

БАКАЛАВРСЬКА РОБОТА

БР.КІ-31.00.00.000 ПЗ

Група КІ-22-2

Гоменюк Максим

2026

Міністерство освіти і науки України
Івано-Франківський національний технічний університет нафти і газу
Інститут інформаційних технологій
Кафедра комп'ютерних систем і мереж

Гоменюк Максим Володимирович

УДК 004.7

БАКАЛАВРСЬКА РОБОТА

Розробка веб-додатку моніторингу та контролю виконання
кваліфікаційних робіт студентами на базі Python та
технології контейнеризації

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 Гоменюк М.В.

(підпис, ініціали та прізвище 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

(посада)

С.І. Мельничук

(підпис)

(дата)

(ініціали та прізвище)

Івано-Франківськ – 2026 рік

Івано-Франківський національний технічний університет нафти і газу

(повне найменування вищого навчального закладу)

Факультет інформаційних технологій

Кафедра комп'ютерних систем і мереж

Освітній ступінь бакалавр

Спеціальність 123 – Комп'ютерна інженерія

(шифр і назва)

ЗАТВЕРДЖУЮ

Завідувач кафедри КСМ

(С.І. Мельничук)

«21» квітня 2026 року

З А В Д А Н Н Я

НА БАКАЛАВРСЬКУ РОБОТУ СТУДЕНТУ

Гоменюку Максиму Володимировичу

(прізвище, ім'я, по батькові)

1. Тема проекту (роботи) Розробка веб-додатку моніторингу та контролю виконання кваліфікаційних робіт студентами на базі Python та технології контейнеризації.

керівник проекту (роботи) Гуменюк Т. В., доцент.

затверджені наказом вищого навчального закладу від 21.04.2026 №180/7

2. Строк подання студентом роботи 11 червня 2026 р

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

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити) 1. Аналіз предметної області та вибір технології. 2. Проективання архітектури системи та бази даних. 3. Програмна реалізація та розгортання веб-додатку.

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

6. Консультанти розділів роботи

7. Дата видачі завдання 2 лютого 2026 р..

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломного проекту (роботи)	Строк виконання етапів проекту (роботи)	Примітка
1	<i>Збір інформації, вивчення літератури та пошук додаткової інформації</i>	<i>Лютий, 2026</i>	
2	<i>Аналіз предметної області та вибір технології</i>	<i>Березень, 2026</i>	
3	<i>Проектування архітектури системи та бази даних</i>	<i>Квітень, 2026</i>	
4	<i>Програмна реалізація та розгортання веб-додатку</i>	<i>Травень, 2026</i>	
5	<i>Оформлення пояснювальної записки</i>	<i>Червень, 2026</i>	

Студент _____
(підпис)

Гоменюк М. В.
(прізвище та ініціали)

Керівник роботи _____
(підпис)

Гоменюк Т. В.
(прізвище та ініціали)

АНОТАЦІЯ

Бакалаврська робота присвячена розробці спеціалізованого веб-додатку для автоматизації процесу моніторингу та контролю виконання кваліфікаційних робіт студентами кафедри. Актуальність роботи пояснюється відсутністю на ринку готових рішень, що поєднують спеціалізовану рольову модель, автоматичний розрахунок прогресу і генерацію академічної звітності в єдиному інструменті.

У роботі проведено аналіз існуючих систем управління проектами (Trello, Jira, GitHub Projects) та навчальних платформ (Google Classroom, Moodle), виявлено їх недоліки у контексті кафедрального дипломного процесу і сформовано повний перелік вимог до розроблюваної системи.

Спроековано трирівневу клієнт-серверну архітектуру з Blueprint-декомпозицією серверної частини на чотири модулі: автентифікації, адміністратора, керівника і студента. Розроблено реляційну модель бази даних у складі чотирьох сутностей із скінченим автоматом статусів центральної таблиці. Реалізовано алгоритм розрахунку прогресу на основі вагових коефіцієнтів підтверджених етапів зі складністю $O(n)$.

Практична реалізація виконана мовою Python з використанням мікрофреймворку Flask 3.0.3, СУБД PostgreSQL 15, ORM SQLAlchemy 2.0, CSS-фреймворку Bootstrap 5.3 та Jinja2-шаблонізатора. Асинхронна взаємодія між кабінетом керівника і сервером реалізована через нативний Fetch API. Генерація зведених звітів виконується у форматах CSV та Excel (openpyxl). Проведено функціональне тестування засобами pytest: 10 тест-кейсів, усі PASSED. Система упакована у Docker-контейнери і розгортається на Linux-сервері засобами Docker Compose.

