "member_id",
416 |             name="uq_council_memberships_council_member",
```

```
        ),
    )

    id: Mapped[int] = mapped_column(primary_key=True)
    council_id: Mapped[int] = mapped_column(
        ForeignKey("councils.id", ondelete="CASCADE"),
        index=True,
    )
    member_id: Mapped[int] = mapped_column(
        ForeignKey("council_members.id", ondelete="RESTRICT"),
        index=True,
    )
    role: Mapped[CouncilRole] = mapped_column(
        enum_type(CouncilRole, "council_role", 16)
    )
    is_external: Mapped[bool] = mapped_column(Boolean, default=False)
    qualification_confirmed: Mapped[bool] = mapped_column(Boolean,
default=False)
    conflict_of_interest_absent: Mapped[bool] = mapped_column(
        Boolean,
        default=False,
    )

    council: Mapped[Council] = relationship(back_populates="memberships")
    member: Mapped[CouncilMember] =
relationship(back_populates="memberships")


class MemberAvailability(TimestampMixin, Base):
    __tablename__ = "member_availability"
    __table_args__ = (
        UniqueConstraint(
            "member_id",
            "starts_at",
            "ends_at",
            "kind",
            name="uq_member_availability_interval",
        ),
        CheckConstraint("ends_at > starts_at", name="valid_interval"),
    )

    id: Mapped[int] = mapped_column(primary_key=True)
    member_id: Mapped[int] = mapped_column(
        ForeignKey("council_members.id", ondelete="CASCADE"),
        index=True,
    )
    starts_at: Mapped[datetime] = mapped_column(DateTime(timezone=True))
    ends_at: Mapped[datetime] = mapped_column(DateTime(timezone=True))
    kind: Mapped[AvailabilityKind] = mapped_column(
        enum_type(AvailabilityKind, "availability_kind", 16)
    )

    member: Mapped[CouncilMember] =
relationship(back_populates="availability")


class Room(TimestampMixin, Base):
    __tablename__ = "rooms"
    __table_args__ = (
        CheckConstraint("capacity > 0", name="positive_capacity"),
    )

    id: Mapped[int] = mapped_column(primary_key=True)
```

```
477 |     name: Mapped[str] = mapped_column(String(255), unique=True)
478 |     location: Mapped[str | None] = mapped_column(String(500),
nullable=True)
479 |     online_url: Mapped[str | None] = mapped_column(String(1000),
nullable=True)
480 |     capacity: Mapped[int] = mapped_column(Integer, default=1)
481 |     is_active: Mapped[bool] = mapped_column(Boolean, default=True)
482 |
483 |     schedule_events: Mapped[list["ScheduleEvent"]] = relationship(
484 |         back_populates="room"
485 |     )
486 |
487 |
488 | class AppSetting(TimestampMixin, Base):
489 |     __tablename__ = "app_settings"
490 |
491 |     key: Mapped[str] = mapped_column(String(100), primary_key=True)
492 |     value: Mapped[dict[str, Any]] = mapped_column(JSON, nullable=False)
493 |     description: Mapped[str | None] = mapped_column(String(500),
nullable=True)
494 |
495 |
496 | class TimeSlot(TimestampMixin, Base):
497 |     __tablename__ = "time_slots"
498 |     __table_args__ = (
499 |         UniqueConstraint("starts_at", "ends_at",
name="uq_time_slots_interval"),
500 |         CheckConstraint("ends_at > starts_at", name="valid_interval"),
501 |     )
502 |
503 |     id: Mapped[int] = mapped_column(primary_key=True)
504 |     starts_at: Mapped[datetime] = mapped_column(
505 |         DateTime(timezone=True),
506 |         index=True,
507 |     )
508 |     ends_at: Mapped[datetime] = mapped_column(DateTime(timezone=True))
509 |     is_active: Mapped[bool] = mapped_column(Boolean, default=True)
510 |
511 |     schedule_events: Mapped[list["ScheduleEvent"]] = relationship(
512 |         back_populates="slot"
513 |     )
514 |
515 |
516 | class ScheduleVersion(TimestampMixin, Base):
517 |     __tablename__ = "schedule_versions"
518 |     __table_args__ = (
519 |         UniqueConstraint("version_number",
name="uq_schedule_versions_number"),
520 |         CheckConstraint("version_number > 0",
name="positive_version_number"),
521 |         CheckConstraint("period_end >= period_start",
name="valid_period"),
522 |     )
523 |
524 |     id: Mapped[int] = mapped_column(primary_key=True)
525 |     version_number: Mapped[int] = mapped_column(Integer)
526 |     status: Mapped[ScheduleStatus] = mapped_column(
527 |         enum_type(ScheduleStatus, "schedule_status", 16),
528 |         default=ScheduleStatus.DRAFT,
529 |         index=True,
530 |     )
531 |     period_start: Mapped[date] = mapped_column(Date)
532 |     period_end: Mapped[date] = mapped_column(Date)
```

```
    score: Mapped[int | None] = mapped_column(Integer, nullable=True)
    created_by_id: Mapped[int] = mapped_column(
        ForeignKey("users.id", ondelete="RESTRICT")
    )
    published_at: Mapped[datetime | None] = mapped_column(
        DateTime(timezone=True),
        nullable=True,
    )

    created_by: Mapped[User] = relationship()
    events: Mapped[list["ScheduleEvent"]] = relationship(
        back_populates="schedule_version",
        cascade="all, delete-orphan",
    )


class ScheduleEvent(TimestampMixin, Base):
    __tablename__ = "schedule_events"
    __table_args__ = (
        UniqueConstraint(
            "schedule_version_id",
            "candidate_id",
            name="uq_schedule_events_version_candidate",
        ),
        UniqueConstraint(
            "schedule_version_id",
            "slot_id",
            "room_id",
            name="uq_schedule_events_version_slot_room",
        ),
    )

    id: Mapped[int] = mapped_column(primary_key=True)
    schedule_version_id: Mapped[int] = mapped_column(
        ForeignKey("schedule_versions.id", ondelete="CASCADE"),
        index=True,
    )
    candidate_id: Mapped[int] = mapped_column(
        ForeignKey("candidates.id", ondelete="CASCADE"),
        index=True,
    )
    council_id: Mapped[int] = mapped_column(
        ForeignKey("councils.id", ondelete="RESTRICT"),
        index=True,
    )
    slot_id: Mapped[int] = mapped_column(
        ForeignKey("time_slots.id", ondelete="RESTRICT"),
        index=True,
    )
    room_id: Mapped[int] = mapped_column(
        ForeignKey("rooms.id", ondelete="RESTRICT"),
        index=True,
    )
    status: Mapped[ScheduleEventStatus] = mapped_column(
        enum_type(ScheduleEventStatus, "schedule_event_status", 16),
        default=ScheduleEventStatus.PLANNED,
    )
    is_locked: Mapped[bool] = mapped_column(Boolean, default=False)
    change_penalty: Mapped[int] = mapped_column(Integer, default=0)

    schedule_version: Mapped[ScheduleVersion] =
relationship(back_populates="events")
    candidate: Mapped[Candidate] = relationship()
```

```
595 |     council: Mapped[Council] =
relationship(back_populates="schedule_events")
596 |     slot: Mapped[TimeSlot] =
relationship(back_populates="schedule_events")
597 |     room: Mapped[Room] = relationship(back_populates="schedule_events")
598 |
599 |
600 | class AuditLog(Base):
601 |     __tablename__ = "audit_log"
602 |
603 |     id: Mapped[int] = mapped_column(BigInteger, primary_key=True)
604 |     actor_id: Mapped[int | None] = mapped_column(
605 |         ForeignKey("users.id", ondelete="SET NULL"),
606 |         nullable=True,
607 |         index=True,
608 |     )
609 |     action: Mapped[str] = mapped_column(String(100), index=True)
610 |     entity_type: Mapped[str] = mapped_column(String(100))
611 |     entity_id: Mapped[str | None] = mapped_column(String(64),
nullable=True)
612 |     occurred_at: Mapped[datetime] = mapped_column(
613 |         DateTime(timezone=True),
614 |         server_default=func.now(),
615 |         index=True,
616 |     )
617 |     changes: Mapped[dict[str, Any] | None] = mapped_column(JSON,
nullable=True)
618 |     ip_address: Mapped[str | None] = mapped_column(String(45),
nullable=True)
619 |
620 |     actor: Mapped[User | None] = relationship()
621 |
622 |
623 | Index(
624 |     "ix_attestation_stages_candidate_status",
625 |     AttestationStage.candidate_id,
626 |     AttestationStage.status,
627 | )
628 | Index(
629 |     "ix_schedule_events_slot_room",
630 |     ScheduleEvent.slot_id,
631 |     ScheduleEvent.room_id,
632 | )
```

## Лістинг А.2.2 - Перелічення ролей, статусів і типів (app/models/enums.py)

```
 1 | from enum import StrEnum
 2 |
 3 |
 4 | class UserRole(StrEnum):
 5 |     ADMIN = "admin"
 6 |     OPERATOR = "operator"
 7 |     CANDIDATE = "candidate"
 8 |     SUPERVISOR = "supervisor"
 9 |     COUNCIL_MEMBER = "council_member"
10 |
11 |
12 | class ImportStatus(StrEnum):
13 |     PENDING = "pending"
14 |     VALIDATED = "validated"
15 |     APPLIED = "applied"
```

```
16 |     FAILED = "failed"
17 |
18 |
19 | class StageStatus(StrEnum):
20 |     PLANNED = "planned"
21 |     IN_PROGRESS = "in_progress"
22 |     COMPLETED = "completed"
23 |     OVERDUE = "overdue"
24 |     NEEDS_REVISION = "needs_revision"
25 |
26 |
27 | class RuleUnit(StrEnum):
28 |     CALENDAR_DAYS = "calendar_days"
29 |     WORKING_DAYS = "working_days"
30 |     WEEKS = "weeks"
31 |     MONTHS = "months"
32 |
33 |
34 | class RuleDirection(StrEnum):
35 |     BEFORE = "before"
36 |     AFTER = "after"
37 |
38 |
39 | class CouncilStatus(StrEnum):
40 |     DRAFT = "draft"
41 |     FORMED = "formed"
42 |     ACTIVE = "active"
43 |     COMPLETED = "completed"
44 |     CANCELLED = "cancelled"
45 |
46 |
47 | class CouncilRole(StrEnum):
48 |     CHAIR = "chair"
49 |     REVIEWER = "reviewer"
50 |     OPPONENT = "opponent"
51 |
52 |
53 | class AvailabilityKind(StrEnum):
54 |     AVAILABLE = "available"
55 |     UNAVAILABLE = "unavailable"
56 |     PREFERRED = "preferred"
57 |
58 |
59 | class ScheduleStatus(StrEnum):
60 |     DRAFT = "draft"
61 |     APPROVED = "approved"
62 |     PUBLISHED = "published"
63 |     CANCELLED = "cancelled"
64 |
65 |
66 | class ScheduleEventStatus(StrEnum):
67 |     PLANNED = "planned"
68 |     CONFIRMED = "confirmed"
69 |     COMPLETED = "completed"
70 |     CANCELLED = "cancelled"
```

## Лістинг А.2.3 - Початкова міграція структури бази даних (alembic/versions/572c3715004e_initial_schema.py)

```
1 | """initial schema
```

```
  2 |
  3 | Revision ID: 572c3715004e
  4 | Revises:
  5 | Create Date: 2026-06-14 04:05:28.305042
  6 | """
  7 | from typing import Sequence, Union
  8 |
  9 | from alembic import op
 10 | import sqlalchemy as sa
 11 |
 12 |
 13 |
 14 | revision: str = '572c3715004e'
 15 | down_revision: Union[str, None] = None
 16 | branch_labels: Union[str, Sequence[str], None] = None
 17 | depends_on: Union[str, Sequence[str], None] = None
 18 |
 19 |
 20 | def upgrade() -> None:
 21 |     # ### commands auto generated by Alembic - please adjust! ###
 22 |     op.create_table('attestation_rules',
 23 |     sa.Column('id', sa.Integer(), nullable=False),
 24 |     sa.Column('code', sa.String(length=64), nullable=False),
 25 |     sa.Column('title', sa.String(length=500), nullable=False),
 26 |     sa.Column('trigger_code', sa.String(length=64), nullable=False),
 27 |     sa.Column('duration_value', sa.Integer(), nullable=False),
 28 |     sa.Column('unit', sa.Enum('calendar_days', 'working_days', 'weeks',
'months', name='rule_unit', native_enum=False, create_constraint=True,
length=24), nullable=False),
 29 |     sa.Column('direction', sa.Enum('before', 'after',
name='rule_direction', native_enum=False, create_constraint=True, length=12),
nullable=False),
 30 |     sa.Column('source_reference', sa.String(length=500), nullable=False),
 31 |     sa.Column('effective_from', sa.Date(), nullable=False),
 32 |     sa.Column('effective_to', sa.Date(), nullable=True),
 33 |     sa.Column('is_active', sa.Boolean(), nullable=False),
 34 |     sa.Column('created_at', sa.DateTime(timezone=True),
server_default=sa.text('(CURRENT_TIMESTAMP)'), nullable=False),
 35 |     sa.Column('updated_at', sa.DateTime(timezone=True),
server_default=sa.text('(CURRENT_TIMESTAMP)'), nullable=False),
 36 |     sa.CheckConstraint('duration_value >= 0',
name=op.f('ck_attestation_rules_non_negative_duration')),
 37 |     sa.CheckConstraint('effective_to IS NULL OR effective_to >=
effective_from', name=op.f('ck_attestation_rules_valid_effective_period')),
 38 |     sa.PrimaryKeyConstraint('id', name=op.f('pk_attestation_rules')),
 39 |     sa.UniqueConstraint('code', name=op.f('uq_attestation_rules_code'))
 40 |     )
 41 |     op.create_table('departments',
 42 |     sa.Column('id', sa.Integer(), nullable=False),
 43 |     sa.Column('name', sa.String(length=500), nullable=False),
 44 |     sa.Column('created_at', sa.DateTime(timezone=True),
server_default=sa.text('(CURRENT_TIMESTAMP)'), nullable=False),
 45 |     sa.Column('updated_at', sa.DateTime(timezone=True),
server_default=sa.text('(CURRENT_TIMESTAMP)'), nullable=False),
 46 |     sa.PrimaryKeyConstraint('id', name=op.f('pk_departments')),
 47 |     sa.UniqueConstraint('name', name=op.f('uq_departments_name'))
 48 |     )
 49 |     op.create_table('educational_programs',
 50 |     sa.Column('id', sa.Integer(), nullable=False),
 51 |     sa.Column('edebo_program_id', sa.String(length=64), nullable=True),
 52 |     sa.Column('name', sa.String(length=500), nullable=False),
 53 |     sa.Column('specialty', sa.String(length=500), nullable=False),
 54 |     sa.Column('degree_level', sa.String(length=255), nullable=True),
```

```
 55 |     sa.Column('created_at', sa.DateTime(timezone=True),
server_default=sa.text('(CURRENT_TIMESTAMP)'), nullable=False),
 56 |     sa.Column('updated_at', sa.DateTime(timezone=True),
server_default=sa.text('(CURRENT_TIMESTAMP)'), nullable=False),
 57 |     sa.PrimaryKeyConstraint('id', name=op.f('pk_educational_programs')),
 58 |     sa.UniqueConstraint('edebo_program_id',
name='uq_educational_programs_edebo_program_id')
 59 |     )
 60 |     op.create_table('rooms',
 61 |     sa.Column('id', sa.Integer(), nullable=False),
 62 |     sa.Column('name', sa.String(length=255), nullable=False),
 63 |     sa.Column('location', sa.String(length=500), nullable=True),
 64 |     sa.Column('online_url', sa.String(length=1000), nullable=True),
 65 |     sa.Column('capacity', sa.Integer(), nullable=False),
 66 |     sa.Column('is_active', sa.Boolean(), nullable=False),
 67 |     sa.Column('created_at', sa.DateTime(timezone=True),
server_default=sa.text('(CURRENT_TIMESTAMP)'), nullable=False),
 68 |     sa.Column('updated_at', sa.DateTime(timezone=True),
server_default=sa.text('(CURRENT_TIMESTAMP)'), nullable=False),
 69 |     sa.CheckConstraint('capacity > 0',
name=op.f('ck_rooms_positive_capacity')),
 70 |     sa.PrimaryKeyConstraint('id', name=op.f('pk_rooms')),
 71 |     sa.UniqueConstraint('name', name=op.f('uq_rooms_name'))
 72 |     )
 73 |     op.create_table('time_slots',
 74 |     sa.Column('id', sa.Integer(), nullable=False),
 75 |     sa.Column('starts_at', sa.DateTime(timezone=True), nullable=False),
 76 |     sa.Column('ends_at', sa.DateTime(timezone=True), nullable=False),
 77 |     sa.Column('is_active', sa.Boolean(), nullable=False),
 78 |     sa.Column('created_at', sa.DateTime(timezone=True),
server_default=sa.text('(CURRENT_TIMESTAMP)'), nullable=False),
 79 |     sa.Column('updated_at', sa.DateTime(timezone=True),
server_default=sa.text('(CURRENT_TIMESTAMP)'), nullable=False),
 80 |     sa.CheckConstraint('ends_at > starts_at',
name=op.f('ck_time_slots_valid_interval')),
 81 |     sa.PrimaryKeyConstraint('id', name=op.f('pk_time_slots')),
 82 |     sa.UniqueConstraint('starts_at', 'ends_at',
name='uq_time_slots_interval')
 83 |     )
 84 |     op.create_index(op.f('ix_time_slots_starts_at'), 'time_slots',
['starts_at'], unique=False)
 85 |     op.create_table('users',
 86 |     sa.Column('id', sa.Integer(), nullable=False),
 87 |     sa.Column('email', sa.String(length=255), nullable=False),
 88 |     sa.Column('full_name', sa.String(length=255), nullable=False),
 89 |     sa.Column('google_subject', sa.String(length=255), nullable=True),
 90 |     sa.Column('is_active', sa.Boolean(), nullable=False),
 91 |     sa.Column('created_at', sa.DateTime(timezone=True),
server_default=sa.text('(CURRENT_TIMESTAMP)'), nullable=False),
 92 |     sa.Column('updated_at', sa.DateTime(timezone=True),
server_default=sa.text('(CURRENT_TIMESTAMP)'), nullable=False),
 93 |     sa.PrimaryKeyConstraint('id', name=op.f('pk_users')),
 94 |     sa.UniqueConstraint('google_subject',
name=op.f('uq_users_google_subject'))
 95 |     )
 96 |     op.create_index(op.f('ix_users_email'), 'users', ['email'],
unique=True)
 97 |     op.create_table('audit_log',
 98 |     sa.Column('id', sa.BigInteger(), nullable=False),
 99 |     sa.Column('actor_id', sa.Integer(), nullable=True),
100 |     sa.Column('action', sa.String(length=100), nullable=False),
101 |     sa.Column('entity_type', sa.String(length=100), nullable=False),
102 |     sa.Column('entity_id', sa.String(length=64), nullable=True),
```

```
103 |     sa.Column('occurred_at', sa.DateTime(timezone=True),
server_default=sa.text('(CURRENT_TIMESTAMP)'), nullable=False),
104 |     sa.Column('changes', sa.JSON(), nullable=True),
105 |     sa.Column('ip_address', sa.String(length=45), nullable=True),
106 |     sa.ForeignKeyConstraint(['actor_id'], ['users.id'],
name=op.f('fk_audit_log_actor_id_users'), ondelete='SET NULL'),
107 |     sa.PrimaryKeyConstraint('id', name=op.f('pk_audit_log'))
108 |     )
109 |     op.create_index(op.f('ix_audit_log_action'), 'audit_log', ['action'],
unique=False)
110 |     op.create_index(op.f('ix_audit_log_actor_id'), 'audit_log',
['actor_id'], unique=False)
111 |     op.create_index(op.f('ix_audit_log_occurred_at'), 'audit_log',
['occurred_at'], unique=False)
112 |     op.create_table('council_members',
113 |     sa.Column('id', sa.Integer(), nullable=False),
114 |     sa.Column('user_id', sa.Integer(), nullable=True),
115 |     sa.Column('full_name', sa.String(length=255), nullable=False),
116 |     sa.Column('email', sa.String(length=255), nullable=True),
117 |     sa.Column('institution', sa.String(length=500), nullable=False),
118 |     sa.Column('academic_degree', sa.String(length=255), nullable=True),
119 |     sa.Column('specialty', sa.String(length=500), nullable=True),
120 |     sa.Column('created_at', sa.DateTime(timezone=True),
server_default=sa.text('(CURRENT_TIMESTAMP)'), nullable=False),
121 |     sa.Column('updated_at', sa.DateTime(timezone=True),
server_default=sa.text('(CURRENT_TIMESTAMP)'), nullable=False),
122 |     sa.ForeignKeyConstraint(['user_id'], ['users.id'],
name=op.f('fk_council_members_user_id_users'), ondelete='SET NULL'),
123 |     sa.PrimaryKeyConstraint('id', name=op.f('pk_council_members')),
124 |     sa.UniqueConstraint('user_id',
name=op.f('uq_council_members_user_id'))
125 |     )
126 |     op.create_index(op.f('ix_council_members_full_name'),
'council_members', ['full_name'], unique=False)
127 |     op.create_table('import_batches',
128 |     sa.Column('id', sa.Integer(), nullable=False),
129 |     sa.Column('file_name', sa.String(length=255), nullable=False),
130 |     sa.Column('file_checksum', sa.String(length=64), nullable=False),
131 |     sa.Column('detected_encoding', sa.String(length=32), nullable=False),
132 |     sa.Column('status', sa.Enum('pending', 'validated', 'applied',
'failed', name='import_status', native_enum=False, create_constraint=True,
length=16), nullable=False),
133 |     sa.Column('total_rows', sa.Integer(), nullable=False),
134 |     sa.Column('inserted_rows', sa.Integer(), nullable=False),
135 |     sa.Column('updated_rows', sa.Integer(), nullable=False),
136 |     sa.Column('skipped_rows', sa.Integer(), nullable=False),
137 |     sa.Column('error_rows', sa.Integer(), nullable=False),
138 |     sa.Column('imported_by_id', sa.Integer(), nullable=False),
139 |     sa.Column('applied_at', sa.DateTime(timezone=True), nullable=True),
140 |     sa.Column('created_at', sa.DateTime(timezone=True),
server_default=sa.text('(CURRENT_TIMESTAMP)'), nullable=False),
141 |     sa.Column('updated_at', sa.DateTime(timezone=True),
server_default=sa.text('(CURRENT_TIMESTAMP)'), nullable=False),
142 |     sa.CheckConstraint('total_rows >= 0 AND inserted_rows >= 0 AND
updated_rows >= 0 AND skipped_rows >= 0 AND error_rows >= 0',
name=op.f('ck_import_batches_non_negative_counters')),
143 |     sa.ForeignKeyConstraint(['imported_by_id'], ['users.id'],
name=op.f('fk_import_batches_imported_by_id_users')),
144 |     sa.PrimaryKeyConstraint('id', name=op.f('pk_import_batches'))
145 |     )
146 |     op.create_index(op.f('ix_import_batches_file_checksum'),
'import_batches', ['file_checksum'], unique=False)
147 |     op.create_table('schedule_versions',
```

```
148 |     sa.Column('id', sa.Integer(), nullable=False),
149 |     sa.Column('version_number', sa.Integer(), nullable=False),
150 |     sa.Column('status', sa.Enum('draft', 'approved', 'published',
'cancelled', name='schedule_status', native_enum=False, create_constraint=True,
length=16), nullable=False),
151 |     sa.Column('period_start', sa.Date(), nullable=False),
152 |     sa.Column('period_end', sa.Date(), nullable=False),
153 |     sa.Column('score', sa.Integer(), nullable=True),
154 |     sa.Column('created_by_id', sa.Integer(), nullable=False),
155 |     sa.Column('published_at', sa.DateTime(timezone=True), nullable=True),
156 |     sa.Column('created_at', sa.DateTime(timezone=True),
server_default=sa.text('(CURRENT_TIMESTAMP)'), nullable=False),
157 |     sa.Column('updated_at', sa.DateTime(timezone=True),
server_default=sa.text('(CURRENT_TIMESTAMP)'), nullable=False),
158 |     sa.CheckConstraint('period_end >= period_start',
name=op.f('ck_schedule_versions_valid_period')),
159 |     sa.CheckConstraint('version_number > 0',
name=op.f('ck_schedule_versions_positive_version_number')),
160 |     sa.ForeignKeyConstraint(['created_by_id'], ['users.id'],
name=op.f('fk_schedule_versions_created_by_id_users'), ondelete='RESTRICT'),
161 |     sa.PrimaryKeyConstraint('id', name=op.f('pk_schedule_versions')),
162 |     sa.UniqueConstraint('version_number',
name='uq_schedule_versions_number')
163 |     )
164 |     op.create_index(op.f('ix_schedule_versions_status'),
'schedule_versions', ['status'], unique=False)
165 |     op.create_table('user_roles',
166 |     sa.Column('user_id', sa.Integer(), nullable=False),
167 |     sa.Column('role', sa.Enum('admin', 'operator', 'candidate',
'supervisor', 'council_member', name='user_role', native_enum=False,
create_constraint=True, length=24), nullable=False),
168 |     sa.ForeignKeyConstraint(['user_id'], ['users.id'],
name=op.f('fk_user_roles_user_id_users'), ondelete='CASCADE'),
169 |     sa.PrimaryKeyConstraint('user_id', 'role', name=op.f('pk_user_roles'))
170 |     )
171 |     op.create_table('candidates',
172 |     sa.Column('id', sa.Integer(), nullable=False),
173 |     sa.Column('edebo_card_id', sa.String(length=64), nullable=False),
174 |     sa.Column('full_name', sa.String(length=255), nullable=False),
175 |     sa.Column('study_status', sa.String(length=255), nullable=False),
176 |     sa.Column('study_start', sa.Date(), nullable=True),
177 |     sa.Column('study_end', sa.Date(), nullable=True),
178 |     sa.Column('study_form', sa.String(length=255), nullable=True),
179 |     sa.Column('course', sa.String(length=32), nullable=True),
180 |     sa.Column('group_name', sa.String(length=255), nullable=True),
181 |     sa.Column('restored_for_defense', sa.Boolean(), nullable=False),
182 |     sa.Column('source_updated_at', sa.DateTime(timezone=True),
nullable=True),
183 |     sa.Column('thesis_title', sa.Text(), nullable=True),
184 |     sa.Column('supervisor_name', sa.String(length=255), nullable=True),
185 |     sa.Column('user_id', sa.Integer(), nullable=True),
186 |     sa.Column('department_id', sa.Integer(), nullable=True),
187 |     sa.Column('educational_program_id', sa.Integer(), nullable=True),
188 |     sa.Column('source_import_id', sa.Integer(), nullable=True),
189 |     sa.Column('created_at', sa.DateTime(timezone=True),
server_default=sa.text('(CURRENT_TIMESTAMP)'), nullable=False),
190 |     sa.Column('updated_at', sa.DateTime(timezone=True),
server_default=sa.text('(CURRENT_TIMESTAMP)'), nullable=False),
191 |     sa.CheckConstraint('study_end IS NULL OR study_start IS NULL OR
study_end >= study_start', name=op.f('ck_candidates_valid_study_period')),
192 |     sa.ForeignKeyConstraint(['department_id'], ['departments.id'],
name=op.f('fk_candidates_department_id_departments'), ondelete='SET NULL'),
```

```
193 |     sa.ForeignKeyConstraint(['educational_program_id'],
['educational_programs.id'],
name=op.f('fk_candidates_educational_program_id_educational_programs'),
ondelete='SET NULL'),
194 |     sa.ForeignKeyConstraint(['source_import_id'], ['import_batches.id'],
name=op.f('fk_candidates_source_import_id_import_batches'), ondelete='SET
NULL'),
195 |     sa.ForeignKeyConstraint(['user_id'], ['users.id'],
name=op.f('fk_candidates_user_id_users'), ondelete='SET NULL'),
196 |     sa.PrimaryKeyConstraint('id', name=op.f('pk_candidates')),
197 |     sa.UniqueConstraint('user_id', name=op.f('uq_candidates_user_id'))
198 |     )
199 |     op.create_index(op.f('ix_candidates_edebo_card_id'), 'candidates',
['edebo_card_id'], unique=True)
200 |     op.create_index(op.f('ix_candidates_full_name'), 'candidates',
['full_name'], unique=False)
201 |     op.create_index(op.f('ix_candidates_study_end'), 'candidates',
['study_end'], unique=False)
202 |     op.create_table('import_errors',
203 |     sa.Column('id', sa.Integer(), nullable=False),
204 |     sa.Column('batch_id', sa.Integer(), nullable=False),
205 |     sa.Column('row_number', sa.Integer(), nullable=False),
206 |     sa.Column('field_name', sa.String(length=255), nullable=True),
207 |     sa.Column('message', sa.Text(), nullable=False),
208 |     sa.Column('raw_value', sa.Text(), nullable=True),
209 |     sa.CheckConstraint('row_number > 0',
name=op.f('ck_import_errors_positive_row_number')),
210 |     sa.ForeignKeyConstraint(['batch_id'], ['import_batches.id'],
name=op.f('fk_import_errors_batch_id_import_batches'), ondelete='CASCADE'),
211 |     sa.PrimaryKeyConstraint('id', name=op.f('pk_import_errors')),
212 |     sa.UniqueConstraint('batch_id', 'row_number', 'field_name',
name='uq_import_errors_row_field')
213 |     )
214 |     op.create_index(op.f('ix_import_errors_batch_id'), 'import_errors',
['batch_id'], unique=False)
215 |     op.create_table('member_availability',
216 |     sa.Column('id', sa.Integer(), nullable=False),
217 |     sa.Column('member_id', sa.Integer(), nullable=False),
218 |     sa.Column('starts_at', sa.DateTime(timezone=True), nullable=False),
219 |     sa.Column('ends_at', sa.DateTime(timezone=True), nullable=False),
220 |     sa.Column('kind', sa.Enum('available', 'unavailable', 'preferred',
name='availability_kind', native_enum=False, create_constraint=True, length=16),
nullable=False),
221 |     sa.Column('created_at', sa.DateTime(timezone=True),
server_default=sa.text('(CURRENT_TIMESTAMP)'), nullable=False),
222 |     sa.Column('updated_at', sa.DateTime(timezone=True),
server_default=sa.text('(CURRENT_TIMESTAMP)'), nullable=False),
223 |     sa.CheckConstraint('ends_at > starts_at',
name=op.f('ck_member_availability_valid_interval')),
224 |     sa.ForeignKeyConstraint(['member_id'], ['council_members.id'],
name=op.f('fk_member_availability_member_id_council_members'),
ondelete='CASCADE'),
225 |     sa.PrimaryKeyConstraint('id', name=op.f('pk_member_availability')),
226 |     sa.UniqueConstraint('member_id', 'starts_at', 'ends_at', 'kind',
name='uq_member_availability_interval')
227 |     )
228 |     op.create_index(op.f('ix_member_availability_member_id'),
'member_availability', ['member_id'], unique=False)
229 |     op.create_table('attestation_stages',
230 |     sa.Column('id', sa.Integer(), nullable=False),
231 |     sa.Column('candidate_id', sa.Integer(), nullable=False),
232 |     sa.Column('rule_id', sa.Integer(), nullable=True),
233 |     sa.Column('code', sa.String(length=64), nullable=False),
```

```
234 |     sa.Column('title', sa.String(length=500), nullable=False),
235 |     sa.Column('planned_date', sa.Date(), nullable=True),
236 |     sa.Column('actual_date', sa.Date(), nullable=True),
237 |     sa.Column('status', sa.Enum('planned', 'in_progress', 'completed',
'overdue', 'needs_revision', name='stage_status', native_enum=False,
create_constraint=True, length=24), nullable=False),
238 |     sa.Column('responsible_user_id', sa.Integer(), nullable=True),
239 |     sa.Column('comment', sa.Text(), nullable=True),
240 |     sa.Column('created_at', sa.DateTime(timezone=True),
server_default=sa.text('(CURRENT_TIMESTAMP)'), nullable=False),
241 |     sa.Column('updated_at', sa.DateTime(timezone=True),
server_default=sa.text('(CURRENT_TIMESTAMP)'), nullable=False),
242 |     sa.CheckConstraint("actual_date IS NULL OR status IN ('completed',
'needs_revision')",
name=op.f('ck_attestation_stages_actual_date_matches_status')),
243 |     sa.ForeignKeyConstraint(['candidate_id'], ['candidates.id'],
name=op.f('fk_attestation_stages_candidate_id_candidates'), ondelete='CASCADE'),
244 |     sa.ForeignKeyConstraint(['responsible_user_id'], ['users.id'],
name=op.f('fk_attestation_stages_responsible_user_id_users'), ondelete='SET
NULL'),
245 |     sa.ForeignKeyConstraint(['rule_id'], ['attestation_rules.id'],
name=op.f('fk_attestation_stages_rule_id_attestation_rules'), ondelete='SET
NULL'),
246 |     sa.PrimaryKeyConstraint('id', name=op.f('pk_attestation_stages')),
247 |     sa.UniqueConstraint('candidate_id', 'code',
name='uq_attestation_stages_candidate_code')
248 |     )
249 |     op.create_index(op.f('ix_attestation_stages_candidate_id'),
'attestation_stages', ['candidate_id'], unique=False)
250 |     op.create_index('ix_attestation_stages_candidate_status',
'attestation_stages', ['candidate_id', 'status'], unique=False)
251 |     op.create_index(op.f('ix_attestation_stages_planned_date'),
'attestation_stages', ['planned_date'], unique=False)
252 |     op.create_index(op.f('ix_attestation_stages_status'),
'attestation_stages', ['status'], unique=False)
253 |     op.create_table('councils',
254 |     sa.Column('id', sa.Integer(), nullable=False),
255 |     sa.Column('candidate_id', sa.Integer(), nullable=False),
256 |     sa.Column('sequence_number', sa.Integer(), nullable=False),
257 |     sa.Column('status', sa.Enum('draft', 'formed', 'active', 'completed',
'cancelled', name='council_status', native_enum=False, create_constraint=True,
length=16), nullable=False),
258 |     sa.Column('order_number', sa.String(length=100), nullable=True),
259 |     sa.Column('order_date', sa.Date(), nullable=True),
260 |     sa.Column('publication_date', sa.Date(), nullable=True),
261 |     sa.Column('created_at', sa.DateTime(timezone=True),
server_default=sa.text('(CURRENT_TIMESTAMP)'), nullable=False),
262 |     sa.Column('updated_at', sa.DateTime(timezone=True),
server_default=sa.text('(CURRENT_TIMESTAMP)'), nullable=False),
263 |     sa.CheckConstraint('sequence_number > 0',
name=op.f('ck_councils_positive_sequence_number')),
264 |     sa.ForeignKeyConstraint(['candidate_id'], ['candidates.id'],
name=op.f('fk_councils_candidate_id_candidates'), ondelete='CASCADE'),
265 |     sa.PrimaryKeyConstraint('id', name=op.f('pk_councils')),
266 |     sa.UniqueConstraint('candidate_id', 'sequence_number',
name='uq_councils_candidate_sequence')
267 |     )
268 |     op.create_index(op.f('ix_councils_candidate_id'), 'councils',
['candidate_id'], unique=False)
269 |     op.create_table('council_memberships',
270 |     sa.Column('id', sa.Integer(), nullable=False),
271 |     sa.Column('council_id', sa.Integer(), nullable=False),
272 |     sa.Column('member_id', sa.Integer(), nullable=False),
```

```
273 |     sa.Column('role', sa.Enum('chair', 'reviewer', 'opponent',
name='council_role', native_enum=False, create_constraint=True, length=16),
nullable=False),
274 |     sa.Column('is_external', sa.Boolean(), nullable=False),
275 |     sa.Column('qualification_confirmed', sa.Boolean(), nullable=False),
276 |     sa.Column('conflict_of_interest_absent', sa.Boolean(),
nullable=False),
277 |     sa.ForeignKeyConstraint(['council_id'], ['councils.id'],
name=op.f('fk_council_memberships_council_id_councils'), ondelete='CASCADE'),
278 |     sa.ForeignKeyConstraint(['member_id'], ['council_members.id'],
name=op.f('fk_council_memberships_member_id_council_members'),
ondelete='RESTRICT'),
279 |     sa.PrimaryKeyConstraint('id', name=op.f('pk_council_memberships')),
280 |     sa.UniqueConstraint('council_id', 'member_id',
name='uq_council_memberships_council_member')
281 |     )
282 |     op.create_index(op.f('ix_council_memberships_council_id'),
'council_memberships', ['council_id'], unique=False)
283 |     op.create_index(op.f('ix_council_memberships_member_id'),
'council_memberships', ['member_id'], unique=False)
284 |     op.create_table('documents',
285 |     sa.Column('id', sa.Integer(), nullable=False),
286 |     sa.Column('stage_id', sa.Integer(), nullable=False),
287 |     sa.Column('uploaded_by_id', sa.Integer(), nullable=False),
288 |     sa.Column('original_name', sa.String(length=255), nullable=False),
289 |     sa.Column('storage_key', sa.String(length=500), nullable=False),
290 |     sa.Column('content_type', sa.String(length=255), nullable=False),
291 |     sa.Column('size_bytes', sa.BigInteger(), nullable=False),
292 |     sa.Column('checksum_sha256', sa.String(length=64), nullable=False),
293 |     sa.Column('created_at', sa.DateTime(timezone=True),
server_default=sa.text('(CURRENT_TIMESTAMP)'), nullable=False),
294 |     sa.Column('updated_at', sa.DateTime(timezone=True),
server_default=sa.text('(CURRENT_TIMESTAMP)'), nullable=False),
295 |     sa.CheckConstraint('size_bytes >= 0',
name=op.f('ck_documents_non_negative_size')),
296 |     sa.ForeignKeyConstraint(['stage_id'], ['attestation_stages.id'],
name=op.f('fk_documents_stage_id_attestation_stages'), ondelete='CASCADE'),
297 |     sa.ForeignKeyConstraint(['uploaded_by_id'], ['users.id'],
name=op.f('fk_documents_uploaded_by_id_users'), ondelete='RESTRICT'),
298 |     sa.PrimaryKeyConstraint('id', name=op.f('pk_documents')),
299 |     sa.UniqueConstraint('storage_key',
name=op.f('uq_documents_storage_key'))
300 |     )
301 |     op.create_index(op.f('ix_documents_checksum_sha256'), 'documents',
['checksum_sha256'], unique=False)
302 |     op.create_index(op.f('ix_documents_stage_id'), 'documents',
['stage_id'], unique=False)
303 |     op.create_table('schedule_events',
304 |     sa.Column('id', sa.Integer(), nullable=False),
305 |     sa.Column('schedule_version_id', sa.Integer(), nullable=False),
306 |     sa.Column('candidate_id', sa.Integer(), nullable=False),
307 |     sa.Column('council_id', sa.Integer(), nullable=False),
308 |     sa.Column('slot_id', sa.Integer(), nullable=False),
309 |     sa.Column('room_id', sa.Integer(), nullable=False),
310 |     sa.Column('status', sa.Enum('planned', 'confirmed', 'completed',
'cancelled', name='schedule_event_status', native_enum=False,
create_constraint=True, length=16), nullable=False),
311 |     sa.Column('is_locked', sa.Boolean(), nullable=False),
312 |     sa.Column('change_penalty', sa.Integer(), nullable=False),
313 |     sa.Column('created_at', sa.DateTime(timezone=True),
server_default=sa.text('(CURRENT_TIMESTAMP)'), nullable=False),
314 |     sa.Column('updated_at', sa.DateTime(timezone=True),
server_default=sa.text('(CURRENT_TIMESTAMP)'), nullable=False),
```

```
315 |     sa.ForeignKeyConstraint(['candidate_id'], ['candidates.id'],
name=op.f('fk_schedule_events_candidate_id_candidates'), ondelete='CASCADE'),
316 |     sa.ForeignKeyConstraint(['council_id'], ['councils.id'],
name=op.f('fk_schedule_events_council_id_councils'), ondelete='RESTRICT'),
317 |     sa.ForeignKeyConstraint(['room_id'], ['rooms.id'],
name=op.f('fk_schedule_events_room_id_rooms'), ondelete='RESTRICT'),
318 |     sa.ForeignKeyConstraint(['schedule_version_id'],
['schedule_versions.id'],
name=op.f('fk_schedule_events_schedule_version_id_schedule_versions'),
ondelete='CASCADE'),
319 |     sa.ForeignKeyConstraint(['slot_id'], ['time_slots.id'],
name=op.f('fk_schedule_events_slot_id_time_slots'), ondelete='RESTRICT'),
320 |     sa.PrimaryKeyConstraint('id', name=op.f('pk_schedule_events')),
321 |     sa.UniqueConstraint('schedule_version_id', 'candidate_id',
name='uq_schedule_events_version_candidate'),
322 |     sa.UniqueConstraint('schedule_version_id', 'slot_id', 'room_id',
name='uq_schedule_events_version_slot_room')
323 |     )
324 |     op.create_index(op.f('ix_schedule_events_candidate_id'),
'schedule_events', ['candidate_id'], unique=False)
325 |     op.create_index(op.f('ix_schedule_events_council_id'),
'schedule_events', ['council_id'], unique=False)
326 |     op.create_index(op.f('ix_schedule_events_room_id'), 'schedule_events',
['room_id'], unique=False)
327 |     op.create_index(op.f('ix_schedule_events_schedule_version_id'),
'schedule_events', ['schedule_version_id'], unique=False)
328 |     op.create_index(op.f('ix_schedule_events_slot_id'), 'schedule_events',
['slot_id'], unique=False)
329 |     op.create_index('ix_schedule_events_slot_room', 'schedule_events',
['slot_id', 'room_id'], unique=False)
330 |     # ### end Alembic commands ###
331 |
332 |
333 | def downgrade() -> None:
334 |     # ### commands auto generated by Alembic - please adjust! ###
335 |     op.drop_index('ix_schedule_events_slot_room',
table_name='schedule_events')
336 |     op.drop_index(op.f('ix_schedule_events_slot_id'),
table_name='schedule_events')
337 |     op.drop_index(op.f('ix_schedule_events_schedule_version_id'),
table_name='schedule_events')
338 |     op.drop_index(op.f('ix_schedule_events_room_id'),
table_name='schedule_events')
339 |     op.drop_index(op.f('ix_schedule_events_council_id'),
table_name='schedule_events')
340 |     op.drop_index(op.f('ix_schedule_events_candidate_id'),
table_name='schedule_events')
341 |     op.drop_table('schedule_events')
342 |     op.drop_index(op.f('ix_documents_stage_id'), table_name='documents')
343 |     op.drop_index(op.f('ix_documents_checksum_sha256'),
table_name='documents')
344 |     op.drop_table('documents')
345 |     op.drop_index(op.f('ix_council_memberships_member_id'),
table_name='council_memberships')
346 |     op.drop_index(op.f('ix_council_memberships_council_id'),
table_name='council_memberships')
347 |     op.drop_table('council_memberships')
348 |     op.drop_index(op.f('ix_councils_candidate_id'), table_name='councils')
349 |     op.drop_table('councils')
350 |     op.drop_index(op.f('ix_attestation_stages_status'),
table_name='attestation_stages')
351 |     op.drop_index(op.f('ix_attestation_stages_planned_date'),
table_name='attestation_stages')
```

```
352 |     op.drop_index('ix_attestation_stages_candidate_status',
table_name='attestation_stages')
353 |     op.drop_index(op.f('ix_attestation_stages_candidate_id'),
table_name='attestation_stages')
354 |     op.drop_table('attestation_stages')
355 |     op.drop_index(op.f('ix_member_availability_member_id'),
table_name='member_availability')
356 |     op.drop_table('member_availability')
357 |     op.drop_index(op.f('ix_import_errors_batch_id'),
table_name='import_errors')
358 |     op.drop_table('import_errors')
359 |     op.drop_index(op.f('ix_candidates_study_end'),
table_name='candidates')
360 |     op.drop_index(op.f('ix_candidates_full_name'),
table_name='candidates')
361 |     op.drop_index(op.f('ix_candidates_edebo_card_id'),
table_name='candidates')
362 |     op.drop_table('candidates')
363 |     op.drop_table('user_roles')
364 |     op.drop_index(op.f('ix_schedule_versions_status'),
table_name='schedule_versions')
365 |     op.drop_table('schedule_versions')
366 |     op.drop_index(op.f('ix_import_batches_file_checksum'),
table_name='import_batches')
367 |     op.drop_table('import_batches')
368 |     op.drop_index(op.f('ix_council_members_full_name'),
table_name='council_members')
369 |     op.drop_table('council_members')
370 |     op.drop_index(op.f('ix_audit_log_occurred_at'),
table_name='audit_log')
371 |     op.drop_index(op.f('ix_audit_log_actor_id'), table_name='audit_log')
372 |     op.drop_index(op.f('ix_audit_log_action'), table_name='audit_log')
373 |     op.drop_table('audit_log')
374 |     op.drop_index(op.f('ix_users_email'), table_name='users')
375 |     op.drop_table('users')
376 |     op.drop_index(op.f('ix_time_slots_starts_at'),
table_name='time_slots')
377 |     op.drop_table('time_slots')
378 |     op.drop_table('rooms')
379 |     op.drop_table('educational_programs')
380 |     op.drop_table('departments')
381 |     op.drop_table('attestation_rules')
382 |     # ### end Alembic commands ###
```

## Лістинг А.2.4 - Міграція операційних налаштувань (alembic/versions/9c1d2e3f4a5b_app_settings.py)

```
 1 | """app settings
 2 |
 3 | Revision ID: 9c1d2e3f4a5b
 4 | Revises: 572c3715004e
 5 | Create Date: 2026-06-14 22:25:00.000000
 6 | """
 7 |
 8 | from typing import Sequence, Union
 9 |
10 | import sqlalchemy as sa
11 | from alembic import op
12 |
13 |
```

```
revision: str = "9c1d2e3f4a5b"
down_revision: Union[str, None] = "572c3715004e"
branch_labels: Union[str, Sequence[str], None] = None
depends_on: Union[str, Sequence[str], None] = None


def upgrade() -> None:
    op.create_table(
        "app_settings",
        sa.Column("key", sa.String(length=100), nullable=False),
        sa.Column("value", sa.JSON(), nullable=False),
        sa.Column("description", sa.String(length=500), nullable=True),
        sa.Column(
            "created_at",
            sa.DateTime(timezone=True),
            server_default=sa.text("(CURRENT_TIMESTAMP)"),
            nullable=False,
        ),
        sa.Column(
            "updated_at",
            sa.DateTime(timezone=True),
            server_default=sa.text("(CURRENT_TIMESTAMP)"),
            nullable=False,
        ),
        sa.PrimaryKeyConstraint("key", name=op.f("pk_app_settings")),
    )


def downgrade() -> None:
    op.drop_table("app_settings")
```

## Лістинг А.2.5 - Міграція публічних реквізитів РСВР (alembic/versions/b7d8e9f0a1b2_council_public_details.py)

```
"""council public details

Revision ID: b7d8e9f0a1b2
Revises: 9c1d2e3f4a5b
Create Date: 2026-06-14 23:10:00.000000
"""

from typing import Sequence, Union

import sqlalchemy as sa
from alembic import op


revision: str = "b7d8e9f0a1b2"
down_revision: Union[str, None] = "9c1d2e3f4a5b"
branch_labels: Union[str, Sequence[str], None] = None
depends_on: Union[str, Sequence[str], None] = None


def upgrade() -> None:
    op.add_column(
        "councils",
        sa.Column(
            "public_details",
            sa.JSON(),
            server_default=sa.text("'{}'"),
```

```
27 |             nullable=False,
28 |         ),
29 |     )
30 |     op.alter_column("councils", "public_details", server_default=None)
31 |
32 |
33 | def downgrade() -> None:
34 |     op.drop_column("councils", "public_details")
```

### Лістинг А.2.6 - Міграція публічних полів членів ради (alembic/versions/c8d9e0f1a2b3_council_member_public_fields.py)

```
 1 | """council member public fields
 2 |
 3 | Revision ID: c8d9e0f1a2b3
 4 | Revises: b7d8e9f0a1b2
 5 | Create Date: 2026-06-14 23:30:00.000000
 6 | """
 7 |
 8 | from collections.abc import Sequence
 9 |
10 | import sqlalchemy as sa
11 | from alembic import op
12 |
13 |
14 | revision: str = "c8d9e0f1a2b3"
15 | down_revision: str | None = "b7d8e9f0a1b2"
16 | branch_labels: str | Sequence[str] | None = None
17 | depends_on: str | Sequence[str] | None = None
18 |
19 |
20 | def upgrade() -> None:
21 |     op.add_column(
22 |         "council_members",
23 |         sa.Column("position", sa.String(length=1000), nullable=True),
24 |     )
25 |     op.add_column(
26 |         "council_members",
27 |         sa.Column("profile_url", sa.String(length=1000), nullable=True),
28 |     )
29 |
30 |
31 | def downgrade() -> None:
32 |     op.drop_column("council_members", "profile_url")
33 |     op.drop_column("council_members", "position")
```

## ДОДАТОК А.3

### Модуль перевірки та застосування CSV-імпорту ЄДЕБО

У додатку наведено код розбору CSV-файлів ЄДЕБО, перевірки структури, кодування, дублікатів і застосування імпорту до бази даних.

## Лістинг А.3.1 - Перевірка та нормалізація CSV-файлу ЄДЕБО (app/services/edebo_import.py)

```
  1 | from __future__ import annotations
  2 |
  3 | import csv
  4 | import hashlib
  5 | import io
  6 | from dataclasses import dataclass
  7 | from datetime import date, datetime
  8 |
  9 | from app.schemas import (
 10 |     CandidatePreview,
 11 |     ImportIssue,
 12 |     ImportValidationResponse,
 13 | )
 14 |
 15 |
 16 | FIELD_MAP = {
 17 |     "edebo_card_id": "ID картки",
 18 |     "study_status": "Статус навчання",
 19 |     "full_name": "Здобувач",
 20 |     "study_start": "Початок навчання",
 21 |     "study_end": "Завершення навчання",
 22 |     "department": "Структурний підрозділ",
 23 |     "degree_level": "Освітній ступінь (рівень)",
 24 |     "study_form": "Форма навчання",
 25 |     "specialty": "Спеціальність",
 26 |     "edebo_program_id": "ID ОП",
 27 |     "educational_program": "Освітня програма",
 28 |     "course": "Курс",
 29 |     "group_name": "Група",
 30 |     "source_updated_at": "Час останньої зміни",
 31 |     "restored_for_defense": "Поновлення для захисту дисертації",
 32 | }
 33 |
 34 | REQUIRED_FIELDS = {
 35 |     "edebo_card_id",
 36 |     "study_status",
 37 |     "full_name",
 38 |     "study_start",
 39 |     "study_end",
 40 |     "degree_level",
 41 |     "specialty",
 42 |     "educational_program",
 43 | }
 44 |
 45 | SENSITIVE_COLUMNS = {
 46 |     "Дата народження",
 47 |     "Серія документа",
 48 |     "Номер документа",
 49 |     "Дата видачі",
 50 |     "Дійсний до",
 51 |     "Стать",
 52 |     "Громадянство",
 53 |     "РНОКПП",
 54 |     "Валідний РНОКПП",
 55 | }
 56 |
 57 |
 58 | @dataclass(frozen=True)
```

```
 59 | class NormalizedCandidate:
 60 |     edebo_card_id: str
 61 |     full_name: str
 62 |     study_status: str
 63 |     study_start: date
 64 |     study_end: date
 65 |     department: str | None
 66 |     degree_level: str
 67 |     study_form: str | None
 68 |     specialty: str
 69 |     edebo_program_id: str | None
 70 |     educational_program: str
 71 |     course: str | None
 72 |     group_name: str | None
 73 |     source_updated_at: datetime | None
 74 |     restored_for_defense: bool
 75 |
 76 |
 77 | @dataclass(frozen=True)
 78 | class ParsedImport:
 79 |     records: list[NormalizedCandidate]
 80 |     report: ImportValidationResponse
 81 |
 82 |
 83 | def _decode(data: bytes) -> tuple[str, str]:
 84 |     encodings = ["utf-8-sig"]
 85 |     if data.startswith((b"\xff\xfe", b"\xfe\xff")):
 86 |         encodings.append("utf-16")
 87 |     encodings.append("cp1251")
 88 |     for encoding in encodings:
 89 |         try:
 90 |             text = data.decode(encoding)
 91 |         except UnicodeDecodeError:
 92 |             continue
 93 |         if "\x00" not in text[:200]:
 94 |             return text, encoding
 95 |     raise ValueError("Не вдалося визначити кодування CSV-файлу.")
 96 |
 97 |
 98 | def _parse_date(value: str, field_name: str) -> date:
 99 |     try:
100 |         return date.fromisoformat(value.strip())
101 |     except ValueError as exc:
102 |         raise ValueError(
103 |             f"Поле «{field_name}» повинно мати формат YYYY-MM-DD."
104 |         ) from exc
105 |
106 |
107 | def _parse_datetime(value: str) -> datetime | None:
108 |     value = value.strip()
109 |     if not value:
110 |         return None
111 |     try:
112 |         return datetime.fromisoformat(value)
113 |     except ValueError as exc:
114 |         raise ValueError(
115 |             "Поле «Час останньої зміни» повинно мати формат "
116 |             "YYYY-MM-DD HH:MM:SS."
117 |         ) from exc
118 |
119 |
120 | def _parse_bool(value: str) -> bool:
121 |     normalized = value.strip().casefold()
```

```
122 |     if normalized in {"так", "yes", "true", "1"}:
123 |         return True
124 |     if normalized in {"ні", "no", "false", "0", ""}:
125 |         return False
126 |     raise ValueError("Логічне поле повинно містити «Так» або «Ні».")
127 |
128 |
129 | def _mask_name(full_name: str) -> str:
130 |     parts = [part for part in full_name.split() if part]
131 |     if not parts:
132 |         return ""
133 |     return " ".join(
134 |         part if index == 0 else f"{part[0]}."
135 |         for index, part in enumerate(parts)
136 |     )
137 |
138 |
139 | def parse_edebo_csv(file_name: str, data: bytes) -> ParsedImport:
140 |     if not data:
141 |         raise ValueError("CSV-файл порожній.")
142 |
143 |     text, encoding = _decode(data)
144 |     try:
145 |         dialect = csv.Sniffer().sniff(text[:4096], delimiters=";,\t|")
146 |         delimiter = dialect.delimiter
147 |     except csv.Error:
148 |         first_line = text.splitlines()[0] if text.splitlines() else ""
149 |         delimiter = max(";,\t|", key=first_line.count)
150 |         if first_line.count(delimiter) == 0:
151 |             delimiter = ";"
152 |
153 |     reader = csv.DictReader(io.StringIO(text), delimiter=delimiter)
154 |     headers = [header.strip() for header in (reader.fieldnames or [])]
155 |     missing_columns = [
156 |         FIELD_MAP[field]
157 |         for field in REQUIRED_FIELDS
158 |         if FIELD_MAP[field] not in headers
159 |     ]
160 |     if missing_columns:
161 |         raise ValueError(
162 |             "Відсутні обов'язкові колонки: " + ",
".join(sorted(missing_columns))
163 |         )
164 |
165 |     records: list[NormalizedCandidate] = []
166 |     issues: list[ImportIssue] = []
167 |     seen_ids: set[str] = set()
168 |     duplicate_ids = 0
169 |     total_rows = 0
170 |
171 |     for row_number, row in enumerate(reader, start=2):
172 |         total_rows += 1
173 |         try:
174 |             values = {
175 |                 field: (row.get(source_header) or "").strip()
176 |                 for field, source_header in FIELD_MAP.items()
177 |             }
178 |             empty_required = [
179 |                 FIELD_MAP[field]
180 |                 for field in REQUIRED_FIELDS
181 |                 if not values[field]
182 |             ]
183 |             if empty_required:
```

```
184 |                 raise ValueError(
185 |                     "Не заповнено обов'язкові поля: "
186 |                     + ", ".join(sorted(empty_required))
187 |                 )
188 |             card_id = values["edebo_card_id"]
189 |             if card_id in seen_ids:
190 |                 duplicate_ids += 1
191 |                 raise ValueError("Дубль значення «ID картки» у файлі.")
192 |             seen_ids.add(card_id)
193 |
194 |             record = NormalizedCandidate(
195 |                 edebo_card_id=card_id,
196 |                 full_name=values["full_name"],
197 |                 study_status=values["study_status"],
198 |                 study_start=_parse_date(
199 |                     values["study_start"],
200 |                     FIELD_MAP["study_start"],
201 |                 ),
202 |                 study_end=_parse_date(
203 |                     values["study_end"],
204 |                     FIELD_MAP["study_end"],
205 |                 ),
206 |                 department=values["department"] or None,
207 |                 degree_level=values["degree_level"],
208 |                 study_form=values["study_form"] or None,
209 |                 specialty=values["specialty"],
210 |                 edebo_program_id=values["edebo_program_id"] or None,
211 |                 educational_program=values["educational_program"],
212 |                 course=values["course"] or None,
213 |                 group_name=values["group_name"] or None,
214 |
source_updated_at=_parse_datetime(values["source_updated_at"]),
215 |                 restored_for_defense=_parse_bool(
216 |                     values["restored_for_defense"]
217 |                 ),
218 |             )
219 |             if record.study_end < record.study_start:
220 |                 raise ValueError(
221 |                     "Дата завершення навчання передує даті початку
навчання."
222 |                 )
223 |             records.append(record)
224 |         except ValueError as exc:
225 |             issues.append(
226 |                 ImportIssue(
227 |                     row_number=row_number,
228 |                     message=str(exc),
229 |                 )
230 |             )
231 |
232 |     report = ImportValidationResponse(
233 |         file_name=file_name,
234 |         checksum_sha256=hashlib.sha256(data).hexdigest(),
235 |         detected_encoding=encoding,
236 |         delimiter=delimiter,
237 |         column_count=len(headers),
238 |         total_rows=total_rows,
239 |         valid_rows=len(records),
240 |         error_rows=len(issues),
241 |         duplicate_ids=duplicate_ids,
242 |         mapped_columns={
243 |             field: header
244 |             for field, header in FIELD_MAP.items()
```

```
245 |             if header in headers
246 |         },
247 |         ignored_sensitive_columns=sorted(
248 |             column for column in SENSITIVE_COLUMNS if column in headers
249 |         ),
250 |         issues=issues[:50],
251 |         preview=[
252 |             CandidatePreview(
253 |                 edebo_card_id=record.edebo_card_id,
254 |                 full_name=_mask_name(record.full_name),
255 |                 study_status=record.study_status,
256 |                 study_start=record.study_start.isoformat(),
257 |                 study_end=record.study_end.isoformat(),
258 |                 educational_program=record.educational_program,
259 |                 restored_for_defense=record.restored_for_defense,
260 |             )
261 |             for record in records[:5]
262 |         ],
263 |     )
264 |     return ParsedImport(records=records, report=report)
```

## Лістинг А.3.2 - Збереження результатів імпорту до PostgreSQL (app/repositories/imports.py)

```
  1 | from __future__ import annotations
  2 |
  3 | from datetime import UTC, datetime
  4 |
  5 | from sqlalchemy import select
  6 | from sqlalchemy.ext.asyncio import AsyncSession
  7 | from sqlalchemy.orm import selectinload
  8 |
  9 | from app.models import (
 10 |     Candidate,
 11 |     Department,
 12 |     EducationalProgram,
 13 |     ImportBatch,
 14 |     User,
 15 |     UserRoleAssignment,
 16 | )
 17 | from app.models.enums import ImportStatus
 18 | from app.schemas import ImportApplyResponse
 19 | from app.security import Principal
 20 | from app.services.edebo_import import ParsedImport
 21 |
 22 |
 23 | async def apply_candidate_import(
 24 |     session: AsyncSession,
 25 |     parsed: ParsedImport,
 26 |     principal: Principal,
 27 | ) -> ImportApplyResponse:
 28 |     if parsed.report.error_rows:
 29 |         raise ValueError(
 30 |             "Пакет із помилками не можна застосувати до бази даних."
 31 |         )
 32 |
 33 |     async with session.begin():
 34 |         user = await _get_or_create_user(session, principal)
 35 |         batch = ImportBatch(
 36 |             file_name=parsed.report.file_name,
```

```
 37 |             file_checksum=parsed.report.checksum_sha256,
 38 |             detected_encoding=parsed.report.detected_encoding,
 39 |             status=ImportStatus.VALIDATED,
 40 |             total_rows=parsed.report.total_rows,
 41 |             inserted_rows=0,
 42 |             updated_rows=0,
 43 |             skipped_rows=0,
 44 |             error_rows=0,
 45 |             imported_by=user,
 46 |         )
 47 |         session.add(batch)
 48 |         await session.flush()
 49 |
 50 |         departments: dict[str, Department] = {}
 51 |         programs: dict[tuple[str | None, str, str], EducationalProgram] =
{}
 52 |
 53 |         for record in parsed.records:
 54 |             department = None
 55 |             if record.department:
 56 |                 department = departments.get(record.department)
 57 |                 if department is None:
 58 |                     department = await _get_or_create_department(
 59 |                         session,
 60 |                         record.department,
 61 |                     )
 62 |                     departments[record.department] = department
 63 |
 64 |             program_key = (
 65 |                 record.edebo_program_id,
 66 |                 record.educational_program,
 67 |                 record.specialty,
 68 |             )
 69 |             program = programs.get(program_key)
 70 |             if program is None:
 71 |                 program = await _get_or_create_program(
 72 |                     session,
 73 |                     edebo_program_id=record.edebo_program_id,
 74 |                     name=record.educational_program,
 75 |                     specialty=record.specialty,
 76 |                     degree_level=record.degree_level,
 77 |                 )
 78 |                 programs[program_key] = program
 79 |
 80 |             candidate = await session.scalar(
 81 |                 select(Candidate).where(
 82 |                     Candidate.edebo_card_id == record.edebo_card_id
 83 |                 )
 84 |             )
 85 |             if candidate is None:
 86 |                 candidate = Candidate(
 87 |                     edebo_card_id=record.edebo_card_id,
 88 |                     full_name=record.full_name,
 89 |                     study_status=record.study_status,
 90 |                     study_start=record.study_start,
 91 |                     study_end=record.study_end,
 92 |                 )
 93 |                 session.add(candidate)
 94 |                 batch.inserted_rows += 1
 95 |             else:
 96 |                 batch.updated_rows += 1
 97 |
 98 |             candidate.full_name = record.full_name
```

```
            candidate.study_status = record.study_status
            candidate.study_start = record.study_start
            if candidate.study_end is None:
                candidate.study_end = record.study_end
            candidate.study_form = record.study_form
            candidate.course = record.course
            candidate.group_name = record.group_name
            candidate.restored_for_defense = record.restored_for_defense
            candidate.source_updated_at = record.source_updated_at
            candidate.department = department
            candidate.educational_program = program
            candidate.source_import = batch

        batch.status = ImportStatus.APPLIED
        batch.applied_at = datetime.now(UTC)
        await session.flush()

        result = ImportApplyResponse(
            batch_id=batch.id,
            total_rows=batch.total_rows,
            inserted_rows=batch.inserted_rows,
            updated_rows=batch.updated_rows,
            error_rows=batch.error_rows,
        )
    return result


async def _get_or_create_user(
    session: AsyncSession,
    principal: Principal,
) -> User:
    user = await session.scalar(
        select(User)
        .options(selectinload(User.roles))
        .where(User.google_subject == principal.subject)
    )
    if user is None and principal.email:
        user = await session.scalar(
            select(User)
            .options(selectinload(User.roles))
            .where(User.email == principal.email)
        )
    if user is None:
        email = principal.email or f"{principal.subject}@example.invalid"
        user = User(
            email=email,
            full_name=principal.email or principal.subject,
            google_subject=principal.subject,
            roles=[
                UserRoleAssignment(role=role)
                for role in sorted(principal.roles, key=lambda item:
item.value)
            ],
        )
        session.add(user)
        await session.flush()
        return user
    elif user.google_subject is None:
        user.google_subject = principal.subject

    existing_roles = {assignment.role for assignment in user.roles}
    for role in principal.roles - existing_roles:
        user.roles.append(UserRoleAssignment(role=role))
```

```
161 |     return user
162 |
163 |
164 | async def _get_or_create_department(
165 |     session: AsyncSession,
166 |     name: str,
167 | ) -> Department:
168 |     department = await session.scalar(
169 |         select(Department).where(Department.name == name)
170 |     )
171 |     if department is None:
172 |         department = Department(name=name)
173 |         session.add(department)
174 |         await session.flush()
175 |     return department
176 |
177 |
178 | async def _get_or_create_program(
179 |     session: AsyncSession,
180 |     *,
181 |     edebo_program_id: str | None,
182 |     name: str,
183 |     specialty: str,
184 |     degree_level: str,
185 | ) -> EducationalProgram:
186 |     if edebo_program_id:
187 |         program = await session.scalar(
188 |             select(EducationalProgram).where(
189 |                 EducationalProgram.edebo_program_id == edebo_program_id
190 |             )
191 |         )
192 |     else:
193 |         program = await session.scalar(
194 |             select(EducationalProgram).where(
195 |                 EducationalProgram.name == name,
196 |                 EducationalProgram.specialty == specialty,
197 |             )
198 |         )
199 |     if program is None:
200 |         program = EducationalProgram(
201 |             edebo_program_id=edebo_program_id,
202 |             name=name,
203 |             specialty=specialty,
204 |             degree_level=degree_level,
205 |         )
206 |         session.add(program)
207 |         await session.flush()
208 |     else:
209 |         program.name = name
210 |         program.specialty = specialty
211 |         program.degree_level = degree_level
212 |     return program
```

## ДОДАТОК А.4

### Формування індивідуального плану, нормативний контроль і РСВР

У додатку подано основний сервіс предметної логіки, який формує етапи атестації, контролює нормативні строки, працює зі складом разової ради, документами, графіками та аудитом.

**Лістинг А.4.1 - Сервіс атестації, нормативного контролю та РСВР (app/services/attestation.py)**

```
 1 | from __future__ import annotations
 2 |
 3 | import hashlib
 4 | import re
 5 | import time as system_time
 6 | from dataclasses import dataclass
 7 | from datetime import UTC, date, datetime, time, timedelta
 8 | from itertools import cycle
 9 | from threading import Lock
10 |
11 | from sqlalchemy import delete as sql_delete
12 | from sqlalchemy import func, select
13 | from sqlalchemy.ext.asyncio import AsyncSession
14 | from sqlalchemy.orm import selectinload
15 |
16 | from app.config import settings
17 | from app.models import (
18 |     AppSetting,
19 |     AttestationRule,
20 |     AttestationStage,
21 |     AuditLog,
22 |     Candidate,
23 |     Council,
24 |     CouncilMember,
25 |     CouncilMembership,
26 |     Document,
27 |     MemberAvailability,
28 |     Room,
29 |     ScheduleEvent,
30 |     ScheduleVersion,
31 |     TimeSlot,
32 |     User,
33 |     UserRoleAssignment,
34 | )
35 | from app.models.enums import (
36 |     AvailabilityKind,
37 |     CouncilRole,
38 |     CouncilStatus,
39 |     RuleDirection,
40 |     RuleUnit,
41 |     ScheduleEventStatus,
42 |     ScheduleStatus,
43 |     StageStatus,
44 | )
45 | from app.schemas import (
46 |     AssignmentPayload,
```

```
  47 |     AuditLogEntry,
  48 |     AuditLogResponse,
  49 |     CandidateDetailResponse,
  50 |     CandidateListResponse,
  51 |     CandidatePayload,
  52 |     CandidateScheduleSummary,
  53 |     CandidateSummary,
  54 |     CandidateUpdatePayload,
  55 |     ConflictListResponse,
  56 |     ConflictSummary,
  57 |     CouncilCompositionPayload,
  58 |     CouncilMetadataPayload,
  59 |     CouncilListResponse,
  60 |     CouncilMemberInput,
  61 |     CouncilMemberSummary,
  62 |     CouncilMaterialResponse,
  63 |     CouncilSummary,
  64 |     DeleteResponse,
  65 |     DocumentSummary,
  66 |     GeneratePlansResponse,
  67 |     NormativeControlResponse,
  68 |     NormativeIssue,
  69 |     OperationalSettingsResponse,
  70 |     OperationalSettingsUpdateRequest,
  71 |     ReplanScheduleRequest,
  72 |     RoomAdminSummary,
  73 |     RoomCreateRequest,
  74 |     RoomPayload,
  75 |     RoomUpdateRequest,
  76 |     ScheduleGenerateRequest,
  77 |     ScheduleListResponse,
  78 |     ScheduleSolveRequest,
  79 |     ScheduleSolveResponse,
  80 |     ScheduleState,
  81 |     ScheduleSummary,
  82 |     ScheduleVersionResponse,
  83 |     SlotPayload,
  84 |     StageEventPayload,
  85 |     StageEventResponse,
  86 |     StageDocumentPayload,
  87 |     StageDocumentResponse,
  88 |     StageListResponse,
  89 |     StageSummary,
  90 |     SystemSettingsResponse,
  91 |     WorkspaceSummaryResponse,
  92 | )
  93 | from app.security import Principal
  94 | from app.services.scheduling import solve_schedule
  95 |
  96 |
  97 | ROOM_NAMES = ("0314", "0509", "1214")
  98 | HOME_INSTITUTION = "ІФНТУНГ"
  99 | HOME_INSTITUTION_ALIASES = {
 100 |     "ifntung",
 101 |     "івано-франківський національний технічний університет нафти і газу",
 102 | }
 103 | DEFAULT_EXTERNAL_INSTITUTIONS = (
 104 |     "Національний університет «Львівська політехніка»",
 105 |     "КПІ ім. Ігоря Сікорського",
 106 |     "Харківський національний університет радіоелектроніки",
 107 |     "Тернопільський національний технічний університет імені Івана
Пулюя",
 108 | )
```

```
 109 | DEFENSE_STARTS = (
 110 |     time(9, 0),
 111 |     time(12, 30),
 112 |     time(16, 0),
 113 | )
 114 | DEFENSE_DURATION = timedelta(hours=3)
 115 | DEFENSE_DEADLINE_BEFORE_STUDY_END_DAYS = 15
 116 | OPERATIONAL_SETTINGS_KEY = "operational"
 117 | DEFAULT_OPERATIONAL_SETTINGS = {
 118 |     "defense_duration_minutes": 180,
 119 |     "default_planning_days": 10,
 120 |     "default_max_member_defenses_per_day": 4,
 121 |     "defense_start_times": ["09:00", "12:30", "16:00"],
 122 | }
 123 | _AUDIT_ID_LOCK = Lock()
 124 | _LAST_AUDIT_ID = 0
 125 |
 126 |
 127 | @dataclass(frozen=True)
 128 | class NormativeRuleSeed:
 129 |     code: str
 130 |     title: str
 131 |     trigger_code: str
 132 |     duration_value: int
 133 |     unit: RuleUnit
 134 |     direction: RuleDirection
 135 |     responsible: str
 136 |     source_reference: str
 137 |
 138 |
 139 | @dataclass(frozen=True)
 140 | class OperationalSettings:
 141 |     defense_duration_minutes: int
 142 |     default_planning_days: int
 143 |     default_max_member_defenses_per_day: int
 144 |     defense_start_times: tuple[time, ...]
 145 |
 146 |     @property
 147 |     def defense_duration(self) -> timedelta:
 148 |         return timedelta(minutes=self.defense_duration_minutes)
 149 |
 150 |     def to_response(self) -> OperationalSettingsResponse:
 151 |         return OperationalSettingsResponse(
 152 |             defense_duration_minutes=self.defense_duration_minutes,
 153 |             default_planning_days=self.default_planning_days,
 154 |             default_max_member_defenses_per_day=(
 155 |                 self.default_max_member_defenses_per_day
 156 |             ),
 157 |             defense_start_times=[
 158 |                 item.strftime("%H:%M") for item in
self.defense_start_times
 159 |             ],
 160 |         )
 161 |
 162 |
 163 | RULE_SEEDS = (
 164 |     NormativeRuleSeed(
 165 |         code="dissertation_format",
 166 |         title="Оформлення дисертації відповідно до вимог МОН та ОНП;
перевірка публікацій",
 167 |         trigger_code="study_end",
 168 |         duration_value=9,
 169 |         unit=RuleUnit.MONTHS,
```

```
 170 |         direction=RuleDirection.BEFORE,
 171 |         responsible="Здобувач, науковий керівник",
 172 |         source_reference="Постанова КМУ № 261; Постанова КМУ № 44;
правила ІФНТУНГ 2026",
 173 |     ),
 174 |     NormativeRuleSeed(
 175 |         code="readiness_review",
 176 |         title="Визначення стану готовності дисертації до захисту",
 177 |         trigger_code="dissertation_format",
 178 |         duration_value=0,
 179 |         unit=RuleUnit.CALENDAR_DAYS,
 180 |         direction=RuleDirection.AFTER,
 181 |         responsible="Науковий керівник",
 182 |         source_reference="Правила ІФНТУНГ 2026, розд. I, п. 2",
 183 |     ),
 184 |     NormativeRuleSeed(
 185 |         code="onp_certificate",
 186 |         title="Отримання довідки про виконання ОНП і висновку наукового
керівника",
 187 |         trigger_code="study_end",
 188 |         duration_value=9,
 189 |         unit=RuleUnit.MONTHS,
 190 |         direction=RuleDirection.BEFORE,
 191 |         responsible="ВАіД / відповідальний підрозділ, науковий керівник",
 192 |         source_reference="Правила ІФНТУНГ 2026, розд. I, п. 3",
 193 |     ),
 194 |     NormativeRuleSeed(
 195 |         code="novelty_statement_request",
 196 |         title="Заява про отримання висновку щодо наукової новизни та
значення результатів",
 197 |         trigger_code="onp_certificate",
 198 |         duration_value=1,
 199 |         unit=RuleUnit.MONTHS,
 200 |         direction=RuleDirection.AFTER,
 201 |         responsible="Здобувач",
 202 |         source_reference="Правила ІФНТУНГ 2026, розд. I, п. 4",
 203 |     ),
 204 |     NormativeRuleSeed(
 205 |         code="public_presentation",
 206 |         title="Публічна презентація результатів дисертації та
обговорення",
 207 |         trigger_code="novelty_statement_request",
 208 |         duration_value=1,
 209 |         unit=RuleUnit.MONTHS,
 210 |         direction=RuleDirection.AFTER,
 211 |         responsible="Кафедра / структурний підрозділ, здобувач",
 212 |         source_reference="Правила ІФНТУНГ 2026, розд. I, п. 8",
 213 |     ),
 214 |     NormativeRuleSeed(
 215 |         code="novelty_statement",
 216 |         title="Надання висновку про наукову новизну, теоретичне та
практичне значення",
 217 |         trigger_code="public_presentation",
 218 |         duration_value=2,
 219 |         unit=RuleUnit.WEEKS,
 220 |         direction=RuleDirection.AFTER,
 221 |         responsible="Кафедра / структурний підрозділ",
 222 |         source_reference="Правила ІФНТУНГ 2026, розд. I, п. 8",
 223 |     ),
 224 |     NormativeRuleSeed(
 225 |         code="council_proposals",
 226 |         title="Формування пропозицій щодо складу разової ради та
перевірка кандидатів",
```

```
 227 |         trigger_code="novelty_statement",
 228 |         duration_value=0,
 229 |         unit=RuleUnit.CALENDAR_DAYS,
 230 |         direction=RuleDirection.AFTER,
 231 |         responsible="Структурний підрозділ, науковий керівник, ВАіД",
 232 |         source_reference="Правила ІФНТУНГ 2026, розд. I, п. 7",
 233 |     ),
 234 |     NormativeRuleSeed(
 235 |         code="dissertation_qes",
 236 |         title="Накладення КЕП з кваліфікованою електронною позначкою часу
на PDF/A-файл",
 237 |         trigger_code="novelty_statement",
 238 |         duration_value=2,
 239 |         unit=RuleUnit.WEEKS,
 240 |         direction=RuleDirection.AFTER,
 241 |         responsible="Здобувач",
 242 |         source_reference="Правила ІФНТУНГ 2026, розд. II, п. 1-2",
 243 |     ),
 244 |     NormativeRuleSeed(
 245 |         code="council_creation_application",
 246 |         title="Заява до вченої ради про утворення разової ради з доданням
документів",
 247 |         trigger_code="novelty_statement",
 248 |         duration_value=2,
 249 |         unit=RuleUnit.WEEKS,
 250 |         direction=RuleDirection.AFTER,
 251 |         responsible="Здобувач через ВАіД",
 252 |         source_reference="Правила ІФНТУНГ 2026, розд. II, п. 1",
 253 |     ),
 254 |     NormativeRuleSeed(
 255 |         code="academic_council_decision",
 256 |         title="Утворення разової ради рішенням вченої ради закладу",
 257 |         trigger_code="council_creation_application",
 258 |         duration_value=2,
 259 |         unit=RuleUnit.MONTHS,
 260 |         direction=RuleDirection.AFTER,
 261 |         responsible="Вчена рада закладу",
 262 |         source_reference="Правила ІФНТУНГ 2026, розд. II, п. 3",
 263 |     ),
 264 |     NormativeRuleSeed(
 265 |         code="council_order",
 266 |         title="Наказ керівника закладу про утворення разової ради",
 267 |         trigger_code="academic_council_decision",
 268 |         duration_value=0,
 269 |         unit=RuleUnit.CALENDAR_DAYS,
 270 |         direction=RuleDirection.AFTER,
 271 |         responsible="Ректор / керівник закладу",
 272 |         source_reference="Правила ІФНТУНГ 2026, розд. II, п. 3",
 273 |     ),
 274 |     NormativeRuleSeed(
 275 |         code="materials_publication",
 276 |         title="Оприлюднення дисертації, висновку, складу ради; внесення
інформації до NAQA.Svr",
 277 |         trigger_code="council_order",
 278 |         duration_value=5,
 279 |         unit=RuleUnit.WORKING_DAYS,
 280 |         direction=RuleDirection.AFTER,
 281 |         responsible="Заклад, ВАіД",
 282 |         source_reference="Правила ІФНТУНГ 2026, розд. II, п. 5-8",
 283 |     ),
 284 |     NormativeRuleSeed(
 285 |         code="mon_check",
```

```
 286 |         title="Перевірка МОН відповідності складу разової ради вимогам
законодавства",
 287 |         trigger_code="materials_publication",
 288 |         duration_value=1,
 289 |         unit=RuleUnit.MONTHS,
 290 |         direction=RuleDirection.AFTER,
 291 |         responsible="МОН",
 292 |         source_reference="Правила ІФНТУНГ 2026, розд. II, п. 9",
 293 |     ),
 294 |     NormativeRuleSeed(
 295 |         code="reviews",
 296 |         title="Подання рецензій і відгуків з КЕП; перевірка академічної
доброчесності",
 297 |         trigger_code="materials_publication",
 298 |         duration_value=45,
 299 |         unit=RuleUnit.CALENDAR_DAYS,
 300 |         direction=RuleDirection.AFTER,
 301 |         responsible="Рецензенти, офіційні опоненти, разова рада",
 302 |         source_reference="Правила ІФНТУНГ 2026, розд. III, п. 12",
 303 |     ),
 304 |     NormativeRuleSeed(
 305 |         code="defense_date_assignment",
 306 |         title="Призначення дати, часу і місця публічного захисту;
оприлюднення рецензій/відгуків",
 307 |         trigger_code="reviews",
 308 |         duration_value=3,
 309 |         unit=RuleUnit.WORKING_DAYS,
 310 |         direction=RuleDirection.AFTER,
 311 |         responsible="Разова рада, заклад",
 312 |         source_reference="Правила ІФНТУНГ 2026, розд. III, п. 14",
 313 |     ),
 314 |     NormativeRuleSeed(
 315 |         code="defense",
 316 |         title="Публічний захист дисертації з трансляцією та
відеозаписом",
 317 |         trigger_code="reviews",
 318 |         duration_value=2,
 319 |         unit=RuleUnit.WEEKS,
 320 |         direction=RuleDirection.AFTER,
 321 |         responsible="Разова рада, здобувач, заклад",
 322 |         source_reference="Правила ІФНТУНГ 2026, розд. III, п. 14-17",
 323 |     ),
 324 |     NormativeRuleSeed(
 325 |         code="council_decision",
 326 |         title="Відкрите голосування та оформлення рішення разової ради за
формою МОН",
 327 |         trigger_code="defense",
 328 |         duration_value=3,
 329 |         unit=RuleUnit.WORKING_DAYS,
 330 |         direction=RuleDirection.AFTER,
 331 |         responsible="Разова рада, голова ради",
 332 |         source_reference="Правила ІФНТУНГ 2026, розд. III, п. 17, 19",
 333 |     ),
 334 |     NormativeRuleSeed(
 335 |         code="result_publication",
 336 |         title="Оприлюднення рішення і відеозапису; внесення результатів
захисту до NAQA.Svr",
 337 |         trigger_code="defense",
 338 |         duration_value=5,
 339 |         unit=RuleUnit.WORKING_DAYS,
 340 |         direction=RuleDirection.AFTER,
 341 |         responsible="Заклад, ВАіД",
 342 |         source_reference="Правила ІФНТУНГ 2026, розд. III, п. 21",
```

```
 343 |     ),
 344 |     NormativeRuleSeed(
 345 |         code="diploma_order",
 346 |         title="Наказ про видачу диплома доктора філософії та додатка до
нього",
 347 |         trigger_code="defense",
 348 |         duration_value=30,
 349 |         unit=RuleUnit.CALENDAR_DAYS,
 350 |         direction=RuleDirection.AFTER,
 351 |         responsible="Керівник закладу, ВАіД",
 352 |         source_reference="Правила ІФНТУНГ 2026, розд. III, п. 22",
 353 |     ),
 354 |     NormativeRuleSeed(
 355 |         code="archive_access",
 356 |         title="Додавання рішення до примірника дисертації та особової
справи; відкритий доступ",
 357 |         trigger_code="diploma_order",
 358 |         duration_value=10,
 359 |         unit=RuleUnit.WORKING_DAYS,
 360 |         direction=RuleDirection.AFTER,
 361 |         responsible="Заклад, бібліотека, ВАіД",
 362 |         source_reference="Правила ІФНТУНГ 2026, розд. III, п. 24-26",
 363 |     ),
 364 | )
 365 |
 366 |
 367 | DEFAULT_COUNCIL_MEMBERS = (
 368 |     "Бойко Андрій Петрович",
 369 |     "Василенко Олена Ігорівна",
 370 |     "Гнатюк Сергій Михайлович",
 371 |     "Данилюк Марія Романівна",
 372 |     "Коваленко Ірина Степанівна",
 373 |     "Литвин Олег Васильович",
 374 |     "Мельник Наталія Богданівна",
 375 |     "Петренко Юрій Андрійович",
 376 |     "Савчук Вікторія Іванівна",
 377 |     "Шевченко Максим Олександрович",
 378 |     "Яворський Тарас Володимирович",
 379 |     "Ільницька Галина Мирославівна",
 380 | )
 381 |
 382 |
 383 | async def get_workspace_summary(session: AsyncSession) ->
WorkspaceSummaryResponse:
 384 |     candidates = await _count(session, Candidate)
 385 |     stages = await _count(session, AttestationStage)
 386 |     councils = await _count(session, Council)
 387 |     schedules = await _count(session, ScheduleVersion)
 388 |     published = await session.scalar(
 389 |         select(func.count())
 390 |         .select_from(ScheduleVersion)
 391 |         .where(ScheduleVersion.status == ScheduleStatus.PUBLISHED)
 392 |     )
 393 |     rooms = await session.scalars(
 394 |         select(Room).where(Room.is_active.is_(True)).order_by(Room.name)
 395 |     )
 396 |     return WorkspaceSummaryResponse(
 397 |         candidates=candidates,
 398 |         stages=stages,
 399 |         councils=councils,
 400 |         schedules=schedules,
 401 |         published_schedules=published or 0,
 402 |         rooms=[room.name for room in rooms],
```

```
    )


async def get_system_settings(session: AsyncSession) ->
SystemSettingsResponse:
    async with session.begin():
        await _ensure_rooms(session)
        operational = await _load_operational_settings(session)
        rooms = await _room_admin_summaries(session)
    return _settings_response(operational, rooms)


async def update_operational_settings(
    session: AsyncSession,
    principal: Principal,
    payload: OperationalSettingsUpdateRequest,
) -> SystemSettingsResponse:
    async with session.begin():
        user = await _get_or_create_user(session, principal)
        record = await _ensure_operational_setting(session)
        current = _normalize_operational_settings_value(record.value)
        updated = _merge_operational_settings(current, payload)
        record.value = updated
        _add_audit(
            session,
            user,
            "update_operational_settings",
            "app_settings",
            OPERATIONAL_SETTINGS_KEY,
            updated,
        )
        operational = _operational_settings_from_value(updated)
        rooms = await _room_admin_summaries(session)
    return _settings_response(operational, rooms)


async def create_room_setting(
    session: AsyncSession,
    principal: Principal,
    payload: RoomCreateRequest,
) -> RoomAdminSummary:
    async with session.begin():
        user = await _get_or_create_user(session, principal)
        room_name = _normalize_room_name(payload.name)
        await _ensure_room_name_available(session, room_name)
        room = Room(
            name=room_name,
            location=payload.location,
            online_url=payload.online_url,
            capacity=payload.capacity,
            is_active=payload.is_active,
        )
        session.add(room)
        await session.flush()
        _add_audit(
            session,
            user,
            "create_room",
            "rooms",
            str(room.id),
            _room_audit_payload(room),
        )
        return _room_to_admin_summary(room, 0)
```

```
 465 |
 466 |
 467 | async def update_room_setting(
 468 |     session: AsyncSession,
 469 |     principal: Principal,
 470 |     room_id: int,
 471 |     payload: RoomUpdateRequest,
 472 | ) -> RoomAdminSummary:
 473 |     async with session.begin():
 474 |         user = await _get_or_create_user(session, principal)
 475 |         room = await session.get(Room, room_id)
 476 |         if room is None:
 477 |             raise LookupError(room_id)
 478 |         if payload.name is not None:
 479 |             room_name = _normalize_room_name(payload.name)
 480 |             await _ensure_room_name_available(session, room_name,
room_id)
 481 |             room.name = room_name
 482 |         if payload.location is not None:
 483 |             room.location = payload.location
 484 |         if payload.online_url is not None:
 485 |             room.online_url = payload.online_url
 486 |         if payload.capacity is not None:
 487 |             room.capacity = payload.capacity
 488 |         if payload.is_active is not None:
 489 |             room.is_active = payload.is_active
 490 |         await session.flush()
 491 |         _add_audit(
 492 |             session,
 493 |             user,
 494 |             "update_room",
 495 |             "rooms",
 496 |             str(room.id),
 497 |             _room_audit_payload(room),
 498 |         )
 499 |         event_count = await _room_event_count(session, room.id)
 500 |         return _room_to_admin_summary(room, event_count)
 501 |
 502 |
 503 | async def deactivate_room_setting(
 504 |     session: AsyncSession,
 505 |     principal: Principal,
 506 |     room_id: int,
 507 | ) -> RoomAdminSummary:
 508 |     return await update_room_setting(
 509 |         session,
 510 |         principal,
 511 |         room_id,
 512 |         RoomUpdateRequest(is_active=False),
 513 |     )
 514 |
 515 |
 516 | async def list_candidates(session: AsyncSession) ->
CandidateListResponse:
 517 |     candidates = (
 518 |         await session.scalars(
 519 |             select(Candidate)
 520 |             .options(
 521 |                 selectinload(Candidate.educational_program),
 522 |                 selectinload(Candidate.department),
 523 |                 selectinload(Candidate.stages),
 524 |                 selectinload(Candidate.councils),
 525 |             )
```

```
            .order_by(Candidate.full_name)
        )
    ).all()
    return CandidateListResponse(
        items=[
            _candidate_to_summary(candidate)
            for candidate in candidates
        ]
    )


async def get_candidate_detail(
    session: AsyncSession,
    candidate_id: int,
) -> CandidateDetailResponse:
    candidate = await session.scalar(
        select(Candidate)
        .options(
            selectinload(Candidate.educational_program),
            selectinload(Candidate.department),
            selectinload(Candidate.stages)
            .selectinload(AttestationStage.rule),
            selectinload(Candidate.stages)
            .selectinload(AttestationStage.documents),
            selectinload(Candidate.councils)
            .selectinload(Council.memberships)
            .selectinload(CouncilMembership.member),
        )
        .where(Candidate.id == candidate_id)
    )
    if candidate is None:
        raise LookupError(candidate_id)

    schedules = (
        await session.scalars(
            select(ScheduleEvent)
            .options(
                selectinload(ScheduleEvent.schedule_version),
                selectinload(ScheduleEvent.slot),
                selectinload(ScheduleEvent.room),
            )
            .where(ScheduleEvent.candidate_id == candidate.id)
            .order_by(ScheduleEvent.id.desc())
        )
    ).all()
    defense_start, defense_end =
_candidate_defense_schedule_window(candidate)
    primary_council = _candidate_primary_council_model(candidate)
    stages = sorted(
        candidate.stages,
        key=lambda item: (item.planned_date or date.max, item.id or 0),
    )
    return CandidateDetailResponse(
        candidate=_candidate_to_summary(candidate),
        thesis_title=candidate.thesis_title,
        supervisor_name=candidate.supervisor_name,
        study_form=candidate.study_form,
        course=candidate.course,
        group_name=candidate.group_name,
        restored_for_defense=candidate.restored_for_defense,
        progress_percent=_candidate_progress_percent(candidate),
        stage_counts=_candidate_stage_counts(candidate),
```

```
        primary_council_id=primary_council.id if primary_council else
None,
        council_status=primary_council.status.value if primary_council
else None,
        council_order_number=(
            primary_council.order_number if primary_council else None
        ),
        council_order_date=primary_council.order_date if primary_council
else None,
        council_publication_date=(
            primary_council.publication_date if primary_council else None
        ),
        council_public_details=(
            primary_council.public_details if primary_council else {}
        ),
        defense_window_start=defense_start,
        defense_window_end=defense_end,
        stages=[_stage_to_summary(stage) for stage in stages],
        issues=_candidate_normative_issues(candidate),
        council_members=_candidate_council_members(candidate),
        documents=_candidate_documents(candidate),
        schedules=[
            CandidateScheduleSummary(
                schedule_id=str(event.schedule_version_id),
                state=event.schedule_version.status.value,
                event_id=event.id,
                starts_at=event.slot.starts_at,
                ends_at=event.slot.ends_at,
                room_name=event.room.name,
                locked=event.is_locked,
            )
            for event in schedules
        ],
    )


async def update_candidate_profile(
    session: AsyncSession,
    principal: Principal,
    candidate_id: int,
    payload: CandidateUpdatePayload,
) -> CandidateDetailResponse:
    async with session.begin():
        user = await _get_or_create_user(session, principal)
        candidate = await session.scalar(
            select(Candidate).where(Candidate.id == candidate_id)
        )
        if candidate is None:
            raise LookupError(candidate_id)
        candidate.thesis_title = payload.thesis_title
        candidate.supervisor_name = payload.supervisor_name
        _add_audit(
            session,
            user,
            "update_candidate_profile",
            "candidates",
            str(candidate.id),
            {
                "thesis_title": payload.thesis_title,
                "supervisor_name": payload.supervisor_name,
            },
        )
    return await get_candidate_detail(session, candidate_id)
```

```
| 
| 
| async def delete_candidate(
|     session: AsyncSession,
|     principal: Principal,
|     candidate_id: int,
| ) -> DeleteResponse:
|     async with session.begin():
|         user = await _get_or_create_user(session, principal)
|         candidate = await session.scalar(
|             select(Candidate).where(Candidate.id == candidate_id)
|         )
|         if candidate is None:
|             raise LookupError(candidate_id)
| 
|         stage_ids = list(
|             await session.scalars(
|                 select(AttestationStage.id)
|                 .where(AttestationStage.candidate_id == candidate_id)
|             )
|         )
|         council_ids = list(
|             await session.scalars(
|                 select(Council.id)
|                 .where(Council.candidate_id == candidate_id)
|             )
|         )
|         schedule_event_count = await session.scalar(
|             select(func.count())
|             .select_from(ScheduleEvent)
|             .where(ScheduleEvent.candidate_id == candidate_id)
|         )
|         document_count = 0
|         if stage_ids:
|             document_count = await session.scalar(
|                 select(func.count())
|                 .select_from(Document)
|                 .where(Document.stage_id.in_(stage_ids))
|             ) or 0
| 
|         related_counts = {
|             "schedule_events": schedule_event_count or 0,
|             "councils": len(council_ids),
|             "stages": len(stage_ids),
|             "documents": document_count,
|         }
|         candidate_name = candidate.full_name
| 
|         await session.execute(
|             sql_delete(ScheduleEvent)
|             .where(ScheduleEvent.candidate_id == candidate_id)
|         )
|         if council_ids:
|             await session.execute(
|                 sql_delete(CouncilMembership)
|                 .where(CouncilMembership.council_id.in_(council_ids))
|             )
|         await session.execute(
|             sql_delete(Council)
|             .where(Council.candidate_id == candidate_id)
|         )
|         if stage_ids:
|             await session.execute(
```

```
                sql_delete(Document)
                .where(Document.stage_id.in_(stage_ids))
            )
        await session.execute(
            sql_delete(AttestationStage)
            .where(AttestationStage.candidate_id == candidate_id)
        )
        await session.execute(
            sql_delete(Candidate)
            .where(Candidate.id == candidate_id)
        )
        _add_audit(
            session,
            user,
            "delete_candidate",
            "candidates",
            str(candidate_id),
            {
                "candidate_name": candidate_name,
                "related_counts": related_counts,
            },
        )

    return DeleteResponse(
        entity_type="candidates",
        id=str(candidate_id),
        related_counts=related_counts,
    )


async def update_candidate_council(
    session: AsyncSession,
    principal: Principal,
    candidate_id: int,
    payload: CouncilCompositionPayload,
) -> CandidateDetailResponse:
    _validate_council_composition(payload)
    async with session.begin():
        user = await _get_or_create_user(session, principal)
        candidate = await session.scalar(
            select(Candidate)
            .options(
                selectinload(Candidate.councils)
                .selectinload(Council.memberships)
            )
            .where(Candidate.id == candidate_id)
        )
        if candidate is None:
            raise LookupError(candidate_id)

        council = _candidate_primary_council_model(candidate)
        if council is None:
            council = Council(
                candidate=candidate,
                sequence_number=1,
                status=CouncilStatus.FORMED,
            )
            session.add(council)
            await session.flush()
        else:
            council.status = CouncilStatus.FORMED
            council.memberships.clear()
            await session.flush()
```

```
773 |
774 |         member_names: list[str] = []
775 |         for member_payload in payload.members:
776 |             member = await _get_or_create_council_member_from_payload(
777 |                 session,
778 |                 member_payload,
779 |             )
780 |             member_names.append(member.full_name)
781 |             session.add(
782 |                 CouncilMembership(
783 |                     council=council,
784 |                     member=member,
785 |                     role=member_payload.role,
786 |                     is_external=member_payload.is_external,
787 |                     qualification_confirmed=(
788 |                         member_payload.qualification_confirmed
789 |                     ),
790 |                     conflict_of_interest_absent=(
791 |                         member_payload.conflict_of_interest_absent
792 |                     ),
793 |                 )
794 |             )
795 |         _add_audit(
796 |             session,
797 |             user,
798 |             "update_candidate_council",
799 |             "councils",
800 |             str(council.id),
801 |             {
802 |                 "candidate_id": candidate.id,
803 |                 "candidate_name": candidate.full_name,
804 |                 "members": member_names,
805 |             },
806 |         )
807 |
808 |     return await get_candidate_detail(session, candidate_id)
809 |
810 |
811 | async def update_candidate_council_metadata(
812 |     session: AsyncSession,
813 |     principal: Principal,
814 |     candidate_id: int,
815 |     payload: CouncilMetadataPayload,
816 | ) -> CandidateDetailResponse:
817 |     async with session.begin():
818 |         user = await _get_or_create_user(session, principal)
819 |         candidate = await session.scalar(
820 |             select(Candidate)
821 |             .options(selectinload(Candidate.councils))
822 |             .where(Candidate.id == candidate_id)
823 |         )
824 |         if candidate is None:
825 |             raise LookupError(candidate_id)
826 |
827 |         council = _candidate_primary_council_model(candidate)
828 |         if council is None:
829 |             council = Council(
830 |                 candidate=candidate,
831 |                 sequence_number=1,
832 |                 status=CouncilStatus.FORMED,
833 |             )
834 |             session.add(council)
835 |             await session.flush()
```

```
        elif council.status == CouncilStatus.DRAFT:
            council.status = CouncilStatus.FORMED

        council.order_number = _empty_to_none(payload.order_number)
        council.order_date = payload.order_date
        council.publication_date = payload.publication_date
        council.public_details = _public_details_from_payload(payload)
        _add_audit(
            session,
            user,
            "update_candidate_council_metadata",
            "councils",
            str(council.id),
            {
                "candidate_id": candidate.id,
                "candidate_name": candidate.full_name,
                "order_number": council.order_number,
                "order_date": (
                    council.order_date.isoformat()
                    if council.order_date
                    else None
                ),
                "publication_date": (
                    council.publication_date.isoformat()
                    if council.publication_date
                    else None
                ),
                "council_code":
council.public_details.get("council_code"),
            },
        )

    return await get_candidate_detail(session, candidate_id)


async def list_stages(session: AsyncSession) -> StageListResponse:
    stages = (
        await session.scalars(
            select(AttestationStage)
            .options(
                selectinload(AttestationStage.candidate),

selectinload(AttestationStage.candidate).selectinload(Candidate.stages),
                selectinload(AttestationStage.rule),
            )
            .order_by(AttestationStage.planned_date, AttestationStage.id)
        )
    ).all()
    return StageListResponse(
        items=[_stage_to_summary(stage) for stage in stages]
    )


async def list_normative_control(session: AsyncSession) ->
NormativeControlResponse:
    candidates = (
        await session.scalars(
            select(Candidate)
            .options(

selectinload(Candidate.stages).selectinload(AttestationStage.rule)
            )
            .order_by(Candidate.full_name)
```