Ключові слова: веб-додаток, моніторинг дипломних робіт, Flask, PostgreSQL, Docker, рольова модель доступу, прогрес виконання, кваліфікаційна робота.

ABSTRACT

The bachelor's thesis is dedicated to the development of a specialized web application for automating the monitoring and control of thesis execution by department students. The relevance of the work is driven by the absence of ready-made solutions that combine a specialized role model, automatic progress calculation, and academic report generation in a single tool.

The work includes an analysis of existing project management systems (Trello, Jira, GitHub Projects) and learning platforms (Google Classroom, Moodle), identifying their shortcomings in the context of the department's thesis supervision process, and forming a complete list of requirements for the system under development.

A three-tier client-server architecture was designed with Blueprint decomposition of the server side into four modules: authentication, administrator, supervisor, and student. A relational database model was developed comprising four entities with a finite state machine for the central table's status attribute. A progress calculation algorithm based on weighted coefficients of confirmed stages with $O(n)$ complexity was implemented.

The practical implementation was carried out in Python using the Flask 3.0.3 microframework, PostgreSQL 15 database, SQLAlchemy 2.0 ORM, Bootstrap 5.3 CSS framework, and Jinja2 template engine. Asynchronous interaction between the supervisor's dashboard and the server was implemented via the native Fetch API. Summary reports are generated in CSV and Excel formats using openpyxl. Functional testing was conducted using pytest: 10 test cases, all PASSED. The system is packaged in Docker containers and deployed on a Linux server via Docker Compose.

Keywords: web application, thesis monitoring, Flask, PostgreSQL, Docker, role-based access control, execution progress, bachelor's thesis.

ЗМІСТ

ВСТУП.....	5
РОЗДІЛ 1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ВИБІР ТЕХНОЛОГІЙ	8
1.1 Аналіз існуючих рішень для управління проєктами	8
1.2 Формування вимог до системи моніторингу кваліфікаційних робіт.....	10
1.3 Вибір та обґрунтування технологічного стеку.....	14
2 ПРОЄКТУВАННЯ АРХІТЕКТУРИ СИСТЕМИ ТА БАЗИ ДАНИХ	20
2.1 Проєктування архітектури клієнт-серверного веб-додатку.....	20
2.2 Розробка моделі бази даних	23
2.3 Проєктування алгоритму розрахунку прогресу	28
2.4 Проєктування системи ролей та розмежування прав доступу	31
3 ПРОГРАМНА РЕАЛІЗАЦІЯ ТА РОЗГОРТАННЯ ВЕБ-ДОДАТКУ	35
3.1 Реалізація серверної частини та взаємодія з БД	35
3.2 Розробка інтерфейсу користувача	40
3.3 Реалізація функціоналу відміток етапів та генерації статистики	53
3.4 Тестування роботи системи	57
3.5 Розгортання у Docker-контейнерах	60
ВИСНОВКИ.....	68
ПЕРЕЛІК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ	70
ДОДАТКИ	
БІБЛІОГРАФІЧНА ДОВІДКА	

					БР.КІ-31.00.00.000 ПЗ			
Змн.	Арк.	№ докум.	Підпис	Дата				
Розроб.		Гоменюк М.В.			Розробка веб-додатку моніторингу та контролю виконання кваліфікаційних робіт студентами на базі Python та технології контейнеризації	Літ.	Арк.	Аркуші
Перевір.		Гуменюк Т. В.					3	69
Реценз.		Гарасимів В.М				ІФНТУНГ, КІ-22-2		
Н. Контр.		Лазорів А.М.						
Затверд.		Мельничук С.І.						

ПЕРЕЛІК ОСНОВНИХ ПОЗНАЧЕНЬ І СКОРОЧЕНЬ

API (Application Programming Interface) - програмний інтерфейс застосунку.

БД (База даних).

ER-діаграма (Entity-Relationship diagram) - діаграма “сутність-зв'язок”.

HTML (HyperText Markup Language) - мова гіпертекстової розмітки.

HTTP (Hypertext Transfer Protocol) - протокол передачі гіпертексту.

CSS (Cascading Style Sheets) - каскадні таблиці стилів.

JS (JavaScript) - мова програмування для розробки інтерактивних елементів веб-сторінок.

JSON (JavaScript Object Notation) - формат обміну даними.

ORM (Object-Relational Mapping) - об'єктно-реляційне відображення.

REST (Representational State Transfer) - архітектурний стиль API.

СКБД / СУБД - Система (управління) баз даних.

UI (User Interface) - інтерфейс користувача.

UX (User Experience) - досвід користувача.

WSGI (Web Server Gateway Interface) - стандарт взаємодії Python-сервера з веб-сервером.

URL (Uniform Resource Locator) - уніфікований локатор ресурсу (веб-адреса).