```
 895 |         )
 896 |     ).all()
 897 |     issues: list[NormativeIssue] = []
 898 |     for candidate in candidates:
 899 |         issues.extend(_candidate_normative_issues(candidate))
 900 |     if not issues:
 901 |         issues.append(
 902 |             NormativeIssue(
 903 |                 candidate_id=0,
 904 |                 candidate_name="Система",
 905 |                 stage_id=0,
 906 |                 code="normative_control",
 907 |                 title="Контроль нормативних строків",
 908 |                 severity="info",
 909 |                 message="Порушень або наближення граничних строків не
виявлено.",
 910 |             )
 911 |         )
 912 |     return NormativeControlResponse(items=issues)
 913 |
 914 |
 915 | async def generate_individual_plans(
 916 |     session: AsyncSession,
 917 |     principal: Principal,
 918 | ) -> GeneratePlansResponse:
 919 |     async with session.begin():
 920 |         user = await _get_or_create_user(session, principal)
 921 |         rules_created = await _ensure_rules(session)
 922 |         await _ensure_rooms(session)
 923 |         await _ensure_council_members(session)
 924 |
 925 |         rules = {
 926 |             rule.code: rule
 927 |             for rule in await session.scalars(select(AttestationRule))
 928 |         }
 929 |         candidates = (
 930 |             await session.scalars(
 931 |                 select(Candidate)
 932 |                 .options(selectinload(Candidate.stages),
selectinload(Candidate.councils))
 933 |                 .order_by(Candidate.full_name)
 934 |             )
 935 |         ).all()
 936 |
 937 |         stages_created = 0
 938 |         for candidate in candidates:
 939 |             stages_created += _sync_candidate_plan(session, candidate,
rules)
 940 |             if not candidate.councils:
 941 |                 await _create_default_council(session, candidate)
 942 |
 943 |         _add_audit(
 944 |             session,
 945 |             user,
 946 |             "generate_individual_plans",
 947 |             "attestation_stages",
 948 |             None,
 949 |             {"candidates": len(candidates), "stages_created":
stages_created},
 950 |         )
 951 |     return GeneratePlansResponse(
 952 |         candidates_processed=len(candidates),
 953 |         stages_created=stages_created,
```

```
 954 |         rules_created=rules_created,
 955 |     )
 956 |
 957 |
 958 | async def record_stage_actual_date(
 959 |     session: AsyncSession,
 960 |     principal: Principal,
 961 |     stage_id: int,
 962 |     payload: StageEventPayload,
 963 | ) -> StageEventResponse:
 964 |     async with session.begin():
 965 |         user = await _get_or_create_user(session, principal)
 966 |         rules_created = await _ensure_rules(session)
 967 |         rules = {
 968 |             rule.code: rule
 969 |             for rule in await session.scalars(select(AttestationRule))
 970 |         }
 971 |         stage = await session.scalar(
 972 |             select(AttestationStage)
 973 |             .options(
 974 |                 selectinload(AttestationStage.rule),
 975 |                 selectinload(AttestationStage.candidate)
 976 |                 .selectinload(Candidate.stages)
 977 |                 .selectinload(AttestationStage.rule),
 978 |             )
 979 |             .where(AttestationStage.id == stage_id)
 980 |         )
 981 |         if stage is None:
 982 |             raise LookupError(stage_id)
 983 |         if stage.candidate.study_end is None:
 984 |             raise ValueError(
 985 |                 "Неможливо перерахувати план без опорної дати завершення
підготовки."
 986 |             )
 987 |
 988 |         stage.actual_date = payload.actual_date
 989 |         _sync_candidate_plan(session, stage.candidate, rules)
 990 |         recalculated_stages = len(stage.candidate.stages)
 991 |         issues = _candidate_normative_issues(stage.candidate)
 992 |         _add_audit(
 993 |             session,
 994 |             user,
 995 |             "record_stage_actual_date",
 996 |             "attestation_stages",
 997 |             str(stage.id),
 998 |             {
 999 |                 "actual_date": payload.actual_date.isoformat(),
1000 |                 "comment": payload.comment,
1001 |                 "rules_created": rules_created,
1002 |                 "recalculated_stages": recalculated_stages,
1003 |             },
1004 |         )
1005 |
1006 |     return StageEventResponse(
1007 |         stage=_stage_to_summary(stage),
1008 |         recalculated_stages=recalculated_stages,
1009 |         issues=issues,
1010 |     )
1011 |
1012 |
1013 | async def register_stage_document(
1014 |     session: AsyncSession,
1015 |     principal: Principal,
```

```
    stage_id: int,
    payload: StageDocumentPayload,
) -> StageDocumentResponse:
    async with session.begin():
        user = await _get_or_create_user(session, principal)
        stage = await session.scalar(
            select(AttestationStage)
            .options(selectinload(AttestationStage.candidate))
            .where(AttestationStage.id == stage_id)
        )
        if stage is None:
            raise LookupError(stage_id)
        storage_key = _registered_document_key(stage_id, payload)
        document = Document(
            stage=stage,
            uploaded_by=user,
            original_name=payload.original_name,
            storage_key=storage_key,
            content_type=(
                payload.content_type
                or ("text/uri-list" if payload.external_url else
"application/octet-stream")
            ),
            size_bytes=0,
            checksum_sha256=hashlib.sha256(storage_key.encode("utf-
8")).hexdigest(),
        )
        session.add(document)
        await session.flush()
        _add_audit(
            session,
            user,
            "register_stage_document",
            "documents",
            str(document.id),
            {
                "candidate_id": stage.candidate_id,
                "stage_id": stage.id,
                "original_name": payload.original_name,
                "external_url": payload.external_url,
            },
        )
    return StageDocumentResponse(document=_document_to_summary(document))


async def create_schedule_from_database(
    session: AsyncSession,
    principal: Principal,
    request: ScheduleGenerateRequest,
) -> ScheduleVersionResponse:
    async with session.begin():
        user = await _get_or_create_user(session, principal)
        await _ensure_rooms(session)
        await _ensure_council_members(session)
        await _ensure_candidate_councils(session)
        candidates = await _load_schedulable_candidates(
            session,
            request.candidate_limit,
            candidate_id=request.candidate_id,
        )
        if not candidates:
            if request.candidate_id is not None:
                raise ValueError(
```

```
1077 |                     f"Аспіранта з ID {request.candidate_id} не знайдено "
1078 |                     "або для нього немає даних для формування графіка."
1079 |                 )
1080 |             raise ValueError(
1081 |                 "У базі даних немає аспірантів для формування графіка. "
1082 |                 "Спочатку застосуйте CSV-імпорт ЄДЕБО."
1083 |             )
1084 |         operational = await _load_operational_settings(session)
1085 |         day_count = request.day_count or
operational.default_planning_days
1086 |         slots = await _ensure_time_slots(
1087 |             session,
1088 |             request.start_date or _default_schedule_start(candidates),
1089 |             day_count,
1090 |             operational.defense_start_times,
1091 |             operational.defense_duration,
1092 |         )
1093 |         rooms = (
1094 |             await session.scalars(
1095 |
select(Room).where(Room.is_active.is_(True)).order_by(Room.name)
1096 |             )
1097 |         ).all()
1098 |         if not rooms:
1099 |             raise ValueError("Немає активних аудиторій для формування
графіка.")
1100 |         solve_request = await _build_solve_request(
1101 |             session,
1102 |             candidates,
1103 |             slots,
1104 |             rooms,
1105 |             selected_room_name=request.room_name,
1106 |             previous_events=[],
1107 |             locked_event_ids=set(request.locked_event_ids),
1108 |             blocked_event_ids=set(request.blocked_event_ids),
1109 |             max_defenses_per_member_per_day=(
1110 |                 request.max_defenses_per_member_per_day
1111 |                 or operational.default_max_member_defenses_per_day
1112 |             ),
1113 |         )
1114 |         response = solve_schedule(solve_request)
1115 |         if response.status != "OPTIMAL":
1116 |             raise ValueError(" ".join(response.diagnostics))
1117 |         version = await _persist_schedule(
1118 |             session=session,
1119 |             user=user,
1120 |             solve_response=response,
1121 |             slots=slots,
1122 |             rooms=rooms,
1123 |             candidates=candidates,
1124 |             previous_version=None,
1125 |         )
1126 |         _add_audit(
1127 |             session,
1128 |             user,
1129 |             "create_schedule",
1130 |             "schedule_versions",
1131 |             str(version.id),
1132 |             {"score": response.score, "events":
len(response.assignments)},
1133 |         )
1134 |     return await get_db_schedule(session, str(version.id))
1135 |
```

```
1136 |
1137 | async def replan_schedule(
1138 |     session: AsyncSession,
1139 |     principal: Principal,
1140 |     schedule_id: str,
1141 |     request: ReplanScheduleRequest,
1142 | ) -> ScheduleVersionResponse:
1143 |     async with session.begin():
1144 |         user = await _get_or_create_user(session, principal)
1145 |         previous_version = await _get_schedule_model(session,
schedule_id)
1146 |         previous_events = list(previous_version.events)
1147 |         candidates = [
1148 |             event.candidate
1149 |             for event in sorted(previous_events, key=lambda item:
item.candidate.full_name)
1150 |         ]
1151 |         if not candidates:
1152 |             raise ValueError("Попередня версія графіка не містить
подій.")
1153 |
1154 |         operational = await _load_operational_settings(session)
1155 |         day_count = request.day_count or
operational.default_planning_days
1156 |         start_date = min(event.slot.starts_at.date() for event in
previous_events)
1157 |         slots = await _ensure_time_slots(
1158 |             session,
1159 |             start_date,
1160 |             day_count,
1161 |             operational.defense_start_times,
1162 |             operational.defense_duration,
1163 |         )
1164 |         rooms = (
1165 |             await session.scalars(
1166 |
select(Room).where(Room.is_active.is_(True)).order_by(Room.name)
1167 |             )
1168 |         ).all()
1169 |         if not rooms:
1170 |             raise ValueError("Немає активних аудиторій для формування
графіка.")
1171 |         solve_request = await _build_solve_request(
1172 |             session,
1173 |             candidates,
1174 |             slots,
1175 |             rooms,
1176 |             selected_room_name=request.room_name,
1177 |             previous_events=previous_events,
1178 |             locked_event_ids=set(request.locked_event_ids),
1179 |             blocked_event_ids=set(request.blocked_event_ids),
1180 |             max_defenses_per_member_per_day=(
1181 |                 request.max_defenses_per_member_per_day
1182 |                 or operational.default_max_member_defenses_per_day
1183 |             ),
1184 |         )
1185 |         response = solve_schedule(solve_request)
1186 |         if response.status != "OPTIMAL":
1187 |             raise ValueError(" ".join(response.diagnostics))
1188 |         version = await _persist_schedule(
1189 |             session=session,
1190 |             user=user,
1191 |             solve_response=response,
```

```
            slots=slots,
            rooms=rooms,
            candidates=candidates,
            previous_version=previous_version,
        )
        _add_audit(
            session,
            user,
            "replan_schedule",
            "schedule_versions",
            str(version.id),
            {
                "source_version": previous_version.id,
                "blocked_event_ids": request.blocked_event_ids,
                "locked_event_ids": request.locked_event_ids,
            },
        )
    return await get_db_schedule(session, str(version.id))


async def list_schedules(session: AsyncSession) -> ScheduleListResponse:
    versions = (
        await session.scalars(
            select(ScheduleVersion)
            .options(selectinload(ScheduleVersion.events))
            .order_by(ScheduleVersion.version_number.desc())
        )
    ).all()
    return ScheduleListResponse(
        items=[
            ScheduleSummary(
                id=str(version.id),
                version_number=version.version_number,
                state=ScheduleState(version.status.value),
                created_at=version.created_at,
                published_at=version.published_at,
                score=version.score,
                event_count=len(version.events),
            )
            for version in versions
        ]
    )


async def get_db_schedule(
    session: AsyncSession,
    schedule_id: str,
    *,
    public_only: bool = False,
) -> ScheduleVersionResponse:
    version = await _get_schedule_model(session, schedule_id)
    if public_only and version.status != ScheduleStatus.PUBLISHED:
        raise LookupError(schedule_id)
    return _version_to_response(version)


async def approve_db_schedule(
    session: AsyncSession,
    principal: Principal,
    schedule_id: str,
) -> ScheduleVersionResponse:
    async with session.begin():
        user = await _get_or_create_user(session, principal)
```

```
1255 |         version = await _get_schedule_model(session, schedule_id)
1256 |         if version.status != ScheduleStatus.DRAFT:
1257 |             raise ValueError("Погодити можна лише чернетку графіка.")
1258 |         version.status = ScheduleStatus.APPROVED
1259 |         for event in version.events:
1260 |             event.status = ScheduleEventStatus.CONFIRMED
1261 |         _add_audit(
1262 |             session,
1263 |             user,
1264 |             "approve_schedule",
1265 |             "schedule_versions",
1266 |             str(version.id),
1267 |             None,
1268 |         )
1269 |     return await get_db_schedule(session, schedule_id)
1270 |
1271 |
1272 | async def publish_db_schedule(
1273 |     session: AsyncSession,
1274 |     principal: Principal,
1275 |     schedule_id: str,
1276 | ) -> ScheduleVersionResponse:
1277 |     async with session.begin():
1278 |         user = await _get_or_create_user(session, principal)
1279 |         version = await _get_schedule_model(session, schedule_id)
1280 |         if version.status != ScheduleStatus.APPROVED:
1281 |             raise ValueError("Публікація дозволена лише після
погодження.")
1282 |         version.status = ScheduleStatus.PUBLISHED
1283 |         version.published_at = datetime.now(UTC)
1284 |         _add_audit(
1285 |             session,
1286 |             user,
1287 |             "publish_schedule",
1288 |             "schedule_versions",
1289 |             str(version.id),
1290 |             None,
1291 |         )
1292 |     return await get_db_schedule(session, schedule_id)
1293 |
1294 |
1295 | async def cancel_db_schedule(
1296 |     session: AsyncSession,
1297 |     principal: Principal,
1298 |     schedule_id: str,
1299 | ) -> ScheduleVersionResponse:
1300 |     async with session.begin():
1301 |         user = await _get_or_create_user(session, principal)
1302 |         version = await _get_schedule_model(session, schedule_id)
1303 |         if version.status == ScheduleStatus.PUBLISHED:
1304 |             raise ValueError("Опублікований графік не можна скасувати без
нової версії.")
1305 |         version.status = ScheduleStatus.CANCELLED
1306 |         for event in version.events:
1307 |             event.status = ScheduleEventStatus.CANCELLED
1308 |         _add_audit(
1309 |             session,
1310 |             user,
1311 |             "cancel_schedule",
1312 |             "schedule_versions",
1313 |             str(version.id),
1314 |             None,
1315 |         )
```

```
    return await get_db_schedule(session, schedule_id)


async def delete_schedule_version(
    session: AsyncSession,
    principal: Principal,
    schedule_id: str,
) -> DeleteResponse:
    try:
        numeric_id = int(schedule_id)
    except ValueError as exc:
        raise LookupError(schedule_id) from exc

    async with session.begin():
        user = await _get_or_create_user(session, principal)
        version = await session.scalar(
            select(ScheduleVersion)
            .where(ScheduleVersion.id == numeric_id)
        )
        if version is None:
            raise LookupError(schedule_id)

        event_count = await session.scalar(
            select(func.count())
            .select_from(ScheduleEvent)
            .where(ScheduleEvent.schedule_version_id == numeric_id)
        )
        related_counts = {"schedule_events": event_count or 0}
        version_number = version.version_number
        status_value = version.status.value

        await session.execute(
            sql_delete(ScheduleEvent)
            .where(ScheduleEvent.schedule_version_id == numeric_id)
        )
        await session.execute(
            sql_delete(ScheduleVersion)
            .where(ScheduleVersion.id == numeric_id)
        )
        _add_audit(
            session,
            user,
            "delete_schedule_version",
            "schedule_versions",
            str(numeric_id),
            {
                "version_number": version_number,
                "status": status_value,
                "related_counts": related_counts,
            },
        )

    return DeleteResponse(
        entity_type="schedule_versions",
        id=str(numeric_id),
        related_counts=related_counts,
    )


async def diagnose_schedule_conflicts(
    session: AsyncSession,
    schedule_id: str,
) -> ConflictListResponse:
```

```
    version = await _get_schedule_model(session, schedule_id)
    operational = await _load_operational_settings(session)
    issues: list[ConflictSummary] = []
    events = list(version.events)
    for event in events:
        minimum_date, deadline_date =
_candidate_defense_schedule_window(event.candidate)
        if not _slot_matches_defense_window(event.slot, minimum_date,
deadline_date):
            window = (
                f"{minimum_date.isoformat()} -
{deadline_date.isoformat()}"
                if minimum_date and deadline_date
                else "не визначено"
            )
            issues.append(
                ConflictSummary(
                    severity="error",
                    message=(
                        f"Захист здобувача {event.candidate.full_name}
виходить "
                        f"за нормативне вікно {window}."
                    ),
                )
            )
        if event.slot.ends_at - event.slot.starts_at <
operational.defense_duration:
            issues.append(
                ConflictSummary(
                    severity="error",
                    message=(
                        f"Захист здобувача {event.candidate.full_name}
має тривалість "
                        f"менше налаштованих
{operational.defense_duration_minutes} хвилин."
                    ),
                )
            )
    for index, left in enumerate(events):
        for right in events[index + 1 :]:
            if _overlaps(left.slot, right.slot) and left.room_id ==
right.room_id:
                issues.append(
                    ConflictSummary(
                        severity="error",
                        message=(
                            f"Аудиторія {left.room.name} має одночасні
захисти: "
                            f"{left.candidate.full_name} і
{right.candidate.full_name}."
                        ),
                    )
                )
            left_members = {item.member_id for item in
left.council.memberships}
            right_members = {item.member_id for item in
right.council.memberships}
            common_members = left_members & right_members
            if _overlaps(left.slot, right.slot) and common_members:
                member_names = sorted(
                    item.member.full_name
                    for item in left.council.memberships
                    if item.member_id in common_members
```

```
                )
                issues.append(
                    ConflictSummary(
                        severity="error",
                        message=(
                            f"Член разової ради {', '.join(member_names)}
"
                            f"одночасно залучений до двох захистів: "
                            f"{left.candidate.full_name} і
{right.candidate.full_name}."
                        ),
                    )
                )
    if not issues:
        issues.append(
            ConflictSummary(
                severity="info",
                message="Конфліктів аудиторій і членів рад у цій версії
не виявлено.",
            )
        )
    return ConflictListResponse(items=issues)


async def list_councils(session: AsyncSession) -> CouncilListResponse:
    councils = (
        await session.scalars(
            select(Council)
            .options(
                selectinload(Council.candidate),
                selectinload(Council.memberships),
            )
            .order_by(Council.id)
        )
    ).all()
    return CouncilListResponse(
        items=[
            CouncilSummary(
                id=council.id,
                candidate_id=council.candidate_id,
                candidate_name=council.candidate.full_name,
                status=council.status.value,
                member_count=len(council.memberships),
                order_number=council.order_number,
                order_date=council.order_date,
            )
            for council in councils
        ]
    )


async def list_public_council_cards(session: AsyncSession) ->
list[dict[str, object]]:
    councils = (
        await session.scalars(
            select(Council)
            .options(
                selectinload(Council.candidate)
                .selectinload(Candidate.educational_program),
                selectinload(Council.candidate)
                .selectinload(Candidate.department),
                selectinload(Council.memberships)
                .selectinload(CouncilMembership.member),
```

```
1489 |             )
1490 |             .where(
1491 |                 Council.status.in_(
1492 |                     [
1493 |                         CouncilStatus.FORMED,
1494 |                         CouncilStatus.ACTIVE,
1495 |                         CouncilStatus.COMPLETED,
1496 |                     ]
1497 |                 )
1498 |             )
1499 |             .order_by(Council.order_date.desc(), Council.id.desc())
1500 |         )
1501 |     ).all()
1502 |     cards: list[dict[str, object]] = []
1503 |     for council in councils:
1504 |         try:
1505 |             _validate_council_model(council)
1506 |         except ValueError:
1507 |             continue
1508 |         candidate = council.candidate
1509 |         memberships = sorted(
1510 |             council.memberships,
1511 |             key=lambda item: (_council_role_order(item.role), item.id),
1512 |         )
1513 |         members = [
1514 |             {
1515 |                 "role": _council_role_label(membership.role),
1516 |                 "full_name": membership.member.full_name,
1517 |                 "institution": membership.member.institution,
1518 |                 "academic_degree": membership.member.academic_degree,
1519 |                 "specialty": membership.member.specialty,
1520 |                 "position": membership.member.position,
1521 |                 "profile_url": membership.member.profile_url,
1522 |                 "is_external": membership.is_external,
1523 |                 "qualification_confirmed":
membership.qualification_confirmed,
1524 |                 "conflict_of_interest_absent": (
1525 |                     membership.conflict_of_interest_absent
1526 |                 ),
1527 |             }
1528 |             for membership in memberships
1529 |         ]
1530 |         chair = next(
1531 |             (
1532 |                 member["full_name"]
1533 |                 for member in members
1534 |                 if member["role"] == "Голова ради"
1535 |             ),
1536 |             "не вказано",
1537 |         )
1538 |         opponents = [
1539 |             str(member["full_name"])
1540 |             for member in members
1541 |             if member["role"] == "Офіційний опонент"
1542 |         ]
1543 |         details = council.public_details or {}
1544 |         cards.append(
1545 |             {
1546 |                 "id": council.id,
1547 |                 "council_code": details.get("council_code") or f"РСВР
№{council.id}",
1548 |                 "candidate_id": candidate.id,
1549 |                 "candidate_name": candidate.full_name,
```

```
1550 |                 "candidate_name_genitive":
_person_name_genitive(candidate.full_name),
1551 |                 "program": (
1552 |                     candidate.educational_program.name
1553 |                     if candidate.educational_program
1554 |                     else "освітньо-наукову програму не вказано"
1555 |                 ),
1556 |                 "department": (
1557 |                     candidate.department.name
1558 |                     if candidate.department
1559 |                     else "структурний підрозділ не вказано"
1560 |                 ),
1561 |                 "thesis_title": candidate.thesis_title or "тему
дисертації не вказано",
1562 |                 "supervisor_name": (
1563 |                     candidate.supervisor_name or "наукового керівника не
вказано"
1564 |                 ),
1565 |                 "knowledge_field": (
1566 |                     details.get("knowledge_field")
1567 |                     or "12 – Інформаційні технології"
1568 |                 ),
1569 |                 "specialty": (
1570 |                     details.get("specialty")
1571 |                     or "123 – Комп'ютерна інженерія"
1572 |                 ),
1573 |                 "status": _council_status_label(council.status),
1574 |                 "order_number": council.order_number,
1575 |                 "order_date": council.order_date,
1576 |                 "publication_date": council.publication_date,
1577 |                 "order_url": details.get("order_url"),
1578 |                 "announcement_url": details.get("announcement_url"),
1579 |                 "ukrintei_card_number":
details.get("ukrintei_card_number"),
1580 |                 "ukrintei_card_url": details.get("ukrintei_card_url"),
1581 |                 "nrat_url": details.get("nrat_url"),
1582 |                 "naqa_id": details.get("naqa_id"),
1583 |                 "naqa_url": details.get("naqa_url"),
1584 |                 "defense_datetime": details.get("defense_datetime"),
1585 |                 "defense_datetime_text": _format_public_datetime(
1586 |                     details.get("defense_datetime")
1587 |                 ),
1588 |                 "defense_room": details.get("defense_room"),
1589 |                 "defense_platform": details.get("defense_platform") or
"Zoom",
1590 |                 "conference_url": details.get("conference_url"),
1591 |                 "conference_id": details.get("conference_id"),
1592 |                 "conference_passcode":
details.get("conference_passcode"),
1593 |                 "livestream_url": details.get("livestream_url"),
1594 |                 "dissertation_url": details.get("dissertation_url"),
1595 |                 "dissertation_signature_url": details.get(
1596 |                     "dissertation_signature_url"
1597 |                 ),
1598 |                 "novelty_conclusion_url":
details.get("novelty_conclusion_url"),
1599 |                 "video_url": details.get("video_url"),
1600 |                 "decision_url": details.get("decision_url"),
1601 |                 "video_stamp_note": details.get("video_stamp_note"),
1602 |                 "reviews": details.get("reviews") or [],
1603 |                 "member_count": len(members),
1604 |                 "chair": chair,
1605 |                 "opponents": opponents,
```

```
                "members": members,
            }
        )
    return cards


async def build_council_material(
    session: AsyncSession,
    council_id: int,
) -> CouncilMaterialResponse:
    council = await session.scalar(
        select(Council)
        .options(
selectinload(Council.candidate).selectinload(Candidate.educational_program),
selectinload(Council.memberships).selectinload(CouncilMembership.member),
        )
        .where(Council.id == council_id)
    )
    if council is None:
        raise LookupError(council_id)
    candidate = council.candidate
    program = (
        candidate.educational_program.name
        if candidate.educational_program
        else "освітньо-наукова програма не вказана"
    )
    member_lines = "\n".join(
        f"- {membership.member.full_name}, роль: {membership.role.value};
"
        f"установа: {membership.member.institution}."
        for membership in council.memberships
    )
    body = (
        f"Проєкт інформаційного повідомлення\n\n"
        f"Здобувач: {candidate.full_name}.\n"
        f"Освітня програма: {program}.\n"
        f"Тема дисертації: {candidate.thesis_title or 'уточнюється'}.\n"
        f"Науковий керівник: {candidate.supervisor_name or
'уточнюється'}.\n\n"
        f"Склад разової спеціалізованої вченої ради:\n{member_lines}\n\n"
        "Матеріал підготовлено як службовий проєкт для подальшої
перевірки "
        "відділом аспірантури та оприлюднення після ухвалення
відповідного рішення."
    )
    return CouncilMaterialResponse(
        council_id=council.id,
        candidate_name=candidate.full_name,
        title=f"Інформація про разову раду здобувача
{candidate.full_name}",
        body=body,
    )


async def list_audit_log(session: AsyncSession) -> AuditLogResponse:
    entries = (
        await session.scalars(
select(AuditLog).order_by(AuditLog.occurred_at.desc()).limit(50)
        )
    ).all()
```

```
1661 |     return AuditLogResponse(
1662 |         items=[
1663 |             AuditLogEntry(
1664 |                 id=entry.id,
1665 |                 action=entry.action,
1666 |                 entity_type=entry.entity_type,
1667 |                 entity_id=entry.entity_id,
1668 |                 occurred_at=entry.occurred_at,
1669 |             )
1670 |             for entry in entries
1671 |         ]
1672 |     )
1673 |
1674 |
1675 | def _candidate_to_summary(candidate: Candidate) -> CandidateSummary:
1676 |     return CandidateSummary(
1677 |         id=candidate.id,
1678 |         edebo_card_id=candidate.edebo_card_id,
1679 |         full_name=candidate.full_name,
1680 |         study_status=candidate.study_status,
1681 |         study_start=candidate.study_start,
1682 |         study_end=candidate.study_end,
1683 |         educational_program=(
1684 |             candidate.educational_program.name
1685 |             if candidate.educational_program
1686 |             else None
1687 |         ),
1688 |         department=candidate.department.name if candidate.department else
None,
1689 |         stage_count=len(candidate.stages),
1690 |         council_status=(
1691 |             candidate.councils[0].status.value if candidate.councils else
None
1692 |         ),
1693 |     )
1694 |
1695 |
1696 | def _candidate_progress_percent(candidate: Candidate) -> int:
1697 |     if not candidate.stages:
1698 |         return 0
1699 |     completed = sum(
1700 |         1
1701 |         for stage in candidate.stages
1702 |         if stage.status in {StageStatus.COMPLETED,
StageStatus.NEEDS_REVISION}
1703 |     )
1704 |     return round(completed * 100 / len(candidate.stages))
1705 |
1706 |
1707 | def _candidate_stage_counts(candidate: Candidate) -> dict[str, int]:
1708 |     counts: dict[str, int] = {
1709 |         "planned": 0,
1710 |         "completed": 0,
1711 |         "overdue": 0,
1712 |         "needs_revision": 0,
1713 |     }
1714 |     for stage in candidate.stages:
1715 |         counts[stage.status.value] = counts.get(stage.status.value, 0) +
1
1716 |     return counts
1717 |
1718 |
```

```
1719 | def _candidate_council_members(candidate: Candidate) ->
list[CouncilMemberSummary]:
1720 |     councils = sorted(candidate.councils, key=lambda item:
item.sequence_number)
1721 |     if not councils:
1722 |         return []
1723 |     memberships = sorted(
1724 |         councils[0].memberships,
1725 |         key=lambda item: (item.role.value, item.id),
1726 |     )
1727 |     return [
1728 |         CouncilMemberSummary(
1729 |             id=membership.member.id,
1730 |             full_name=membership.member.full_name,
1731 |             role=membership.role.value,
1732 |             institution=membership.member.institution,
1733 |             academic_degree=membership.member.academic_degree,
1734 |             specialty=membership.member.specialty,
1735 |             is_external=membership.is_external,
1736 |             qualification_confirmed=membership.qualification_confirmed,
1737 |
conflict_of_interest_absent=membership.conflict_of_interest_absent,
1738 |         )
1739 |         for membership in memberships
1740 |     ]
1741 |
1742 |
1743 | def _candidate_primary_council_id(candidate: Candidate) -> int | None:
1744 |     councils = sorted(candidate.councils, key=lambda item:
item.sequence_number)
1745 |     return councils[0].id if councils else None
1746 |
1747 |
1748 | def _candidate_primary_council_model(candidate: Candidate) -> Council |
None:
1749 |     councils = sorted(candidate.councils, key=lambda item:
item.sequence_number)
1750 |     return councils[0] if councils else None
1751 |
1752 |
1753 | def _council_role_order(role: CouncilRole) -> int:
1754 |     order = {
1755 |         CouncilRole.CHAIR: 0,
1756 |         CouncilRole.REVIEWER: 1,
1757 |         CouncilRole.OPPONENT: 2,
1758 |     }
1759 |     return order.get(role, 99)
1760 |
1761 |
1762 | def _council_role_label(role: CouncilRole) -> str:
1763 |     labels = {
1764 |         CouncilRole.CHAIR: "Голова ради",
1765 |         CouncilRole.REVIEWER: "Рецензент",
1766 |         CouncilRole.OPPONENT: "Офіційний опонент",
1767 |     }
1768 |     return labels.get(role, "Член ради")
1769 |
1770 |
1771 | def _council_status_label(status: CouncilStatus) -> str:
1772 |     labels = {
1773 |         CouncilStatus.DRAFT: "проєкт",
1774 |         CouncilStatus.FORMED: "утворена",
1775 |         CouncilStatus.ACTIVE: "активна",
```

```
1776 |         CouncilStatus.COMPLETED: "завершена",
1777 |         CouncilStatus.CANCELLED: "скасована",
1778 |     }
1779 |     return labels.get(status, "невідомо")
1780 |
1781 |
1782 | def _candidate_documents(candidate: Candidate) -> list[DocumentSummary]:
1783 |     documents = [
1784 |         _document_to_summary(document)
1785 |         for stage in candidate.stages
1786 |         for document in stage.documents
1787 |     ]
1788 |     fallback = datetime(1970, 1, 1, tzinfo=UTC)
1789 |     return sorted(documents, key=lambda item: (item.created_at or
fallback, item.id))
1790 |
1791 |
1792 | def _document_to_summary(document: Document) -> DocumentSummary:
1793 |     return DocumentSummary(
1794 |         id=document.id,
1795 |         stage_id=document.stage_id,
1796 |         stage_title=document.stage.title,
1797 |         original_name=document.original_name,
1798 |         content_type=document.content_type,
1799 |         size_bytes=document.size_bytes,
1800 |         storage_key=document.storage_key,
1801 |         created_at=document.created_at,
1802 |     )
1803 |
1804 |
1805 | def _registered_document_key(
1806 |     stage_id: int,
1807 |     payload: StageDocumentPayload,
1808 | ) -> str:
1809 |     if payload.external_url:
1810 |         return payload.external_url
1811 |     stamp = datetime.now(UTC).strftime("%Y%m%d%H%M%S%f")
1812 |     slug = re.sub(r"[^A-Za-z0-9_.-]+", "_",
payload.original_name).strip("_")
1813 |     external_hash = hashlib.sha256(
1814 |         (payload.external_url or payload.original_name).encode("utf-8")
1815 |     ).hexdigest()[:12]
1816 |     return f"registered/stages/{stage_id}/{stamp}-{external_hash}-{slug
or 'document'}"
1817 |
1818 |
1819 | def _validate_council_composition(payload: CouncilCompositionPayload) ->
None:
1820 |     counts = {role: 0 for role in CouncilRole}
1821 |     names: set[str] = set()
1822 |     opponent_institutions: set[str] = set()
1823 |     for member in payload.members:
1824 |         counts[member.role] += 1
1825 |         normalized_name = _normalize_member_name(member.full_name)
1826 |         if normalized_name.casefold() in names:
1827 |             raise ValueError("Одна й та сама особа не може дублюватися в
одній раді.")
1828 |         names.add(normalized_name.casefold())
1829 |
1830 |         institution_key = _normalize_institution(member.institution)
1831 |         if member.role in {CouncilRole.CHAIR, CouncilRole.REVIEWER}:
1832 |             if not _is_home_institution(member.institution):
1833 |                 raise ValueError(
```

```
1834 |                     "Голова разової ради та рецензенти можуть бути тільки
з ІФНТУНГ."
1835 |                 )
1836 |             if member.is_external:
1837 |                 raise ValueError(
1838 |                     "Голова разової ради та рецензенти не позначаються як
зовнішні члени."
1839 |                 )
1840 |         if member.role == CouncilRole.OPPONENT:
1841 |             if _is_home_institution(member.institution):
1842 |                 raise ValueError("Офіційні опоненти не можуть бути з
ІФНТУНГ.")
1843 |             if not member.is_external:
1844 |                 raise ValueError("Офіційні опоненти мають бути зовнішніми
членами ради.")
1845 |             if institution_key in opponent_institutions:
1846 |                 raise ValueError("Офіційні опоненти не можуть бути з
одного закладу.")
1847 |             opponent_institutions.add(institution_key)
1848 |
1849 |     allowed_role_counts = (
1850 |         counts[CouncilRole.CHAIR] == 1
1851 |         and (
1852 |             (
1853 |                 counts[CouncilRole.REVIEWER] == 2
1854 |                 and counts[CouncilRole.OPPONENT] == 2
1855 |             )
1856 |             or (
1857 |                 counts[CouncilRole.REVIEWER] == 1
1858 |                 and counts[CouncilRole.OPPONENT] == 3
1859 |             )
1860 |         )
1861 |     )
1862 |     if not allowed_role_counts:
1863 |         raise ValueError(
1864 |             "Склад разової ради повинен містити 1 голову та один із
варіантів: "
1865 |             "2 рецензенти і 2 офіційні опоненти або 1 рецензент і 3
офіційні опоненти."
1866 |         )
1867 |
1868 |
1869 | async def _get_or_create_council_member_from_payload(
1870 |     session: AsyncSession,
1871 |     payload: CouncilMemberInput,
1872 | ) -> CouncilMember:
1873 |     normalized_name = _normalize_member_name(payload.full_name)
1874 |     member = await session.scalar(
1875 |         select(CouncilMember)
1876 |         .where(func.lower(CouncilMember.full_name) ==
normalized_name.casefold())
1877 |     )
1878 |     if member is None:
1879 |         member = CouncilMember(
1880 |             full_name=normalized_name,
1881 |             email=payload.email,
1882 |             institution=payload.institution,
1883 |             academic_degree=payload.academic_degree,
1884 |             specialty=payload.specialty,
1885 |             position=_empty_to_none(payload.position),
1886 |             profile_url=_empty_to_none(payload.profile_url),
1887 |         )
1888 |         session.add(member)
```

```
1889 |         await session.flush()
1890 |         return member
1891 |     member.email = payload.email or member.email
1892 |     member.institution = payload.institution
1893 |     member.academic_degree = payload.academic_degree
1894 |     member.specialty = payload.specialty
1895 |     member.position = _empty_to_none(payload.position)
1896 |     member.profile_url = _empty_to_none(payload.profile_url)
1897 |     return member
1898 |
1899 |
1900 | def _normalize_member_name(value: str) -> str:
1901 |     return " ".join(value.split())
1902 |
1903 |
1904 | def _empty_to_none(value: str | None) -> str | None:
1905 |     if value is None:
1906 |         return None
1907 |     value = value.strip()
1908 |     return value or None
1909 |
1910 |
1911 | def _public_details_from_payload(payload: CouncilMetadataPayload) ->
dict[str, object]:
1912 |     raw = payload.model_dump(
1913 |         mode="json",
1914 |         exclude={"order_number", "order_date", "publication_date"},
1915 |     )
1916 |     details: dict[str, object] = {}
1917 |     for key, value in raw.items():
1918 |         if key == "reviews":
1919 |             reviews = [
1920 |                 {
1921 |                     review_key: review_value
1922 |                     for review_key, review_value in review.items()
1923 |                     if review_value not in (None, "")
1924 |                 }
1925 |                 for review in value
1926 |                 if review.get("member_name")
1927 |             ]
1928 |             if reviews:
1929 |                 details[key] = reviews
1930 |             continue
1931 |         if value in (None, ""):
1932 |             continue
1933 |         details[key] = value
1934 |     return details
1935 |
1936 |
1937 | def _person_name_genitive(full_name: str) -> str:
1938 |     # Для публічного тексту краще мати окреме поле відмінка, але поки
1939 |     # використовуємо ПІБ із картки, щоб не робити ненадійне автоматичне
1940 |     # відмінювання українських імен.
1941 |     return full_name
1942 |
1943 |
1944 | def _format_public_datetime(value: object) -> str | None:
1945 |     if not value:
1946 |         return None
1947 |     try:
1948 |         if isinstance(value, datetime):
1949 |             moment = value
1950 |         else:
```

```
            moment = datetime.fromisoformat(str(value).replace("Z",
"+00:00"))
    except ValueError:
        return str(value)
    return moment.strftime("%d.%m.%Y p. o %H:%M")


def _normalize_institution(value: str) -> str:
    return " ".join(value.split()).casefold()


def _is_home_institution(value: str) -> bool:
    return _normalize_institution(value) in HOME_INSTITUTION_ALIASES


def _stage_to_summary(stage: AttestationStage) -> StageSummary:
    normative_status, normative_message, deadline_date =
_stage_normative_state(stage)
    return StageSummary(
        id=stage.id,
        candidate_id=stage.candidate_id,
        candidate_name=stage.candidate.full_name,
        code=stage.code,
        title=stage.title,
        responsible=stage.comment,
        planned_date=stage.planned_date,
        actual_date=stage.actual_date,
        status=stage.status.value,
        source_reference=stage.rule.source_reference if stage.rule else
None,
        normative_status=normative_status,
        normative_message=normative_message,
        deadline_date=deadline_date,
    )


def _sync_candidate_plan(
    session: AsyncSession,
    candidate: Candidate,
    rules: dict[str, AttestationRule],
) -> int:
    base_date = candidate.study_end or date.today()
    known_dates: dict[str, date] = {"study_end": base_date}
    stages_by_code = {stage.code: stage for stage in candidate.stages}
    created = 0

    for seed in RULE_SEEDS:
        trigger_date = known_dates.get(seed.trigger_code, base_date)
        planned_date = _apply_rule_duration(trigger_date, seed)
        stage = stages_by_code.get(seed.code)

        if stage is None:
            stage = AttestationStage(
                candidate=candidate,
                rule=rules[seed.code],
                code=seed.code,
                title=seed.title,
                planned_date=planned_date,
                status=StageStatus.PLANNED,
                comment=seed.responsible,
            )
            session.add(stage)
            stages_by_code[seed.code] = stage
```

```
            created += 1
        else:
            stage.title = seed.title
            stage.rule = rules[seed.code]
            stage.comment = seed.responsible
            stage.planned_date = planned_date

        stage.status = _stage_runtime_status(stage)
        known_dates[seed.code] = stage.actual_date or stage.planned_date
or planned_date

    return created


def _stage_runtime_status(stage: AttestationStage) -> StageStatus:
    normative_status, _, _ = _stage_normative_state(stage)
    if stage.actual_date is not None:
        if normative_status == "violation":
            return StageStatus.NEEDS_REVISION
        return StageStatus.COMPLETED
    if normative_status == "overdue":
        return StageStatus.OVERDUE
    return StageStatus.PLANNED


def _candidate_normative_issues(candidate: Candidate) ->
list[NormativeIssue]:
    issues: list[NormativeIssue] = []
    if candidate.study_end is None:
        for stage in candidate.stages:
            issues.append(
                NormativeIssue(
                    candidate_id=candidate.id,
                    candidate_name=candidate.full_name,
                    stage_id=stage.id,
                    code=stage.code,
                    title=stage.title,
                    severity="error",
                    message=(
                        "Не задано незмінну опорну дату завершення
підготовки "
                        "аспіранта, тому нормативний контроль
неможливий."
                    ),
                    planned_date=stage.planned_date,
                    actual_date=stage.actual_date,
                    deadline_date=stage.planned_date,
                )
            )
        return issues

    for stage in candidate.stages:
        status, message, deadline = _stage_normative_state(stage)
        if status not in {"ok", "pending"}:
            issues.append(
                NormativeIssue(
                    candidate_id=candidate.id,
                    candidate_name=candidate.full_name,
                    stage_id=stage.id,
                    code=stage.code,
                    title=stage.title,
                    severity="error" if status in {"violation",
"overdue"} else "warning",
```

```
2069 |                     message=message or "Потрібна перевірка нормативного
строку.",
2070 |                     planned_date=stage.planned_date,
2071 |                     actual_date=stage.actual_date,
2072 |                     deadline_date=deadline,
2073 |                 )
2074 |             )
2075 |     return issues
2076 |
2077 |
2078 | def _stage_normative_state(
2079 |     stage: AttestationStage,
2080 | ) -> tuple[str, str | None, date | None]:
2081 |     if stage.planned_date is None:
2082 |         return "pending", "Планову дату етапу ще не визначено.", None
2083 |
2084 |     minimum_date, deadline_date = _stage_normative_window(stage)
2085 |     if minimum_date and deadline_date and minimum_date > deadline_date:
2086 |         return (
2087 |             "violation",
2088 |             (
2089 |                 f"Нормативне вікно неможливе: мінімально допустима дата "
2090 |                 f"{minimum_date.isoformat()} пізніша за граничний строк "
2091 |                 f"{deadline_date.isoformat()}."
2092 |             ),
2093 |             deadline_date,
2094 |         )
2095 |
2096 |     today = date.today()
2097 |     if stage.actual_date is not None:
2098 |         if minimum_date and stage.actual_date < minimum_date:
2099 |             return (
2100 |                 "violation",
2101 |                 (
2102 |                     f"Фактична дата {stage.actual_date.isoformat()}
раніша за "
2103 |                     f"нормативно допустиму дату
{minimum_date.isoformat()}."
2104 |                 ),
2105 |                 deadline_date,
2106 |             )
2107 |         if deadline_date and stage.actual_date > deadline_date:
2108 |             return (
2109 |                 "violation",
2110 |                 (
2111 |                     f"Фактична дата {stage.actual_date.isoformat()}
перевищує "
2112 |                     f"граничний строк {deadline_date.isoformat()}."
2113 |                 ),
2114 |                 deadline_date,
2115 |             )
2116 |         return "ok", "Етап виконано без порушення нормативного строку.",
deadline_date
2117 |
2118 |     if deadline_date and deadline_date < today:
2119 |         return (
2120 |             "overdue",
2121 |             f"Граничний строк {deadline_date.isoformat()} минув, а етап
не зафіксовано.",
2122 |             deadline_date,
2123 |         )
2124 |     if deadline_date and deadline_date <= today + timedelta(days=7):
2125 |         return (
```

```
2126 |             "risk",
2127 |             f"До граничного строку {deadline_date.isoformat()} залишилося
не більше 7 днів.",
2128 |             deadline_date,
2129 |         )
2130 |     return "pending", "Етап очікує виконання у межах нормативного
строку.", deadline_date
2131 |
2132 |
2133 | def _stage_normative_window(stage: AttestationStage) -> tuple[date |
None, date | None]:
2134 |     if stage.code == "defense":
2135 |         reviews_stage = _candidate_stage_by_code(stage.candidate,
"reviews")
2136 |         reviews_date = None
2137 |         if reviews_stage is not None:
2138 |             reviews_date = reviews_stage.actual_date or
reviews_stage.planned_date
2139 |         minimum_date = (
2140 |             reviews_date + timedelta(weeks=2)
2141 |             if reviews_date is not None
2142 |             else stage.planned_date
2143 |         )
2144 |         deadline_date = (
2145 |             stage.candidate.study_end
2146 |             - timedelta(days=DEFENSE_DEADLINE_BEFORE_STUDY_END_DAYS)
2147 |             if stage.candidate.study_end is not None
2148 |             else stage.planned_date
2149 |         )
2150 |         return minimum_date, deadline_date
2151 |     return None, stage.planned_date
2152 |
2153 |
2154 | def _candidate_stage_by_code(
2155 |     candidate: Candidate,
2156 |     code: str,
2157 | ) -> AttestationStage | None:
2158 |     for stage in candidate.stages:
2159 |         if stage.code == code:
2160 |             return stage
2161 |     return None
2162 |
2163 |
2164 | async def _count(session: AsyncSession, model: type) -> int:
2165 |     return await session.scalar(select(func.count()).select_from(model))
or 0
2166 |
2167 |
2168 | def _apply_rule_duration(base_date: date, seed: NormativeRuleSeed) ->
date:
2169 |     sign = -1 if seed.direction == RuleDirection.BEFORE else 1
2170 |     if seed.unit == RuleUnit.CALENDAR_DAYS:
2171 |         return base_date + timedelta(days=sign * seed.duration_value)
2172 |     if seed.unit == RuleUnit.WEEKS:
2173 |         return base_date + timedelta(weeks=sign * seed.duration_value)
2174 |     if seed.unit == RuleUnit.WORKING_DAYS:
2175 |         return _add_working_days(base_date, sign * seed.duration_value)
2176 |     if seed.unit == RuleUnit.MONTHS:
2177 |         return _add_months(base_date, sign * seed.duration_value)
2178 |     return base_date
2179 |
2180 |
2181 | def _add_working_days(base_date: date, days: int) -> date:
```

```
2182 |     if days == 0:
2183 |         return base_date
2184 |     current = base_date
2185 |     step = 1 if days > 0 else -1
2186 |     remaining = abs(days)
2187 |     while remaining:
2188 |         current += timedelta(days=step)
2189 |         if current.weekday() < 5:
2190 |             remaining -= 1
2191 |     return current
2192 |
2193 |
2194 | def _add_months(base_date: date, months: int) -> date:
2195 |     month_index = base_date.month - 1 + months
2196 |     year = base_date.year + month_index // 12
2197 |     month = month_index % 12 + 1
2198 |     month_lengths = (
2199 |         31,
2200 |         29 if _is_leap_year(year) else 28,
2201 |         31,
2202 |         30,
2203 |         31,
2204 |         30,
2205 |         31,
2206 |         31,
2207 |         30,
2208 |         31,
2209 |         30,
2210 |         31,
2211 |     )
2212 |     day = min(base_date.day, month_lengths[month - 1])
2213 |     return date(year, month, day)
2214 |
2215 |
2216 | def _is_leap_year(year: int) -> bool:
2217 |     return year % 4 == 0 and (year % 100 != 0 or year % 400 == 0)
2218 |
2219 |
2220 | async def _ensure_operational_setting(session: AsyncSession) ->
AppSetting:
2221 |     record = await session.get(AppSetting, OPERATIONAL_SETTINGS_KEY)
2222 |     normalized = _normalize_operational_settings_value(
2223 |         record.value if record is not None else None
2224 |     )
2225 |     if record is None:
2226 |         record = AppSetting(
2227 |             key=OPERATIONAL_SETTINGS_KEY,
2228 |             value=normalized,
2229 |             description="Операційні параметри планування захистів PhD",
2230 |         )
2231 |         session.add(record)
2232 |         await session.flush()
2233 |     elif record.value != normalized:
2234 |         record.value = normalized
2235 |         await session.flush()
2236 |     return record
2237 |
2238 |
2239 | async def _load_operational_settings(session: AsyncSession) ->
OperationalSettings:
2240 |     record = await _ensure_operational_setting(session)
2241 |     return _operational_settings_from_value(record.value)
2242 |
```

```
2243 |
2244 | def _normalize_operational_settings_value(value: dict | None) -> dict:
2245 |     raw = {**DEFAULT_OPERATIONAL_SETTINGS, **(value or {})}
2246 |     duration = int(raw["defense_duration_minutes"])
2247 |     planning_days = int(raw["default_planning_days"])
2248 |     max_member_defenses = int(raw["default_max_member_defenses_per_day"])
2249 |     start_times =
_format_start_times(_parse_start_times(raw["defense_start_times"]))
2250 |     if duration < 180 or duration > 480:
2251 |         duration =
DEFAULT_OPERATIONAL_SETTINGS["defense_duration_minutes"]
2252 |     if planning_days < 1 or planning_days > 120:
2253 |         planning_days =
DEFAULT_OPERATIONAL_SETTINGS["default_planning_days"]
2254 |     if max_member_defenses < 1 or max_member_defenses > 20:
2255 |         max_member_defenses = DEFAULT_OPERATIONAL_SETTINGS[
2256 |             "default_max_member_defenses_per_day"
2257 |         ]
2258 |     return {
2259 |         "defense_duration_minutes": duration,
2260 |         "default_planning_days": planning_days,
2261 |         "default_max_member_defenses_per_day": max_member_defenses,
2262 |         "defense_start_times": start_times,
2263 |     }
2264 |
2265 |
2266 | def _merge_operational_settings(
2267 |     current: dict,
2268 |     payload: OperationalSettingsUpdateRequest,
2269 | ) -> dict:
2270 |     updated = dict(current)
2271 |     if payload.defense_duration_minutes is not None:
2272 |         updated["defense_duration_minutes"] =
payload.defense_duration_minutes
2273 |     if payload.default_planning_days is not None:
2274 |         updated["default_planning_days"] = payload.default_planning_days
2275 |     if payload.default_max_member_defenses_per_day is not None:
2276 |         updated["default_max_member_defenses_per_day"] = (
2277 |             payload.default_max_member_defenses_per_day
2278 |         )
2279 |     if payload.defense_start_times is not None:
2280 |         updated["defense_start_times"] = _format_start_times(
2281 |             _parse_start_times(payload.defense_start_times)
2282 |         )
2283 |     return _normalize_operational_settings_value(updated)
2284 |
2285 |
2286 | def _operational_settings_from_value(value: dict) -> OperationalSettings:
2287 |     normalized = _normalize_operational_settings_value(value)
2288 |     return OperationalSettings(
2289 |         defense_duration_minutes=normalized["defense_duration_minutes"],
2290 |         default_planning_days=normalized["default_planning_days"],
2291 |         default_max_member_defenses_per_day=(
2292 |             normalized["default_max_member_defenses_per_day"]
2293 |         ),
2294 |
defense_start_times=_parse_start_times(normalized["defense_start_times"]),
2295 |     )
2296 |
2297 |
2298 | def _parse_start_times(values: list[str] | tuple[str, ...]) ->
tuple[time, ...]:
2299 |     parsed: list[time] = []
```

```
2300 |     seen: set[str] = set()
2301 |     for value in values:
2302 |         if not re.fullmatch(r"\d{2}:\d{2}", str(value)):
2303 |             raise ValueError("Час початку захистів потрібно задавати у
форматі HH:MM.")
2304 |         try:
2305 |             parsed_time = time.fromisoformat(str(value))
2306 |         except ValueError as exc:
2307 |             raise ValueError(
2308 |                 "Час початку захистів потрібно задавати у форматі HH:MM."
2309 |             ) from exc
2310 |         key = parsed_time.strftime("%H:%M")
2311 |         if key not in seen:
2312 |             parsed.append(parsed_time)
2313 |             seen.add(key)
2314 |     if not parsed:
2315 |         raise ValueError("Потрібно вказати хоча б один час початку
захисту.")
2316 |     return tuple(sorted(parsed))
2317 |
2318 |
2319 | def _format_start_times(values: tuple[time, ...]) -> list[str]:
2320 |     return [item.strftime("%H:%M") for item in values]
2321 |
2322 |
2323 | async def _room_admin_summaries(session: AsyncSession) ->
list[RoomAdminSummary]:
2324 |     rooms = (await
session.scalars(select(Room).order_by(Room.name))).all()
2325 |     count_rows = (
2326 |         await session.execute(
2327 |             select(ScheduleEvent.room_id, func.count(ScheduleEvent.id))
2328 |             .group_by(ScheduleEvent.room_id)
2329 |         )
2330 |     ).all()
2331 |     counts = {room_id: int(count) for room_id, count in count_rows}
2332 |     return [_room_to_admin_summary(room, counts.get(room.id, 0)) for room
in rooms]
2333 |
2334 |
2335 | async def _room_event_count(session: AsyncSession, room_id: int) -> int:
2336 |     return await session.scalar(
2337 |         select(func.count(ScheduleEvent.id)).where(ScheduleEvent.room_id
== room_id)
2338 |     ) or 0
2339 |
2340 |
2341 | def _room_to_admin_summary(room: Room, event_count: int) ->
RoomAdminSummary:
2342 |     return RoomAdminSummary(
2343 |         id=room.id,
2344 |         name=room.name,
2345 |         location=room.location,
2346 |         online_url=room.online_url,
2347 |         capacity=room.capacity,
2348 |         is_active=room.is_active,
2349 |         schedule_event_count=event_count,
2350 |     )
2351 |
2352 |
2353 | def _settings_response(
2354 |     operational: OperationalSettings,
2355 |     rooms: list[RoomAdminSummary],
```

```
2356 | ) -> SystemSettingsResponse:
2357 |     return SystemSettingsResponse(
2358 |         operational=operational.to_response(),
2359 |         rooms=rooms,
2360 |         active_rooms=[room.name for room in rooms if room.is_active],
2361 |         app_environment=settings.app_environment,
2362 |         demo_tokens_enabled=settings.app_environment.casefold() !=
"production",
2363 |         google_allowed_domain=settings.google_allowed_domain,
2364 |         google_oauth_configured=bool(
2365 |             settings.google_client_id and settings.google_client_secret
2366 |         ),
2367 |     )
2368 |
2369 |
2370 | async def _ensure_room_name_available(
2371 |     session: AsyncSession,
2372 |     room_name: str,
2373 |     current_room_id: int | None = None,
2374 | ) -> None:
2375 |     existing = await session.scalar(
2376 |         select(Room).where(func.lower(Room.name) == room_name.casefold())
2377 |     )
2378 |     if existing is not None and existing.id != current_room_id:
2379 |         raise ValueError(f"Аудиторія {room_name} вже існує.")
2380 |
2381 |
2382 | def _normalize_room_name(value: str) -> str:
2383 |     return " ".join(value.split())
2384 |
2385 |
2386 | def _room_audit_payload(room: Room) -> dict:
2387 |     return {
2388 |         "name": room.name,
2389 |         "location": room.location,
2390 |         "online_url": room.online_url,
2391 |         "capacity": room.capacity,
2392 |         "is_active": room.is_active,
2393 |     }
2394 |
2395 |
2396 | async def _get_or_create_user(session: AsyncSession, principal:
Principal) -> User:
2397 |     user = await session.scalar(
2398 |         select(User)
2399 |         .options(selectinload(User.roles))
2400 |         .where(User.google_subject == principal.subject)
2401 |     )
2402 |     if user is None and principal.email:
2403 |         user = await session.scalar(
2404 |             select(User)
2405 |             .options(selectinload(User.roles))
2406 |             .where(User.email == principal.email)
2407 |         )
2408 |     if user is None:
2409 |         user = User(
2410 |             email=principal.email or
f"{principal.subject}@example.invalid",
2411 |             full_name=principal.email or principal.subject,
2412 |             google_subject=principal.subject,
2413 |             roles=[
2414 |                 UserRoleAssignment(role=role)
```

```
2415 |                 for role in sorted(principal.roles, key=lambda item:
item.value)
2416 |             ],
2417 |         )
2418 |         session.add(user)
2419 |         await session.flush()
2420 |         return user
2421 |     if user.google_subject is None:
2422 |         user.google_subject = principal.subject
2423 |     existing_roles = {assignment.role for assignment in user.roles}
2424 |     for role in principal.roles - existing_roles:
2425 |         user.roles.append(UserRoleAssignment(role=role))
2426 |     return user
2427 |
2428 |
2429 | async def _ensure_rules(session: AsyncSession) -> int:
2430 |     created = 0
2431 |     existing = {
2432 |         rule.code: rule
2433 |         for rule in await session.scalars(select(AttestationRule))
2434 |     }
2435 |     for seed in RULE_SEEDS:
2436 |         if seed.code in existing:
2437 |             rule = existing[seed.code]
2438 |             rule.title = seed.title
2439 |             rule.trigger_code = seed.trigger_code
2440 |             rule.duration_value = seed.duration_value
2441 |             rule.unit = seed.unit
2442 |             rule.direction = seed.direction
2443 |             rule.source_reference = seed.source_reference
2444 |             continue
2445 |         session.add(
2446 |             AttestationRule(
2447 |                 code=seed.code,
2448 |                 title=seed.title,
2449 |                 trigger_code=seed.trigger_code,
2450 |                 duration_value=seed.duration_value,
2451 |                 unit=seed.unit,
2452 |                 direction=seed.direction,
2453 |                 source_reference=seed.source_reference,
2454 |                 effective_from=date(2025, 10, 1),
2455 |                 is_active=True,
2456 |             )
2457 |         )
2458 |         created += 1
2459 |     await session.flush()
2460 |     return created
2461 |
2462 |
2463 | async def _ensure_rooms(session: AsyncSession) -> None:
2464 |     existing = {
2465 |         room.name: room
2466 |         for room in await session.scalars(select(Room))
2467 |     }
2468 |     for name in ROOM_NAMES:
2469 |         if name not in existing:
2470 |             session.add(
2471 |                 Room(
2472 |                     name=name,
2473 |                     location=f"Аудиторія {name}",
2474 |                     capacity=30,
2475 |                     is_active=True,
2476 |                 )
```

```
            )
    await session.flush()


async def _ensure_council_members(session: AsyncSession) -> None:
    existing = {
        member.full_name: member
        for member in await session.scalars(select(CouncilMember))
    }
    for index, name in enumerate(DEFAULT_COUNCIL_MEMBERS):
        institution = _default_council_member_institution(index)
        member = existing.get(name)
        if member is None:
            session.add(
                CouncilMember(
                    full_name=name,
                    email=None,
                    institution=institution,
                    academic_degree="кандидат/доктор наук",
                    specialty="Комп'ютерна інженерія",
                )
            )
        else:
            member.institution = institution
    await session.flush()


def _default_council_member_institution(index: int) -> str:
    if index < 4:
        return HOME_INSTITUTION
    return DEFAULT_EXTERNAL_INSTITUTIONS[
        (index - 4) % len(DEFAULT_EXTERNAL_INSTITUTIONS)
    ]


async def _ensure_candidate_councils(session: AsyncSession) -> None:
    await _ensure_council_members(session)
    candidates = (
        await session.scalars(
            select(Candidate)
            .options(selectinload(Candidate.councils))
            .order_by(Candidate.full_name)
        )
    ).all()
    for candidate in candidates:
        if not candidate.councils:
            await _create_default_council(session, candidate)
    await session.flush()


async def _create_default_council(
    session: AsyncSession,
    candidate: Candidate,
) -> Council:
    members = (
        await
session.scalars(select(CouncilMember).order_by(CouncilMember.full_name))
    ).all()
    if len(members) < 5:
        await _ensure_council_members(session)
        members = (
            await session.scalars(
                select(CouncilMember).order_by(CouncilMember.full_name)
```

```
2539 |             )
2540 |         ).all()
2541 |     internal_members = [
2542 |         member for member in members if
_is_home_institution(member.institution)
2543 |     ]
2544 |     external_members = [
2545 |         member for member in members if not
_is_home_institution(member.institution)
2546 |     ]
2547 |     if len(internal_members) < 3 or len(external_members) < 2:
2548 |         raise ValueError(
2549 |             "Недостатньо членів ради для формування нормативного складу:
"
2550 |             "потрібно щонайменше 3 представники ІФНТУНГ і 2 зовнішні
опоненти."
2551 |         )
2552 |     internal_start = (candidate.id or 1) % len(internal_members)
2553 |     external_start = (candidate.id or 1) % len(external_members)
2554 |     selected_internal = [
2555 |         internal_members[(internal_start + offset) %
len(internal_members)]
2556 |         for offset in range(3)
2557 |     ]
2558 |     selected_external: list[CouncilMember] = []
2559 |     used_external_institutions: set[str] = set()
2560 |     for offset in range(len(external_members)):
2561 |         member = external_members[(external_start + offset) %
len(external_members)]
2562 |         institution_key = _normalize_institution(member.institution)
2563 |         if institution_key in used_external_institutions:
2564 |             continue
2565 |         selected_external.append(member)
2566 |         used_external_institutions.add(institution_key)
2567 |         if len(selected_external) == 2:
2568 |             break
2569 |     if len(selected_external) < 2:
2570 |         raise ValueError(
2571 |             "Недостатньо зовнішніх опонентів із різних закладів для
формування ради."
2572 |         )
2573 |     council = Council(
2574 |         candidate=candidate,
2575 |         sequence_number=1,
2576 |         status=CouncilStatus.FORMED,
2577 |         memberships=[
2578 |             CouncilMembership(
2579 |                 member=selected_internal[0],
2580 |                 role=CouncilRole.CHAIR,
2581 |                 is_external=False,
2582 |                 qualification_confirmed=True,
2583 |                 conflict_of_interest_absent=True,
2584 |             ),
2585 |             CouncilMembership(
2586 |                 member=selected_internal[1],
2587 |                 role=CouncilRole.REVIEWER,
2588 |                 is_external=False,
2589 |                 qualification_confirmed=True,
2590 |                 conflict_of_interest_absent=True,
2591 |             ),
2592 |             CouncilMembership(
2593 |                 member=selected_internal[2],
2594 |                 role=CouncilRole.REVIEWER,
```

```
                is_external=False,
                qualification_confirmed=True,
                conflict_of_interest_absent=True,
            ),
            CouncilMembership(
                member=selected_external[0],
                role=CouncilRole.OPPONENT,
                is_external=True,
                qualification_confirmed=True,
                conflict_of_interest_absent=True,
            ),
            CouncilMembership(
                member=selected_external[1],
                role=CouncilRole.OPPONENT,
                is_external=True,
                qualification_confirmed=True,
                conflict_of_interest_absent=True,
            ),
        ],
    )
    session.add(council)
    await session.flush()
    return council


async def _load_schedulable_candidates(
    session: AsyncSession,
    limit: int,
    *,
    candidate_id: int | None = None,
) -> list[Candidate]:
    query = (
        select(Candidate)
        .options(
            selectinload(Candidate.councils)
            .selectinload(Council.memberships)
            .selectinload(CouncilMembership.member),
            selectinload(Candidate.stages),
        )
        .order_by(Candidate.full_name)
    )
    if candidate_id is not None:
        query = query.where(Candidate.id == candidate_id)
    else:
        query = query.limit(limit)
    return list(
        (
            await session.scalars(query)
        ).all()
    )


def _default_schedule_start(candidates: list[Candidate]) -> date:
    defense_dates = [
        stage.planned_date
        for candidate in candidates
        for stage in candidate.stages
        if stage.code == "defense" and stage.planned_date is not None
    ]
    if defense_dates:
        return min(defense_dates)
    return date.today() + timedelta(days=14)

```

```
2658 |
2659 | async def _ensure_time_slots(
2660 |     session: AsyncSession,
2661 |     start_date: date,
2662 |     day_count: int,
2663 |     defense_start_times: tuple[time, ...],
2664 |     defense_duration: timedelta,
2665 | ) -> list[TimeSlot]:
2666 |     existing = {
2667 |         (slot.starts_at.replace(tzinfo=None),
slot.ends_at.replace(tzinfo=None)): slot
2668 |         for slot in await session.scalars(select(TimeSlot))
2669 |     }
2670 |     current = start_date
2671 |     for _ in range(day_count):
2672 |         if current.weekday() < 5:
2673 |             for starts in defense_start_times:
2674 |                 starts_at = datetime.combine(current, starts)
2675 |                 ends_at = starts_at + defense_duration
2676 |                 key = (starts_at, ends_at)
2677 |                 if key not in existing:
2678 |                     slot = TimeSlot(
2679 |                         starts_at=starts_at,
2680 |                         ends_at=ends_at,
2681 |                         is_active=True,
2682 |                     )
2683 |                     session.add(slot)
2684 |                     existing[key] = slot
2685 |         current += timedelta(days=1)
2686 |     await session.flush()
2687 |     return list(
2688 |         (
2689 |             await session.scalars(
2690 |                 select(TimeSlot)
2691 |                 .where(TimeSlot.starts_at >= datetime.combine(start_date,
time.min))
2692 |                 .order_by(TimeSlot.starts_at)
2693 |             )
2694 |         ).all()
2695 |     )
2696 |
2697 |
2698 | async def _build_solve_request(
2699 |     session: AsyncSession,
2700 |     candidates: list[Candidate],
2701 |     slots: list[TimeSlot],
2702 |     rooms: list[Room],
2703 |     selected_room_name: str | None,
2704 |     previous_events: list[ScheduleEvent],
2705 |     locked_event_ids: set[int],
2706 |     blocked_event_ids: set[int],
2707 |     max_defenses_per_member_per_day: int,
2708 | ) -> ScheduleSolveRequest:
2709 |     previous_by_candidate = {event.candidate_id: event for event in
previous_events}
2710 |     blocked_slots_by_candidate: dict[int, set[int]] = {}
2711 |     for event in previous_events:
2712 |         if event.id in blocked_event_ids:
2713 |             blocked_slots_by_candidate.setdefault(event.candidate_id,
set()).add(
2714 |                 event.slot_id
2715 |             )
2716 |
```

```
2717 |     member_ids = sorted(
2718 |         {
2719 |             str(membership.member_id)
2720 |             for candidate in candidates
2721 |             for council in candidate.councils
2722 |             for membership in council.memberships
2723 |         }
2724 |     )
2725 |     slot_ids = {str(slot.id) for slot in slots if slot.is_active}
2726 |     availability = {member_id: set(slot_ids) for member_id in member_ids}
2727 |     await _apply_member_availability(session, availability, slots)
2728 |     allowed_room_ids = _selected_room_ids(rooms, selected_room_name)
2729 |
2730 |     candidate_payloads: list[CandidatePayload] = []
2731 |     slot_cycle = cycle(slots)
2732 |     for candidate in candidates:
2733 |         council = _primary_council(candidate)
2734 |         _validate_council_model(council)
2735 |         member_tuple = tuple(
2736 |             str(membership.member_id)
2737 |             for membership in sorted(council.memberships, key=lambda
item: item.id)
2738 |         )
2739 |         defense_min, defense_deadline =
_candidate_defense_schedule_window(candidate)
2740 |         allowed = {
2741 |             str(slot.id)
2742 |             for slot in slots
2743 |             if slot.id not in
blocked_slots_by_candidate.get(candidate.id, set())
2744 |             and _slot_matches_defense_window(slot, defense_min,
defense_deadline)
2745 |         }
2746 |         if not allowed:
2747 |             window = (
2748 |                 f" з {defense_min.isoformat()} до
{defense_deadline.isoformat()}"
2749 |                 if defense_min and defense_deadline
2750 |                 else ""
2751 |             )
2752 |             raise ValueError(
2753 |                 f"Для здобувача {candidate.full_name} немає допустимих
слотів "
2754 |                 f"у нормативному вікні захисту{window}."
2755 |             )
2756 |         preferred_slot = next(slot_cycle)
2757 |         if str(preferred_slot.id) not in allowed:
2758 |             preferred_slot = next(slot for slot in slots if str(slot.id)
in allowed)
2759 |         candidate_payloads.append(
2760 |             CandidatePayload(
2761 |                 id=str(candidate.id),
2762 |                 name=candidate.full_name,
2763 |                 council_members=member_tuple,
2764 |                 allowed_slots=frozenset(allowed),
2765 |                 preferred_slots=frozenset({str(preferred_slot.id)}),
2766 |                 allowed_rooms=frozenset(allowed_room_ids),
2767 |             )
2768 |         )
2769 |
2770 |     previous_payloads = [
2771 |         AssignmentPayload(
2772 |             candidate_id=str(event.candidate_id),
```

```
2773 |             slot_id=str(event.slot_id),
2774 |             room_id=str(event.room_id),
2775 |             locked=event.is_locked or event.id in locked_event_ids,
2776 |         )
2777 |         for event in previous_events
2778 |         if event.id not in blocked_event_ids
2779 |     ]
2780 |     return ScheduleSolveRequest(
2781 |         slots=[
2782 |             SlotPayload(id=str(slot.id), starts_at=slot.starts_at,
ends_at=slot.ends_at)
2783 |             for slot in slots
2784 |             if slot.is_active
2785 |         ],
2786 |         rooms=[RoomPayload(id=str(room.id), name=room.name) for room in
rooms],
2787 |         member_availability=availability,
2788 |         candidates=candidate_payloads,
2789 |         previous_assignments=previous_payloads,
2790 |         max_defenses_per_member_per_day=max_defenses_per_member_per_day,
2791 |     )
2792 |
2793 |
2794 | def _selected_room_ids(rooms: list[Room], selected_room_name: str | None)
-> set[str]:
2795 |     if selected_room_name is None or not selected_room_name.strip():
2796 |         return set()
2797 |     normalized = selected_room_name.strip()
2798 |     selected = {str(room.id) for room in rooms if room.name ==
normalized}
2799 |     if not selected:
2800 |         available = ", ".join(room.name for room in rooms) or "немає
активних аудиторій"
2801 |         raise ValueError(
2802 |             f"Аудиторію {normalized} не знайдено. Доступні аудиторії:
{available}."
2803 |         )
2804 |     return selected
2805 |
2806 |
2807 | def _candidate_defense_schedule_window(
2808 |     candidate: Candidate,
2809 | ) -> tuple[date | None, date | None]:
2810 |     defense_stage = _candidate_stage_by_code(candidate, "defense")
2811 |     if defense_stage is None:
2812 |         return None, None
2813 |     return _stage_normative_window(defense_stage)
2814 |
2815 |
2816 | def _slot_matches_defense_window(
2817 |     slot: TimeSlot,
2818 |     minimum_date: date | None,
2819 |     deadline_date: date | None,
2820 | ) -> bool:
2821 |     slot_date = slot.starts_at.date()
2822 |     if minimum_date is not None and slot_date < minimum_date:
2823 |         return False
2824 |     if deadline_date is not None and slot_date > deadline_date:
2825 |         return False
2826 |     return True
2827 |
2828 |
2829 | async def _apply_member_availability(
```

```
2830 |     session: AsyncSession,
2831 |     availability: dict[str, set[str]],
2832 |     slots: list[TimeSlot],
2833 | ) -> None:
2834 |     rules = (
2835 |         await session.scalars(
2836 |             select(MemberAvailability).where(
2837 |                 MemberAvailability.member_id.in_(
2838 |                     [int(member_id) for member_id in availability]
2839 |                 )
2840 |             )
2841 |         )
2842 |     ).all()
2843 |     explicit_available = {
2844 |         str(rule.member_id)
2845 |         for rule in rules
2846 |         if rule.kind == AvailabilityKind.AVAILABLE
2847 |     }
2848 |     for member_id in explicit_available:
2849 |         availability[member_id] = set()
2850 |     for rule in rules:
2851 |         member_id = str(rule.member_id)
2852 |         matching_slots = {
2853 |             str(slot.id)
2854 |             for slot in slots
2855 |             if slot.starts_at < rule.ends_at and rule.starts_at <
slot.ends_at
2856 |         }
2857 |         if rule.kind == AvailabilityKind.UNAVAILABLE:
2858 |             availability[member_id] -= matching_slots
2859 |         elif rule.kind in {AvailabilityKind.AVAILABLE,
AvailabilityKind.PREFERRED}:
2860 |             availability[member_id] |= matching_slots
2861 |
2862 |
2863 | async def _persist_schedule(
2864 |     *,
2865 |     session: AsyncSession,
2866 |     user: User,
2867 |     solve_response: ScheduleSolveResponse,
2868 |     slots: list[TimeSlot],
2869 |     rooms: list[Room],
2870 |     candidates: list[Candidate],
2871 |     previous_version: ScheduleVersion | None,
2872 | ) -> ScheduleVersion:
2873 |     max_version = await
session.scalar(select(func.max(ScheduleVersion.version_number)))
2874 |     slot_map = {str(slot.id): slot for slot in slots}
2875 |     room_map = {str(room.id): room for room in rooms}
2876 |     candidate_map = {str(candidate.id): candidate for candidate in
candidates}
2877 |     councils = {candidate.id: _primary_council(candidate) for candidate
in candidates}
2878 |     period_start = min(slot.starts_at.date() for slot in slots)
2879 |     period_end = max(slot.ends_at.date() for slot in slots)
2880 |     version = ScheduleVersion(
2881 |         version_number=(max_version or 0) + 1,
2882 |         status=ScheduleStatus.DRAFT,
2883 |         period_start=period_start,
2884 |         period_end=period_end,
2885 |         score=solve_response.score,
2886 |         created_by=user,
2887 |     )
```

```
    session.add(version)
    await session.flush()

    previous_locked = {
        (event.candidate_id, event.slot_id, event.room_id)
        for event in previous_version.events
    } if previous_version is not None else set()

    for assignment in solve_response.assignments:
        candidate = candidate_map[assignment.candidate_id]
        slot = slot_map[assignment.slot_id]
        room = room_map[assignment.room_id]
        is_locked = (
            assignment.locked
            or (candidate.id, slot.id, room.id) in previous_locked
        )
        session.add(
            ScheduleEvent(
                schedule_version=version,
                candidate=candidate,
                council=councils[candidate.id],
                slot=slot,
                room=room,
                status=ScheduleEventStatus.PLANNED,
                is_locked=is_locked,
                change_penalty=0 if is_locked else 1,
            )
        )
    await session.flush()
    return version


async def _get_schedule_model(
    session: AsyncSession,
    schedule_id: str,
) -> ScheduleVersion:
    try:
        numeric_id = int(schedule_id)
    except ValueError as exc:
        raise LookupError(schedule_id) from exc
    version = await session.scalar(
        select(ScheduleVersion)
        .options(
            selectinload(ScheduleVersion.events)
            .selectinload(ScheduleEvent.candidate),
            selectinload(ScheduleVersion.events)
            .selectinload(ScheduleEvent.candidate)
            .selectinload(Candidate.stages),
            selectinload(ScheduleVersion.events)
            .selectinload(ScheduleEvent.candidate)
            .selectinload(Candidate.councils)
            .selectinload(Council.memberships),
            selectinload(ScheduleVersion.events)
            .selectinload(ScheduleEvent.council)
            .selectinload(Council.memberships)
            .selectinload(CouncilMembership.member),

selectinload(ScheduleVersion.events).selectinload(ScheduleEvent.slot),

selectinload(ScheduleVersion.events).selectinload(ScheduleEvent.room),
        )
        .where(ScheduleVersion.id == numeric_id)
    )
```

```
2949 |     if version is None:
2950 |         raise LookupError(schedule_id)
2951 |     return version
2952 |
2953 |
2954 | def _version_to_response(version: ScheduleVersion) ->
ScheduleVersionResponse:
2955 |     events = sorted(
2956 |         version.events,
2957 |         key=lambda item: (item.slot.starts_at, item.room.name,
item.candidate.full_name),
2958 |     )
2959 |     return ScheduleVersionResponse(
2960 |         id=str(version.id),
2961 |         state=ScheduleState(version.status.value),
2962 |         created_at=version.created_at,
2963 |         approved_at=(
2964 |             version.updated_at
2965 |             if version.status in {ScheduleStatus.APPROVED,
ScheduleStatus.PUBLISHED}
2966 |             else None
2967 |         ),
2968 |         published_at=version.published_at,
2969 |         result=ScheduleSolveResponse(
2970 |             status="OPTIMAL" if events else "EMPTY",
2971 |             assignments=[
2972 |                 AssignmentPayload(
2973 |                     event_id=event.id,
2974 |                     candidate_id=str(event.candidate_id),
2975 |                     candidate_name=event.candidate.full_name,
2976 |                     slot_id=str(event.slot_id),
2977 |                     slot_label=_format_slot(event.slot),
2978 |                     room_id=str(event.room_id),
2979 |                     room_name=event.room.name,
2980 |                     locked=event.is_locked,
2981 |                 )
2982 |                 for event in events
2983 |             ],
2984 |             score=version.score or 0,
2985 |             explored_nodes=0,
2986 |             elapsed_ms=0,
2987 |             diagnostics=[],
2988 |         ),
2989 |     )
2990 |
2991 |
2992 | def _primary_council(candidate: Candidate) -> Council:
2993 |     if not candidate.councils:
2994 |         raise ValueError(f"Для аспіранта {candidate.full_name} не
сформовано раду.")
2995 |     return sorted(candidate.councils, key=lambda item:
item.sequence_number)[0]
2996 |
2997 |
2998 | def _validate_council_model(council: Council) -> None:
2999 |     _validate_council_composition(
3000 |         CouncilCompositionPayload(
3001 |             members=[
3002 |                 CouncilMemberInput(
3003 |                     role=membership.role,
3004 |                     full_name=membership.member.full_name,
3005 |                     email=membership.member.email,
3006 |             institution=membership.member.institution,
```

```
            academic_degree=membership.member.academic_degree,
            specialty=membership.member.specialty,
            position=membership.member.position,
            profile_url=membership.member.profile_url,
            is_external=membership.is_external,
            qualification_confirmed=membership.qualification_confirmed,
                    conflict_of_interest_absent=(
                        membership.conflict_of_interest_absent
                    ),
                )
                for membership in sorted(council.memberships, key=lambda item: item.id)
            ],
        )
    )


def _format_slot(slot: TimeSlot) -> str:
    return f"{slot.starts_at:%d.%m.%Y %H:%M} - {slot.ends_at:%H:%M}"


def _overlaps(left: TimeSlot, right: TimeSlot) -> bool:
    return left.starts_at < right.ends_at and right.starts_at < left.ends_at


def _add_audit(
    session: AsyncSession,
    user: User,
    action: str,
    entity_type: str,
    entity_id: str | None,
    changes: dict | None,
) -> None:
    session.add(
        AuditLog(
            id=_next_audit_id(),
            actor=user,
            action=action,
            entity_type=entity_type,
            entity_id=entity_id,
            changes=changes,
        )
    )


def _next_audit_id() -> int:
    global _LAST_AUDIT_ID
    with _AUDIT_ID_LOCK:
        current = system_time.time_ns()
        _LAST_AUDIT_ID = max(current, _LAST_AUDIT_ID + 1)
        return _LAST_AUDIT_ID
```

## ДОДАТОК А.5

### Алгоритм формування графіка та динамічного перепланування

У додатку наведено код планувальника, який перевіряє жорсткі та м'які обмеження, зберігає фіксовані події та мінімізує кількість змін під час перепланування.

**Лістинг А.5.1 - Алгоритм пошуку допустимого графіка (scheduler.py)**

```
  1 | from __future__ import annotations
  2 |
  3 | from dataclasses import dataclass, field
  4 | from datetime import datetime, timedelta
  5 | from time import perf_counter
  6 | from typing import Iterable
  7 |
  8 |
  9 | DEFAULT_DEFENSE_DURATION = timedelta(hours=3)
 10 |
 11 |
 12 | @dataclass(frozen=True)
 13 | class Slot:
 14 |     id: str
 15 |     starts_at: datetime
 16 |     ends_at: datetime | None = None
 17 |
 18 |     def __post_init__(self) -> None:
 19 |         ends_at = self.ends_at or self.starts_at +
DEFAULT_DEFENSE_DURATION
 20 |         if ends_at <= self.starts_at:
 21 |             raise ValueError("Slot end time must be after start time.")
 22 |         object.__setattr__(self, "ends_at", ends_at)
 23 |
 24 |     @property
 25 |     def day(self) -> str:
 26 |         return self.starts_at.date().isoformat()
 27 |
 28 |
 29 | @dataclass(frozen=True)
 30 | class Room:
 31 |     id: str
 32 |     name: str
 33 |
 34 |
 35 | @dataclass(frozen=True)
 36 | class Candidate:
 37 |     id: str
 38 |     name: str
 39 |     council_members: tuple[str, ...]
 40 |     allowed_slots: frozenset[str]
 41 |     preferred_slots: frozenset[str] = frozenset()
 42 |     allowed_rooms: frozenset[str] = frozenset()
 43 |
 44 |
 45 | @dataclass(frozen=True)
 46 | class Assignment:
 47 |     candidate_id: str
```

```
    slot_id: str
    room_id: str
    locked: bool = False


@dataclass
class ScheduleResult:
    assignments: list[Assignment]
    score: int
    explored_nodes: int
    elapsed_ms: float
    status: str
    diagnostics: list[str] = field(default_factory=list)


class DynamicScheduler:
    """Branch-and-bound scheduler with MRV ordering and stability penalties."""

    def __init__(
        self,
        slots: Iterable[Slot],
        rooms: Iterable[Room],
        member_availability: dict[str, set[str]],
        *,
        preference_penalty: int = 10,
        change_penalty: int = 30,
        max_defenses_per_member_per_day: int = 4,
    ) -> None:
        self.slots = {slot.id: slot for slot in slots}
        self.rooms = {room.id: room for room in rooms}
        self.room_order = {room_id: index for index, room_id in enumerate(self.rooms)}
        self.member_availability = member_availability
        self.preference_penalty = preference_penalty
        self.change_penalty = change_penalty
        self.max_defenses_per_member_per_day = max_defenses_per_member_per_day

    def solve(
        self,
        candidates: Iterable[Candidate],
        previous: Iterable[Assignment] = (),
    ) -> ScheduleResult:
        started = perf_counter()
        candidates_by_id = {candidate.id: candidate for candidate in candidates}
        previous_by_candidate = {
            assignment.candidate_id: assignment for assignment in previous
        }

        fixed = [
            assignment
            for assignment in previous_by_candidate.values()
            if assignment.locked and assignment.candidate_id in candidates_by_id
        ]
        diagnostics = self._validate_fixed(fixed, candidates_by_id)
        if diagnostics:
            return ScheduleResult(
                assignments=[],
                score=0,
                explored_nodes=0,
```

```
106 |                 elapsed_ms=(perf_counter() - started) * 1000,
107 |                 status="INFEASIBLE",
108 |                 diagnostics=diagnostics,
109 |             )
110 |
111 |         room_usage: dict[str, list[str]] = {}
112 |         member_usage: dict[str, list[str]] = {}
113 |         member_day_load: dict[tuple[str, str], int] = {}
114 |
115 |         for assignment in fixed:
116 |             candidate = candidates_by_id[assignment.candidate_id]
117 |             room_usage.setdefault(assignment.room_id,
[]).append(assignment.slot_id)
118 |             for member_id in candidate.council_members:
119 |                 member_usage.setdefault(member_id,
[]).append(assignment.slot_id)
120 |                 day = self.slots[assignment.slot_id].day
121 |                 key = (member_id, day)
122 |                 member_day_load[key] = member_day_load.get(key, 0) + 1
123 |
124 |         remaining = [
125 |             candidate for candidate in candidates_by_id.values()
126 |             if candidate.id not in {item.candidate_id for item in fixed}
127 |         ]
128 |         options = {
129 |             candidate.id: self._candidate_options(candidate)
130 |             for candidate in remaining
131 |         }
132 |         for candidate in remaining:
133 |             if not options[candidate.id]:
134 |                 diagnostics.append(
135 |                     f"{candidate.name}: немає слота, доступного для всіх
членів ради."
136 |                 )
137 |
138 |         if diagnostics:
139 |             return ScheduleResult(
140 |                 assignments=[],
141 |                 score=0,
142 |                 explored_nodes=0,
143 |                 elapsed_ms=(perf_counter() - started) * 1000,
144 |                 status="INFEASIBLE",
145 |                 diagnostics=diagnostics,
146 |             )
147 |
148 |         remaining.sort(key=lambda item: (len(options[item.id]), item.id))
149 |         best_assignments: list[Assignment] | None = None
150 |         best_score = 10**9
151 |         explored_nodes = 0
152 |
153 |         def option_cost(candidate: Candidate, slot_id: str, room_id: str)
-> int:
154 |             cost = 0
155 |             if candidate.preferred_slots and slot_id not in
candidate.preferred_slots:
156 |                 cost += self.preference_penalty
157 |             previous_assignment = previous_by_candidate.get(candidate.id)
158 |             if previous_assignment and (
159 |                 previous_assignment.slot_id != slot_id
160 |                 or previous_assignment.room_id != room_id
161 |             ):
162 |                 cost += self.change_penalty
163 |             return cost
```

```
164 |
165 |         def room_rank(candidate: Candidate, room_id: str) -> int:
166 |             if not self.room_order:
167 |                 return 0
168 |             offset = sum(ord(character) for character in candidate.id)
169 |             return (self.room_order[room_id] - offset) %
len(self.room_order)
170 |
171 |         def backtrack(index: int, current: list[Assignment], score: int) -
> None:
172 |             nonlocal best_assignments, best_score, explored_nodes
173 |             explored_nodes += 1
174 |             if score >= best_score:
175 |                 return
176 |             if index == len(remaining):
177 |                 best_assignments = fixed + list(current)
178 |                 best_score = score
179 |                 return
180 |
181 |             candidate = remaining[index]
182 |             ranked_options = sorted(
183 |                 options[candidate.id],
184 |                 key=lambda option: (
185 |                     option_cost(candidate, option[0], option[1]),
186 |                     self.slots[option[0]].starts_at,
187 |                     room_rank(candidate, option[1]),
188 |                     option[1],
189 |                 ),
190 |             )
191 |
192 |             for slot_id, room_id in ranked_options:
193 |                 if self._has_interval_conflict(
194 |                     slot_id,
195 |                     room_usage.get(room_id, []),
196 |                 ):
197 |                     continue
198 |                 if any(
199 |                     self._has_interval_conflict(
200 |                         slot_id,
201 |                         member_usage.get(member_id, []),
202 |                     )
203 |                     for member_id in candidate.council_members
204 |                 ):
205 |                     continue
206 |
207 |                 day = self.slots[slot_id].day
208 |                 if any(
209 |                     member_day_load.get((member_id, day), 0)
210 |                     >= self.max_defenses_per_member_per_day
211 |                     for member_id in candidate.council_members
212 |                 ):
213 |                     continue
214 |
215 |                 room_usage.setdefault(room_id, []).append(slot_id)
216 |                 for member_id in candidate.council_members:
217 |                     member_usage.setdefault(member_id, []).append(slot_id)
218 |                     key = (member_id, day)
219 |                     member_day_load[key] = member_day_load.get(key, 0) + 1
220 |
221 |                 current.append(Assignment(candidate.id, slot_id, room_id))
222 |                 backtrack(
223 |                     index + 1,
224 |                     current,
```

```
                    score + option_cost(candidate, slot_id, room_id),
                )
                current.pop()

                room_usage[room_id].remove(slot_id)
                for member_id in candidate.council_members:
                    member_usage[member_id].remove(slot_id)
                    key = (member_id, day)
                    member_day_load[key] -= 1

        backtrack(0, [], 0)
        elapsed_ms = (perf_counter() - started) * 1000

        if best_assignments is None:
            return ScheduleResult(
                assignments=[],
                score=0,
                explored_nodes=explored_nodes,
                elapsed_ms=elapsed_ms,
                status="INFEASIBLE",
                diagnostics=[
                    "Сукупність обмежень не допускає повного розкладу. "
                    "Необхідно додати слоти, аудиторії або змінити
доступність членів ради."
                ],
            )

        return ScheduleResult(
            assignments=sorted(
                best_assignments,
                key=lambda item: (
                    self.slots[item.slot_id].starts_at,
                    item.room_id,
                ),
            ),
            score=best_score,
            explored_nodes=explored_nodes,
            elapsed_ms=elapsed_ms,
            status="OPTIMAL",
        )

    def _candidate_options(self, candidate: Candidate) -> list[tuple[str,
str]]:
        result: list[tuple[str, str]] = []
        for slot_id in candidate.allowed_slots:
            if slot_id not in self.slots:
                continue
            if not all(
                slot_id in self.member_availability.get(member_id, set())
                for member_id in candidate.council_members
            ):
                continue
            room_ids = candidate.allowed_rooms or frozenset(self.rooms)
            for room_id in room_ids:
                if room_id not in self.rooms:
                    continue
                result.append((slot_id, room_id))
        return result

    def _has_interval_conflict(
        self,
        slot_id: str,
        used_slot_ids: Iterable[str],
```

```
286 |     ) -> bool:
287 |         return any(
288 |             self._slots_overlap(slot_id, used_slot_id)
289 |             for used_slot_id in used_slot_ids
290 |         )
291 |
292 |     def _slots_overlap(self, left_id: str, right_id: str) -> bool:
293 |         left = self.slots[left_id]
294 |         right = self.slots[right_id]
295 |         return left.starts_at < right.ends_at and right.starts_at <
left.ends_at
296 |
297 |     def _validate_fixed(
298 |         self,
299 |         fixed: list[Assignment],
300 |         candidates_by_id: dict[str, Candidate],
301 |     ) -> list[str]:
302 |         diagnostics: list[str] = []
303 |         room_usage: dict[str, list[str]] = {}
304 |         member_usage: dict[str, list[str]] = {}
305 |         member_day_load: dict[tuple[str, str], int] = {}
306 |         for assignment in fixed:
307 |             candidate = candidates_by_id[assignment.candidate_id]
308 |             if assignment.slot_id not in self.slots:
309 |                 diagnostics.append(
310 |                     f"Зафіксований слот {assignment.slot_id} відсутній у
довіднику."
311 |                 )
312 |                 continue
313 |             if assignment.room_id not in self.rooms:
314 |                 diagnostics.append(
315 |                     f"Зафіксована аудиторія {assignment.room_id} відсутня
у довіднику."
316 |                 )
317 |                 continue
318 |             if candidate.allowed_rooms and assignment.room_id not in
candidate.allowed_rooms:
319 |                 diagnostics.append(
320 |                     f"{candidate.name}: зафіксована аудиторія
{assignment.room_id} "
321 |                     "не належить до дозволених аудиторій для захисту."
322 |                 )
323 |             if assignment.slot_id not in candidate.allowed_slots:
324 |                 diagnostics.append(
325 |                     f"{candidate.name}: зафіксований слот
{assignment.slot_id} "
326 |                     "не належить допустимому періоду."
327 |                 )
328 |             if self._has_interval_conflict(
329 |                 assignment.slot_id,
330 |                 room_usage.get(assignment.room_id, []),
331 |             ):
332 |                 diagnostics.append(
333 |                     f"Подвійне бронювання аудиторії {assignment.room_id}."
334 |                 )
335 |             room_usage.setdefault(assignment.room_id,
[]).append(assignment.slot_id)
336 |             day = self.slots[assignment.slot_id].day
337 |             for member_id in candidate.council_members:
338 |                 if assignment.slot_id not in self.member_availability.get(
339 |                     member_id, set()
340 |                 ):
341 |                     diagnostics.append(
```

```
342 |                         f"Член ради {member_id} недоступний у
зафіксованому "
343 |                         f"слоті {assignment.slot_id}."
344 |                     )
345 |                 if self._has_interval_conflict(
346 |                     assignment.slot_id,
347 |                     member_usage.get(member_id, []),
348 |                 ):
349 |                     diagnostics.append(
350 |                         f"Член ради {member_id} має два одночасні
захисти."
351 |                     )
352 |                 member_usage.setdefault(member_id,
[]).append(assignment.slot_id)
353 |                 day_key = (member_id, day)
354 |                 member_day_load[day_key] = member_day_load.get(day_key, 0)
+ 1
355 |                 if (
356 |                     member_day_load[day_key]
357 |                     > self.max_defenses_per_member_per_day
358 |                 ):
359 |                     diagnostics.append(
360 |                         f"Член ради {member_id} перевищує денний ліміт
захистів."
361 |                     )
362 |         return diagnostics
```

## Лістинг А.5.2 - Сервісний шар взаємодії з планувальником (app/services/scheduling.py)

```
  1 | from __future__ import annotations
  2 |
  3 | from app.schemas import (
  4 |     AssignmentPayload,
  5 |     CandidatePayload,
  6 |     RoomPayload,
  7 |     ScheduleSolveRequest,
  8 |     ScheduleSolveResponse,
  9 |     SlotPayload,
 10 | )
 11 | from sample_data import build_dataset
 12 | from scheduler import (
 13 |     DEFAULT_DEFENSE_DURATION,
 14 |     Assignment,
 15 |     Candidate,
 16 |     DynamicScheduler,
 17 |     Room,
 18 |     Slot,
 19 | )
 20 |
 21 |
 22 | def solve_schedule(payload: ScheduleSolveRequest) ->
ScheduleSolveResponse:
 23 |     _validate_unique_ids(payload)
 24 |     slot_map = {item.id: item for item in payload.slots}
 25 |     room_map = {item.id: item for item in payload.rooms}
 26 |     candidate_map = {item.id: item for item in payload.candidates}
 27 |     scheduler = DynamicScheduler(
 28 |         slots=[
```

```
            Slot(id=item.id, starts_at=item.starts_at,
ends_at=item.ends_at)
            for item in payload.slots
        ],
        rooms=[Room(id=item.id, name=item.name) for item in
payload.rooms],
        member_availability={
            member_id: set(slot_ids)
            for member_id, slot_ids in payload.member_availability.items()
        },
        preference_penalty=payload.preference_penalty,
        change_penalty=payload.change_penalty,
        max_defenses_per_member_per_day=(
            payload.max_defenses_per_member_per_day
        ),
    )
    result = scheduler.solve(
        candidates=[
            Candidate(
                id=item.id,
                name=item.name,
                council_members=item.council_members,
                allowed_slots=item.allowed_slots,
                preferred_slots=item.preferred_slots,
                allowed_rooms=item.allowed_rooms,
            )
            for item in payload.candidates
        ],
        previous=[
            Assignment(
                candidate_id=item.candidate_id,
                slot_id=item.slot_id,
                room_id=item.room_id,
                locked=item.locked,
            )
            for item in payload.previous_assignments
        ],
    )
    return ScheduleSolveResponse(
        status=result.status,
        assignments=[
            AssignmentPayload(
                candidate_id=item.candidate_id,
                candidate_name=(
                    candidate_map[item.candidate_id].name
                    if item.candidate_id in candidate_map
                    else None
                ),
                slot_id=item.slot_id,
                slot_label=(
                    _format_slot_label(slot_map[item.slot_id])
                    if item.slot_id in slot_map
                    else None
                ),
                room_id=item.room_id,
                room_name=(
                    room_map[item.room_id].name
                    if item.room_id in room_map
                    else None
                ),
                locked=item.locked,
            )
            for item in result.assignments
```

```
 90 |         ],
 91 |         score=result.score,
 92 |         explored_nodes=result.explored_nodes,
 93 |         elapsed_ms=round(result.elapsed_ms, 3),
 94 |         diagnostics=result.diagnostics,
 95 |     )
 96 |
 97 |
 98 | def build_demo_request(candidate_count: int) -> ScheduleSolveRequest:
 99 |     slots, rooms, availability, candidates =
build_dataset(candidate_count)
100 |     return ScheduleSolveRequest(
101 |         slots=[
102 |             SlotPayload(id=item.id, starts_at=item.starts_at,
ends_at=item.ends_at)
103 |             for item in slots
104 |         ],
105 |         rooms=[RoomPayload(id=item.id, name=item.name) for item in rooms],
106 |         member_availability=availability,
107 |         candidates=[
108 |             CandidatePayload(
109 |                 id=item.id,
110 |                 name=item.name,
111 |                 council_members=item.council_members,
112 |                 allowed_slots=item.allowed_slots,
113 |                 preferred_slots=item.preferred_slots,
114 |                 allowed_rooms=item.allowed_rooms,
115 |             )
116 |             for item in candidates
117 |         ],
118 |     )
119 |
120 |
121 | def _validate_unique_ids(payload: ScheduleSolveRequest) -> None:
122 |     groups = {
123 |         "слотів": [item.id for item in payload.slots],
124 |         "аудиторій": [item.id for item in payload.rooms],
125 |         "аспірантів": [item.id for item in payload.candidates],
126 |     }
127 |     for label, values in groups.items():
128 |         if len(values) != len(set(values)):
129 |             raise ValueError(f"Ідентифікатори {label} повинні бути
унікальними.")
130 |
131 |
132 | def _format_slot_label(slot: SlotPayload) -> str:
133 |     ends_at = slot.ends_at or slot.starts_at + DEFAULT_DEFENSE_DURATION
134 |     return (
135 |         f"{slot.starts_at.strftime('%d.%m.%Y %H:%M')}"
136 |         f" - {ends_at.strftime('%H:%M')}"
137 |     )
```

## ДОДАТОК А.6

### Авторизація, ролі та корпоративна пошта

У додатку подано код JWT-авторизації, перевірки ролей доступу та інтеграції Google OAuth для входу через корпоративну поштову адресу.

## Лістинг А.6.1 - JWT-токени та рольова модель доступу (app/security.py)

```
 1 | from __future__ import annotations
 2 |
 3 | from dataclasses import dataclass
 4 | from datetime import UTC, datetime, timedelta
 5 | from typing import Callable
 6 |
 7 | from fastapi import Depends, HTTPException, status
 8 | from fastapi.security import HTTPAuthorizationCredentials, HTTPBearer
 9 | from jose import JWTError, jwt
10 |
11 | from app.config import settings
12 | from app.models.enums import UserRole
13 |
14 |
15 | bearer_scheme = HTTPBearer(auto_error=False)
16 |
17 |
18 | @dataclass(frozen=True)
19 | class Principal:
20 |     subject: str
21 |     email: str | None
22 |     roles: frozenset[UserRole]
23 |
24 |
25 | def create_access_token(
26 |     subject: str,
27 |     roles: set[UserRole],
28 |     *,
29 |     email: str | None = None,
30 | ) -> tuple[str, int]:
31 |     expires_in = settings.access_token_minutes * 60
32 |     now = datetime.now(UTC)
33 |     payload = {
34 |         "sub": subject,
35 |         "email": email,
36 |         "roles": sorted(role.value for role in roles),
37 |         "iat": now,
38 |         "exp": now + timedelta(seconds=expires_in),
39 |     }
40 |     token = jwt.encode(
41 |         payload,
42 |         settings.jwt_secret_key,
43 |         algorithm=settings.jwt_algorithm,
44 |     )
45 |     return token, expires_in
46 |
47 |
48 | async def get_current_principal(
49 |     credentials: HTTPAuthorizationCredentials | None =
Depends(bearer_scheme),
50 | ) -> Principal:
51 |     if credentials is None:
52 |         raise HTTPException(
53 |             status_code=status.HTTP_401_UNAUTHORIZED,
54 |             detail="Потрібен Bearer-токен.",
55 |             headers={"WWW-Authenticate": "Bearer"},
56 |         )
57 |     try:
58 |         payload = jwt.decode(
59 |             credentials.credentials,
60 |             settings.jwt_secret_key,
```

```
61 |             algorithms=[settings.jwt_algorithm],
62 |         )
63 |         subject = payload.get("sub")
64 |         role_values = payload.get("roles", [])
65 |         if not subject or not isinstance(role_values, list):
66 |             raise JWTError("Invalid claims")
67 |         roles = frozenset(UserRole(value) for value in role_values)
68 |     except (JWTError, ValueError, TypeError) as exc:
69 |         raise HTTPException(
70 |             status_code=status.HTTP_401_UNAUTHORIZED,
71 |             detail="Токен недійсний або прострочений.",
72 |             headers={"WWW-Authenticate": "Bearer"},
73 |         ) from exc
74 |     return Principal(
75 |         subject=subject,
76 |         email=payload.get("email"),
77 |         roles=roles,
78 |     )
79 |
80 |
81 | def require_roles(*allowed_roles: UserRole) -> Callable:
82 |     allowed = frozenset(allowed_roles)
83 |
84 |     async def dependency(
85 |         principal: Principal = Depends(get_current_principal),
86 |     ) -> Principal:
87 |         if not principal.roles.intersection(allowed):
88 |             raise HTTPException(
89 |                 status_code=status.HTTP_403_FORBIDDEN,
90 |                 detail="Роль користувача не має права виконувати цю
операцію.",
91 |             )
92 |         return principal
93 |
94 |     return dependency
```

## Лістинг А.6.2 - Google OAuth для корпоративної пошти (app/services/google_auth.py)

```
  1 | from __future__ import annotations
  2 |
  3 | import json
  4 | import urllib.parse
  5 | import urllib.request
  6 | from dataclasses import dataclass
  7 | from typing import Any
  8 |
  9 | from sqlalchemy import select
 10 | from sqlalchemy.ext.asyncio import AsyncSession
 11 | from sqlalchemy.orm import selectinload
 12 |
 13 | from app.config import settings
 14 | from app.models import User, UserRoleAssignment
 15 | from app.models.enums import UserRole
 16 | from app.security import create_access_token
 17 |
 18 |
 19 | GOOGLE_AUTHORIZATION_ENDPOINT =
"https://accounts.google.com/o/oauth2/v2/auth"
 20 | GOOGLE_TOKEN_ENDPOINT = "https://oauth2.googleapis.com/token"
```

```
 21 | GOOGLE_TOKENINFO_ENDPOINT = "https://oauth2.googleapis.com/tokeninfo"
 22 | GOOGLE_SCOPES = "openid email profile"
 23 |
 24 |
 25 | class GoogleAuthError(ValueError):
 26 |     """Raised when Google Workspace authentication cannot be completed."""
 27 |
 28 |
 29 | @dataclass(frozen=True)
 30 | class GoogleIdentity:
 31 |     subject: str
 32 |     email: str
 33 |     full_name: str
 34 |     hosted_domain: str | None
 35 |
 36 |
 37 | def google_auth_configured() -> bool:
 38 |     return bool(settings.google_client_id and
settings.google_client_secret)
 39 |
 40 |
 41 | def build_google_authorization_url(state: str) -> str:
 42 |     if not google_auth_configured():
 43 |         raise GoogleAuthError(
 44 |             "Google OAuth не налаштовано: задайте GOOGLE_CLIENT_ID і
GOOGLE_CLIENT_SECRET."
 45 |         )
 46 |     params = {
 47 |         "client_id": settings.google_client_id,
 48 |         "redirect_uri": settings.google_redirect_uri,
 49 |         "response_type": "code",
 50 |         "scope": GOOGLE_SCOPES,
 51 |         "state": state,
 52 |         "access_type": "online",
 53 |         "prompt": "select_account",
 54 |     }
 55 |     if settings.google_allowed_domain:
 56 |         params["hd"] = settings.google_allowed_domain
 57 |     return
f"{GOOGLE_AUTHORIZATION_ENDPOINT}?{urllib.parse.urlencode(params)}"
 58 |
 59 |
 60 | def exchange_code_for_identity(code: str) -> GoogleIdentity:
 61 |     token_payload = {
 62 |         "code": code,
 63 |         "client_id": settings.google_client_id,
 64 |         "client_secret": settings.google_client_secret,
 65 |         "redirect_uri": settings.google_redirect_uri,
 66 |         "grant_type": "authorization_code",
 67 |     }
 68 |     token_response = _post_form(GOOGLE_TOKEN_ENDPOINT, token_payload)
 69 |     id_token = token_response.get("id_token")
 70 |     if not isinstance(id_token, str) or not id_token:
 71 |         raise GoogleAuthError("Google не повернув id_token для перевірки
користувача.")
 72 |
 73 |     claims = _get_json(
 74 |         f"{GOOGLE_TOKENINFO_ENDPOINT}?{urllib.parse.urlencode({'id_token':
id_token})}"
 75 |     )
 76 |     if claims.get("aud") != settings.google_client_id:
 77 |         raise GoogleAuthError("id_token видано не для цього застосунку.")
 78 |     if str(claims.get("email_verified", "")).casefold() != "true":
```

```
 79 |         raise GoogleAuthError("Корпоративну адресу не підтверджено
Google.")
 80 |
 81 |     email = str(claims.get("email", "")).strip().casefold()
 82 |     if not email or "@" not in email:
 83 |         raise GoogleAuthError("Google не повернув коректну адресу
електронної пошти.")
 84 |
 85 |     hosted_domain = claims.get("hd")
 86 |     _ensure_allowed_domain(email, str(hosted_domain) if hosted_domain else
None)
 87 |     subject = str(claims.get("sub", "")).strip()
 88 |     if not subject:
 89 |         raise GoogleAuthError("Google не повернув стабільний ідентифікатор
користувача.")
 90 |     return GoogleIdentity(
 91 |         subject=subject,
 92 |         email=email,
 93 |         full_name=str(claims.get("name") or email),
 94 |         hosted_domain=str(hosted_domain) if hosted_domain else None,
 95 |     )
 96 |
 97 |
 98 | async def issue_local_token_for_google_identity(
 99 |     session: AsyncSession,
100 |     identity: GoogleIdentity,
101 | ) -> tuple[str, int, set[UserRole]]:
102 |     async with session.begin():
103 |         user = await session.scalar(
104 |             select(User)
105 |             .options(selectinload(User.roles))
106 |             .where(User.google_subject == identity.subject)
107 |         )
108 |         if user is None:
109 |             user = await session.scalar(
110 |                 select(User)
111 |                 .options(selectinload(User.roles))
112 |                 .where(User.email == identity.email)
113 |             )
114 |
115 |         roles = _roles_for_email(identity.email)
116 |         if user is None:
117 |             user = User(
118 |                 email=identity.email,
119 |                 full_name=identity.full_name,
120 |                 google_subject=identity.subject,
121 |                 roles=[
122 |                     UserRoleAssignment(role=role)
123 |                     for role in sorted(roles, key=lambda item: item.value)
124 |                 ],
125 |             )
126 |             session.add(user)
127 |             await session.flush()
128 |         else:
129 |             if not user.is_active:
130 |                 raise GoogleAuthError("Обліковий запис деактивовано.")
131 |             user.email = identity.email
132 |             user.full_name = identity.full_name
133 |             user.google_subject = identity.subject
134 |             existing_roles = {assignment.role for assignment in
user.roles}
135 |             if existing_roles:
136 |                 roles |= existing_roles
```

```
137 |             for role in roles - existing_roles:
138 |                 user.roles.append(UserRoleAssignment(role=role))
139 |
140 |     token, expires_in = create_access_token(
141 |         subject=identity.subject,
142 |         roles=roles,
143 |         email=identity.email,
144 |     )
145 |     return token, expires_in, roles
146 |
147 |
148 | def _roles_for_email(email: str) -> set[UserRole]:
149 |     normalized = email.casefold()
150 |     if normalized in settings.google_admin_emails:
151 |         return {UserRole.ADMIN}
152 |     if normalized in settings.google_operator_emails:
153 |         return {UserRole.OPERATOR}
154 |     return {UserRole.CANDIDATE}
155 |
156 |
157 | def _ensure_allowed_domain(email: str, hosted_domain: str | None) -> None:
158 |     allowed_domain = settings.google_allowed_domain
159 |     if not allowed_domain:
160 |         if settings.app_environment.casefold() == "production":
161 |             raise GoogleAuthError("Для production потрібно задати
GOOGLE_ALLOWED_DOMAIN.")
162 |         return
163 |     email_domain = email.rsplit("@", 1)[-1].casefold()
164 |     google_domain = (hosted_domain or "").casefold()
165 |     if email_domain != allowed_domain or (
166 |         google_domain and google_domain != allowed_domain
167 |     ):
168 |         raise GoogleAuthError(
169 |             f"Дозволено вхід лише з корпоративного домену
{allowed_domain}."
170 |         )
171 |
172 |
173 | def _post_form(url: str, payload: dict[str, str]) -> dict[str, Any]:
174 |     data = urllib.parse.urlencode(payload).encode("utf-8")
175 |     request = urllib.request.Request(
176 |         url,
177 |         data=data,
178 |         headers={"Content-Type": "application/x-www-form-urlencoded"},
179 |         method="POST",
180 |     )
181 |     return _open_json(request)
182 |
183 |
184 | def _get_json(url: str) -> dict[str, Any]:
185 |     return _open_json(urllib.request.Request(url, method="GET"))
186 |
187 |
188 | def _open_json(request: urllib.request.Request) -> dict[str, Any]:
189 |     try:
190 |         with urllib.request.urlopen(request, timeout=10) as response:
191 |             return json.loads(response.read().decode("utf-8"))
192 |     except Exception as exc:  # pragma: no cover - depends on Google
availability
193 |         raise GoogleAuthError("Не вдалося перевірити відповідь Google
OAuth.") from exc
```

# ДОДАТОК Б

## Код інтерфейсу користувача та публічних сторінок

У додатку наведено шаблони HTML, клієнтський JavaScript і CSS, що реалізують робочу панель, картку аспіранта, календар графіків, налаштування та публічні сторінки РСВР.

### Лістинг Б.1 - Шаблон робочої панелі оператора (app/templates/dashboard.html)

```
  1 | <!doctype html>
  2 | <html lang="uk">
  3 | <head>
  4 |   <meta charset="utf-8">
  5 |   <meta name="viewport" content="width=device-width, initial-scale=1">
  6 |   <title>Супровід атестації аспірантів</title>
  7 |   <link rel="stylesheet" href="{{ url_for('static', path='styles.css')
}}?v=20260614-public-council-member-details">
  8 | </head>
  9 | <body>
 10 |   <header class="topbar app-topbar">
 11 |     <div>
 12 |       <p class="eyebrow">ІФНТУНГ · відділ аспірантури та докторантури</p>
 13 |       <h1>Супровід атестації аспірантів</h1>
 14 |     </div>
 15 |     <div class="topbar-actions">
 16 |       <span id="authStatus" class="badge">Не авторизовано</span>
 17 |       <a class="button-link secondary"
href="/api/v1/auth/google/login">Корпоративна пошта</a>
 18 |       <button id="getToken" class="secondary">Демо адміністратор</button>
 19 |     </div>
 20 |   </header>
 21 |
 22 |   <main class="layout">
 23 |     <aside class="sidebar">
 24 |       <button class="nav-item active" data-view="overview">Панель</button>
 25 |       <button class="nav-item" data-view="import">Імпорт ЄДЕБО</button>
 26 |       <button class="nav-item" data-view="candidates">Аспіранти</button>
 27 |       <button class="nav-item" data-view="candidate-card">Картка
аспіранта</button>
 28 |       <button class="nav-item" data-view="normative">Контроль
строків</button>
 29 |       <button class="nav-item" data-view="calendar">Календар
захистів</button>
 30 |       <button class="nav-item" data-view="settings">Налаштування</button>
 31 |       <a class="nav-item link" href="/docs">Документація API</a>
 32 |     </aside>
 33 |
 34 |     <section class="content">
 35 |       <div id="diagnostics" class="notice hidden"></div>
 36 |
 37 |       <section id="overview" class="view active">
 38 |         <div class="section-heading">
 39 |           <div>
 40 |             <p class="eyebrow">Оперативний стан</p>
 41 |             <h2>Нормативний супровід захистів доктора філософії</h2>
```

```
 42 |           </div>
 43 |           <button id="refreshWorkspace" class="secondary">Оновити</button>
 44 |         </div>
 45 |
 46 |         <div class="metrics">
 47 |           <article><span>Аспіранти</span><strong id="metricCandidates">—
</strong></article>
 48 |           <article><span>Етапи</span><strong id="metricStages">—
</strong></article>
 49 |           <article><span>Разові ради</span><strong id="metricCouncils">—
</strong></article>
 50 |           <article><span>Версії графіка</span><strong
id="metricSchedules">—</strong></article>
 51 |         </div>
 52 |
 53 |         <div class="dashboard-grid">
 54 |           <article class="panel">
 55 |             <p class="eyebrow">Найближчі дії</p>
 56 |             <h3>Порушення та ризики</h3>
 57 |             <div id="overviewIssues" class="compact-list">Авторизуйтеся та
оновіть дані.</div>
 58 |           </article>
 59 |           <article class="panel">
 60 |             <p class="eyebrow">Робочий сценарій</p>
 61 |             <h3>Що робить оператор</h3>
 62 |             <ol class="steps">
 63 |               <li>Імпортує або оновлює дані з ЄДЕБО.</li>
 64 |               <li>Відкриває картку конкретного аспіранта.</li>
 65 |               <li>Фіксує фактичні дати подій.</li>
 66 |               <li>Контролює нормативні строки та формує захист.</li>
 67 |             </ol>
 68 |           </article>
 69 |         </div>
 70 |       </section>
 71 |
 72 |       <section id="import" class="view">
 73 |         <div class="section-heading">
 74 |           <div>
 75 |             <p class="eyebrow">Вхідні дані</p>
 76 |             <h2>CSV-експорт ЄДЕБО</h2>
 77 |           </div>
 78 |         </div>
 79 |         <p class="lead">Модуль перевіряє структуру, кодування, обов'язкові
поля, дати та дублікати. Опорна дата завершення підготовки не перезаписується
повторним імпортом без явної зміни в картці.</p>
 80 |         <div class="upload-box">
 81 |           <input id="csvFile" type="file" accept=".csv,text/csv">
 82 |           <button id="validateCsv">Перевірити</button>
 83 |           <button id="applyCsv" class="secondary">Застосувати
імпорт</button>
 84 |         </div>
 85 |         <pre id="importResult" class="result">Результат імпорту з'явиться
тут.</pre>
 86 |       </section>
 87 |
 88 |       <section id="candidates" class="view">
 89 |         <div class="section-heading">
 90 |           <div>
 91 |             <p class="eyebrow">Центр роботи системи</p>
 92 |             <h2>Аспіранти</h2>
 93 |           </div>
 94 |           <div class="actions">
```

```
 95 |             <input id="candidateSearch" placeholder="Пошук за ПІБ або
ОНП">
 96 |             <button id="loadCandidates" class="secondary">Оновити</button>
 97 |           </div>
 98 |         </div>
 99 |         <div id="candidateCards" class="card-grid">
100 |           <article class="panel muted-panel">Завантажте список
аспірантів.</article>
101 |         </div>
102 |       </section>
103 |
104 |       <section id="candidate-card" class="view">
105 |         <div class="section-heading">
106 |           <div>
107 |             <p class="eyebrow">Картка аспіранта</p>
108 |             <h2 id="candidateTitle">Оберіть аспіранта</h2>
109 |           </div>
110 |           <div class="actions">
111 |             <select id="candidateQuickSelect">
112 |               <option value="">Оберіть аспіранта</option>
113 |             </select>
114 |             <button id="generatePlans">Сформувати/перерахувати
етапи</button>
115 |             <button id="reloadCandidate" class="secondary">Оновити
картку</button>
116 |           </div>
117 |         </div>
118 |
119 |         <div id="candidateEmpty" class="panel muted-panel">
120 |           Відкрийте вкладку «Аспіранти» і натисніть «Відкрити картку».
121 |         </div>
122 |
123 |         <div id="candidateWorkspace" class="hidden">
124 |           <div class="candidate-hero">
125 |             <div>
126 |               <p class="eyebrow">Профіль</p>
127 |               <h3 id="candidateName">—</h3>
128 |               <p id="candidateMeta" class="lead">—</p>
129 |             </div>
130 |             <div class="progress-box">
131 |               <span>Готовність</span>
132 |               <strong id="candidateProgress">0%</strong>
133 |               <div class="progress-track"><div
id="candidateProgressBar"></div></div>
134 |             </div>
135 |           </div>
136 |
137 |           <div class="detail-grid">
138 |             <article><span>Опорна дата завершення</span><strong
id="candidateStudyEnd">—</strong></article>
139 |             <article><span>Допустиме вікно захисту</span><strong
id="candidateDefenseWindow">—</strong></article>
140 |             <article><span>Поточні зауваження</span><strong
id="candidateIssueCount">—</strong></article>
141 |           </div>
142 |
143 |           <div class="dashboard-grid">
144 |             <article class="panel">
145 |               <p class="eyebrow">Службові дані</p>
146 |               <h3>Тема і керівник</h3>
147 |               <label>Тема дисертації<textarea id="thesisTitle"
rows="4"></textarea></label>
148 |               <label>Науковий керівник<input id="supervisorName"></label>
```

```
149 |               <button id="saveCandidateProfile">Зберегти</button>
150 |             </article>
151 |
152 |             <article class="panel">
153 |               <p class="eyebrow">Нормативний контроль</p>
154 |               <h3>Ризики аспіранта</h3>
155 |               <div id="candidateIssues" class="compact-list">—</div>
156 |             </article>
157 |           </div>
158 |
159 |           <article class="panel">
160 |             <div class="section-heading">
161 |               <div>
162 |                 <p class="eyebrow">20 етапів</p>
163 |                 <h3>Індивідуальна траєкторія атестації</h3>
164 |               </div>
165 |             </div>
166 |             <div id="stageTimeline" class="timeline"></div>
167 |           </article>
168 |
169 |           <div class="dashboard-grid">
170 |             <article class="panel">
171 |               <p class="eyebrow">Разова рада</p>
172 |               <h3>Склад ради</h3>
173 |               <div id="councilMembers" class="compact-list">—</div>
174 |               <h3>Реквізити РСВР для головної сторінки</h3>
175 |               <p class="lead">Ці дані використовуються на публічній
головній сторінці для короткої та повної інформації про утворену разову
спеціалізовану вчену раду.</p>
176 |               <div class="settings-form council-metadata-form">
177 |                 <label>Код ради
178 |                   <input id="councilCode" placeholder="Наприклад: ДФ
20.052.069">
179 |                 </label>
180 |                 <label>Номер наказу
181 |                   <input id="councilOrderNumber" placeholder="Наприклад:
123-р">
182 |                 </label>
183 |                 <label>Дата наказу
184 |                   <input id="councilOrderDate" type="date">
185 |                 </label>
186 |                 <label>Дата оприлюднення
187 |                   <input id="councilPublicationDate" type="date">
188 |                 </label>
189 |                 <label>Посилання на наказ
190 |                   <input id="councilOrderUrl" placeholder="https://...">
191 |                 </label>
192 |                 <label>Повідомлення про утворення ради
193 |                   <input id="councilAnnouncementUrl"
placeholder="https://...">
194 |                 </label>
195 |                 <label>Галузь знань
196 |                   <input id="councilKnowledgeField" value="12 -
Інформаційні технології">
197 |                 </label>
198 |                 <label>Спеціальність
199 |                   <input id="councilSpecialty" value="123 - Комп'ютерна
інженерія">
200 |                 </label>
201 |                 <label>Облікова картка УкрІНТЕІ
202 |                   <input id="ukrinteiCardNumber" placeholder="Наприклад:
0826U001406">
203 |                 </label>
```

```
204 |                 <label>Посилання на УкрІНТЕІ
205 |                   <input id="ukrinteiCardUrl" placeholder="https://...">
206 |                 </label>
207 |                 <label>Посилання НРАТ
208 |                   <input id="nratUrl" placeholder="https://...">
209 |                 </label>
210 |                 <label>ID разової ради NAQA
211 |                   <input id="naqaId" placeholder="Наприклад: PhD 13426">
212 |                 </label>
213 |                 <label>Посилання NAQA
214 |                   <input id="naqaUrl"
placeholder="https://svr.naqa.gov.ua/...">
215 |                 </label>
216 |                 <label>Дата і час захисту
217 |                   <input id="defenseDateTime" type="datetime-local">
218 |                 </label>
219 |                 <label>Аудиторія / місце
220 |                   <input id="defenseRoom" placeholder="Наприклад: ауд.
0314">
221 |                 </label>
222 |                 <label>Платформа
223 |                   <input id="defensePlatform" placeholder="Наприклад:
Zoom">
224 |                 </label>
225 |                 <label>Посилання відеоконференції
226 |                   <input id="conferenceUrl" placeholder="https://...">
227 |                 </label>
228 |                 <label>Ідентифікатор конференції
229 |                   <input id="conferenceId" placeholder="893 9917 7889">
230 |                 </label>
231 |                 <label>Код доступу
232 |                   <input id="conferencePasscode" placeholder="086700">
233 |                 </label>
234 |                 <label>Пряма трансляція
235 |                   <input id="livestreamUrl" placeholder="https://...">
236 |                 </label>
237 |                 <label>Дисертаційна робота
238 |                   <input id="dissertationUrl" placeholder="https://...">
239 |                 </label>
240 |                 <label>КЕП дисертаційної роботи
241 |                   <input id="dissertationSignatureUrl"
placeholder="https://...">
242 |                 </label>
243 |                 <label>Висновок про наукову новизну
244 |                   <input id="noveltyConclusionUrl"
placeholder="https://...">
245 |                 </label>
246 |                 <label>Відеозапис захисту
247 |                   <input id="videoUrl" placeholder="https://...">
248 |                 </label>
249 |                 <label>Рішення РСВР
250 |                   <input id="decisionUrl" placeholder="https://...">
251 |                 </label>
252 |               </div>
253 |               <label class="wide-label">Рецензії та відгуки
254 |                 <textarea id="reviewsText" rows="5" placeholder="Кожен
рядок: ПІБ | Рецензія/Відгук | посилання на файл | посилання на КЕП"></textarea>
255 |               </label>
256 |               <label class="wide-label">Примітка щодо відеозапису /
електронної печатки
257 |                 <textarea id="videoStampNote" rows="3"
placeholder="Наприклад: Відеозапис захисту, на який накладається електронна
печатка ІФНТУНГ..."></textarea>
```

```
258 |               </label>
259 |               <div class="row-actions">
260 |                 <button id="saveCouncilMetadata"
class="secondary">Зберегти реквізити РСВР</button>
261 |               </div>
262 |               <h3>Редагування складу ради</h3>
263 |               <p class="lead">Вкажіть одну голову та один із нормативних
варіантів: 2 рецензенти і 2 офіційні опоненти або 1 рецензент і 3 офіційні
опоненти. Голова й рецензенти мають бути з ІФНТУНГ, офіційні опоненти — із
зовнішніх різних закладів.</p>
264 |               <label class="wide-label">Варіант складу ради
265 |                 <select id="councilVariant">
266 |                   <option value="two-reviewers">2 рецензенти + 2
опоненти</option>
267 |                   <option value="three-opponents">1 рецензент + 3
опоненти</option>
268 |                 </select>
269 |               </label>
270 |               <div id="councilEditor" class="council-editor">
271 |                 <div class="council-member-form" data-council-row="chair">
272 |                   <strong>Голова ради</strong>
273 |                   <input data-council-field="full_name" placeholder="ПІБ">
274 |                   <input data-council-field="institution"
placeholder="Установа" value="ІФНТУНГ">
275 |                   <input data-council-field="academic_degree"
placeholder="Науковий ступінь">
276 |                   <input data-council-field="position" placeholder="Вчене
звання, посада, кафедра">
277 |                   <input data-council-field="profile_url"
placeholder="Посилання на профіль">
278 |                 </div>
279 |                 <div class="council-member-form" data-council-
row="reviewer1">
280 |                   <strong>Рецензент 1</strong>
281 |                   <input data-council-field="full_name" placeholder="ПІБ">
282 |                   <input data-council-field="institution"
placeholder="Установа" value="ІФНТУНГ">
283 |                   <input data-council-field="academic_degree"
placeholder="Науковий ступінь">
284 |                   <input data-council-field="position" placeholder="Вчене
звання, посада, кафедра">
285 |                   <input data-council-field="profile_url"
placeholder="Посилання на профіль">
286 |                 </div>
287 |                 <div class="council-member-form" data-council-
row="reviewer2">
288 |                   <strong>Рецензент 2</strong>
289 |                   <input data-council-field="full_name" placeholder="ПІБ">
290 |                   <input data-council-field="institution"
placeholder="Установа" value="ІФНТУНГ">
291 |                   <input data-council-field="academic_degree"
placeholder="Науковий ступінь">
292 |                   <input data-council-field="position" placeholder="Вчене
звання, посада, кафедра">
293 |                   <input data-council-field="profile_url"
placeholder="Посилання на профіль">
294 |                 </div>
295 |                 <div class="council-member-form" data-council-
row="opponent1">
296 |                   <strong>Офіційний опонент 1</strong>
297 |                   <input data-council-field="full_name" placeholder="ПІБ">
298 |                   <input data-council-field="institution"
placeholder="Зовнішній заклад">
```

```
299 |                   <input data-council-field="academic_degree"
placeholder="Науковий ступінь">
300 |                   <input data-council-field="position" placeholder="Вчене
звання, посада, підрозділ">
301 |                   <input data-council-field="profile_url"
placeholder="Посилання на профіль">
302 |                 </div>
303 |                 <div class="council-member-form" data-council-
row="opponent2">
304 |                   <strong>Офіційний опонент 2</strong>
305 |                   <input data-council-field="full_name" placeholder="ПІБ">
306 |                   <input data-council-field="institution"
placeholder="Інший зовнішній заклад">
307 |                   <input data-council-field="academic_degree"
placeholder="Науковий ступінь">
308 |                   <input data-council-field="position" placeholder="Вчене
звання, посада, підрозділ">
309 |                   <input data-council-field="profile_url"
placeholder="Посилання на профіль">
310 |                 </div>
311 |                 <div class="council-member-form hidden" data-council-
row="opponent3">
312 |                   <strong>Офіційний опонент 3</strong>
313 |                   <input data-council-field="full_name" placeholder="ПІБ">
314 |                   <input data-council-field="institution"
placeholder="Третій зовнішній заклад">
315 |                   <input data-council-field="academic_degree"
placeholder="Науковий ступінь">
316 |                   <input data-council-field="position" placeholder="Вчене
звання, посада, підрозділ">
317 |                   <input data-council-field="profile_url"
placeholder="Посилання на профіль">
318 |                 </div>
319 |               </div>
320 |               <div class="row-actions">
321 |                 <button id="saveCouncilComposition">Зберегти склад
ради</button>
322 |                 <button id="checkCouncilConflicts"
class="secondary">Перевірити накладання</button>
323 |                 <button id="buildCandidateMaterial"
class="secondary">Підготувати повідомлення</button>
324 |               </div>
325 |               <pre id="councilConflictResult" class="result compact-
result">Оберіть або сформуйте активний графік, щоб перевірити накладання членів
рад.</pre>
326 |               <pre id="candidateMaterial" class="result compact-
result"></pre>
327 |             </article>
328 |
329 |             <article class="panel">
330 |               <p class="eyebrow">Документи</p>
331 |               <h3>Зареєстровані матеріали</h3>
332 |               <div id="candidateDocuments" class="compact-list">—</div>
333 |             </article>
334 |           </div>
335 |
336 |           <article class="panel">
337 |             <p class="eyebrow">Планування захисту</p>
338 |             <h3>Графік для обраного аспіранта</h3>
339 |             <p class="lead">Цей блок формує захист для поточного
аспіранта, добирає допустимий час і аудиторію, перевіряє нормативне вікно,
зайнятість аудиторій та накладання роботи членів разової ради.</p>
340 |             <div class="task-grid compact-settings schedule-help-grid">
```

```
341 |               <article><strong>Формування</strong><span>Створює нову
версію графіка за заданою датою, аудиторією і параметрами
планування.</span></article>
342 |               <article><strong>Погодження</strong><span>Фіксує
підготовлену версію графіка перед публікацією.</span></article>
343 |               <article><strong>Перепланування</strong><span>Зберігає
зафіксовані події, вилучає заблоковані і мінімізує кількість
змін.</span></article>
344 |             </div>
345 |             <div class="actions schedule-params">
346 |               <label>Дата початку<input id="scheduleStartDate"
type="date"></label>
347 |               <label>Аудиторія захисту<select
id="preferredRoomName"><option value="">Автоматично</option></select></label>
348 |               <label>Днів планування<input id="dayCount" type="number"
min="1" max="120" value="10"></label>
349 |               <label>Максимум захистів члена ради на день<input
id="maxMemberDefenses" type="number" min="1" max="20" value="4"></label>
350 |               <button id="generateSchedule">Сформувати захист</button>
351 |             </div>
352 |             <div id="candidateSchedules" class="compact-list">—</div>
353 |             <div id="activeScheduleSummary" class="notice subtle-
notice">Активний графік ще не вибрано.</div>
354 |             <div class="workflow-actions">
355 |               <a id="viewSchedule" class="button-link secondary disabled"
aria-disabled="true">Переглянути графік</a>
356 |               <button id="checkActiveSchedule" class="secondary"
disabled>Перевірити конфлікти</button>
357 |               <button id="approveSchedule" class="secondary"
disabled>Погодити</button>
358 |               <button id="publishSchedule" disabled>Опублікувати</button>
359 |             </div>
360 |             <pre id="scheduleConflictResult" class="result compact-
result">Після формування або вибору графіка тут можна переглянути конфлікти
аудиторій, членів ради та нормативного вікна.</pre>
361 |             <h3>Динамічне перепланування</h3>
362 |             <p class="lead">Вкажіть номери подій через кому. Зафіксовані
події залишаються без змін, заблоковані вилучаються з попереднього слота.</p>
363 |             <div class="actions schedule-params">
364 |               <label>Заблоковані події<input id="blockedEvents"
placeholder="Наприклад: 3,5"></label>
365 |               <label>Зафіксовані події<input id="lockedEvents"
placeholder="Наприклад: 1,2"></label>
366 |               <button id="replanSchedule" class="secondary"
disabled>Перепланувати</button>
367 |             </div>
368 |           </article>
369 |         </div>
370 |       </section>
371 |
372 |       <section id="normative" class="view">
373 |         <div class="section-heading">
374 |           <div>
375 |             <p class="eyebrow">Контроль строків</p>
376 |             <h2>Загальний журнал ризиків</h2>
377 |           </div>
378 |           <button id="loadNormative" class="secondary">Оновити
контроль</button>
379 |         </div>
380 |         <div id="normativeResult" class="issue-grid"></div>
381 |       </section>
382 |
383 |       <section id="calendar" class="view">
```

```
384 |         <div class="section-heading">
385 |           <div>
386 |             <p class="eyebrow">Календар</p>
387 |             <h2>Версії графіків захистів</h2>
388 |           </div>
389 |           <button id="loadSchedules" class="secondary">Оновити</button>
390 |         </div>
391 |         <div id="scheduleList" class="card-grid"></div>
392 |         <pre id="conflictResult" class="result">Оберіть або сформуйте
графік, щоб перевірити конфлікти.</pre>
393 |       </section>
394 |
395 |       <section id="settings" class="view">
396 |         <div class="section-heading">
397 |           <div>
398 |             <p class="eyebrow">Налаштування</p>
399 |             <h2>Параметри роботи вебсервісу</h2>
400 |           </div>
401 |           <button id="reloadSettings" class="secondary">Оновити</button>
402 |         </div>
403 |
404 |         <div class="dashboard-grid">
405 |           <article class="panel">
406 |             <p class="eyebrow">Планування засідання РСВР</p>
407 |             <h3>Операційні параметри</h3>
408 |             <div class="settings-form">
409 |               <label>Тривалість засідання, хв
410 |                 <input id="settingDefenseDuration" type="number" min="180"
max="480" step="15">
411 |               </label>
412 |               <label>Днів планування за замовчуванням
413 |                 <input id="settingPlanningDays" type="number" min="1"
max="120">
414 |               </label>
415 |               <label>Макс. захистів члена ради на день
416 |                 <input id="settingMaxMemberDefenses" type="number" min="1"
max="20">
417 |               </label>
418 |               <label>Час початку захистів
419 |                 <input id="settingStartTimes" placeholder="09:00, 12:30,
16:00">
420 |               </label>
421 |             </div>
422 |             <div class="row-actions">
423 |               <button id="saveOperationalSettings">Зберегти
параметри</button>
424 |             </div>
425 |             <p class="lead">Ці значення використовуються під час створення
нових слотів графіка захистів.</p>
426 |           </article>
427 |
428 |           <article class="panel">
429 |             <p class="eyebrow">Авторизація</p>
430 |             <h3>Доступ до системи</h3>
431 |             <div class="task-grid compact-settings">
432 |               <article><strong>Середовище</strong><span
id="settingsEnvironment">—</span></article>
433 |               <article><strong>Демо-токени</strong><span
id="settingsDemoTokens">—</span></article>
434 |               <article><strong>Вхід через Google</strong><span
id="settingsGoogleOauth">—</span></article>
435 |               <article><strong>Домен пошти</strong><span
id="settingsGoogleDomain">—</span></article>
```

```
436 |             </div>
437 |           </article>
438 |         </div>
439 |
440 |         <article class="panel">
441 |           <p class="eyebrow">Аудиторії</p>
442 |           <h3>Керування аудиторіями для захистів</h3>
443 |           <div class="settings-form room-create-form">
444 |             <label>Номер / назва
445 |               <input id="newRoomName" placeholder="Напр. 1401">
446 |             </label>
447 |             <label>Розташування
448 |               <input id="newRoomLocation" placeholder="Корпус, поверх">
449 |             </label>
450 |             <label>Місткість
451 |               <input id="newRoomCapacity" type="number" min="1" max="500"
value="30">
452 |             </label>
453 |             <label>Посилання на трансляцію
454 |               <input id="newRoomOnlineUrl" placeholder="https://...">
455 |             </label>
456 |             <button id="addRoom">Додати аудиторію</button>
457 |           </div>
458 |           <div id="settingsRoomList" class="settings-room-list">
459 |             <div class="muted-panel">Налаштування аудиторій
завантажуються.</div>
460 |           </div>
461 |           <div class="task-grid compact-settings">
462 |             <article><strong>Активні аудиторії</strong><span
id="settingsRooms">0314, 0509, 1214</span></article>
463 |             <article><strong>Використання</strong><span>Аудиторію з
історією графіків можна вимкнути, але не потрібно фізично
видаляти.</span></article>
464 |           </div>
465 |         </article>
466 |       </section>
467 |     </section>
468 |   </main>
469 |
470 |   {% if capture_import_report %}
471 |   <script id="captureImportReport" type="application/json">{{
capture_import_report | tojson }}</script>
472 |   {% endif %}
473 |   <script src="{{ url_for('static', path='app.js') }}?v=20260614-import-
auth-refresh"></script>
474 | </body>
475 | </html>
```

## Лістинг Б.2 - Шаблон публічної головної сторінки РСВР (app/templates/public_home.html)

```
  1 | <!doctype html>
  2 | <html lang="uk">
  3 | <head>
  4 |   <meta charset="utf-8">
  5 |   <meta name="viewport" content="width=device-width, initial-scale=1">
  6 |   <title>Разові спеціалізовані вчені ради</title>
  7 |   <link rel="stylesheet" href="{{ url_for('static', path='styles.css')
}}?v=20260614-public-council-member-details">
  8 | </head>
```

```
  9 | <body class="public-page public-home-page">
 10 |   <main class="public-home">
 11 |     <header class="public-hero">
 12 |       <div>
 13 |         <p class="eyebrow">ІФНТУНГ · атестація здобувачів доктора
філософії</p>
 14 |         <h1>Разові спеціалізовані вчені ради</h1>
 15 |         <p class="lead">На цій сторінці подано відомості про утворені
РСВР. Коротка інформація відображається у картці, повний склад і реквізити
відкриваються через розкривний блок.</p>
 16 |       </div>
 17 |       <a class="button-link secondary" href="/ui">Робоча панель</a>
 18 |     </header>
 19 |
 20 |     {% if council_load_error %}
 21 |       <section class="public-empty warning">
 22 |         <h2>Дані РСВР тимчасово недоступні</h2>
 23 |         <p>Сторінку завантажено, але систему не вдалося під'єднати до бази
даних. Після відновлення підключення відомості про утворені ради з'являться
автоматично.</p>
 24 |       </section>
 25 |     {% elif councils %}
 26 |       <section class="public-council-grid">
 27 |         {% for council in councils %}
 28 |         <article class="public-council-card">
 29 |           <div class="public-council-head">
 30 |             <div>
 31 |               <p class="eyebrow">{{ council.council_code }} · {{
council.status }}</p>
 32 |               <h2>Разова спеціалізована вчена рада {{ council.council_code
}}</h2>
 33 |             </div>
 34 |             <span class="status-pill ok">{{ council.member_count }} членів
ради</span>
 35 |           </div>
 36 |
 37 |           <div class="public-council-summary">
 38 |             <p><strong>Здобувач:</strong> {{ council.candidate_name }}</p>
 39 |             <p><strong>Спеціальність:</strong> {{ council.specialty }}</p>
 40 |             <p><strong>Назва дисертації:</strong> {{ council.thesis_title
}}</p>
 41 |             <p><strong>Наказ:</strong>
 42 |               {% if council.order_number or council.order_date %}
 43 |                 {% if council.order_url %}<a href="{{ council.order_url
}}" target="_blank" rel="noopener">{% endif %}
 44 |                 № {{ council.order_number or "номер не вказано" }}{% if
council.order_date %} від {{ council.order_date }}{% endif %}
 45 |                 {% if council.order_url %}</a>{% endif %}
 46 |               {% else %}
 47 |                 реквізити наказу не вказано
 48 |               {% endif %}
 49 |             </p>
 50 |           </div>
 51 |
 52 |           <details class="public-council-details">
 53 |             <summary>Повна інформація про РСВР</summary>
 54 |             <div class="public-council-full">
 55 |               <section class="public-statement">
 56 |                 <h3>Разова спеціалізована вчена рада {{
council.council_code }}</h3>
 57 |                 <p>з правом прийняття до розгляду та проведення разового
захисту дисертації здобувача {{ council.candidate_name_genitive }} на здобуття
```

```
ступеня доктора філософії з галузі знань {{ council.knowledge_field }} за
спеціальністю {{ council.specialty }}.</p>
 58 |                </section>
 59 |
 60 |                <div class="public-facts">
 61 |                  <p><strong>Здобувач:</strong> {{ council.candidate_name
}}</p>
 62 |                  <p><strong>Структурний підрозділ:</strong> {{
council.department }}</p>
 63 |                  <p><strong>Освітньо-наукова програма:</strong> {{
council.program }}</p>
 64 |                  <p><strong>Тема дисертації:</strong> {{
council.thesis_title }}</p>
 65 |                  <p><strong>Науковий керівник:</strong> {{
council.supervisor_name }}</p>
 66 |                  <p><strong>Дата оприлюднення:</strong> {{
council.publication_date or "не вказано" }}</p>
 67 |                </div>
 68 |
 69 |                <h3>Голова та члени ради</h3>
 70 |                <div class="public-member-list">
 71 |                  {% for member in council.members %}
 72 |                  <div class="public-member-row">
 73 |                    <strong>{{ member.role }}:
 74 |                      {% if member.profile_url %}
 75 |                        <a href="{{ member.profile_url }}" target="_blank"
rel="noopener">{{ member.full_name }}</a>
 76 |                      {% else %}
 77 |                        {{ member.full_name }}
 78 |                      {% endif %}
 79 |                    </strong>
 80 |                    <span>{{ member.institution }}</span>
 81 |                    <small>
 82 |                      {{ member.academic_degree or "науковий ступінь не
вказано" }}
 83 |                      {% if member.position %} · {{ member.position }}{%
endif %}
 84 |                      {% if member.specialty %} · {{ member.specialty }}{%
endif %}
 85 |                    </small>
 86 |                  </div>
 87 |                  {% endfor %}
 88 |                </div>
 89 |
 90 |                <section class="public-registry-block">
 91 |                  <h3>Повідомлення про утворення разової ради</h3>
 92 |                  {% if council.announcement_url %}
 93 |                    <p><a href="{{ council.announcement_url }}"
target="_blank" rel="noopener">Повідомлення про утворення разової ради</a></p>
 94 |                  {% endif %}
 95 |                  <p><strong>Облікова картка УкрІНТЕІ:</strong>
 96 |                    {% if council.ukrintei_card_number and
council.ukrintei_card_url %}
 97 |                      <a href="{{ council.ukrintei_card_url }}"
target="_blank" rel="noopener">{{ council.ukrintei_card_number }}</a>
 98 |                    {% else %}
 99 |                      {{ council.ukrintei_card_number or "не вказано" }}
100 |                    {% endif %}
101 |                  </p>
102 |                  <p><strong>Текст дисертації на сайті НРАТ
УкрІНТЕІ:</strong>
```

```
103 |                     {% if council.nrat_url %}<a href="{{ council.nrat_url
}}" target="_blank" rel="noopener">{{ council.nrat_url }}</a>{% else %}не
вказано{% endif %}
104 |                   </p>
105 |                   <p><strong>ID разової ради:</strong>
106 |                     {% if council.naqa_id and council.naqa_url %}
107 |                       <a href="{{ council.naqa_url }}" target="_blank"
rel="noopener">{{ council.naqa_id }}</a>
108 |                     {% else %}
109 |                       {{ council.naqa_id or "не вказано" }}
110 |                     {% endif %}
111 |                   </p>
112 |                 </section>
113 |
114 |                 <section class="public-broadcast-block">
115 |                   <h3>Посилання на вебсайт, де здійснюватиметься трансляція
захисту дисертації</h3>
116 |                   <p>Захист дисертаційної роботи {{
council.candidate_name_genitive }}{% if council.defense_datetime_text %}
відбудеться {{ council.defense_datetime_text }}{% endif %}{% if
council.defense_platform %} в режимі відеоконференції на платформі {{
council.defense_platform }}{% endif %}{% if council.defense_room %} в {{
council.defense_room }}{% endif %}.</p>
117 |                   <p><strong>Посилання для підключення:</strong>
118 |                     {% if council.conference_url %}<a href="{{
council.conference_url }}" target="_blank" rel="noopener">{{
council.conference_url }}</a>{% else %}не вказано{% endif %}
119 |                   </p>
120 |                   <p><strong>Ідентифікатор конференції:</strong> {{
council.conference_id or "не вказано" }}</p>
121 |                   <p><strong>Код доступу:</strong> {{
council.conference_passcode or "не вказано" }}</p>
122 |                   <p><strong>Пряма трансляція:</strong>
123 |                     {% if council.livestream_url %}<a href="{{
council.livestream_url }}" target="_blank" rel="noopener">{{
council.livestream_url }}</a>{% else %}не вказано{% endif %}
124 |                   </p>
125 |                 </section>
126 |
127 |                 <section class="public-files-block">
128 |                   <h3>Приєднані файли</h3>
129 |                   <p><strong>Дисертація:</strong>
130 |                     {% if council.dissertation_url %}<a href="{{
council.dissertation_url }}" target="_blank" rel="noopener">Дисертаційна
робота</a>{% else %}не вказано{% endif %}
131 |                     {% if council.dissertation_signature_url %} | <a
href="{{ council.dissertation_signature_url }}" target="_blank"
rel="noopener">КЕП дисертаційної роботи</a>{% endif %}
132 |                   </p>
133 |                   <p><strong>Висновок про наукову новизну:</strong>
134 |                     {% if council.novelty_conclusion_url %}<a href="{{
council.novelty_conclusion_url }}" target="_blank" rel="noopener">Висновок</a>{%
else %}не вказано{% endif %}
135 |                   </p>
136 |
137 |                   {% if council.reviews %}
138 |                   <h4>Рецензії та відгуки</h4>
139 |                   <div class="public-review-list">
140 |                     {% for review in council.reviews %}
141 |                     <p>
142 |                       <strong>{{ review.member_name }} ({{ review.kind
}}):</strong>
```

```
143 |                       {% if review.material_url %}<a href="{{
review.material_url }}" target="_blank" rel="noopener">{{ review.kind }}</a>{%
else %}файл не вказано{% endif %}
144 |                       {% if review.signature_url %} | <a href="{{
review.signature_url }}" target="_blank" rel="noopener">КЕП</a>{% endif %}
145 |                     </p>
146 |                     {% endfor %}
147 |                   </div>
148 |                   {% endif %}
149 |                 </section>
150 |
151 |                 <section class="public-decision-block">
152 |                   <h3>Відеозапис та рішення разової спеціалізованої вченої
ради {{ council.council_code }}</h3>
153 |                   <p><strong>Відеозапис захисту:</strong>
154 |                     {% if council.video_url %}<a href="{{ council.video_url
}}" target="_blank" rel="noopener">Відеозапис захисту дисертації</a>{% else %}не
вказано{% endif %}
155 |                   </p>
156 |                   <p><strong>Рішення РСВР:</strong>
157 |                     {% if council.decision_url %}<a href="{{
council.decision_url }}" target="_blank" rel="noopener">Рішення разової
спеціалізованої вченої ради</a>{% else %}не вказано{% endif %}
158 |                   </p>
159 |                   {% if council.video_stamp_note %}
160 |                   <p>{{ council.video_stamp_note }}</p>
161 |                   {% endif %}
162 |                 </section>
163 |               </div>
164 |             </details>
165 |           </article>
166 |           {% endfor %}
167 |         </section>
168 |       {% else %}
169 |         <section class="public-empty">
170 |           <h2>Утворених РСВР ще не оприлюднено</h2>
171 |           <p>Після заповнення складу ради та реквізитів у картці аспіранта
інформація з'явиться на цій сторінці.</p>
172 |         </section>
173 |       {% endif %}
174 |     </main>
175 | </body>
176 | </html>
```

## Лістинг Б.3 - Шаблон публічного перегляду графіка (app/templates/public_schedule.html)

```
 1 | <!doctype html>
 2 | <html lang="uk">
 3 | <head>
 4 |   <meta charset="utf-8">
 5 |   <meta name="viewport" content="width=device-width, initial-scale=1">
 6 |   <title>Опублікований розклад захистів</title>
 7 |   <link rel="stylesheet" href="{{ url_for('static', path='styles.css')
}}?v=20260614-planning-ua">
 8 | </head>
 9 | <body class="public-page">
10 |   {% set state_labels = {
11 |     "draft": "чернетка",
12 |     "approved": "погоджено",
```

```
13 |     "published": "опубліковано",
14 |     "cancelled": "скасовано"
15 |   } %}
16 |   <main class="public-card">
17 |     <p class="eyebrow">Офіційна публічна версія</p>
18 |     <h1>Розклад захистів</h1>
19 |     <p class="lead">Версія {{ schedule.id }}. Стан: {{
state_labels.get(schedule.state.value, "невідомо") }}.</p>
20 |     <div class="table-wrap">
21 |       <table>
22 |         <thead>
23 |           <tr><th>Здобувач</th><th>Дата і час</th><th>Аудиторія</th></tr>
24 |         </thead>
25 |         <tbody>
26 |           {% for item in schedule.result.assignments %}
27 |           <tr>
28 |             <td>{{ item.candidate_name or item.candidate_id }}</td>
29 |             <td>{{ item.slot_label or item.slot_id }}</td>
30 |             <td>{{ item.room_name or item.room_id }}</td>
31 |           </tr>
32 |           {% endfor %}
33 |         </tbody>
34 |       </table>
35 |     </div>
36 |   </main>
37 | </body>
38 | </html>
```

## Лістинг Б.4 - Шаблон службового перегляду графіка (app/templates/schedule_view.html)

```
  1 | <!doctype html>
  2 | <html lang="uk">
  3 | <head>
  4 |   <meta charset="utf-8">
  5 |   <meta name="viewport" content="width=device-width, initial-scale=1">
  6 |   <title>Перегляд графіка захистів</title>
  7 |   <link rel="stylesheet" href="{{ url_for('static', path='styles.css')
}}?v=20260614-planning-ua">
  8 | </head>
  9 | <body>
 10 |   {% set environment_labels = {
 11 |     "development": "розробка",
 12 |     "testing": "тестування",
 13 |     "test": "тестування",
 14 |     "staging": "підготовче середовище",
 15 |     "production": "робоче середовище"
 16 |   } %}
 17 |   <header class="topbar">
 18 |     <div>
 19 |       <p class="eyebrow">Робочий перегляд</p>
 20 |       <h1>Графік захистів</h1>
 21 |     </div>
 22 |     <div class="environment">Середовище: {{
environment_labels.get(app_environment, "інше середовище") }}</div>
 23 |   </header>
 24 |
 25 |   <main class="page">
 26 |     <div class="schedule-toolbar">
```

```
 27 |       <a class="button-link secondary" href="/ui">Повернутися до
панелі</a>
 28 |       <button id="refreshSchedule" class="secondary">Оновити дані</button>
 29 |       <a id="publicScheduleLink" class="button-link disabled" aria-
disabled="true">Публічна версія недоступна</a>
 30 |       <button id="deleteSchedule" class="secondary danger">Видалити
версію</button>
 31 |     </div>
 32 |
 33 |     <section class="panel">
 34 |       <div class="section-heading">
 35 |         <div>
 36 |           <p class="eyebrow">Версія розкладу</p>
 37 |           <h2 id="scheduleTitle">Завантаження...</h2>
 38 |         </div>
 39 |         <span id="scheduleStateBadge" class="badge">—</span>
 40 |       </div>
 41 |
 42 |       <div id="scheduleNotice" class="notice hidden"></div>
 43 |
 44 |       <div class="metrics">
 45 |         <article><span>Стан задачі</span><strong id="solverStatus">—
</strong></article>
 46 |         <article><span>Кількість захистів</span><strong
id="assignmentCount">—</strong></article>
 47 |         <article><span>Оцінка</span><strong id="scheduleScore">—
</strong></article>
 48 |         <article><span>Час, мс</span><strong id="scheduleTime">—
</strong></article>
 49 |       </div>
 50 |
 51 |       <div class="detail-grid">
 52 |         <article>
 53 |           <span>Створено</span>
 54 |           <strong id="createdAt">—</strong>
 55 |         </article>
 56 |         <article>
 57 |           <span>Погоджено</span>
 58 |           <strong id="approvedAt">—</strong>
 59 |         </article>
 60 |         <article>
 61 |           <span>Опубліковано</span>
 62 |           <strong id="publishedAt">—</strong>
 63 |         </article>
 64 |       </div>
 65 |
 66 |       <div class="table-wrap">
 67 |         <table>
 68 |           <thead>
 69 |             <tr>
 70 |               <th>№</th>
 71 |               <th>Здобувач</th>
 72 |               <th>Дата і час</th>
 73 |               <th>Аудиторія</th>
 74 |               <th>Фіксація</th>
 75 |             </tr>
 76 |           </thead>
 77 |           <tbody id="scheduleRows">
 78 |             <tr><td colspan="5" class="empty">Дані графіка
завантажуються.</td></tr>
 79 |           </tbody>
 80 |         </table>
 81 |       </div>
```

```
 82 |     </section>
 83 |   </main>
 84 |
 85 |   <script>
 86 |     const scheduleId = {{ schedule_id | tojson }};
 87 |     const token = localStorage.getItem("attestationAccessToken") || "";
 88 |
 89 |     const stateLabels = {
 90 |       draft: "Чернетка",
 91 |       approved: "Погоджено",
 92 |       published: "Опубліковано",
 93 |       cancelled: "Скасовано"
 94 |     };
 95 |
 96 |     const solverStatusLabels = {
 97 |       OPTIMAL: "Сформовано без конфліктів",
 98 |       FEASIBLE: "Сформовано з обмеженнями",
 99 |       INFEASIBLE: "Не вдалося сформувати",
100 |       UNKNOWN: "Стан розрахунку невідомий"
101 |     };
102 |
103 |     document.getElementById("refreshSchedule").addEventListener("click",
loadSchedule);
104 |     document.getElementById("deleteSchedule").addEventListener("click",
deleteSchedule);
105 |
106 |     async function loadSchedule() {
107 |       hideNotice();
108 |       if (!token) {
109 |         showNotice("Немає активного токена доступу. Поверніться до панелі,
отримайте роль адміністратора і відкрийте графік ще раз.");
110 |         return;
111 |       }
112 |
113 |       const response = await fetch(`/api/v1/schedules/${scheduleId}`, {
114 |         headers: {"Authorization": `Bearer ${token}`}
115 |       });
116 |       const data = await response.json();
117 |       if (!response.ok) {
118 |         showNotice(data.detail || "Не вдалося завантажити графік.");
119 |         return;
120 |       }
121 |       renderSchedule(data);
122 |     }
123 |
124 |     function renderSchedule(data) {
125 |       document.getElementById("scheduleTitle").textContent = `Графік №
${data.id}`;
126 |       document.getElementById("scheduleStateBadge").textContent =
stateLabels[data.state] || "Невідомо";
127 |       document.getElementById("solverStatus").textContent =
solverStatusLabels[data.result.status] || "Невідомий стан розрахунку";
128 |       document.getElementById("assignmentCount").textContent =
data.result.assignments.length;
129 |       document.getElementById("scheduleScore").textContent =
data.result.score;
130 |       document.getElementById("scheduleTime").textContent =
data.result.elapsed_ms;
131 |       document.getElementById("createdAt").textContent =
formatDateTime(data.created_at);
132 |       document.getElementById("approvedAt").textContent =
formatDateTime(data.approved_at);
```

```
133 |       document.getElementById("publishedAt").textContent =
formatDateTime(data.published_at);
134 |
135 |       document.getElementById("scheduleRows").innerHTML =
data.result.assignments.map((item, index) => `
136 |         <tr>
137 |           <td>${index + 1}</td>
138 |           <td>${escapeHtml(item.candidate_name || item.candidate_id)}</td>
139 |           <td>${escapeHtml(item.slot_label || item.slot_id)}</td>
140 |           <td>${escapeHtml(item.room_name || item.room_id)}</td>
141 |           <td>${item.locked ? "Так" : "Ні"}</td>
142 |         </tr>
143 |       `).join("");
144 |
145 |       const publicLink = document.getElementById("publicScheduleLink");
146 |       if (data.state === "published") {
147 |         publicLink.textContent = "Відкрити публічну версію";
148 |         publicLink.href = `/public/schedules/${data.id}`;
149 |         publicLink.classList.remove("disabled");
150 |         publicLink.removeAttribute("aria-disabled");
151 |       } else {
152 |         publicLink.textContent = "Публічна версія недоступна";
153 |         publicLink.classList.add("disabled");
154 |         publicLink.setAttribute("aria-disabled", "true");
155 |         publicLink.removeAttribute("href");
156 |       }
157 |     }
158 |
159 |     async function deleteSchedule() {
160 |       hideNotice();
161 |       if (!token) {
162 |         showNotice("Немає активного токена доступу. Поверніться до панелі,
отримайте роль адміністратора і відкрийте графік ще раз.");
163 |         return;
164 |       }
165 |       if (!confirm(`Видалити версію графіка №${scheduleId}? Події цієї
версії також буде видалено.`)) return;
166 |
167 |       const response = await fetch(`/api/v1/schedules/${scheduleId}`, {
168 |         method: "DELETE",
169 |         headers: {"Authorization": `Bearer ${token}`}
170 |       });
171 |       const data = await response.json();
172 |       if (!response.ok) {
173 |         showNotice(data.detail || "Не вдалося видалити графік.");
174 |         return;
175 |       }
176 |       window.location.replace("/ui");
177 |     }
178 |
179 |     function showNotice(message) {
180 |       const notice = document.getElementById("scheduleNotice");
181 |       notice.textContent = message;
182 |       notice.classList.remove("hidden");
183 |     }
184 |
185 |     function hideNotice() {
186 |       const notice = document.getElementById("scheduleNotice");
187 |       notice.textContent = "";
188 |       notice.classList.add("hidden");
189 |     }
190 |
191 |     function formatDateTime(value) {
```

```
192 |       if (!value) return "—";
193 |       return new Intl.DateTimeFormat("uk-UA", {
194 |         dateStyle: "medium",
195 |         timeStyle: "short"
196 |       }).format(new Date(value));
197 |     }
198 |
199 |     function escapeHtml(value) {
200 |       const element = document.createElement("div");
201 |       element.textContent = value;
202 |       return element.innerHTML;
203 |     }
204 |
205 |     loadSchedule().catch(error => showNotice(error.message));
206 |   </script>
207 | </body>
208 | </html>
```

## Лістинг Б.5 - Клієнтська логіка web-інтерфейсу (app/static/app.js)

```
  1 | const state = {
  2 |   token: localStorage.getItem("attestationAccessToken") || "",
  3 |   candidates: [],
  4 |   selectedCandidateId: null,
  5 |   candidateDetail: null,
  6 |   scheduleId: "",
  7 |   schedules: [],
  8 |   issues: [],
  9 |   rooms: [],
 10 |   settings: null,
 11 |   tokenRefreshPromise: null
 12 | };
 13 |
 14 | const $ = (id) => document.getElementById(id);
 15 |
 16 | const statusLabels = {
 17 |   planned: "заплановано",
 18 |   completed: "виконано",
 19 |   overdue: "прострочено",
 20 |   needs_revision: "потребує перегляду",
 21 |   in_progress: "у роботі"
 22 | };
 23 |
 24 | const normativeLabels = {
 25 |   ok: "вчасно",
 26 |   pending: "очікує",
 27 |   risk: "ризик",
 28 |   overdue: "прострочено",
 29 |   violation: "порушено"
 30 | };
 31 |
 32 | const roleLabels = {
 33 |   chair: "Голова ради",
 34 |   reviewer: "Рецензент",
 35 |   opponent: "Офіційний опонент"
 36 | };
 37 |
 38 | const councilStatusLabels = {
 39 |   draft: "проєкт",
 40 |   formed: "сформовано",
 41 |   active: "активна",
```

```
  42 |   completed: "завершена",
  43 |   cancelled: "скасована"
  44 | };
  45 |
  46 | const scheduleStateLabels = {
  47 |   draft: "чернетка",
  48 |   approved: "погоджено",
  49 |   published: "опубліковано",
  50 |   cancelled: "скасовано"
  51 | };
  52 |
  53 | const solverStatusLabels = {
  54 |   OPTIMAL: "сформовано без конфліктів",
  55 |   FEASIBLE: "сформовано з обмеженнями",
  56 |   INFEASIBLE: "не вдалося сформувати",
  57 |   UNKNOWN: "стан розрахунку невідомий"
  58 | };
  59 |
  60 | const severityLabels = {
  61 |   info: "інформація",
  62 |   warning: "попередження",
  63 |   error: "помилка",
  64 |   success: "успішно"
  65 | };
  66 |
  67 | const environmentLabels = {
  68 |   development: "розробка",
  69 |   testing: "тестування",
  70 |   test: "тестування",
  71 |   staging: "підготовче середовище",
  72 |   production: "робоче середовище"
  73 | };
  74 |
  75 | const accessRoleLabels = {
  76 |   admin: "адміністратор",
  77 |   operator: "оператор відділу аспірантури",
  78 |   candidate: "аспірант",
  79 |   supervisor: "науковий керівник",
  80 |   council_member: "член разової ради"
  81 | };
  82 |
  83 | const councilRows = [
  84 |   {key: "chair", role: "chair", label: "Голова ради", index: 0,
isExternal: false, variants: ["two-reviewers", "three-opponents"]},
  85 |   {key: "reviewer1", role: "reviewer", label: "Рецензент 1", index: 0,
isExternal: false, variants: ["two-reviewers", "three-opponents"]},
  86 |   {key: "reviewer2", role: "reviewer", label: "Рецензент 2", index: 1,
isExternal: false, variants: ["two-reviewers"]},
  87 |   {key: "opponent1", role: "opponent", label: "Офіційний опонент 1",
index: 0, isExternal: true, variants: ["two-reviewers", "three-opponents"]},
  88 |   {key: "opponent2", role: "opponent", label: "Офіційний опонент 2",
index: 1, isExternal: true, variants: ["two-reviewers", "three-opponents"]},
  89 |   {key: "opponent3", role: "opponent", label: "Офіційний опонент 3",
index: 2, isExternal: true, variants: ["three-opponents"]}
  90 | ];
  91 |
  92 | const homeInstitution = "ІФНТУНГ";
  93 |
  94 | document.querySelectorAll(".nav-item[data-view]").forEach((button) => {
  95 |   button.addEventListener("click", () =>
activateView(button.dataset.view));
  96 | });
  97 |
```

```
  98 | $("getToken").addEventListener("click", getDemoToken);
  99 | $("refreshWorkspace").addEventListener("click", refreshAll);
 100 | $("validateCsv").addEventListener("click", () => {
 101 |   validateCsv().catch((error) => showError(error.message));
 102 | });
 103 | $("applyCsv").addEventListener("click", () => {
 104 |   applyCsv().catch((error) => showError(error.message));
 105 | });
 106 | $("loadCandidates").addEventListener("click", loadCandidates);
 107 | $("candidateSearch").addEventListener("input", renderCandidateCards);
 108 | $("candidateCards").addEventListener("click", (event) => {
 109 |   handleCandidateCardAction(event).catch((error) =>
showError(error.message));
 110 | });
 111 | $("candidateQuickSelect").addEventListener("change", () => {
 112 |   const candidateId = Number($("candidateQuickSelect").value);
 113 |   if (candidateId) openCandidate(candidateId);
 114 | });
 115 | $("generatePlans").addEventListener("click", generatePlans);
 116 | $("reloadCandidate").addEventListener("click", () => {
 117 |   if (state.selectedCandidateId) return
openCandidate(state.selectedCandidateId);
 118 |   showError("Спочатку оберіть аспіранта.");
 119 | });
 120 | $("saveCandidateProfile").addEventListener("click",
saveCandidateProfile);
 121 | $("loadNormative").addEventListener("click", loadNormative);
 122 | $("loadSchedules").addEventListener("click", loadSchedules);
 123 | $("reloadSettings").addEventListener("click", loadSettings);
 124 | $("saveOperationalSettings").addEventListener("click",
saveOperationalSettings);
 125 | $("addRoom").addEventListener("click", addRoom);
 126 | $("settingsRoomList").addEventListener("click",
handleSettingsRoomAction);
 127 | $("generateSchedule").addEventListener("click", generateSchedule);
 128 | $("checkActiveSchedule").addEventListener("click",
checkActiveScheduleConflicts);
 129 | $("approveSchedule").addEventListener("click", approveSchedule);
 130 | $("publishSchedule").addEventListener("click", publishSchedule);
 131 | $("replanSchedule").addEventListener("click", replanSchedule);
 132 | $("buildCandidateMaterial").addEventListener("click",
buildCandidateMaterial);
 133 | $("saveCouncilComposition").addEventListener("click",
saveCouncilComposition);
 134 | $("saveCouncilMetadata").addEventListener("click", saveCouncilMetadata);
 135 | $("checkCouncilConflicts").addEventListener("click",
checkCouncilConflicts);
 136 | $("councilVariant").addEventListener("change", renderCouncilVariant);
 137 | $("stageTimeline").addEventListener("click", handleStageAction);
 138 | $("scheduleList").addEventListener("click", handleScheduleListAction);
 139 | $("candidateSchedules").addEventListener("click",
handleCandidateScheduleAction);
 140 | document.addEventListener("click", (event) => {
 141 |   if (!event.target.closest(".dropdown")) closeScheduleMenus();
 142 | });
 143 |
 144 | if (state.token) {
 145 |   markAuthenticated();
 146 |   refreshAll().catch((error) => showError(error.message));
 147 | }
 148 | runCaptureMode().catch((error) => showError(error.message));
 149 |
 150 | function activateView(viewId) {
```

```
 151 |   document.querySelectorAll(".nav-item").forEach((item) =>
item.classList.remove("active"));
 152 |   document.querySelectorAll(".view").forEach((item) =>
item.classList.remove("active"));
 153 |   const nav = document.querySelector(`.nav-item[data-view="${viewId}"]`);
 154 |   if (nav) nav.classList.add("active");
 155 |   const view = $(viewId);
 156 |   if (view) view.classList.add("active");
 157 | }
 158 |
 159 | async function getDemoToken() {
 160 |   try {
 161 |     await issueDemoToken();
 162 |     showInfo("Демо-доступ оновлено.");
 163 |     await refreshAll();
 164 |   } catch (error) {
 165 |     showError(error.message);
 166 |   }
 167 | }
 168 |
 169 | async function issueDemoToken() {
 170 |   const response = await fetch("/api/v1/auth/demo-token", {
 171 |     method: "POST",
 172 |     headers: {"Content-Type": "application/json"},
 173 |     body: JSON.stringify({role: "admin"})
 174 |   });
 175 |   const data = await response.json().catch(() => ({}));
 176 |   if (!response.ok) throw new Error(data.detail || "Не вдалося отримати
демо-токен.");
 177 |   state.token = data.access_token;
 178 |   localStorage.setItem("attestationAccessToken", state.token);
 179 |   localStorage.setItem("attestationRoles", JSON.stringify(["admin"]));
 180 |   localStorage.setItem("attestationTokenExpiresIn",
String(data.expires_in || ""));
 181 |   localStorage.setItem("attestationTokenIssuedAt", new
Date().toISOString());
 182 |   markAuthenticated();
 183 | }
 184 |
 185 | async function refreshDemoTokenSilently() {
 186 |   if (state.tokenRefreshPromise) return state.tokenRefreshPromise;
 187 |   state.tokenRefreshPromise = issueDemoToken()
 188 |     .then(() => true)
 189 |     .catch(() => false)
 190 |     .finally(() => {
 191 |       state.tokenRefreshPromise = null;
 192 |     });
 193 |   return state.tokenRefreshPromise;
 194 | }
 195 |
 196 | function markAuthenticated() {
 197 |   const roles = localStorage.getItem("attestationRoles");
 198 |   $("authStatus").textContent = roles ? `Активні ролі:
${accessRoleSummary(roles)}` : "Авторизовано";
 199 |   $("authStatus").classList.add("success");
 200 | }
 201 |
 202 | async function refreshAll() {
 203 |   if (!ensureToken()) return;
 204 |   await Promise.allSettled([
 205 |     refreshWorkspace(),
 206 |     loadCandidates(),
 207 |     loadNormative(),
```

```
 208 |     loadSchedules(),
 209 |     loadSettings()
 210 |   ]);
 211 |   if (state.selectedCandidateId) await
openCandidate(state.selectedCandidateId, false);
 212 | }
 213 |
 214 | async function refreshWorkspace() {
 215 |   const data = await getJson("/api/v1/workspace");
 216 |   $("metricCandidates").textContent = data.candidates;
 217 |   $("metricStages").textContent = data.stages;
 218 |   $("metricCouncils").textContent = data.councils;
 219 |   $("metricSchedules").textContent = data.schedules;
 220 |   state.rooms = data.rooms.length ? data.rooms : ["0314", "0509",
"1214"];
 221 |   $("settingsRooms").textContent = state.rooms.join(", ");
 222 |   renderRoomSelect();
 223 | }
 224 |
 225 | async function loadSettings() {
 226 |   if (!ensureToken()) return;
 227 |   const data = await getJson("/api/v1/settings");
 228 |   state.settings = data;
 229 |   state.rooms = data.active_rooms.length ? data.active_rooms : ["0314",
"0509", "1214"];
 230 |   renderSettings(data);
 231 |   renderRoomSelect();
 232 | }
 233 |
 234 | async function loadCandidates() {
 235 |   if (!ensureToken()) return;
 236 |   const data = await getJson("/api/v1/candidates");
 237 |   state.candidates = data.items;
 238 |   renderCandidateQuickSelect();
 239 |   renderCandidateCards();
 240 | }
 241 |
 242 | function renderCandidateQuickSelect() {
 243 |   const select = $("candidateQuickSelect");
 244 |   const current = select.value || (state.selectedCandidateId ?
String(state.selectedCandidateId) : "");
 245 |   select.innerHTML = state.candidates.length
 246 |     ? `<option value="">Оберіть аспіранта</option>` +
state.candidates.map((item) => (
 247 |       `<option value="${item.id}">${item.id}:
${escapeHtml(item.full_name)}</option>`
 248 |     )).join("")
 249 |     : `<option value="">Аспірантів у базі немає</option>`;
 250 |   if (current && state.candidates.some((item) => String(item.id) ===
current)) {
 251 |     select.value = current;
 252 |   }
 253 | }
 254 |
 255 | function renderCandidateCards() {
 256 |   const query = $("candidateSearch").value.trim().toLowerCase();
 257 |   const items = state.candidates.filter((item) => {
 258 |     const haystack = `${item.full_name} ${item.educational_program || ""}
${item.department || ""}`.toLowerCase();
 259 |     return !query || haystack.includes(query);
 260 |   });
 261 |   $("candidateCards").innerHTML = items.length
 262 |     ? items.map((item) => `
```

```
 263 |       <article class="candidate-card ${state.selectedCandidateId ===
item.id ? "selected" : ""}">
 264 |         <div>
 265 |           <p class="eyebrow">Картка №${item.id} ·
${escapeHtml(item.study_status)}</p>
 266 |           <h3>${escapeHtml(item.full_name)}</h3>
 267 |           <p>${escapeHtml(item.educational_program || "ОНП не
вказана")}</p>
 268 |           <p class="muted">Завершення підготовки:
${formatDate(item.study_end)}</p>
 269 |         </div>
 270 |         <div class="candidate-card-footer">
 271 |           <span>${item.stage_count} етапів · рада:
${escapeHtml(councilStatusLabel(item.council_status))}</span>
 272 |           <div class="row-actions">
 273 |             <button data-open-candidate="${item.id}">Відкрити
картку</button>
 274 |             <button class="secondary danger" data-delete-
candidate="${item.id}">Видалити</button>
 275 |           </div>
 276 |         </div>
 277 |       </article>
 278 |     `).join("")
 279 |     : `<article class="panel muted-panel">Аспірантів не
знайдено.</article>`;
 280 | }
 281 |
 282 | async function handleCandidateCardAction(event) {
 283 |   const openButton = event.target.closest("[data-open-candidate]");
 284 |   const deleteButton = event.target.closest("[data-delete-candidate]");
 285 |   if (openButton) {
 286 |     event.preventDefault();
 287 |     return openCandidate(Number(openButton.dataset.openCandidate));
 288 |   }
 289 |   if (deleteButton) {
 290 |     event.preventDefault();
 291 |     return deleteCandidate(Number(deleteButton.dataset.deleteCandidate));
 292 |   }
 293 | }
 294 |
 295 | async function openCandidate(candidateId, switchView = true) {
 296 |   if (!ensureToken()) return;
 297 |   const detail = await getJson(`/api/v1/candidates/${candidateId}`);
 298 |   state.selectedCandidateId = candidateId;
 299 |   state.candidateDetail = detail;
 300 |   $("candidateQuickSelect").value = String(candidateId);
 301 |   renderCandidateDetail(detail);
 302 |   renderCandidateCards();
 303 |   if (switchView) activateView("candidate-card");
 304 | }
 305 |
 306 | function renderCandidateDetail(detail) {
 307 |   $("candidateEmpty").classList.add("hidden");
 308 |   $("candidateWorkspace").classList.remove("hidden");
 309 |   $("candidateTitle").textContent = detail.candidate.full_name;
 310 |   $("candidateName").textContent = detail.candidate.full_name;
 311 |   $("candidateMeta").textContent = [
 312 |     detail.candidate.educational_program || "ОНП не вказана",
 313 |     detail.candidate.department || "кафедру не вказано",
 314 |     detail.study_form || "форма навчання не вказана"
 315 |   ].join(" · ");
 316 |   $("candidateProgress").textContent = `${detail.progress_percent}%`;
 317 |   $("candidateProgressBar").style.width = `${detail.progress_percent}%`;
```

```
 318 |   $("candidateStudyEnd").textContent =
formatDate(detail.candidate.study_end);
 319 |   $("candidateDefenseWindow").textContent = detail.defense_window_start
&& detail.defense_window_end
 320 |     ? `${formatDate(detail.defense_window_start)} -
${formatDate(detail.defense_window_end)}`
 321 |     : "потрібно сформувати/уточнити етапи";
 322 |   $("candidateIssueCount").textContent = detail.issues.length;
 323 |   $("thesisTitle").value = detail.thesis_title || "";
 324 |   $("supervisorName").value = detail.supervisor_name || "";
 325 |   const publicDetails = detail.council_public_details || {};
 326 |   $("councilCode").value = publicDetails.council_code || "";
 327 |   $("councilOrderNumber").value = detail.council_order_number || "";
 328 |   $("councilOrderDate").value = detail.council_order_date || "";
 329 |   $("councilPublicationDate").value = detail.council_publication_date ||
"";
 330 |   $("councilOrderUrl").value = publicDetails.order_url || "";
 331 |   $("councilAnnouncementUrl").value = publicDetails.announcement_url ||
"";
 332 |   $("councilKnowledgeField").value = publicDetails.knowledge_field || "12
– Інформаційні технології";
 333 |   $("councilSpecialty").value = publicDetails.specialty || "123 –
Комп'ютерна інженерія";
 334 |   $("ukrinteiCardNumber").value = publicDetails.ukrintei_card_number ||
"";
 335 |   $("ukrinteiCardUrl").value = publicDetails.ukrintei_card_url || "";
 336 |   $("nratUrl").value = publicDetails.nrat_url || "";
 337 |   $("naqaId").value = publicDetails.naqa_id || "";
 338 |   $("naqaUrl").value = publicDetails.naqa_url || "";
 339 |   $("defenseDateTime").value =
dateTimeLocalValue(publicDetails.defense_datetime);
 340 |   $("defenseRoom").value = publicDetails.defense_room || "";
 341 |   $("defensePlatform").value = publicDetails.defense_platform || "Zoom";
 342 |   $("conferenceUrl").value = publicDetails.conference_url || "";
 343 |   $("conferenceId").value = publicDetails.conference_id || "";
 344 |   $("conferencePasscode").value = publicDetails.conference_passcode ||
"";
 345 |   $("livestreamUrl").value = publicDetails.livestream_url || "";
 346 |   $("dissertationUrl").value = publicDetails.dissertation_url || "";
 347 |   $("dissertationSignatureUrl").value =
publicDetails.dissertation_signature_url || "";
 348 |   $("noveltyConclusionUrl").value = publicDetails.novelty_conclusion_url
|| "";
 349 |   $("videoUrl").value = publicDetails.video_url || "";
 350 |   $("decisionUrl").value = publicDetails.decision_url || "";
 351 |   $("reviewsText").value = reviewsToText(publicDetails.reviews || []);
 352 |   $("videoStampNote").value = publicDetails.video_stamp_note || "";
 353 |   if (detail.defense_window_start && !$("scheduleStartDate").value) {
 354 |     $("scheduleStartDate").value = detail.defense_window_start;
 355 |   }
 356 |   if (!detail.schedules.some((item) => item.schedule_id ===
state.scheduleId)) {
 357 |     state.scheduleId = detail.schedules[0]?.schedule_id || "";
 358 |   }
 359 |   if (state.scheduleId) {
 360 |     const activeSchedule = detail.schedules.find((item) =>
item.schedule_id === state.scheduleId);
 361 |     renderActiveScheduleSummary(activeSchedule);
 362 |   } else {
 363 |     renderActiveScheduleSummary(null);
 364 |   }
 365 |   renderCandidateIssues(detail.issues);
 366 |   renderStageTimeline(detail.stages);
```

```
  renderCouncilMembers(detail.council_members);
  renderCouncilEditor(detail.council_members);
  renderCandidateDocuments(detail.documents);
  renderCandidateSchedules(detail.schedules);
}

function renderCandidateIssues(issues) {
  $("candidateIssues").innerHTML = issues.length
    ? issues.map((item) => `
      <div class="issue-item ${escapeHtml(item.severity)}">
        <strong>${escapeHtml(item.title)}</strong>
        <span>${escapeHtml(item.message)}</span>
      </div>
    `).join("")
    : `<div class="issue-item info"><strong>Порушень не виявлено</strong><span>Поточні етапи вкладаються в нормативні строки.</span></div>`;
}

function renderStageTimeline(stages) {
  $("stageTimeline").innerHTML = stages.length
    ? stages.map((stage, index) => `
      <article class="timeline-item ${escapeHtml(stage.normative_status)}">
        <div class="timeline-index">${index + 1}</div>
        <div class="timeline-body">
          <div class="timeline-head">
            <div>
              <strong>${escapeHtml(stage.title)}</strong>
              <span>${escapeHtml(stage.responsible || "відповідального не вказано")}</span>
            </div>
            <span class="status-pill ${escapeHtml(stage.normative_status)}">${escapeHtml(normativeLabels[stage.normative_status] || "невідомо")}</span>
          </div>
          <div class="timeline-dates">
            <span>План: ${formatDate(stage.planned_date)}</span>
            <span>Факт: ${formatDate(stage.actual_date)}</span>
            <span>Гранично: ${formatDate(stage.deadline_date)}</span>
            <span>Стан: ${escapeHtml(statusLabels[stage.status] || "невідомо")}</span>
          </div>
          <p>${escapeHtml(stage.normative_message || stage.source_reference || "")}</p>
          <div class="stage-actions">
            <input id="actual-${stage.id}" type="date" value="${escapeHtml(stage.actual_date || "")}">
            <button class="secondary" data-fix-stage="${stage.id}">Зафіксувати подію</button>
            <input id="doc-name-${stage.id}" placeholder="Назва документа">
            <input id="doc-url-${stage.id}" placeholder="Посилання або службовий ключ">
            <button class="secondary" data-register-doc="${stage.id}">Додати документ</button>
          </div>
        </div>
      </article>
    `).join("")
    : `<div class="muted-panel">Етапи ще не сформовано. Натисніть «Сформувати/перерахувати етапи».</div>`;
}
```

```
 416 |
 417 | function renderCouncilMembers(members) {
 418 |   $("councilMembers").innerHTML = members.length
 419 |     ? members.map((item) => `
 420 |       <div class="list-row">
 421 |         <strong>${escapeHtml(roleLabels[item.role] || "Член ради")}:
${item.profile_url ? `<a href="${escapeHtml(item.profile_url)}" target="_blank"
rel="noopener">${escapeHtml(item.full_name)}</a>` :
escapeHtml(item.full_name)}</strong>
 422 |         <span>${escapeHtml(item.institution)} ·
${escapeHtml(item.academic_degree || "ступінь не вказано")}${item.position ? ` ·
${escapeHtml(item.position)}` : ""}</span>
 423 |         <small>Кваліфікація: ${item.qualification_confirmed ? "так" :
"ні"}; конфлікт інтересів відсутній: ${item.conflict_of_interest_absent ? "так"
: "ні"}</small>
 424 |       </div>
 425 |     `).join("")
 426 |     : `<div class="muted-panel">Разову раду ще не сформовано.</div>`;
 427 | }
 428 |
 429 | function renderRoomSelect() {
 430 |   const select = $("preferredRoomName");
 431 |   const current = select.value;
 432 |   select.innerHTML = `<option value="">Автоматично</option>` +
state.rooms.map((room) => (
 433 |     `<option value="${escapeHtml(room)}">${escapeHtml(room)}</option>`
 434 |   )).join("");
 435 |   if (current && state.rooms.includes(current)) {
 436 |     select.value = current;
 437 |   }
 438 | }
 439 |
 440 | function renderSettings(data) {
 441 |   $("settingDefenseDuration").value =
data.operational.defense_duration_minutes;
 442 |   $("settingPlanningDays").value =
data.operational.default_planning_days;
 443 |   $("settingMaxMemberDefenses").value =
data.operational.default_max_member_defenses_per_day;
 444 |   $("settingStartTimes").value =
data.operational.defense_start_times.join(", ");
 445 |   $("dayCount").value = data.operational.default_planning_days;
 446 |   $("maxMemberDefenses").value =
data.operational.default_max_member_defenses_per_day;
 447 |   $("settingsRooms").textContent = data.active_rooms.length ?
data.active_rooms.join(", ") : "Немає активних аудиторій";
 448 |   $("settingsEnvironment").textContent =
environmentLabel(data.app_environment);
 449 |   $("settingsDemoTokens").textContent = data.demo_tokens_enabled ?
"доступні" : "вимкнені";
 450 |   $("settingsGoogleOauth").textContent = data.google_oauth_configured ?
"налаштовано" : "не налаштовано";
 451 |   $("settingsGoogleDomain").textContent = data.google_allowed_domain ||
"не обмежено";
 452 |   renderSettingsRooms(data.rooms);
 453 | }
 454 |
 455 | function renderSettingsRooms(rooms) {
 456 |   $("settingsRoomList").innerHTML = rooms.length
 457 |     ? rooms.map((room) => `
 458 |       <div class="settings-room-row ${room.is_active ? "" : "inactive"}"
data-room-id="${room.id}">
```

```
 459 |         <input data-room-field="name" value="${escapeHtml(room.name)}"
aria-label="Назва аудиторії">
 460 |         <input data-room-field="location"
value="${escapeHtml(room.location || "")}" placeholder="Розташування" aria-
label="Розташування">
 461 |         <input data-room-field="capacity" type="number" min="1" max="500"
value="${room.capacity}" aria-label="Місткість">
 462 |         <input data-room-field="online_url"
value="${escapeHtml(room.online_url || "")}" placeholder="https://..." aria-
label="Посилання на трансляцію">
 463 |         <label class="inline-check"><input data-room-field="is_active"
type="checkbox" ${room.is_active ? "checked" : ""}> активна</label>
 464 |         <span class="room-usage">подій:
${room.schedule_event_count}</span>
 465 |         <div class="row-actions">
 466 |           <button class="secondary" data-save-
room="${room.id}">Зберегти</button>
 467 |           <button class="secondary danger" data-disable-room="${room.id}"
${room.is_active ? "" : "disabled"}>Вимкнути</button>
 468 |         </div>
 469 |       </div>
 470 |     `).join("")
 471 |     : `<div class="muted-panel">Аудиторій ще не додано.</div>`;
 472 | }
 473 |
 474 | async function saveOperationalSettings() {
 475 |   const payload = {
 476 |     defense_duration_minutes: Number($("settingDefenseDuration").value),
 477 |     default_planning_days: Number($("settingPlanningDays").value),
 478 |     default_max_member_defenses_per_day:
Number($("settingMaxMemberDefenses").value),
 479 |     defense_start_times: $("settingStartTimes").value
 480 |       .split(",")
 481 |       .map((item) => item.trim())
 482 |       .filter(Boolean)
 483 |   };
 484 |   const data = await patchJson("/api/v1/settings/operational", payload);
 485 |   state.settings = data;
 486 |   state.rooms = data.active_rooms;
 487 |   renderSettings(data);
 488 |   renderRoomSelect();
 489 |   showInfo("Операційні параметри планування збережено.");
 490 | }
 491 |
 492 | async function addRoom() {
 493 |   const payload = {
 494 |     name: $("newRoomName").value.trim(),
 495 |     location: $("newRoomLocation").value.trim() || null,
 496 |     capacity: Number($("newRoomCapacity").value),
 497 |     online_url: $("newRoomOnlineUrl").value.trim() || null,
 498 |     is_active: true
 499 |   };
 500 |   if (!payload.name) return showError("Вкажіть номер або назву
аудиторії.");
 501 |   await postJson("/api/v1/settings/rooms", payload);
 502 |   $("newRoomName").value = "";
 503 |   $("newRoomLocation").value = "";
 504 |   $("newRoomCapacity").value = "30";
 505 |   $("newRoomOnlineUrl").value = "";
 506 |   showInfo("Аудиторію додано.");
 507 |   await loadSettings();
 508 |   await refreshWorkspace();
 509 | }
```

```
 510 |
 511 | async function handleSettingsRoomAction(event) {
 512 |   const saveId = event.target.dataset.saveRoom;
 513 |   const disableId = event.target.dataset.disableRoom;
 514 |   if (saveId) return saveRoom(Number(saveId));
 515 |   if (disableId) return disableRoom(Number(disableId));
 516 | }
 517 |
 518 | async function saveRoom(roomId) {
 519 |   const row = document.querySelector(`[data-room-id="${roomId}"]`);
 520 |   const payload = collectRoomPayload(row);
 521 |   await patchJson(`/api/v1/settings/rooms/${roomId}`, payload);
 522 |   showInfo("Аудиторію оновлено.");
 523 |   await loadSettings();
 524 |   await refreshWorkspace();
 525 | }
 526 |
 527 | async function disableRoom(roomId) {
 528 |   if (!confirm("Вимкнути аудиторію для нових графіків захистів?"))
return;
 529 |   await deleteJson(`/api/v1/settings/rooms/${roomId}`);
 530 |   showInfo("Аудиторію вимкнено для нових графіків.");
 531 |   await loadSettings();
 532 |   await refreshWorkspace();
 533 | }
 534 |
 535 | function collectRoomPayload(row) {
 536 |   return {
 537 |     name: row.querySelector('[data-room-field="name"]').value.trim(),
 538 |     location: row.querySelector('[data-room-
field="location"]').value.trim() || null,
 539 |     capacity: Number(row.querySelector('[data-room-
field="capacity"]').value),
 540 |     online_url: row.querySelector('[data-room-
field="online_url"]').value.trim() || null,
 541 |     is_active: row.querySelector('[data-room-field="is_active"]').checked
 542 |   };
 543 | }
 544 |
 545 | function activeCouncilRows() {
 546 |   const variant = $("councilVariant").value || "two-reviewers";
 547 |   return councilRows.filter((row) => row.variants.includes(variant));
 548 | }
 549 |
 550 | function inferCouncilVariant(members) {
 551 |   const reviewers = members.filter((item) => item.role ===
"reviewer").length;
 552 |   const opponents = members.filter((item) => item.role ===
"opponent").length;
 553 |   if (reviewers === 1 && opponents === 3) return "three-opponents";
 554 |   return "two-reviewers";
 555 | }
 556 |
 557 | function renderCouncilVariant() {
 558 |   const activeKeys = new Set(activeCouncilRows().map((row) => row.key));
 559 |   councilRows.forEach((row) => {
 560 |     const container = document.querySelector(`[data-council-
row="${row.key}"]`);
 561 |     if (!container) return;
 562 |     const active = activeKeys.has(row.key);
 563 |     container.classList.toggle("hidden", !active);
 564 |     container.querySelectorAll("input").forEach((input) => {
 565 |       input.disabled = !active;
```

```
 566 |     });
 567 |     if (active) applyCouncilRoleDefaults(row, container);
 568 |   });
 569 | }
 570 |
 571 | function applyCouncilRoleDefaults(row, container) {
 572 |   const institution = container.querySelector('[data-council-
field="institution"]');
 573 |   if (row.role === "chair" || row.role === "reviewer") {
 574 |     institution.value = homeInstitution;
 575 |     institution.readOnly = true;
 576 |     return;
 577 |   }
 578 |   institution.readOnly = false;
 579 |   if (normalizeInstitution(institution.value) ===
normalizeInstitution(homeInstitution)) {
 580 |     institution.value = "";
 581 |   }
 582 | }
 583 |
 584 | function renderCouncilEditor(members) {
 585 |   $("councilVariant").value = inferCouncilVariant(members);
 586 |   const grouped = {
 587 |     chair: members.filter((item) => item.role === "chair"),
 588 |     reviewer: members.filter((item) => item.role === "reviewer"),
 589 |     opponent: members.filter((item) => item.role === "opponent")
 590 |   };
 591 |   councilRows.forEach((row) => {
 592 |     const member = grouped[row.role][row.index] || null;
 593 |     const container = document.querySelector(`[data-council-
row="${row.key}"]`);
 594 |     if (!container) return;
 595 |     const fullName = container.querySelector('[data-council-
field="full_name"]');
 596 |     const institution = container.querySelector('[data-council-
field="institution"]');
 597 |     const degree = container.querySelector('[data-council-
field="academic_degree"]');
 598 |     const position = container.querySelector('[data-council-
field="position"]');
 599 |     const profileUrl = container.querySelector('[data-council-
field="profile_url"]');
 600 |     fullName.value = member?.full_name || "";
 601 |     institution.value = member?.institution || (row.role === "opponent" ?
"" : homeInstitution);
 602 |     degree.value = member?.academic_degree || "кандидат/доктор наук";
 603 |     position.value = member?.position || "";
 604 |     profileUrl.value = member?.profile_url || "";
 605 |   });
 606 |   renderCouncilVariant();
 607 | }
 608 |
 609 | function renderCandidateDocuments(documents) {
 610 |   $("candidateDocuments").innerHTML = documents.length
 611 |     ? documents.map((item) => `
 612 |       <div class="list-row">
 613 |         <strong>${escapeHtml(item.original_name)}</strong>
 614 |         <span>${escapeHtml(item.stage_title)}</span>
 615 |         <small>${escapeHtml(item.storage_key)}</small>
 616 |       </div>
 617 |     `).join("")
 618 |     : `<div class="muted-panel">Документи ще не зареєстровано.</div>`;
 619 | }
```

```
 620 |
 621 | function renderCandidateSchedules(schedules) {
 622 |   $("candidateSchedules").innerHTML = schedules.length
 623 |     ? schedules.map((item) => `
 624 |       <div class="list-row">
 625 |         <strong>Графік №${item.schedule_id}:
${formatDateTime(item.starts_at)} - ${formatTime(item.ends_at)}</strong>
 626 |         <span>Аудиторія ${escapeHtml(item.room_name)} · стан:
${escapeHtml(scheduleStateLabel(item.state))} · подія №${item.event_id}</span>
 627 |         <div class="row-actions">
 628 |           <button class="secondary" data-select-candidate-
schedule="${item.schedule_id}">Зробити активним</button>
 629 |           <a class="button-link secondary"
href="/ui/schedules/${item.schedule_id}">Переглянути</a>
 630 |           <button class="secondary danger" data-delete-candidate-
schedule="${item.schedule_id}">Видалити графік</button>
 631 |         </div>
 632 |       </div>
 633 |     `).join("")
 634 |     : `<div class="muted-panel">Захист для цього аспіранта ще не
заплановано.</div>`;
 635 | }
 636 |
 637 | function renderActiveScheduleSummary(schedule) {
 638 |   if (!schedule) {
 639 |     $("activeScheduleSummary").textContent = "Активний графік ще не
вибрано.";
 640 |     $("approveSchedule").disabled = true;
 641 |     $("checkActiveSchedule").disabled = true;
 642 |     $("publishSchedule").disabled = true;
 643 |     $("replanSchedule").disabled = true;
 644 |     $("scheduleConflictResult").textContent = "Після формування або
вибору графіка тут можна переглянути конфлікти аудиторій, членів ради та
нормативного вікна.";
 645 |     $("viewSchedule").classList.add("disabled");
 646 |     $("viewSchedule").setAttribute("aria-disabled", "true");
 647 |     $("viewSchedule").removeAttribute("href");
 648 |     return;
 649 |   }
 650 |   $("activeScheduleSummary").textContent = `Активний графік
№${schedule.schedule_id}: ${formatDateTime(schedule.starts_at)} -
${formatTime(schedule.ends_at)}, аудиторія ${schedule.room_name}, стан:
${scheduleStateLabel(schedule.state)}.`;
 651 |   $("approveSchedule").disabled = schedule.state !== "draft";
 652 |   $("checkActiveSchedule").disabled = false;
 653 |   $("publishSchedule").disabled = schedule.state !== "approved";
 654 |   $("replanSchedule").disabled = false;
 655 |   $("viewSchedule").href = `/ui/schedules/${schedule.schedule_id}`;
 656 |   $("viewSchedule").classList.remove("disabled");
 657 |   $("viewSchedule").removeAttribute("aria-disabled");
 658 | }
 659 |
 660 | async function saveCandidateProfile() {
 661 |   if (!state.selectedCandidateId) return showError("Спочатку оберіть
аспіранта.");
 662 |   const detail = await
patchJson(`/api/v1/candidates/${state.selectedCandidateId}`, {
 663 |     thesis_title: $("thesisTitle").value.trim() || null,
 664 |     supervisor_name: $("supervisorName").value.trim() || null
 665 |   });
 666 |   state.candidateDetail = detail;
 667 |   renderCandidateDetail(detail);
 668 |   showInfo("Службові дані картки збережено.");
```

```
 669 | }
 670 |
 671 | async function saveCouncilComposition() {
 672 |   if (!state.selectedCandidateId) return showError("Спочатку оберіть
аспіранта.");
 673 |   const members = activeCouncilRows().map((row) => {
 674 |     const container = document.querySelector(`[data-council-
row="${row.key}"]`);
 675 |     const fullName = container.querySelector('[data-council-
field="full_name"]').value.trim();
 676 |     const institution = container.querySelector('[data-council-
field="institution"]').value.trim();
 677 |     const degree = container.querySelector('[data-council-
field="academic_degree"]').value.trim();
 678 |     const position = container.querySelector('[data-council-
field="position"]').value.trim();
 679 |     const profileUrl = container.querySelector('[data-council-
field="profile_url"]').value.trim();
 680 |     return {
 681 |       role: row.role,
 682 |       full_name: fullName,
 683 |       institution: institution || (row.role === "opponent" ? "" :
homeInstitution),
 684 |       academic_degree: degree || "кандидат/доктор наук",
 685 |       specialty: "Комп'ютерна інженерія",
 686 |       position: position || null,
 687 |       profile_url: profileUrl || null,
 688 |       is_external: row.isExternal,
 689 |       qualification_confirmed: true,
 690 |       conflict_of_interest_absent: true
 691 |     };
 692 |   });
 693 |   const missing = members.filter((item) => !item.full_name);
 694 |   if (missing.length) return showError("Заповніть ПІБ усіх членів разової
ради.");
 695 |   const missingInstitution = members.filter((item) => !item.institution);
 696 |   if (missingInstitution.length) return showError("Заповніть установи для
всіх членів разової ради.");
 697 |   const invalidInternal = members.filter((item) => (
 698 |     (item.role === "chair" || item.role === "reviewer")
 699 |     && normalizeInstitution(item.institution) !==
normalizeInstitution(homeInstitution)
 700 |   ));
 701 |   if (invalidInternal.length) return showError("Голова ради та рецензенти
можуть бути тільки з ІФНТУНГ.");
 702 |   const opponents = members.filter((item) => item.role === "opponent");
 703 |   if (opponents.some((item) => normalizeInstitution(item.institution) ===
normalizeInstitution(homeInstitution))) {
 704 |     return showError("Офіційні опоненти не можуть бути з ІФНТУНГ.");
 705 |   }
 706 |   const opponentInstitutions = opponents.map((item) =>
normalizeInstitution(item.institution));
 707 |   if (opponentInstitutions.length !== new Set(opponentInstitutions).size)
{
 708 |     return showError("Офіційні опоненти не можуть бути з одного
закладу.");
 709 |   }
 710 |
 711 |   const detail = await
putJson(`/api/v1/candidates/${state.selectedCandidateId}/council`, {members});
 712 |   state.candidateDetail = detail;
 713 |   renderCandidateDetail(detail);
```

```
  714 |   showInfo("Склад разової ради збережено. Під час формування графіка ці
члени ради будуть враховані для перевірки накладань.");
  715 | }
  716 |
  717 | async function saveCouncilMetadata() {
  718 |   if (!state.selectedCandidateId) return showError("Спочатку оберіть
аспіранта.");
  719 |   const detail = await
patchJson(`/api/v1/candidates/${state.selectedCandidateId}/council/metadata`, {
  720 |     council_code: emptyToNull($("councilCode").value),
  721 |     order_number: $("councilOrderNumber").value.trim() || null,
  722 |     order_date: $("councilOrderDate").value || null,
  723 |     publication_date: $("councilPublicationDate").value || null,
  724 |     order_url: emptyToNull($("councilOrderUrl").value),
  725 |     announcement_url: emptyToNull($("councilAnnouncementUrl").value),
  726 |     knowledge_field: emptyToNull($("councilKnowledgeField").value),
  727 |     specialty: emptyToNull($("councilSpecialty").value),
  728 |     ukrintei_card_number: emptyToNull($("ukrinteiCardNumber").value),
  729 |     ukrintei_card_url: emptyToNull($("ukrinteiCardUrl").value),
  730 |     nrat_url: emptyToNull($("nratUrl").value),
  731 |     naqa_id: emptyToNull($("naqaId").value),
  732 |     naqa_url: emptyToNull($("naqaUrl").value),
  733 |     defense_datetime: $("defenseDateTime").value || null,
  734 |     defense_room: emptyToNull($("defenseRoom").value),
  735 |     defense_platform: emptyToNull($("defensePlatform").value),
  736 |     conference_url: emptyToNull($("conferenceUrl").value),
  737 |     conference_id: emptyToNull($("conferenceId").value),
  738 |     conference_passcode: emptyToNull($("conferencePasscode").value),
  739 |     livestream_url: emptyToNull($("livestreamUrl").value),
  740 |     dissertation_url: emptyToNull($("dissertationUrl").value),
  741 |     dissertation_signature_url:
emptyToNull($("dissertationSignatureUrl").value),
  742 |     novelty_conclusion_url: emptyToNull($("noveltyConclusionUrl").value),
  743 |     video_url: emptyToNull($("videoUrl").value),
  744 |     decision_url: emptyToNull($("decisionUrl").value),
  745 |     video_stamp_note: emptyToNull($("videoStampNote").value),
  746 |     reviews: parseReviewsText($("reviewsText").value)
  747 |   });
  748 |   state.candidateDetail = detail;
  749 |   renderCandidateDetail(detail);
  750 |   showInfo("Реквізити РСВР збережено. Дані оновляться на головній
сторінці.");
  751 | }
  752 |
  753 | async function checkCouncilConflicts() {
  754 |   if (!state.scheduleId) {
  755 |     $("councilConflictResult").textContent = "Спочатку сформуйте або
оберіть активний графік.";
  756 |     return showError("Спочатку сформуйте або оберіть активний графік.");
  757 |   }
  758 |   const data = await
getJson(`/api/v1/schedules/${state.scheduleId}/conflicts`);
  759 |   const memberConflicts = data.items.filter((item) => (
  760 |     item.message.includes("Член разової ради")
  761 |     || item.message.includes("членів рад")
  762 |   ));
  763 |   $("councilConflictResult").textContent = memberConflicts.length
  764 |     ? memberConflicts.map(formatConflictItem).join("\n")
  765 |     : "Накладань роботи членів разових рад в активному графіку не
виявлено.";
  766 | }
  767 |
  768 | async function generatePlans() {
```

```
  769 |   if (!ensureToken()) return;
  770 |   const data = await postJson("/api/v1/attestation/plans/generate", {});
  771 |   showInfo(`Опрацьовано аспірантів: ${data.candidates_processed};
створено етапів: ${data.stages_created}; додано правил:
${data.rules_created}.`);
  772 |   await loadCandidates();
  773 |   if (state.selectedCandidateId) await
openCandidate(state.selectedCandidateId, false);
  774 |   await loadNormative();
  775 | }
  776 |
  777 | async function handleStageAction(event) {
  778 |   const fixId = event.target.dataset.fixStage;
  779 |   const docId = event.target.dataset.registerDoc;
  780 |   if (fixId) return fixStageActualDate(Number(fixId));
  781 |   if (docId) return registerStageDocument(Number(docId));
  782 | }
  783 |
  784 | async function fixStageActualDate(stageId) {
  785 |   const value = $(`actual-${stageId}`).value;
  786 |   if (!value) return showError("Вкажіть фактичну дату події.");
  787 |   await postJson(`/api/v1/stages/${stageId}/actual-date`, {actual_date:
value});
  788 |   showInfo("Фактичну дату зафіксовано, залежні етапи перераховано.");
  789 |   await openCandidate(state.selectedCandidateId, false);
  790 |   await loadNormative();
  791 | }
  792 |
  793 | async function registerStageDocument(stageId) {
  794 |   const name = $(`doc-name-${stageId}`).value.trim();
  795 |   const url = $(`doc-url-${stageId}`).value.trim();
  796 |   if (!name) return showError("Вкажіть назву документа.");
  797 |   await postJson(`/api/v1/stages/${stageId}/documents`, {
  798 |     original_name: name,
  799 |     external_url: url || null
  800 |   });
  801 |   showInfo("Документ зареєстровано в картці аспіранта.");
  802 |   await openCandidate(state.selectedCandidateId, false);
  803 | }
  804 |
  805 | async function generateSchedule() {
  806 |   if (!state.selectedCandidateId) return showError("Спочатку оберіть
аспіранта.");
  807 |   const payload = {
  808 |     candidate_id: state.selectedCandidateId,
  809 |     candidate_limit: 1,
  810 |     day_count: Number($("dayCount").value),
  811 |     max_defenses_per_member_per_day: Number($("maxMemberDefenses").value)
  812 |   };
  813 |   const startDate = $("scheduleStartDate").value;
  814 |   if (startDate) payload.start_date = startDate;
  815 |   const roomName = $("preferredRoomName").value;
  816 |   if (roomName) payload.room_name = roomName;
  817 |   const data = await postJson("/api/v1/schedules/generate", payload);
  818 |   state.scheduleId = data.id;
  819 |   showInfo(`Сформовано графік №${data.id} для обраного аспіранта.`);
  820 |   renderScheduleActions(data);
  821 |   await openCandidate(state.selectedCandidateId, false);
  822 |   await loadSchedules();
  823 | }
  824 |
  825 | function renderScheduleActions(schedule) {
  826 |   state.scheduleId = schedule.id;
```

```
  $("approveSchedule").disabled = schedule.state !== "draft";
  $("checkActiveSchedule").disabled = false;
  $("publishSchedule").disabled = schedule.state !== "approved";
  $("replanSchedule").disabled = false;
  $("viewSchedule").href = `/ui/schedules/${schedule.id}`;
  $("viewSchedule").classList.remove("disabled");
  $("viewSchedule").removeAttribute("aria-disabled");
  const firstAssignment = schedule.result.assignments[0];
  const place = firstAssignment
    ? ` ${firstAssignment.slot_label || "час не визначено"}, аудиторія
${firstAssignment.room_name || firstAssignment.room_id}.`
    : "";
  $("activeScheduleSummary").textContent = `Активний графік
№${schedule.id}. Стан: ${scheduleStateLabel(schedule.state)}; подій:
${schedule.result.assignments.length}.${place}`;
}

async function checkActiveScheduleConflicts() {
  if (!state.scheduleId) {
    $("scheduleConflictResult").textContent = "Спочатку сформуйте або
оберіть активний графік.";
    return showError("Спочатку сформуйте або оберіть активний графік.");
  }
  const data = await
getJson(`/api/v1/schedules/${state.scheduleId}/conflicts`);
  $("scheduleConflictResult").textContent = data.items.length
    ? data.items.map(formatConflictItem).join("\n")
    : "Конфліктів для активного графіка не виявлено.";
  showInfo("Перевірку конфліктів активного графіка виконано.");
}

async function approveSchedule() {
  if (!state.scheduleId) return showError("Спочатку сформуйте графік.");
  const data = await
postJson(`/api/v1/schedules/${state.scheduleId}/approve`, {});
  renderScheduleActions(data);
  showInfo("Графік погоджено.");
  await loadSchedules();
}

async function publishSchedule() {
  if (!state.scheduleId) return showError("Спочатку сформуйте графік.");
  const data = await
postJson(`/api/v1/schedules/${state.scheduleId}/publish`, {});
  renderScheduleActions(data);
  showInfo("Графік опубліковано.");
  await loadSchedules();
}

async function replanSchedule() {
  if (!state.scheduleId) return showError("Спочатку сформуйте або оберіть
активний графік.");
  const data = await
postJson(`/api/v1/schedules/${state.scheduleId}/replan`, {
    blocked_event_ids: parseIds($("blockedEvents").value),
    locked_event_ids: parseIds($("lockedEvents").value),
    room_name: $("preferredRoomName").value || null,
    day_count: Number($("dayCount").value),
    max_defenses_per_member_per_day: Number($("maxMemberDefenses").value)
  });
  state.scheduleId = data.id;
  renderScheduleActions(data);
  showInfo(`Створено нову версію графіка №${data.id}.`);
```

```
 881 |   if (state.selectedCandidateId) await
openCandidate(state.selectedCandidateId, false);
 882 |   await loadSchedules();
 883 | }
 884 |
 885 | async function loadSchedules() {
 886 |   if (!ensureToken()) return;
 887 |   const data = await getJson("/api/v1/schedules");
 888 |   state.schedules = data.items;
 889 |   renderScheduleList();
 890 | }
 891 |
 892 | function renderScheduleList() {
 893 |   $("scheduleList").innerHTML = state.schedules.length
 894 |     ? state.schedules.map((item) => `
 895 |       <article class="panel schedule-card">
 896 |         <p class="eyebrow">Версія ${item.version_number}</p>
 897 |         <h3>Графік №${item.id}</h3>
 898 |         <p>Стан: ${escapeHtml(scheduleStateLabel(item.state))} · подій:
${item.event_count} · оцінка: ${item.score ?? "—"}</p>
 899 |         <div class="schedule-menu-wrap">
 900 |           <div class="dropdown">
 901 |             <button class="secondary dropdown-toggle" data-toggle-
schedule-menu="${item.id}" aria-expanded="false">Дії</button>
 902 |             <div class="dropdown-menu" data-schedule-menu="${item.id}">
 903 |               <button data-select-schedule="${item.id}">Зробити
активним</button>
 904 |               <a class="dropdown-item"
href="/ui/schedules/${item.id}">Відкрити</a>
 905 |               <button class="secondary" data-check-
conflicts="${item.id}">Конфлікти</button>
 906 |               <button class="secondary" data-approve-list-
schedule="${item.id}" ${item.state !== "draft" ? "disabled" :
""}>Погодити</button>
 907 |               <button class="secondary" data-publish-list-
schedule="${item.id}" ${item.state !== "approved" ? "disabled" :
""}>Опублікувати</button>
 908 |               <button class="secondary danger" data-delete-list-
schedule="${item.id}">Видалити</button>
 909 |             </div>
 910 |           </div>
 911 |         </div>
 912 |       </article>
 913 |     `).join("")
 914 |     : `<article class="panel muted-panel">Графіки ще не
створено.</article>`;
 915 | }
 916 |
 917 | async function handleScheduleListAction(event) {
 918 |   const toggleId = event.target.dataset.toggleScheduleMenu;
 919 |   const scheduleId = event.target.dataset.checkConflicts;
 920 |   const selectId = event.target.dataset.selectSchedule;
 921 |   const approveId = event.target.dataset.approveListSchedule;
 922 |   const publishId = event.target.dataset.publishListSchedule;
 923 |   const deleteId = event.target.dataset.deleteListSchedule;
 924 |   if (toggleId) {
 925 |     event.stopPropagation();
 926 |     return toggleScheduleMenu(toggleId);
 927 |   }
 928 |   if (selectId || approveId || publishId || deleteId || scheduleId)
closeScheduleMenus();
 929 |   if (selectId) return setActiveSchedule(selectId);
 930 |   if (approveId) return approveScheduleById(approveId);
```

```
 931 |   if (publishId) return publishScheduleById(publishId);
 932 |   if (deleteId) return deleteSchedule(deleteId);
 933 |   if (scheduleId) {
 934 |     const data = await
getJson(`/api/v1/schedules/${scheduleId}/conflicts`);
 935 |     $("conflictResult").textContent = data.items.length
 936 |       ? data.items.map(formatConflictItem).join("\n")
 937 |       : "Конфліктів для цього графіка не виявлено.";
 938 |   }
 939 | }
 940 |
 941 | function toggleScheduleMenu(scheduleId) {
 942 |   const menu = document.querySelector(`[data-schedule-
menu="${scheduleId}"]`);
 943 |   const toggle = document.querySelector(`[data-toggle-schedule-
menu="${scheduleId}"]`);
 944 |   if (!menu || !toggle) return;
 945 |   const shouldOpen = !menu.classList.contains("open");
 946 |   closeScheduleMenus();
 947 |   if (shouldOpen) {
 948 |     menu.classList.add("open");
 949 |     toggle.setAttribute("aria-expanded", "true");
 950 |   }
 951 | }
 952 |
 953 | function closeScheduleMenus() {
 954 |   document.querySelectorAll("[data-schedule-menu]").forEach((menu) =>
menu.classList.remove("open"));
 955 |   document.querySelectorAll("[data-toggle-schedule-
menu]").forEach((toggle) => {
 956 |     toggle.setAttribute("aria-expanded", "false");
 957 |   });
 958 | }
 959 |
 960 | async function handleCandidateScheduleAction(event) {
 961 |   const scheduleId = event.target.dataset.selectCandidateSchedule;
 962 |   const deleteId = event.target.dataset.deleteCandidateSchedule;
 963 |   if (deleteId) return deleteSchedule(deleteId);
 964 |   if (scheduleId) await setActiveSchedule(scheduleId);
 965 | }
 966 |
 967 | async function setActiveSchedule(scheduleId) {
 968 |   const data = await getJson(`/api/v1/schedules/${scheduleId}`);
 969 |   renderScheduleActions(data);
 970 |   showInfo(`Графік №${scheduleId} вибрано активним.`);
 971 |   if (state.selectedCandidateId) await
openCandidate(state.selectedCandidateId, false);
 972 | }
 973 |
 974 | async function approveScheduleById(scheduleId) {
 975 |   state.scheduleId = scheduleId;
 976 |   await approveSchedule();
 977 | }
 978 |
 979 | async function publishScheduleById(scheduleId) {
 980 |   state.scheduleId = scheduleId;
 981 |   await publishSchedule();
 982 | }
 983 |
 984 | async function deleteCandidate(candidateId) {
 985 |   if (!ensureToken()) return;
 986 |   const candidate = state.candidates.find((item) => item.id ===
candidateId);
```

```
 987 |   const label = candidate?.full_name || `картка №${candidateId}`;
 988 |   const confirmed = confirm(
 989 |     `Видалити аспіранта "${label}" з бази? Будуть видалені його етапи,
документи, разова рада та події у графіках.`
 990 |   );
 991 |   if (!confirmed) return;
 992 |   try {
 993 |     const data = await deleteJson(`/api/v1/candidates/${candidateId}`);
 994 |     showInfo(`Аспіранта "${label}" видалено. Пов'язаних етапів:
${data.related_counts.stages || 0}; подій графіка:
${data.related_counts.schedule_events || 0}.`);
 995 |     if (state.selectedCandidateId === candidateId)
resetCandidateWorkspace();
 996 |     await refreshAll();
 997 |   } catch (error) {
 998 |     showError(error.message || "Не вдалося видалити аспіранта.");
 999 |   }
1000 | }
1001 |
1002 | async function deleteSchedule(scheduleId) {
1003 |   if (!ensureToken()) return;
1004 |   const confirmed = confirm(
1005 |     `Видалити версію графіка №${scheduleId}? Події цієї версії також буде
видалено.`
1006 |   );
1007 |   if (!confirmed) return;
1008 |   try {
1009 |     const data = await deleteJson(`/api/v1/schedules/${scheduleId}`);
1010 |     showInfo(`Графік №${scheduleId} видалено. Подій видалено:
${data.related_counts.schedule_events || 0}.`);
1011 |     if (String(state.scheduleId) === String(scheduleId)) {
1012 |       state.scheduleId = "";
1013 |       renderActiveScheduleSummary(null);
1014 |     }
1015 |     await loadSchedules();
1016 |     await refreshWorkspace();
1017 |     if (state.selectedCandidateId) await
openCandidate(state.selectedCandidateId, false);
1018 |   } catch (error) {
1019 |     showError(error.message || "Не вдалося видалити графік.");
1020 |   }
1021 | }
1022 |
1023 | function resetCandidateWorkspace() {
1024 |   state.selectedCandidateId = null;
1025 |   state.candidateDetail = null;
1026 |   state.scheduleId = "";
1027 |   $("candidateTitle").textContent = "Оберіть аспіранта";
1028 |   $("candidateEmpty").classList.remove("hidden");
1029 |   $("candidateWorkspace").classList.add("hidden");
1030 |   $("candidateQuickSelect").value = "";
1031 | }
1032 |
1033 | async function buildCandidateMaterial() {
1034 |   const detail = state.candidateDetail;
1035 |   if (!detail || !detail.primary_council_id) return showError("Для
аспіранта ще не сформовано разову раду.");
1036 |   const data = await
getJson(`/api/v1/councils/${detail.primary_council_id}/materials`);
1037 |   $("candidateMaterial").textContent = `${data.title}\n\n${data.body}`;
1038 | }
1039 |
1040 | async function loadNormative() {
```

```
1041 |   if (!ensureToken()) return;
1042 |   const data = await getJson("/api/v1/attestation/normative-control");
1043 |   state.issues = data.items;
1044 |   renderNormativeIssues();
1045 | }
1046 |
1047 | function renderNormativeIssues() {
1048 |   const actionable = state.issues.filter((item) => item.candidate_id !==
0);
1049 |   $("overviewIssues").innerHTML = actionable.length
1050 |     ? actionable.slice(0, 6).map((item) => issueCard(item)).join("")
1051 |     : `<div class="issue-item info"><strong>Критичних ризиків
немає</strong><span>Поточний стан не містить зауважень.</span></div>`;
1052 |   $("normativeResult").innerHTML = state.issues.length
1053 |     ? state.issues.map((item) => issueCard(item, true)).join("")
1054 |     : `<article class="panel muted-panel">Дані нормативного контролю
відсутні.</article>`;
1055 | }
1056 |
1057 | function issueCard(item, full = false) {
1058 |   return `
1059 |     <article class="issue-item ${escapeHtml(item.severity)}">
1060 |       <strong>${escapeHtml(item.candidate_name)} ·
${escapeHtml(item.title)}</strong>
1061 |       <span>${escapeHtml(item.message)}</span>
1062 |       ${full ? `<small>План: ${formatDate(item.planned_date)} · факт:
${formatDate(item.actual_date)} · гранично:
${formatDate(item.deadline_date)}</small>` : ""}
1063 |     </article>
1064 |   `;
1065 | }
1066 |
1067 | async function validateCsv() {
1068 |   if (!ensureToken()) return;
1069 |   const file = selectedCsv();
1070 |   if (!file) return showError("Оберіть CSV-файл.");
1071 |   const data = await uploadCsv("/api/v1/imports/validate", file);
1072 |   $("importResult").textContent = JSON.stringify({
1073 |     "Файл": data.file_name,
1074 |     "Усього рядків": data.total_rows,
1075 |     "Коректних рядків": data.valid_rows,
1076 |     "Рядків з помилками": data.error_rows,
1077 |     "Дублікатів": data.duplicate_ids,
1078 |     "Кодування": data.detected_encoding,
1079 |     "Проігноровані чутливі колонки": data.ignored_sensitive_columns,
1080 |     "Проблеми": data.issues.slice(0, 20).map((item) => ({
1081 |       "Рядок": item.row_number,
1082 |       "Поле": item.field_name || "—",
1083 |       "Повідомлення": item.message
1084 |     }))
1085 |   }, null, 2);
1086 |   showInfo("CSV-файл перевірено. Результат виведено нижче.");
1087 | }
1088 |
1089 | async function applyCsv() {
1090 |   if (!ensureToken()) return;
1091 |   const file = selectedCsv();
1092 |   if (!file) return showError("Оберіть CSV-файл.");
1093 |   const data = await uploadCsv("/api/v1/imports/apply", file);
1094 |   $("importResult").textContent = JSON.stringify({
1095 |     "Пакет імпорту": data.batch_id,
1096 |     "Усього рядків": data.total_rows,
1097 |     "Додано записів": data.inserted_rows,
```

```
1098 |     "Оновлено записів": data.updated_rows,
1099 |     "Рядків з помилками": data.error_rows
1100 |   }, null, 2);
1101 |   showInfo("Імпорт застосовано. Тепер сформуйте етапи або відкрийте
картку аспіранта.");
1102 |   await refreshAll();
1103 | }
1104 |
1105 | function selectedCsv() {
1106 |   return $("csvFile").files[0];
1107 | }
1108 |
1109 | function parseIds(value) {
1110 |   return value
1111 |     .split(",")
1112 |     .map((item) => Number(item.trim()))
1113 |     .filter((item) => Number.isInteger(item) && item > 0);
1114 | }
1115 |
1116 | function councilStatusLabel(value) {
1117 |   if (!value) return "не сформовано";
1118 |   return councilStatusLabels[value] || "невідомо";
1119 | }
1120 |
1121 | function scheduleStateLabel(value) {
1122 |   if (!value) return "невідомо";
1123 |   return scheduleStateLabels[value] || "невідомо";
1124 | }
1125 |
1126 | function solverStatusLabel(value) {
1127 |   if (!value) return "невідомо";
1128 |   return solverStatusLabels[value] || "невідомо";
1129 | }
1130 |
1131 | function severityLabel(value) {
1132 |   if (!value) return "інформація";
1133 |   return severityLabels[value] || "інформація";
1134 | }
1135 |
1136 | function environmentLabel(value) {
1137 |   if (!value) return "не вказано";
1138 |   return environmentLabels[value] || "інше середовище";
1139 | }
1140 |
1141 | function emptyToNull(value) {
1142 |   const normalized = (value || "").trim();
1143 |   return normalized || null;
1144 | }
1145 |
1146 | function dateTimeLocalValue(value) {
1147 |   if (!value) return "";
1148 |   return String(value).slice(0, 16);
1149 | }
1150 |
1151 | function reviewsToText(reviews) {
1152 |   return reviews.map((item) => [
1153 |     item.member_name || "",
1154 |     item.kind || "Рецензія",
1155 |     item.material_url || "",
1156 |     item.signature_url || ""
1157 |   ].join(" | ")).join("\n");
1158 | }
1159 |
```

```
function parseReviewsText(value) {
  return value
    .split(/\r?\n/)
    .map((line) => line.trim())
    .filter(Boolean)
    .map((line) => {
      const [memberName, kind, materialUrl, signatureUrl] =
line.split("|").map((item) => item.trim());
      return {
        member_name: memberName,
        kind: kind || "Рецензія",
        material_url: materialUrl || null,
        signature_url: signatureUrl || null
      };
    })
    .filter((item) => item.member_name);
}

function accessRoleSummary(value) {
  try {
    const roles = JSON.parse(value);
    if (!Array.isArray(roles) || !roles.length) return "не вказано";
    return roles.map((item) => accessRoleLabels[item] || "невідома
роль").join(", ");
  } catch {
    return "не вказано";
  }
}

function formatConflictItem(item) {
  return `[${severityLabel(item.severity)}] ${item.message}`;
}

async function getJson(url) {
  const response = await authenticatedFetch(url);
  return parseResponse(response);
}

async function postJson(url, body) {
  const response = await authenticatedFetch(url, {
    method: "POST",
    headers: {"Content-Type": "application/json"},
    body: JSON.stringify(body)
  });
  return parseResponse(response);
}

async function patchJson(url, body) {
  const response = await authenticatedFetch(url, {
    method: "PATCH",
    headers: {"Content-Type": "application/json"},
    body: JSON.stringify(body)
  });
  return parseResponse(response);
}

async function putJson(url, body) {
  const response = await authenticatedFetch(url, {
    method: "PUT",
    headers: {"Content-Type": "application/json"},
    body: JSON.stringify(body)
  });
  return parseResponse(response);
```

```
1221 | }
1222 |
1223 | async function deleteJson(url) {
1224 |   const response = await authenticatedFetch(url, {
1225 |     method: "DELETE"
1226 |   });
1227 |   return parseResponse(response);
1228 | }
1229 |
1230 | async function uploadCsv(url, file) {
1231 |   const form = new FormData();
1232 |   form.append("file", file);
1233 |   const response = await authenticatedFetch(url, {
1234 |     method: "POST",
1235 |     body: form
1236 |   });
1237 |   return parseResponse(response);
1238 | }
1239 |
1240 | async function authenticatedFetch(url, options = {}, retryOnUnauthorized
= true) {
1241 |   const response = await fetch(url, {
1242 |     ...options,
1243 |     headers: withAuthHeaders(options.headers || {})
1244 |   });
1245 |   if (response.status !== 401 || !retryOnUnauthorized) return response;
1246 |
1247 |   const refreshed = await refreshDemoTokenSilently();
1248 |   if (!refreshed) return response;
1249 |
1250 |   return fetch(url, {
1251 |     ...options,
1252 |     headers: withAuthHeaders(options.headers || {})
1253 |   });
1254 | }
1255 |
1256 | async function parseResponse(response) {
1257 |   const contentType = response.headers.get("content-type") || "";
1258 |   const data = contentType.includes("application/json")
1259 |     ? await response.json()
1260 |     : {detail: await response.text()};
1261 |   if (!response.ok) {
1262 |     if (response.status === 401) clearAuthState();
1263 |     throw showError(data.detail || "Помилка запиту.");
1264 |   }
1265 |   return data;
1266 | }
1267 |
1268 | function authHeaders() {
1269 |   return {"Authorization": `Bearer ${state.token}`};
1270 | }
1271 |
1272 | function withAuthHeaders(headers) {
1273 |   return {...headers, ...authHeaders()};
1274 | }
1275 |
1276 | function clearAuthState() {
1277 |   state.token = "";
1278 |   localStorage.removeItem("attestationAccessToken");
1279 |   localStorage.removeItem("attestationRoles");
1280 |   localStorage.removeItem("attestationTokenExpiresIn");
1281 |   localStorage.removeItem("attestationTokenIssuedAt");
1282 | }
```

```
1283 |
1284 | function ensureToken() {
1285 |   if (state.token) return true;
1286 |   showError("Спочатку увійдіть через корпоративну пошту або отримайте
демо-доступ.");
1287 |   return false;
1288 | }
1289 |
1290 | function showError(message) {
1291 |   $("diagnostics").textContent = message;
1292 |   $("diagnostics").classList.remove("hidden");
1293 |   return new Error(message);
1294 | }
1295 |
1296 | function showInfo(message) {
1297 |   $("diagnostics").textContent = message;
1298 |   $("diagnostics").classList.remove("hidden");
1299 | }
1300 |
1301 | function formatDate(value) {
1302 |   return value || "—";
1303 | }
1304 |
1305 | function formatDateTime(value) {
1306 |   if (!value) return "—";
1307 |   return new Date(value).toLocaleString("uk-UA", {
1308 |     day: "2-digit",
1309 |     month: "2-digit",
1310 |     year: "numeric",
1311 |     hour: "2-digit",
1312 |     minute: "2-digit"
1313 |   });
1314 | }
1315 |
1316 | function formatTime(value) {
1317 |   if (!value) return "—";
1318 |   return new Date(value).toLocaleTimeString("uk-UA", {
1319 |     hour: "2-digit",
1320 |     minute: "2-digit"
1321 |   });
1322 | }
1323 |
1324 | function escapeHtml(value) {
1325 |   const element = document.createElement("div");
1326 |   element.textContent = value ?? "";
1327 |   return element.innerHTML;
1328 | }
1329 |
1330 | function normalizeInstitution(value) {
1331 |   return (value || "").trim().replace(/\s+/g, " ").toLocaleLowerCase("uk-
UA");
1332 | }
1333 |
1334 | async function runCaptureMode() {
1335 |   const mode = new
URLSearchParams(window.location.search).get("capture");
1336 |   if (mode !== "import") return;
1337 |   activateView("import");
1338 |   const source = $("captureImportReport");
1339 |   if (!source) return;
1340 |   const report = JSON.parse(source.textContent);
1341 |   $("importResult").textContent = JSON.stringify({
1342 |     "Усього рядків": report.total_rows,
```

```
1343 |     "Коректних рядків": report.valid_rows,
1344 |     "Рядків з помилками": report.error_rows
1345 |   }, null, 2);
1346 | }
```

## Лістинг Б.6 - Стилі робочого та публічного інтерфейсів (app/static/styles.css)

```
   1 | :root {
   2 |   color-scheme: light;
   3 |   --ink: #172033;
   4 |   --muted: #64748b;
   5 |   --line: #d8e0ec;
   6 |   --panel: #f6f8fc;
   7 |   --paper: #fff;
   8 |   --app-bg: #eef4ff;
   9 |   --primary: #2563eb;
  10 |   --primary-dark: #1d4ed8;
  11 |   --primary-soft: #dbeafe;
  12 |   --success: #15803d;
  13 |   --success-soft: #dcfce7;
  14 |   --warning: #b45309;
  15 |   --warning-soft: #fef3c7;
  16 |   --danger: #b91c1c;
  17 |   --danger-soft: #fee2e2;
  18 |   --violet: #7c3aed;
  19 |   --violet-soft: #ede9fe;
  20 |   --cyan-soft: #cffafe;
  21 |   --shadow: 0 18px 50px rgba(37, 99, 235, .12);
  22 | }
  23 |
  24 | * { box-sizing: border-box; }
  25 |
  26 | body {
  27 |   margin: 0;
  28 |   background: linear-gradient(135deg, #f8fbff 0%, var(--app-bg) 100%);
  29 |   color: var(--ink);
  30 |   font-family: Arial, Helvetica, sans-serif;
  31 | }
  32 |
  33 | button, select, input, textarea { font: inherit; }
  34 |
  35 | button {
  36 |   border: 1px solid var(--primary);
  37 |   background: var(--primary);
  38 |   color: #fff;
  39 |   border-radius: 10px;
  40 |   padding: 10px 16px;
  41 |   cursor: pointer;
  42 |   font-weight: 700;
  43 |   transition: transform .15s ease, box-shadow .15s ease, background .15s
ease;
  44 | }
  45 |
  46 | button:hover:not(:disabled),
  47 | .button-link:hover:not(.disabled) {
  48 |   box-shadow: 0 10px 24px rgba(37, 99, 235, .18);
  49 |   transform: translateY(-1px);
  50 | }
  51 |
  52 | button.secondary {
  53 |   background: #fff;
```

```
  54 |   color: var(--primary-dark);
  55 |   border-color: #b8c7e6;
  56 | }
  57 |
  58 | button.danger,
  59 | .button-link.danger {
  60 |   background: var(--danger);
  61 |   border-color: var(--danger);
  62 |   color: #fff;
  63 | }
  64 |
  65 | button.secondary.danger,
  66 | .button-link.secondary.danger {
  67 |   background: var(--danger-soft);
  68 |   border-color: #fca5a5;
  69 |   color: var(--danger);
  70 | }
  71 |
  72 | button:disabled { cursor: not-allowed; opacity: .45; }
  73 |
  74 | .button-link {
  75 |   display: inline-flex;
  76 |   align-items: center;
  77 |   justify-content: center;
  78 |   border: 1px solid var(--primary);
  79 |   background: var(--primary);
  80 |   color: #fff;
  81 |   border-radius: 10px;
  82 |   padding: 10px 16px;
  83 |   text-decoration: none;
  84 | }
  85 |
  86 | .button-link.secondary {
  87 |   background: #fff;
  88 |   color: var(--primary-dark);
  89 |   border-color: #b8c7e6;
  90 | }
  91 |
  92 | .button-link.disabled {
  93 |   opacity: .45;
  94 |   pointer-events: none;
  95 | }
  96 |
  97 | .topbar {
  98 |   min-height: 112px;
  99 |   display: flex;
 100 |   align-items: center;
 101 |   justify-content: space-between;
 102 |   background: linear-gradient(120deg, #1e3a8a 0%, #2563eb 50%, #7c3aed
100%);
 103 |   border-bottom: 0;
 104 |   color: #fff;
 105 |   padding: 24px 40px;
 106 |   box-shadow: var(--shadow);
 107 | }
 108 |
 109 | .app-topbar { gap: 20px; }
 110 |
 111 | .topbar .eyebrow {
 112 |   color: #dbeafe;
 113 | }
 114 |
 115 | .topbar-actions {
```

```
116 |   display: flex;
117 |   align-items: center;
118 |   gap: 10px;
119 |   flex-wrap: wrap;
120 |   justify-content: flex-end;
121 | }
122 |
123 | h1, h2, p { margin-top: 0; }
124 | h1 { margin-bottom: 0; font-size: 30px; }
125 | h2 { margin-bottom: 18px; font-size: 24px; }
126 | h3 { margin: 26px 0 10px; font-size: 18px; }
127 |
128 | .eyebrow {
129 |   margin-bottom: 7px;
130 |   color: #6b7a90;
131 |   font-size: 12px;
132 |   font-weight: 700;
133 |   letter-spacing: .08em;
134 |   text-transform: uppercase;
135 | }
136 |
137 | .environment, .badge {
138 |   border: 1px solid #bdd0f4;
139 |   border-radius: 999px;
140 |   background: #fff;
141 |   color: var(--primary-dark);
142 |   padding: 8px 12px;
143 |   font-size: 13px;
144 | }
145 |
146 | .badge.success {
147 |   background: var(--success);
148 |   border-color: var(--success);
149 |   color: #fff;
150 | }
151 |
152 | .layout {
153 |   display: grid;
154 |   grid-template-columns: 220px minmax(0, 1fr);
155 |   min-height: calc(100vh - 112px);
156 | }
157 |
158 | .sidebar {
159 |   background: rgba(255, 255, 255, .82);
160 |   border-right: 1px solid var(--line);
161 |   padding: 30px 18px;
162 |   backdrop-filter: blur(12px);
163 | }
164 |
165 | .nav-item {
166 |   display: block;
167 |   width: 100%;
168 |   border: 0;
169 |   border-radius: 12px;
170 |   border-bottom: 0;
171 |   background: transparent;
172 |   color: var(--ink);
173 |   padding: 14px 10px;
174 |   text-align: left;
175 |   text-decoration: none;
176 | }
177 |
178 | .nav-item:hover {
```

```
179 |   background: var(--primary-soft);
180 | }
181 |
182 | .nav-item.active {
183 |   background: var(--primary);
184 |   color: #fff;
185 |   font-weight: 700;
186 | }
187 |
188 | .content { padding: 30px 40px 60px; }
189 |
190 | .page {
191 |   padding: 30px 40px 60px;
192 | }
193 |
194 | .panel {
195 |   background: rgba(255, 255, 255, .94);
196 |   border: 1px solid var(--line);
197 |   border-radius: 18px;
198 |   box-shadow: 0 10px 30px rgba(15, 23, 42, .05);
199 |   padding: 24px;
200 | }
201 |
202 | .status-row {
203 |   display: flex;
204 |   justify-content: flex-end;
205 |   gap: 10px;
206 |   margin-bottom: 28px;
207 | }
208 |
209 | .view { display: none; }
210 | .view.active { display: block; }
211 |
212 | .section-heading {
213 |   display: flex;
214 |   align-items: end;
215 |   justify-content: space-between;
216 |   gap: 20px;
217 | }
218 |
219 | .actions {
220 |   display: flex;
221 |   align-items: end;
222 |   gap: 10px;
223 | }
224 |
225 | .schedule-params {
226 |   flex-wrap: wrap;
227 |   justify-content: flex-end;
228 | }
229 |
230 | .schedule-params label {
231 |   min-width: 180px;
232 | }
233 |
234 | .schedule-params select {
235 |   max-width: 360px;
236 | }
237 |
238 | label {
239 |   display: grid;
240 |   gap: 6px;
241 |   color: var(--muted);
```

```
  font-size: 12px;
}

select, textarea, input[type="file"], input[type="number"],
input[type="date"], input:not([type]) {
  border: 1px solid #b8c7e6;
  background: #fff;
  border-radius: 10px;
  padding: 9px;
  color: #111;
}

select:focus, textarea:focus, input:focus {
  border-color: var(--primary);
  box-shadow: 0 0 0 3px rgba(37, 99, 235, .14);
  outline: none;
}

textarea {
  min-width: 100%;
  resize: vertical;
}

.metrics {
  display: grid;
  grid-template-columns: repeat(4, minmax(0, 1fr));
  gap: 12px;
  margin: 24px 0;
}

.metrics article {
  background: linear-gradient(180deg, #fff 0%, #f8fbff 100%);
  border: 1px solid var(--line);
  border-radius: 18px;
  box-shadow: 0 10px 24px rgba(37, 99, 235, .08);
  padding: 16px;
}

.metrics span {
  display: block;
  margin-bottom: 8px;
  color: var(--muted);
  font-size: 12px;
}

.metrics strong { font-size: 22px; }

.metrics article:nth-child(1) strong { color: var(--primary); }
.metrics article:nth-child(2) strong { color: var(--success); }
.metrics article:nth-child(3) strong { color: var(--violet); }
.metrics article:nth-child(4) strong { color: var(--warning); }

.task-grid {
  display: grid;
  grid-template-columns: repeat(3, minmax(0, 1fr));
  gap: 12px;
}

.task-grid article {
  background: #fff;
  border: 1px solid var(--line);
  border-radius: 16px;
  padding: 16px;
```

```
 304 | }
 305 |
 306 | .task-grid strong {
 307 |   display: block;
 308 |   margin-bottom: 8px;
 309 | }
 310 |
 311 | .task-grid span {
 312 |   color: var(--muted);
 313 |   line-height: 1.45;
 314 | }
 315 |
 316 | .schedule-help-grid {
 317 |   margin: 12px 0 16px;
 318 | }
 319 |
 320 | .schedule-help-grid article {
 321 |   background: linear-gradient(180deg, #f8fbff, #ffffff);
 322 |   border-color: #bfdbfe;
 323 |   box-shadow: 0 8px 20px rgba(37, 99, 235, .06);
 324 | }
 325 |
 326 | .schedule-help-grid strong {
 327 |   color: var(--primary);
 328 | }
 329 |
 330 | .dashboard-grid {
 331 |   display: grid;
 332 |   grid-template-columns: repeat(2, minmax(0, 1fr));
 333 |   gap: 16px;
 334 |   margin: 18px 0;
 335 | }
 336 |
 337 | .card-grid {
 338 |   display: grid;
 339 |   grid-template-columns: repeat(auto-fill, minmax(310px, 1fr));
 340 |   gap: 14px;
 341 | }
 342 |
 343 | .candidate-card {
 344 |   background: #fff;
 345 |   border: 1px solid var(--line);
 346 |   border-radius: 18px;
 347 |   box-shadow: 0 10px 28px rgba(15, 23, 42, .06);
 348 |   display: flex;
 349 |   flex-direction: column;
 350 |   justify-content: space-between;
 351 |   gap: 18px;
 352 |   padding: 18px;
 353 |   min-height: 210px;
 354 | }
 355 |
 356 | .candidate-card.selected {
 357 |   border-color: var(--primary);
 358 |   box-shadow: 0 16px 38px rgba(37, 99, 235, .18), inset 0 0 0 1px var(--
primary);
 359 | }
 360 |
 361 | .candidate-card h3 {
 362 |   margin-top: 0;
 363 | }
 364 |
 365 | .candidate-card-footer {
```

```
  366 |   border-top: 1px solid var(--line);
  367 |   display: flex;
  368 |   align-items: center;
  369 |   justify-content: space-between;
  370 |   gap: 10px;
  371 |   padding-top: 12px;
  372 | }
  373 |
  374 | .row-actions {
  375 |   display: flex;
  376 |   flex-wrap: wrap;
  377 |   gap: 8px;
  378 |   margin-top: 10px;
  379 | }
  380 |
  381 | .steps {
  382 |   color: var(--muted);
  383 |   line-height: 1.7;
  384 |   padding-left: 22px;
  385 | }
  386 |
  387 | .muted, .muted-panel {
  388 |   color: var(--muted);
  389 | }
  390 |
  391 | .muted-panel {
  392 |   background: #fff;
  393 |   border: 1px dashed #b8c7e6;
  394 |   border-radius: 16px;
  395 |   padding: 18px;
  396 | }
  397 |
  398 | .table-wrap {
  399 |   overflow-x: auto;
  400 |   border: 1px solid var(--line);
  401 |   border-radius: 18px;
  402 |   background: #fff;
  403 | }
  404 |
  405 | table {
  406 |   width: 100%;
  407 |   border-collapse: collapse;
  408 | }
  409 |
  410 | th, td {
  411 |   border-bottom: 1px solid var(--line);
  412 |   padding: 12px 14px;
  413 |   text-align: left;
  414 | }
  415 |
  416 | th { background: var(--primary-soft); color: var(--primary-dark); font-
size: 13px; }
  417 | tr:last-child td { border-bottom: 0; }
  418 | .empty { color: var(--muted); text-align: center; }
  419 |
  420 | td small {
  421 |   display: block;
  422 |   margin-top: 6px;
  423 |   color: var(--muted);
  424 |   line-height: 1.35;
  425 | }
  426 |
  427 | .inline-action {
```

```
  min-width: 230px;
}

.inline-action input {
  width: 100%;
  margin-bottom: 8px;
}

.status-pill {
  display: inline-block;
  border: 1px solid transparent;
  border-radius: 999px;
  padding: 3px 8px;
  font-size: 12px;
  font-weight: 700;
}

.status-pill.ok { background: var(--success-soft); color: var(--success); border-color: #86efac; }
.status-pill.pending { background: var(--primary-soft); color: var(--primary-dark); border-color: #bfdbfe; }
.status-pill.risk { background: var(--warning-soft); color: var(--warning); border-color: #fcd34d; }
.status-pill.overdue, .status-pill.violation { background: var(--danger-soft); color: var(--danger); border-color: #fca5a5; }

.workflow-actions {
  display: flex;
  justify-content: flex-end;
  gap: 10px;
  margin-top: 18px;
}

.schedule-toolbar {
  display: flex;
  justify-content: flex-end;
  gap: 10px;
  margin-bottom: 18px;
}

.detail-grid {
  display: grid;
  grid-template-columns: repeat(3, minmax(0, 1fr));
  gap: 12px;
  margin: 0 0 24px;
}

.detail-grid article {
  border: 1px solid var(--line);
  padding: 14px 16px;
}

.detail-grid span {
  display: block;
  margin-bottom: 7px;
  color: var(--muted);
  font-size: 12px;
}

.detail-grid strong {
  font-size: 16px;
  font-weight: 700;
}
```

```
 487 |
 488 | .candidate-hero {
 489 |   background: linear-gradient(120deg, #eff6ff 0%, #f5f3ff 100%);
 490 |   border: 1px solid #bfdbfe;
 491 |   border-radius: 22px;
 492 |   display: flex;
 493 |   align-items: center;
 494 |   justify-content: space-between;
 495 |   gap: 20px;
 496 |   padding: 24px;
 497 | }
 498 |
 499 | .progress-box {
 500 |   min-width: 220px;
 501 | }
 502 |
 503 | .progress-box span {
 504 |   color: var(--muted);
 505 |   display: block;
 506 |   font-size: 12px;
 507 |   margin-bottom: 8px;
 508 | }
 509 |
 510 | .progress-box strong {
 511 |   display: block;
 512 |   font-size: 34px;
 513 |   margin-bottom: 10px;
 514 | }
 515 |
 516 | .progress-track {
 517 |   background: #fff;
 518 |   border: 1px solid #bfdbfe;
 519 |   border-radius: 999px;
 520 |   height: 12px;
 521 | }
 522 |
 523 | .progress-track div {
 524 |   background: linear-gradient(90deg, var(--primary) 0%, var(--success)
100%);
 525 |   border-radius: 999px;
 526 |   height: 100%;
 527 |   width: 0;
 528 | }
 529 |
 530 | .compact-list {
 531 |   display: grid;
 532 |   gap: 10px;
 533 | }
 534 |
 535 | .list-row,
 536 | .issue-item {
 537 |   background: #fff;
 538 |   border: 1px solid var(--line);
 539 |   border-radius: 14px;
 540 |   padding: 12px;
 541 | }
 542 |
 543 | .list-row strong,
 544 | .issue-item strong {
 545 |   display: block;
 546 |   margin-bottom: 5px;
 547 | }
 548 |
```

```
.list-row span,
.issue-item span,
.list-row small,
.issue-item small {
  color: var(--muted);
  display: block;
  line-height: 1.35;
}

.issue-grid {
  display: grid;
  grid-template-columns: repeat(auto-fill, minmax(340px, 1fr));
  gap: 12px;
}

.issue-item.error {
  background: var(--danger-soft);
  border-color: #fecaca;
  border-left: 5px solid var(--danger);
}

.issue-item.warning {
  background: var(--warning-soft);
  border-color: #fde68a;
  border-left: 5px solid var(--warning);
}

.issue-item.info {
  background: var(--cyan-soft);
  border-color: #67e8f9;
}

.issue-item.success {
  background: var(--success-soft);
  border-color: #86efac;
}

.council-editor {
  display: grid;
  gap: 10px;
  margin: 12px 0;
}

.wide-label {
  display: grid;
  gap: 6px;
  margin: 10px 0 12px;
}

.settings-form {
  display: grid;
  gap: 12px;
  grid-template-columns: repeat(auto-fit, minmax(190px, 1fr));
  margin: 12px 0;
}

.room-create-form {
  align-items: end;
}

.settings-room-list {
  display: grid;
  gap: 10px;
```

```
  612 |   margin: 14px 0;
  613 | }
  614 |
  615 | .settings-room-row {
  616 |   align-items: end;
  617 |   background: #f8fbff;
  618 |   border: 1px solid var(--line);
  619 |   border-radius: 16px;
  620 |   display: grid;
  621 |   gap: 10px;
  622 |   grid-template-columns: minmax(90px, .7fr) minmax(160px, 1fr) 95px
minmax(180px, 1.2fr) 95px 80px auto;
  623 |   padding: 12px;
  624 | }
  625 |
  626 | .settings-room-row.inactive {
  627 |   background: #f8fafc;
  628 |   opacity: .72;
  629 | }
  630 |
  631 | .inline-check {
  632 |   align-items: center;
  633 |   display: flex;
  634 |   gap: 6px;
  635 |   margin: 0;
  636 | }
  637 |
  638 | .inline-check input {
  639 |   width: auto;
  640 | }
  641 |
  642 | .room-usage {
  643 |   color: var(--muted);
  644 |   font-size: .9rem;
  645 | }
  646 |
  647 | .compact-settings {
  648 |   grid-template-columns: repeat(auto-fit, minmax(180px, 1fr));
  649 | }
  650 |
  651 | .council-member-form {
  652 |   background: #f8fbff;
  653 |   border: 1px solid var(--line);
  654 |   border-radius: 14px;
  655 |   display: grid;
  656 |   gap: 8px;
  657 |   grid-template-columns: 150px minmax(180px, 1fr) minmax(120px, .7fr)
minmax(140px, .8fr);
  658 |   padding: 12px;
  659 | }
  660 |
  661 | .council-member-form strong {
  662 |   align-self: center;
  663 | }
  664 |
  665 | .timeline {
  666 |   display: grid;
  667 |   gap: 12px;
  668 | }
  669 |
  670 | .timeline-item {
  671 |   background: #fff;
  672 |   border: 1px solid var(--line);
```

```
673 |   border-radius: 18px;
674 |   display: grid;
675 |   grid-template-columns: 48px minmax(0, 1fr);
676 | }
677 |
678 | .timeline-item.violation,
679 | .timeline-item.overdue {
680 |   background: #fff7f7;
681 |   border-color: #fecaca;
682 | }
683 |
684 | .timeline-item.risk {
685 |   background: #fffbeb;
686 |   border-color: #fde68a;
687 | }
688 |
689 | .timeline-item.ok {
690 |   background: #f0fdf4;
691 |   border-color: #bbf7d0;
692 | }
693 |
694 | .timeline-index {
695 |   align-items: center;
696 |   background: var(--primary);
697 |   border-radius: 18px 0 0 18px;
698 |   color: #fff;
699 |   display: flex;
700 |   font-weight: 700;
701 |   justify-content: center;
702 | }
703 |
704 | .timeline-item.ok .timeline-index { background: var(--success); }
705 | .timeline-item.risk .timeline-index { background: var(--warning); }
706 | .timeline-item.overdue .timeline-index,
707 | .timeline-item.violation .timeline-index { background: var(--danger); }
708 |
709 | .timeline-body {
710 |   padding: 14px;
711 | }
712 |
713 | .timeline-head {
714 |   display: flex;
715 |   justify-content: space-between;
716 |   gap: 14px;
717 | }
718 |
719 | .timeline-head strong,
720 | .timeline-head span {
721 |   display: block;
722 | }
723 |
724 | .timeline-head span,
725 | .timeline-dates,
726 | .timeline-body p {
727 |   color: var(--muted);
728 | }
729 |
730 | .timeline-dates {
731 |   display: flex;
732 |   flex-wrap: wrap;
733 |   gap: 12px;
734 |   margin: 10px 0;
735 |   font-size: 13px;
```

```
 736 | }
 737 |
 738 | .stage-actions {
 739 |   border-top: 1px solid var(--line);
 740 |   display: grid;
 741 |   gap: 8px;
 742 |   grid-template-columns: 160px auto minmax(160px, 1fr) minmax(160px, 1fr)
auto;
 743 |   margin-top: 12px;
 744 |   padding-top: 12px;
 745 | }
 746 |
 747 | .compact-result {
 748 |   max-height: 260px;
 749 |   min-height: 90px;
 750 | }
 751 |
 752 | .schedule-card .actions {
 753 |   margin-top: 12px;
 754 | }
 755 |
 756 | .schedule-menu-wrap {
 757 |   display: flex;
 758 |   justify-content: flex-end;
 759 |   margin-top: 14px;
 760 | }
 761 |
 762 | .dropdown {
 763 |   position: relative;
 764 | }
 765 |
 766 | .dropdown-toggle {
 767 |   min-width: 110px;
 768 | }
 769 |
 770 | .dropdown-toggle::after {
 771 |   content: " v";
 772 |   font-size: 12px;
 773 | }
 774 |
 775 | .dropdown-menu {
 776 |   background: #fff;
 777 |   border: 1px solid var(--line);
 778 |   border-radius: 14px;
 779 |   box-shadow: 0 18px 40px rgba(15, 23, 42, .18);
 780 |   display: none;
 781 |   min-width: 230px;
 782 |   padding: 8px;
 783 |   position: absolute;
 784 |   right: 0;
 785 |   top: calc(100% + 8px);
 786 |   z-index: 30;
 787 | }
 788 |
 789 | .dropdown-menu.open {
 790 |   display: grid;
 791 |   gap: 6px;
 792 | }
 793 |
 794 | .dropdown-menu button,
 795 | .dropdown-item {
 796 |   border-radius: 10px;
 797 |   justify-content: flex-start;
```

```
798 |   text-align: left;
799 |   width: 100%;
800 | }
801 |
802 | .dropdown-item {
803 |   align-items: center;
804 |   background: #fff;
805 |   border: 1px solid #b8c7e6;
806 |   color: var(--primary-dark);
807 |   display: flex;
808 |   font-weight: 700;
809 |   min-height: 42px;
810 |   padding: 10px 16px;
811 |   text-decoration: none;
812 | }
813 |
814 | .dropdown-item:hover {
815 |   background: var(--primary-soft);
816 | }
817 |
818 | .notice {
819 |   border-left: 4px solid var(--primary);
820 |   background: var(--primary-soft);
821 |   border-radius: 12px;
822 |   color: var(--primary-dark);
823 |   margin: 16px 0;
824 |   padding: 12px 16px;
825 | }
826 |
827 | .subtle-notice {
828 |   background: #fff;
829 |   border: 1px solid var(--line);
830 |   border-left: 4px solid var(--violet);
831 |   color: var(--ink);
832 | }
833 |
834 | .hidden { display: none; }
835 | .lead { max-width: 780px; color: #444; line-height: 1.55; }
836 |
837 | .upload-box {
838 |   display: flex;
839 |   gap: 10px;
840 |   background: #fff;
841 |   border: 1px dashed #93c5fd;
842 |   border-radius: 18px;
843 |   padding: 22px;
844 |   margin: 22px 0;
845 | }
846 |
847 | .result {
848 |   min-height: 170px;
849 |   overflow: auto;
850 |   border: 1px solid var(--line);
851 |   border-radius: 16px;
852 |   background: #f8fafc;
853 |   padding: 16px;
854 |   white-space: pre-wrap;
855 | }
856 |
857 | .public-link {
858 |   display: inline-block;
859 |   border-bottom: 1px solid #111;
860 |   color: #111;
```

```
  861 |   padding: 8px 0;
  862 |   text-decoration: none;
  863 | }
  864 |
  865 | .public-link.disabled {
  866 |   color: var(--muted);
  867 |   pointer-events: none;
  868 | }
  869 |
  870 | .public-page { background: #eee; padding: 40px; }
  871 | .public-card { max-width: 920px; margin: auto; background: #fff; border-
radius: 18px; padding: 40px; }
  872 |
  873 | .public-home-page {
  874 |   background: linear-gradient(135deg, #eff6ff, #f8fafc);
  875 | }
  876 |
  877 | .public-home {
  878 |   max-width: 1180px;
  879 |   margin: 0 auto;
  880 | }
  881 |
  882 | .public-hero {
  883 |   align-items: flex-start;
  884 |   background: #fff;
  885 |   border: 1px solid var(--line);
  886 |   border-radius: 28px;
  887 |   box-shadow: var(--shadow);
  888 |   display: flex;
  889 |   gap: 24px;
  890 |   justify-content: space-between;
  891 |   margin-bottom: 22px;
  892 |   padding: 34px;
  893 | }
  894 |
  895 | .public-hero h1 {
  896 |   font-size: clamp(30px, 4vw, 52px);
  897 |   margin: 4px 0 12px;
  898 | }
  899 |
  900 | .public-council-grid {
  901 |   display: grid;
  902 |   gap: 18px;
  903 | }
  904 |
  905 | .public-council-card,
  906 | .public-empty {
  907 |   background: #fff;
  908 |   border: 1px solid var(--line);
  909 |   border-radius: 24px;
  910 |   box-shadow: var(--shadow);
  911 |   padding: 24px;
  912 | }
  913 |
  914 | .public-empty.warning {
  915 |   background: #fffbeb;
  916 |   border-color: #fde68a;
  917 | }
  918 |
  919 | .public-council-head {
  920 |   align-items: flex-start;
  921 |   display: flex;
  922 |   gap: 18px;
```

```
  justify-content: space-between;
}

.public-council-head h2 {
  margin: 4px 0 10px;
}

.public-council-summary {
  display: grid;
  gap: 10px;
  grid-template-columns: repeat(auto-fit, minmax(240px, 1fr));
  margin: 12px 0 16px;
}

.public-council-summary p,
.public-facts p {
  margin: 0;
}

.public-council-details {
  border-top: 1px solid var(--line);
  padding-top: 14px;
}

.public-council-details summary {
  color: var(--primary);
  cursor: pointer;
  font-weight: 800;
}

.public-council-full {
  display: grid;
  gap: 18px;
  margin-top: 16px;
}

.public-facts {
  background: #f8fbff;
  border: 1px solid #bfdbfe;
  border-radius: 18px;
  display: grid;
  gap: 10px;
  padding: 16px;
}

.public-statement,
.public-registry-block,
.public-broadcast-block,
.public-files-block,
.public-decision-block {
  background: #fff;
  border: 1px solid var(--line);
  border-radius: 18px;
  display: grid;
  gap: 10px;
  padding: 16px;
}

.public-statement {
  background: #f8fbff;
  border-color: #bfdbfe;
}

```

```
.public-statement h3,
.public-registry-block h3,
.public-broadcast-block h3,
.public-files-block h3,
.public-decision-block h3 {
  margin: 0;
}

.public-statement p,
.public-registry-block p,
.public-broadcast-block p,
.public-files-block p,
.public-decision-block p,
.public-review-list p {
  margin: 0;
}

.public-member-list {
  display: grid;
  gap: 10px;
}

.public-member-row {
  border: 1px solid var(--line);
  border-radius: 16px;
  display: grid;
  gap: 4px;
  padding: 14px;
}

.public-member-row span,
.public-member-row small {
  color: var(--muted);
}

@media (max-width: 800px) {
  .topbar, .section-heading, .status-row, .candidate-hero, .public-hero, .public-council-head { align-items: stretch; flex-direction: column; }
  .layout { grid-template-columns: 1fr; }
  .sidebar { border-right: 0; border-bottom: 1px solid var(--line); }
  .content, .page { padding: 24px 18px; }
  .metrics { grid-template-columns: repeat(2, minmax(0, 1fr)); }
  .task-grid, .dashboard-grid, .issue-grid { grid-template-columns: 1fr; }
  .detail-grid { grid-template-columns: 1fr; }
  .actions, .upload-box, .schedule-toolbar, .workflow-actions {
    align-items: stretch;
    flex-direction: column;
  }
  .candidate-card-footer,
  .timeline-head {
    align-items: stretch;
    flex-direction: column;
  }
  .stage-actions {
    grid-template-columns: 1fr;
  }
  .council-member-form {
    grid-template-columns: 1fr;
  }
}
```

## ДОДАТОК В

### Контейнерне розгортання FastAPI-сервісу та PostgreSQL

У додатку наведено конфігурації Dockerfile, Docker Compose, entrypoint-скрипт і приклад змінних середовища для запуску web-сервісу та бази даних PostgreSQL.

### Лістинг В.1 - Образ FastAPI-сервісу (Dockerfile)

```
FROM python:3.13-slim

ENV PYTHONDONTWRITEBYTECODE=1 \
    PYTHONUNBUFFERED=1

WORKDIR /app

COPY requirements.txt .
RUN pip install --no-cache-dir -r requirements.txt

RUN useradd --create-home --uid 10001 appuser

COPY --chown=appuser:appuser . .
RUN chmod +x /app/docker-entrypoint.sh

USER appuser

EXPOSE 8000

HEALTHCHECK --interval=30s --timeout=5s --retries=3 \
    CMD python -c "import urllib.request;
urllib.request.urlopen('http://127.0.0.1:8000/health', timeout=3)"

ENTRYPOINT ["/app/docker-entrypoint.sh"]
```

### Лістинг В.2 - Композиція сервісів app і db (docker-compose.yml)

```
services:
  app:
    build:
      context: .
    env_file:
      - .env
    depends_on:
      db:
        condition: service_healthy
    ports:
      - "8000:8000"
    healthcheck:
      test:
        [
          "CMD",
          "python",
          "-c",
          "import urllib.request;
urllib.request.urlopen('http://127.0.0.1:8000/health', timeout=3)",
```

```
19 |         ]
20 |       interval: 30s
21 |       timeout: 5s
22 |       retries: 3
23 |     restart: unless-stopped
24 |     networks:
25 |       - attestation_net
26 |
27 |   db:
28 |     image: postgres:17-alpine
29 |     environment:
30 |       POSTGRES_DB: ${POSTGRES_DB}
31 |       POSTGRES_USER: ${POSTGRES_USER}
32 |       POSTGRES_PASSWORD: ${POSTGRES_PASSWORD}
33 |     volumes:
34 |       - pg_data:/var/lib/postgresql/data
35 |     healthcheck:
36 |       test: ["CMD-SHELL", "pg_isready -U ${POSTGRES_USER} -d
${POSTGRES_DB}"]
37 |       interval: 10s
38 |       timeout: 5s
39 |       retries: 5
40 |     restart: unless-stopped
41 |     networks:
42 |       - attestation_net
43 |
44 | volumes:
45 |   pg_data:
46 |
47 | networks:
48 |   attestation_net:
49 |     driver: bridge
```

## Лістинг В.3 - Стартовий скрипт контейнера app (docker-entrypoint.sh)

```
1 | #!/bin/sh
2 | set -eu
3 |
4 | alembic upgrade head
5 | exec uvicorn app.main:app --host 0.0.0.0 --port 8000
```

## Лістинг В.4 - Приклад змінних середовища (.env.example)

```
 1 | POSTGRES_DB=attestation
 2 | POSTGRES_USER=attestation
 3 | POSTGRES_PASSWORD=change-me
 4 | DATABASE_URL=postgresql+asyncpg://attestation:change-me@db:5432/attestation
 5 | SQL_ECHO=false
 6 | APP_ENV=development
 7 | JWT_SECRET_KEY=replace-with-a-long-random-secret
 8 | JWT_ALGORITHM=HS256
 9 | ACCESS_TOKEN_MINUTES=60
10 | MAX_UPLOAD_BYTES=5242880
```

# ДОДАТОК Г

## ТЕХНІЧНЕ ЗАВДАННЯ
### Назва проєкту

Web-додаток для динамічного планування, нормативного контролю та супроводу атестації аспірантів під час захисту дисертації доктора філософії.

**1. Підстава для розробки**

Розробка виконується в межах бакалаврської роботи, присвяченої створенню інформаційної системи для автоматизованого планування етапів підготовки та проведення захисту дисертацій докторів філософії.

Система повинна враховувати:

Порядок підготовки здобувачів вищої освіти ступеня доктора філософії, затверджений постановою КМУ № 261 [1];

Порядок присудження ступеня доктора філософії, затверджений постановою КМУ № 44 [2];

внутрішні правила ІФНТУНГ щодо процедури захисту дисертацій у 2026 році;

дані аспірантів, експортовані з ЄДЕБО;

потребу відділу аспірантури та докторантури контролювати строки, події, документи, склад разових рад і графік захистів.

**2. Мета розробки**

Метою є створення web-додатка, який не просто відображає календар, а забезпечує повний супровід процесу атестації аспіранта: від завершення освітньо-наукової програми до видачі диплома доктора філософії.

Ключова функція системи: автоматичний контроль нормативних строків і динамічний перерахунок наступних етапів після фіксації фактичної дати певної події відділом аспірантури та докторантури.

**3. Основна ідея системи**

Система має працювати навколо картки конкретного аспіранта.

Для кожного аспіранта створюється індивідуальна траєкторія атестації. Базовою незмінною датою є дата завершення підготовки аспіранта. Вона імпортується з ЄДЕБО або вноситься відповідальним працівником і не повинна випадково змінюватися під час повторного імпорту.

Після настання певної події працівник відділу аспірантури фіксує її фактичну дату. Система автоматично:

перераховує залежні майбутні етапи;

перевіряє, чи не порушено нормативний строк;

показує ризики наближення граничних дат;

забороняє або попереджає про неможливість формування допустимого графіка;

зберігає історію змін.

**4. Користувачі системи**

У системі повинні бути такі ролі:

Адміністратор системи

Керує користувачами, ролями, довідниками, нормативними правилами, аудиторіями, глобальними налаштуваннями.

Оператор відділу аспірантури та докторантури

Основний користувач системи. Імпортує дані ЄДЕБО, веде картки аспірантів, фіксує фактичні дати подій, формує графік захистів, контролює порушення строків.

Аспірант

Переглядає власну картку, етапи, строки, статус документів, дату захисту.

Науковий керівник

Переглядає етапи аспірантів, за яких відповідає, статус готовності дисертації, строки подання висновків.

Член разової спеціалізованої вченої ради

Переглядає захисти, у яких бере участь, дату, час, аудиторію, матеріали, статус рецензій або відгуків.

Авторизація повинна здійснюватися через корпоративну електронну адресу на базі Google Workspace / Gmail-домену закладу.

**5. Основні модулі системи**

**5.1. Модуль авторизації**

Система повинна підтримувати:

вхід через корпоративну Google-пошту;

перевірку домену електронної адреси;

автоматичне створення локального користувача після першого входу;

призначення ролі адміністратором або за попередньо заданими правилами;

заборону доступу користувачам поза корпоративним доменом.

**5.2. Модуль імпорту ЄДЕБО**

Модуль повинен забезпечити:

завантаження CSV-файлів;

визначення кодування;

перевірку структури колонок;

перевірку обов’язкових полів;

перевірку дат початку і завершення навчання;

виявлення дублікатів;

попередній перегляд імпорту;

звіт про помилки;

збереження валідних даних у базі.

Дата завершення підготовки аспіранта після первинного внесення повинна вважатися опорною і не перезаписуватися автоматично повторним імпортом без явного підтвердження оператора.

**5.3. Картка аспіранта**

Картка аспіранта є центральним екраном системи.

Вона повинна містити:

ПІБ аспіранта;

освітньо-наукову програму;

спеціальність;

кафедру;

форму навчання;

дату початку підготовки;

дату завершення підготовки;

тему дисертації;

наукового керівника;

статус проходження атестації;

перелік етапів;

документи;

склад разової ради;

історію змін;

попередження про нормативні ризики.

**5.4. Модуль етапів атестації**

Система повинна підтримувати повний набір етапів:

Оформлення дисертації та перевірка публікацій.

Визначення стану готовності дисертації.

Отримання довідки про виконання ОНП і висновку керівника.
Заява про отримання висновку щодо новизни.
Публічна презентація результатів.
Надання висновку про наукову новизну.
Формування пропозицій щодо складу разової ради.
Накладення КЕП на PDF/A-файл дисертації.
Заява до вченої ради про утворення разової ради.
Рішення вченої ради про утворення ради.
Наказ керівника закладу.
Оприлюднення матеріалів і внесення даних до NAQA.Svr.
Перевірка МОН складу ради.
Подання рецензій і відгуків.
Призначення дати, часу і місця захисту.
Публічний захист.
Голосування та оформлення рішення ради.
Оприлюднення рішення і відеозапису.
Наказ про видачу диплома.
Архівування матеріалів і забезпечення відкритого доступу.
Для кожного етапу повинні зберігатися:
назва;
відповідальний;
планова дата;
фактична дата;
нормативне джерело;
статус;
залежність від іншого етапу;
правило розрахунку строку;
коментар оператора.

**5.5. Модуль нормативного контролю**

Це ключовий модуль системи.
Він повинен:
автоматично розраховувати планові дати;
перераховувати майбутні етапи після фіксації фактичної дати події;
контролювати мінімальні та максимальні строки;
виявляти порушення;
показувати ризики наближення граничних строків;
формувати список проблем по аспіранту;
формувати загальний список проблем по всіх аспірантах.
Статуси контролю:
Заплановано;
Виконано вчасно;
Наближається граничний строк;
Прострочено;
Порушено нормативний строк;
Неможливо розрахувати через відсутність базової дати.

**5.6. Модуль разової спеціалізованої вченої ради**

Система повинна дозволяти:
створювати склад ради для конкретного аспіранта;
вказувати голову ради;
вказувати рецензентів;
вказувати офіційних опонентів;
перевіряти повноту складу ради;

зберігати письмові згоди;
перевіряти відсутність конфлікту інтересів;
формувати проєкт інформаційного повідомлення про разову раду.

**5.7. Модуль планування захисту**

Графік повинен формуватися насамперед для одного обраного аспіранта.
Оператор має обрати аспіранта, після чого система повинна показати:
допустимий період захисту;
доступні аудиторії;
доступність членів ради;
нормативні обмеження;
рекомендовані дати;
причини, якщо графік сформувати неможливо.
Для захистів використовуються аудиторії:
0314;
0509;
1214.
Тривалість одного захисту — не менше 3 годин.
Система повинна перевіряти:
зайнятість аудиторії;
зайнятість членів ради;
нормативне вікно захисту;
тривалість захисту;
накладання подій;
наявність усіх попередніх етапів;
наявність рецензій і відгуків;
завершення перевірки МОН або настання відповідного строку.

**5.8. Модуль динамічного перепланування**

Система повинна дозволяти:
змінити дату події;
зафіксувати подію як незмінну;
перенести захист;
перерахувати залежні етапи;
зберегти попередню версію графіка;
порівняти стару і нову версію;
показати, які дати змінилися;
пояснити причину неможливості перепланування.

**5.9. Модуль документів**

Для кожного аспіранта та етапу система повинна дозволяти прикріплювати або реєструвати:
заяву;
довідку про виконання ОНП;
висновок наукового керівника;
висновок про наукову новизну;
PDF/A дисертації;
КЕП;
згоди членів ради;
рецензії;
відгуки;
рішення ради;
накази;
посилання на відеозапис;
посилання на оприлюднені матеріали.

**6. Вимоги до інтерфейсу**

Інтерфейс повинен бути не технічним, а адміністративно зрозумілим.

Основні сторінки:

Головна панель

Загальна кількість аспірантів, активні захисти, прострочені строки, ризики, найближчі події.

Аспіранти

Таблиця з пошуком, фільтрами і переходом у картку аспіранта.

Картка аспіранта

Основний робочий екран.

Етапи аспіранта

Вертикальна шкала етапів із плановими і фактичними датами.

Нормативний контроль

Список порушень і ризиків.

Разова рада

Склад ради, ролі, документи, перевірки.

Планування захисту

Вибір аспіранта, допустимі дати, аудиторії, члени ради, формування графіка.

Календар захистів

Загальний календар усіх захистів.

Документи

Документи по аспірантах і етапах.

Налаштування

Аудиторії, користувачі, ролі, нормативні правила.

**7. Вимоги до бази даних**

База даних повинна містити сутності:

користувачі;

ролі;

аспіранти;

освітні програми;

кафедри;

етапи атестації;

нормативні правила;

фактичні події;

документи;

разові ради;

члени рад;

доступність членів рад;

аудиторії;

часові слоти;

графіки;

версії графіків;

журнал дій.

**8. Вимоги до backend**

Backend повинен бути реалізований на Python з використанням FastAPI.

Потрібні API:

авторизація через Google;

імпорт CSV;

перегляд аспірантів;

перегляд картки аспіранта;

формування етапів;

фіксація фактичної дати події;

перерахунок плану;

нормативний контроль;
створення разової ради;
планування захисту одного аспіранта;
перепланування;
перегляд календаря;
підготовка інформаційних матеріалів;
журнал дій.

**9. Вимоги до frontend**

Frontend повинен бути побудований як повноцінний web-інтерфейс, а не як одна технічна сторінка з таблицями.

Бажана структура:
ліва навігація;
окрема картка аспіранта;
візуальна шкала етапів;
кольоровий статус нормативного контролю;
календар;
модальні вікна для фіксації подій;
зрозумілі попередження;
фільтри;
пошук;
кнопки дій відповідно до ролі користувача.

**10. Нефункціональні вимоги**

Система повинна:
працювати у браузері;
підтримувати PostgreSQL;
запускатися через Docker Compose;
мати захищену авторизацію;
зберігати історію змін;
не втрачати попередні версії графіків;
мати зрозумілі повідомлення про помилки;
підтримувати українську мову інтерфейсу;
бути придатною для демонстрації в межах бакалаврської роботи.

**11. Очікуваний результат**

Результатом розробки має бути web-додаток, у якому оператор відділу аспірантури може:

Увійти через корпоративну пошту.
Імпортувати аспірантів з ЄДЕБО.
Відкрити картку конкретного аспіранта.
Побачити повний нормативний план атестації.
Зафіксувати фактичну дату події.
Отримати автоматичний перерахунок наступних етапів.
Побачити порушення або ризики строків.
Сформувати склад разової ради.
Обрати дату, час і аудиторію захисту.
Сформувати графік захисту конкретного аспіранта.
Перепланувати графік у разі зміни обставин.
Підготувати інформаційні матеріали.
Опублікувати або передати графік для подальшого використання.

**12. Критерії приймання**

Проєкт можна вважати виконаним, якщо:
імпорт із CSV працює коректно;
для аспіранта формується 20 нормативних етапів;

дата завершення підготовки є незмінною опорною датою;
після фіксації фактичної дати система перераховує наступні етапи;
система виявляє порушення нормативних строків;
графік формується для одного обраного аспіранта;
захист не може бути запланований поза нормативним вікном;
тривалість захисту становить не менше 3 годин;
аудиторії 0314, 0509, 1214 використовуються як доступні місця захисту;
є авторизація через корпоративну Google-пошту;
є зрозумілий web-інтерфейс;
система запускається через Docker Compose;
реалізовано тести основного функціоналу.

# ДОДАТОК Д
# ER-СХЕМА І СЛОВНИК ДАНИХ

У додатку наведено узагальнену ER-схему та словник даних PostgreSQL-бази web-додатка. Схема побудована за фактичними ORM-моделями SQLAlchemy, які використовуються застосунком.

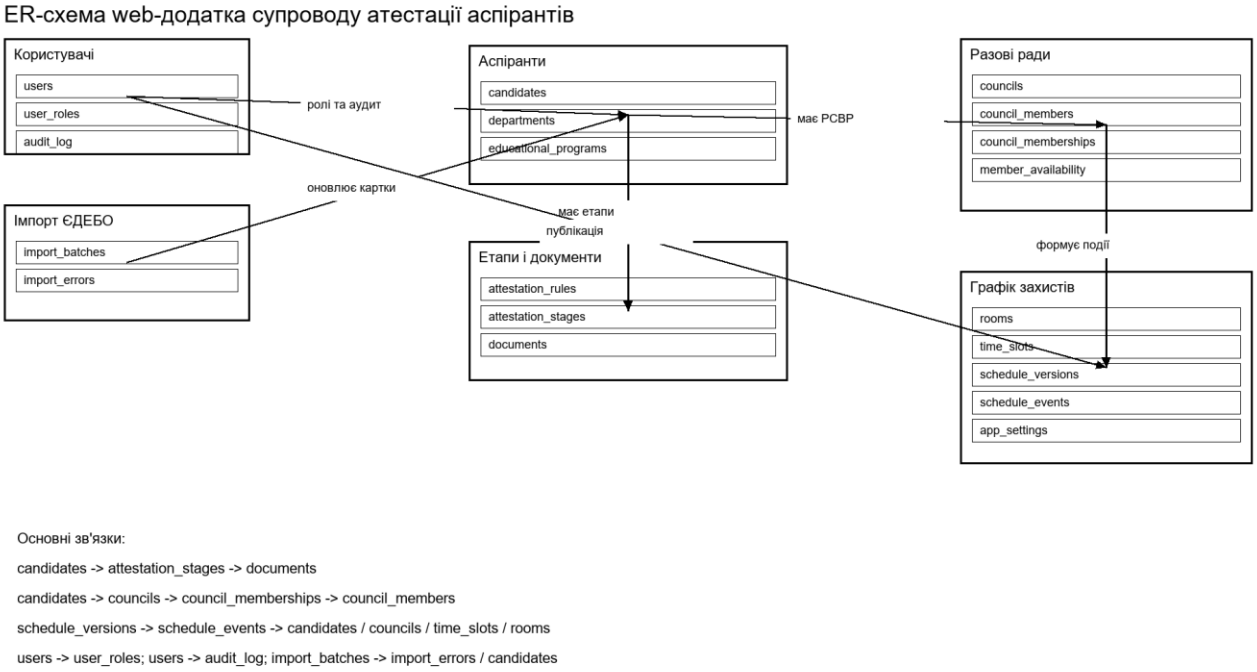


Рисунок Д.1 - ER-схема бази даних web-додатка

**Таблиця Д.1 - Основні зв'язки ER-моделі**

| Зв'язок | Тип | Зміст |
|---|---|---|
| users -> user_roles | 1:M | Користувач може мати декілька ролей. |
| users -> candidates | 1:0..1 | Користувач може бути пов'язаний із карткою аспіранта. |
| departments -> candidates | 1:M | Підрозділ містить картки аспірантів. |
| educational_programs -> candidates | 1:M | Освітня програма пов'язана з багатьма аспірантами. |
| import_batches -> import_errors | 1:M | Пакет імпорту містить помилки перевірки. |
| import_batches -> candidates | 1:M | Імпорт є джерелом оновлення карток. |
| candidates -> attestation_stages | 1:M | Для аспіранта формується послідовність етапів. |
| attestation_rules -> attestation_stages | 1:M | Етап може бути розрахований за нормативним правилом. |
| attestation_stages -> documents | 1:M | До етапу можуть додаватися документи. |
| candidates -> councils | 1:M | Аспірант може мати історію утворення РСВР. |
| councils -> council_memberships | 1:M | Рада має склад учасників. |
| council_members -> council_memberships | 1:M | Одна особа може входити до різних рад. |
| council_members -> member_availability | 1:M | Для члена ради фіксуються інтервали доступності. |
| schedule_versions -> schedule_events | 1:M | Версія графіка складається з подій. |

| Зв'язок | Тип | Зміст |
|---|---|---|
| schedule_events -> candidates/councils/time_slots/rooms | M:1 | Подія захисту пов'язує аспіранта, раду, час і аудиторію. |

## Таблиця Д.2 - Словник сутностей бази даних

| Таблиця | PK | FK | Призначення |
|---|---|---|---|
| users | id | - | облікові записи користувачів і зв'язок із Google Workspace |
| user_roles | user_id, role | - | призначення ролей користувачам |
| departments | id | - | довідник структурних підрозділів |
| educational_programs | id | - | освітньо-наукові програми та спеціальності |
| import_batches | id | - | пакети імпорту CSV-файлів ЄДЕБО |
| import_errors | id | - | помилки, виявлені під час перевірки CSV |
| candidates | id | - | картки аспірантів і опорні дати підготовки |
| attestation_rules | id | - | нормативні правила розрахунку строків |
| attestation_stages | id | - | індивідуальні етапи атестації аспіранта |
| documents | id | - | метадані службових документів |
| councils | id | - | разові спеціалізовані вчені ради |
| council_members | id | - | довідник членів рад |
| council_memberships | id | - | участь особи у конкретній раді та її роль |
| member_availability | id | - | доступність членів рад у часових інтервалах |
| rooms | id | - | аудиторії та онлайн-посилання для захистів |
| app_settings | key | - | операційні налаштування планування |
| time_slots | id | - | часові слоти захистів |
| schedule_versions | id | - | версії графіків захистів |
| schedule_events | id | - | окремі події захисту в межах версії графіка |
| audit_log | id | - | журнал критичних дій користувачів |

## Таблиця Д.3 - Словник полів бази даних

| Таблиця | Поле | Тип | Обмеження | Призначення |
|---|---|---|---|---|
| users | id | int | PK | внутрішній первинний ключ |
| users | email | str | UNIQUE, INDEX | електронна адреса користувача |
| users | full_name | str | - | повне ім'я особи |
| users | google_subject | str \| None | UNIQUE, NULL | стабільний ідентифікатор Google-акаунта |
| users | is_active | bool | NOT NULL | ознака активності запису |
| user_roles | user_id | int | PK | посилання на користувача |
| user_roles | role | UserRole | PK | роль у системі або раді |
| departments | id | int | PK | внутрішній первинний ключ |
| departments | name | str | UNIQUE | назва довідкового об'єкта |

| Таблиця | Поле | Тип | Обмеження | Призначення |
|---|---|---|---|---|
| educational_programs | id | int | PK | внутрішній первинний ключ |
| educational_programs | edebo_program_id | str \| None | NULL | ідентифікатор освітньої програми в ЄДЕБО |
| educational_programs | name | str | - | назва довідкового об'єкта |
| educational_programs | specialty | str | - | спеціальність або галузь спеціалізації |
| educational_programs | degree_level | str \| None | NULL | рівень освіти |
| import_batches | id | int | PK | внутрішній первинний ключ |
| import_batches | file_name | str | - | назва імпортованого файлу |
| import_batches | file_checksum | str | INDEX | контрольна сума файлу |
| import_batches | detected_encoding | str | - | визначене кодування CSV |
| import_batches | status | ImportStatus | - | поточний стан сутності |
| import_batches | total_rows | int | - | кількість рядків у файлі |
| import_batches | inserted_rows | int | - | кількість доданих записів |
| import_batches | updated_rows | int | - | кількість оновлених записів |
| import_batches | skipped_rows | int | - | кількість пропущених записів |
| import_batches | error_rows | int | - | кількість помилкових записів |
| import_batches | imported_by_id | int | - | користувач, який виконав імпорт |
| import_batches | applied_at | datetime \| None | NULL | дата й час застосування імпорту |
| import_errors | id | int | PK | внутрішній первинний ключ |
| import_errors | batch_id | int | INDEX | посилання на пакет імпорту |
| import_errors | row_number | int | - | номер рядка CSV |
| import_errors | field_name | str \| None | NULL | поле, у якому знайдено помилку |
| import_errors | message | str | - | текст повідомлення |
| import_errors | raw_value | str \| None | NULL | початкове значення з файлу |
| candidates | id | int | PK | внутрішній первинний ключ |
| candidates | edebo_card_id | str | UNIQUE, INDEX | унікальний ідентифікатор картки ЄДЕБО |
| candidates | full_name | str | INDEX | повне ім'я особи |
| candidates | study_status | str | - | статус навчання |
| candidates | study_start | date \| None | NULL | дата початку підготовки |
| candidates | study_end | date \| None | INDEX, NULL | дата завершення підготовки |
| candidates | study_form | str \| None | NULL | форма навчання |
| candidates | course | str \| None | NULL | курс |
| candidates | group_name | str \| None | NULL | група |
| candidates | restored_for_defense | bool | NOT NULL | ознака поновлення для захисту |
| candidates | source_updated_at | datetime \| None | NULL | час оновлення у джерелі |
| candidates | thesis_title | str \| None | NULL | назва дисертаційної роботи |
| candidates | supervisor_name | str \| None | NULL | науковий керівник |
| candidates | user_id | int \| None | UNIQUE, NULL | посилання на користувача |
| candidates | department_id | int \| None | NULL | посилання на структурний підрозділ |
| candidates | educational_program_id | int \| None | NULL | посилання на освітню програму |
| candidates | source_import_id | int \| None | NULL | посилання на останній пакет імпорту |

| Таблиця | Поле | Тип | Обмеження | Призначення |
|---|---|---|---|---|
| attestation_rules | id | int | PK | внутрішній первинний ключ |
| attestation_rules | code | str | UNIQUE | службовий код правила або етапу |
| attestation_rules | title | str | - | назва правила або етапу |
| attestation_rules | trigger_code | str | - | базова подія для розрахунку строку |
| attestation_rules | duration_value | int | - | числове значення тривалості |
| attestation_rules | unit | RuleUnit | - | одиниця вимірювання строку |
| attestation_rules | direction | RuleDirection | - | напрям відліку строку |
| attestation_rules | source_reference | str | - | нормативне посилання |
| attestation_rules | effective_from | date | - | дата початку чинності правила |
| attestation_rules | effective_to | date \| None | NULL | дата завершення чинності правила |
| attestation_rules | is_active | bool | - | ознака активності запису |
| attestation_stages | id | int | PK | внутрішній первинний ключ |
| attestation_stages | candidate_id | int | INDEX | посилання на аспіранта |
| attestation_stages | rule_id | int \| None | NULL | посилання на нормативне правило |
| attestation_stages | code | str | - | службовий код правила або етапу |
| attestation_stages | title | str | - | назва правила або етапу |
| attestation_stages | planned_date | date \| None | INDEX, NULL | планова дата |
| attestation_stages | actual_date | date \| None | NULL | фактична дата |
| attestation_stages | status | StageStatus | INDEX | поточний стан сутності |
| attestation_stages | responsible_user_id | int \| None | NULL | відповідальний користувач |
| attestation_stages | comment | str \| None | NULL | службовий коментар |
| documents | id | int | PK | внутрішній первинний ключ |
| documents | stage_id | int | INDEX | посилання на етап |
| documents | uploaded_by_id | int | - | користувач, який додав документ |
| documents | original_name | str | - | початкова назва документа |
| documents | storage_key | str | UNIQUE | службовий ключ сховища |
| documents | content_type | str | - | MIME-тип документа |
| documents | size_bytes | int | - | розмір файла в байтах |
| documents | checksum_sha256 | str | INDEX | контрольна сума SHA-256 |
| councils | id | int | PK | внутрішній первинний ключ |
| councils | candidate_id | int | INDEX | посилання на аспіранта |
| councils | sequence_number | int | - | порядковий номер ради |
| councils | status | CouncilStatus | - | поточний стан сутності |
| councils | order_number | str \| None | NULL | номер наказу |
| councils | order_date | date \| None | NULL | дата наказу |
| councils | publication_date | date \| None | NULL | дата оприлюднення |
| councils | public_details | dict[str, Any | NOT NULL | публічні реквізити ради у форматі JSON |
| council_members | id | int | PK | внутрішній первинний ключ |
| council_members | user_id | int \| None | UNIQUE, NULL | посилання на користувача |
| council_members | full_name | str | INDEX | повне ім'я особи |
| council_members | email | str \| None | NULL | електронна адреса користувача |

| Таблиця | Поле | Тип | Обмеження | Призначення |
|---|---|---|---|---|
| council_members | institution | str | - | установа члена ради |
| council_members | academic_degree | str \| None | NULL | науковий ступінь |
| council_members | specialty | str \| None | NULL | спеціальність або галузь спеціалізації |
| council_members | position | str \| None | NULL | посада |
| council_members | profile_url | str \| None | NULL | посилання на профіль |
| council_memberships | id | int | PK | внутрішній первинний ключ |
| council_memberships | council_id | int | INDEX | посилання на разову раду |
| council_memberships | member_id | int | INDEX | посилання на члена ради |
| council_memberships | role | CouncilRole | - | роль у системі або раді |
| council_memberships | is_external | bool | - | ознака зовнішнього учасника |
| council_memberships | qualification_confirmed | bool | - | підтвердження кваліфікації |
| council_memberships | conflict_of_interest_absent | bool | - | підтвердження відсутності конфлікту інтересів |
| member_availability | id | int | PK | внутрішній первинний ключ |
| member_availability | member_id | int | INDEX | посилання на члена ради |
| member_availability | starts_at | datetime | - | початок часового інтервалу |
| member_availability | ends_at | datetime | - | завершення часового інтервалу |
| member_availability | kind | AvailabilityKind | - | тип доступності |
| rooms | id | int | PK | внутрішній первинний ключ |
| rooms | name | str | UNIQUE | назва довідкового об'єкта |
| rooms | location | str \| None | NULL | місце проведення |
| rooms | online_url | str \| None | NULL | посилання на онлайн-трансляцію або конференцію |
| rooms | capacity | int | - | місткість аудиторії |
| rooms | is_active | bool | - | ознака активності запису |
| app_settings | key | str | PK | ключ налаштування |
| app_settings | value | dict[str, Any | NOT NULL | значення налаштування |
| app_settings | description | str \| None | NULL | опис налаштування |
| time_slots | id | int | PK | внутрішній первинний ключ |
| time_slots | starts_at | datetime | INDEX | початок часового інтервалу |
| time_slots | ends_at | datetime | - | завершення часового інтервалу |
| time_slots | is_active | bool | - | ознака активності запису |
| schedule_versions | id | int | PK | внутрішній первинний ключ |
| schedule_versions | version_number | int | - | номер версії графіка |
| schedule_versions | status | ScheduleStatus | INDEX | поточний стан сутності |
| schedule_versions | period_start | date | - | початок періоду планування |
| schedule_versions | period_end | date | - | кінець періоду планування |
| schedule_versions | score | int \| None | NULL | оцінка планувальника |
| schedule_versions | created_by_id | int | - | автор версії |
| schedule_versions | published_at | datetime \| None | NULL | дата публікації |
| schedule_events | id | int | PK | внутрішній первинний ключ |
| schedule_events | schedule_version_id | int | INDEX | посилання на версію графіка |

| Таблиця | Поле | Тип | Обмеження | Призначення |
|---|---|---|---|---|
| schedule_events | candidate_id | int | INDEX | посилання на аспіранта |
| schedule_events | council_id | int | INDEX | посилання на разову раду |
| schedule_events | slot_id | int | INDEX | посилання на часовий слот |
| schedule_events | room_id | int | INDEX | посилання на аудиторію |
| schedule_events | status | ScheduleEventStatus | - | поточний стан сутності |
| schedule_events | is_locked | bool | - | ознака зафіксованої події |
| schedule_events | change_penalty | int | - | штраф за зміну призначення |
| audit_log | id | int | PK | внутрішній первинний ключ |
| audit_log | actor_id | int \| None | INDEX, NULL | користувач, який виконав дію |
| audit_log | action | str | INDEX | тип дії |
| audit_log | entity_type | str | - | тип сутності |
| audit_log | entity_id | str \| None | NULL | ідентифікатор сутності |
| audit_log | occurred_at | datetime | INDEX | час виконання дії |
| audit_log | changes | dict[str, Any] \| None | NULL | опис змін у JSON |
| audit_log | ip_address | str \| None | NULL | IP-адреса користувача |

# ДОДАТОК Е

## REST API / OPENAPI

У додатку наведено витяг зі специфікації OpenAPI, сформованої FastAPI-додатком. Повна машинозчитувана специфікація доступна за маршрутом /openapi.json, а інтерактивна документація - за маршрутом /docs.

API використовує префікс /api/v1. Службові маршрути захищені JWT-токеном у заголовку Authorization. Публічні сторінки та оприлюднені матеріали РСВР доступні без службового токена.

**Таблиця Е.1 - Групи маршрутів OpenAPI**

| Група | Кількість маршрутів |
|---|---|
| attestation | 4 |
| audit | 1 |
| authentication | 1 |
| candidates | 4 |
| councils | 4 |
| documents | 1 |
| imports | 2 |
| publication | 1 |
| schedules | 11 |
| settings | 5 |
| system | 1 |
| workspace | 1 |

**Таблиця Е.2 - Витяг REST API**

| Метод | Маршрут | Група | Доступ | Коди |
|---|---|---|---|---|
| POST | /api/v1/auth/demo-token | authentication | Публічний / користувач | 200, 422 |
| POST | /api/v1/imports/validate | imports | Адміністратор, оператор | 200, 422 |
| GET | /api/v1/workspace | workspace | Автентифікований користувач | 200 |
| GET | /api/v1/settings | settings | Адміністратор; частково оператор | 200 |
| PATCH | /api/v1/settings/operational | settings | Адміністратор; частково оператор | 200, 422 |
| POST | /api/v1/settings/rooms | settings | Адміністратор; частково оператор | 201, 422 |
| PATCH | /api/v1/settings/rooms/{room_id} | settings | Адміністратор; частково оператор | 200, 422 |
| DELETE | /api/v1/settings/rooms/{room_id} | settings | Адміністратор; частково оператор | 200, 422 |
| GET | /api/v1/candidates | candidates | Адміністратор, оператор | 200 |
| GET | /api/v1/candidates/{candidate_id} | candidates | Адміністратор, оператор | 200, 422 |
| PATCH | /api/v1/candidates/{candidate_id} | candidates | Адміністратор, оператор | 200, 422 |
| DELETE | /api/v1/candidates/{candidate_id} | candidates | Адміністратор, оператор | 200, 422 |
| PUT | /api/v1/candidates/{candidate_id}/council | councils | Адміністратор, оператор | 200, 422 |
| PATCH | /api/v1/candidates/{candidate_id}/council/metadata | councils | Адміністратор, оператор | 200, 422 |
| GET | /api/v1/stages | attestation | Адміністратор, оператор | 200 |
| GET | /api/v1/attestation/normative-control | attestation | Адміністратор, оператор | 200 |

| Метод | Маршрут | Група | Доступ | Коди |
|---|---|---|---|---|
| POST | /api/v1/attestation/plans/generate | attestation | Адміністратор, оператор | 200 |
| POST | /api/v1/stages/{stage_id}/actual-date | attestation | Адміністратор, оператор | 200, 422 |
| POST | /api/v1/stages/{stage_id}/documents | documents | Адміністратор, оператор | 200, 422 |
| POST | /api/v1/imports/apply | imports | Адміністратор, оператор | 200, 422 |
| GET | /api/v1/schedules | schedules | Адміністратор, оператор | 200 |
| POST | /api/v1/schedules | schedules | Адміністратор, оператор | 201, 422 |
| POST | /api/v1/schedules/generate | schedules | Адміністратор, оператор | 201, 422 |
| POST | /api/v1/schedules/demo | schedules | Адміністратор, оператор | 201, 422 |
| GET | /api/v1/schedules/{schedule_id} | schedules | Адміністратор, оператор | 200, 422 |
| DELETE | /api/v1/schedules/{schedule_id} | schedules | Адміністратор, оператор | 200, 422 |
| POST | /api/v1/schedules/{schedule_id}/approve | schedules | Адміністратор, оператор | 200, 422 |
| POST | /api/v1/schedules/{schedule_id}/publish | schedules | Адміністратор, оператор | 200, 422 |
| POST | /api/v1/schedules/{schedule_id}/replan | schedules | Адміністратор, оператор | 201, 422 |
| POST | /api/v1/schedules/{schedule_id}/cancel | schedules | Адміністратор, оператор | 200, 422 |
| GET | /api/v1/schedules/{schedule_id}/conflicts | schedules | Адміністратор, оператор | 200, 422 |
| GET | /api/v1/public/schedules/{schedule_id} | publication | Публічний доступ | 200, 422 |
| GET | /api/v1/councils | councils | Адміністратор, оператор | 200 |
| GET | /api/v1/councils/{council_id}/materials | councils | Адміністратор, оператор | 200, 422 |
| GET | /api/v1/audit-log | audit | Адміністратор | 200 |
| GET | /health | system | Автентифікований користувач | 200 |

Основні коди відповідей: 200 - успішне виконання; 201 - створення ресурсу; 401 - користувач не автентифікований; 403 - недостатньо прав; 404 - об'єкт не знайдено; 409 - конфлікт станів; 422 - помилка структури або значень вхідних даних.

# ДОДАТОК Ж

# ПРОТОКОЛИ ТЕСТУВАННЯ

Тестування виконано у середовищі розробки проєкту командою python -m pytest -q. Додатково виконано експериментальне оцінювання планувальника командою python run_benchmarks.py.

**Таблиця Ж.1 - Загальний протокол автоматизованого тестування**

| Параметр | Значення |
|---|---|
| Дата виконання | 15.06.2026 |
| Середовище | Windows, локальне .venv проєкту; web-сервіс FastAPI; PostgreSQL у Docker Compose |
| Команда модульних та інтеграційних тестів | python -m pytest -q |
| Результат | 39 passed, 1 warning in 8.49s |
| Попередження | StarletteDeprecationWarning у fastapi.testclient.py; не впливає на функціональність додатка |

**Таблиця Ж.2 - Структура автоматизованих тестів**

| Файл | Кількість | Що перевіряється |
|---|---|---|
| tests/test_main.py | 1 | Перевірка /health endpoint. |
| tests/test_api.py | 10 | Авторизація, ролі, графіки, публікація, імпорт, сторінки UI та OpenAPI. |
| tests/test_importer.py | 4 | Кодування CSV, дублікати, неправильні дати, відсутні колонки. |
| tests/test_import_repository.py | 1 | Транзакційність та ідемпотентність імпорту. |
| tests/test_attestation_workflow.py | 4 | Повний сценарій атестації, склад РСВР, налаштування аудиторій, 15-денний deadline. |
| tests/test_models.py | 20 | Структура ORM-моделей, обмеження та перелічення. |

**Таблиця Ж.3 - Експериментальне оцінювання планувальника**

| Аспірантів | Слотів | Аудиторій | Статус | Оцінка | Вузлів | Median, мс | P95, мс |
|---|---|---|---|---|---|---|---|
| 8 | 8 | 3 | OPTIMAL | 0 | 139 | 0.933 | 1.431 |
| 12 | 12 | 3 | OPTIMAL | 0 | 295 | 2.178 | 3.12 |
| 16 | 16 | 3 | OPTIMAL | 0 | 519 | 3.909 | 5.007 |

**Таблиця Ж.4 - Протокол динамічного перепланування**

| Показник | Значення |
|---|---|
| Статус | OPTIMAL |
| Зафіксованих призначень | 3 |
| Змінених призначень | 0 |
| Повторень | 30 |
| Median, мс | 1.482 |
| P95, мс | 2.424 |
| Оцінка | 0 |

Результати підтверджують працездатність ключових функцій: імпорту ЄДЕБО, нормативного контролю, перевірки складу РСВР, формування графіка, перепланування, рольового доступу та публікації матеріалів.

# ДОДАТОК З
# КОРОТКА ІНСТРУКЦІЯ ОПЕРАТОРА WEB-СЕРВІСУ

Інструкція призначена для працівника відділу аспірантури і докторантури, який веде картки аспірантів, фіксує фактичні події, формує графіки захистів і готує матеріали про РСВР.

**1. Вхід до системи**

Відкрити web-додаток у браузері.

Натиснути кнопку входу через корпоративну пошту або, у демонстраційному режимі, отримати демо-токен адміністратора.

Перевірити, що у верхній частині інтерфейсу відображається активна роль користувача.

**2. Імпорт даних ЄДЕБО**

Перейти до розділу «Імпорт ЄДЕБО».

Вибрати CSV-файл експорту ЄДЕБО.

Натиснути «Перевірити» та переглянути звіт про структуру, кодування, помилки дат і дублікати.

Якщо звіт не містить критичних помилок, натиснути «Застосувати імпорт».

Після імпорту перейти до списку аспірантів і перевірити створені або оновлені картки.

**3. Робота з карткою аспіранта**

У розділі «Аспіранти» знайти потрібного аспіранта за ПІБ або ID.

Відкрити картку та перевірити дату завершення підготовки, спеціальність, освітню програму й форму навчання.

За потреби уточнити тему дисертації, наукового керівника та службові реквізити.

Переглянути допустиме вікно захисту. Система контролює, щоб захист відбувся не пізніше ніж за 15 днів до завершення підготовки.

**4. Фіксація етапів атестації**

У блоці етапів обрати подію, яка фактично настала.

Внести фактичну дату та, за потреби, службовий коментар.

Після збереження перевірити перераховані майбутні етапи й попередження про ризики.

Додати посилання або метадані документа, якщо етап підтверджується файлом.

**5. Заповнення складу РСВР**

У картці аспіранта перейти до блоку «Разова рада».

Заповнити голову ради, рецензентів і офіційних опонентів.

Перевірити допустимий склад: 2 рецензенти і 2 опоненти або 1 рецензент і 3 опоненти.

Переконатися, що голова і рецензенти належать до ІФНТУНГ, а опоненти є зовнішніми і представляють різні заклади.

Натиснути кнопку підготовки інформаційних матеріалів РСВР.

**6. Формування графіка захисту**

У блоці планування вибрати дату початку, аудиторію або режим автоматичного вибору.

За потреби змінити горизонт планування і максимальну кількість захистів члена ради на день.

Натиснути «Сформувати захист».

Якщо система повідомляє про конфлікт, перевірити доступність аудиторій, склад ради, накладання членів ради та нормативне вікно.

**7. Календар, погодження і публікація**

Перейти до розділу «Календар захистів».

Відкрити потрібну версію графіка та переглянути події.

Для чернетки можна виконати перевірку конфліктів, погодження, публікацію або видалення.

Публікувати слід лише перевірену версію, у якій заповнено дату, час, аудиторію, склад ради та публічні реквізити.

**8. Налаштування**

У розділі «Налаштування» можна додати або вимкнути аудиторії.
Змінити тривалість засідання РСВР, стартові години захистів, горизонт планування та денні ліміти. Типове значення тривалості захисту - 180 хвилин.
Після зміни налаштувань повторити формування або перепланування графіка.

**Таблиця 3.1 - Типові проблеми оператора та дії**

| Проблема | Ймовірна причина | Дія оператора |
|---|---|---|
| Не імпортується CSV | Неправильне кодування, відсутні обов'язкові колонки або некоректні дати | Перевірити звіт імпорту і виправити файл або вибрати правильний експорт ЄДЕБО. |
| Не формується графік | Немає допустимого слота, конфлікт аудиторії або члена ради | Переглянути діагностику, змінити аудиторію, дату або склад ради. |
| Захист виходить за межі строків | Дата захисту пізніша ніж завершення підготовки мінус 15 днів | Обрати ранніший слот або переглянути стан попередніх етапів. |
| Опоненти не приймаються системою | Опоненти з одного закладу або з ІФНТУНГ | Замінити опонентів відповідно до вимог складу РСВР. |
| Публікація недоступна | Версія не погоджена або неповні публічні реквізити | Заповнити реквізити РСВР, погодити графік і повторити публікацію. |

## БІБЛІОГРАФІЧНА ДОВІДКА

Тема бакалаврської роботи: ***Розробка web-сервісу динамічного планування підсумкової атестації аспірантів на основі Python, PostgreSQL та Docker***

Обсяг пояснювальної записки 282 аркушів:

28 таблиць;

22 рисунків;

8 додатки.

Дата завершення роботи: *10 червня 2026 р*.

Підпис студента- _____ *Гуменюк А.Л.*