ВСТУП

Актуальність теми. Виконання кваліфікаційної роботи є завершальним і найбільш відповідальним етапом підготовки бакалавра. Водночас традиційний механізм супроводу цього процесу на кафедрі — усні консультації, паперові графіки і ручний контроль з боку наукового керівника — позбавлений систематичності і прозорості: студент не завжди усвідомлює поточний стан виконання роботи, керівник витрачає значний час на з'ясування статусу кожного підопічного, а адміністрація кафедри не має зручного інструменту для формування оперативної звітності. Зростаючий рівень цифровізації вищої освіти і поширення дистанційного навчання роблять відсутність спеціалізованих інструментів моніторингу дипломного процесу особливо відчутною [1, 4].

Аналіз наявних систем управління проектами (Trello, Jira, GitHub Projects) та навчальних платформ (Google Classroom, Moodle) засвідчив, що жодна з них не реалізує повного набору функцій, необхідних для автоматизованого кафедрального контролю: поєднання спеціалізованої рольової моделі “студент / керівник / адміністратор”, автоматичного розрахунку відсотка виконання на основі вагових коефіцієнтів і генерації академічної звітності. Виявлена функціональна прогалина підтверджує потребу в розробці власного спеціалізованого веб-додатку.

Мета роботи — розробка веб-додатку для автоматизованого моніторингу та контролю виконання кваліфікаційних робіт студентами кафедри, що забезпечує прозору взаємодію між студентом і науковим керівником та автоматичний розрахунок поточного прогресу.

Для досягнення поставленої мети вирішено такі завдання:

- проведено аналіз існуючих систем управління проектами та навчальних платформ і обґрунтовано потребу у спеціалізованому рішенні;
- сформовано повний перелік функціональних і нефункціональних вимог до системи у розрізі трьох рівнів доступу;

					БР.КІ-31.00.00.000 ПЗ	Арк.
						5
Змн.	Арк.	№ докум.	Підпис	Дата		

- обрано та обґрунтовано технологічний стек: Python / Flask, PostgreSQL, SQLAlchemy, Bootstrap 5, Docker;
- спроектовано трирівневу клієнт-серверну архітектуру з Blueprint-декомпозицією серверної частини;
- розроблено реляційну модель бази даних у складі чотирьох сутностей і скінченний автомат станів етапу виконання;
- реалізовано алгоритм автоматичного розрахунку відсотка виконання роботи на основі вагових коефіцієнтів;
- розроблено рольові особисті кабінети студента, керівника і адміністратора з відповідним розмежуванням доступу;
- реалізовано функціонал асинхронної зміни статусу етапів (Fetch API) та генерацію зведеної звітності у форматах CSV і Excel;
- проведено функціональне тестування засобами pytest (10 тест-кейсів, усі PASSED);
- налаштовано контейнеризацію додатку засобами Docker та Docker Compose для розгортання на Linux-сервері.

Об'єкт дослідження — процес моніторингу та контролю виконання кваліфікаційних робіт студентами у вищому навчальному закладі.

Предмет дослідження — методи і засоби автоматизації взаємодії між студентом і науковим керівником на основі вебтехнологій розгортання програмного забезпечення у Docker-контейнерах.

Методи дослідження. У роботі застосовано методи системного аналізу предметної галузі, об'єктно-орієнтованого проектування (ORM-моделювання, Blueprint-декомпозиція), реляційного моделювання даних (ER-діаграми, нормалізація), теорії скінченних автоматів (машина станів етапу) та методи автоматизованого тестування програмного забезпечення.

Практична цінність роботи полягає у тому, що розроблений веб-додаток може бути безпосередньо впроваджений на кафедрі для автоматизації дипломного процесу: система розгортається на Linux-сервері однією командою

					БР.КІ-31.00.00.000 ПЗ	Арк.
						6
Змн.	Арк.	№ докум.	Підпис	Дата		

Docker Compose, не потребує ліцензійних витрат і адаптована до специфіки кафедрального планування із налаштовуваними ваговими коефіцієнтами етапів.

					БР.КІ-31.00.00.000 ПЗ	Арк.
						7
Змн.	Арк.	№ докум.	Підпис	Дата		

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ВИБІР ТЕХНОЛОГІЙ

1.1 Аналіз існуючих систем управління проектами

Перш ніж розпочати проектування власної системи, необхідно дослідити наявні інструменти управління завданнями та проектами, оцінити ступінь їх придатності для супроводу навчального процесу і виявити функціональні прогалини, що й обумовлюють доцільність розробки спеціалізованого рішення.

Trello — веб-орієнтована дошка завдань, побудована на методології Kanban [1]. Система надає можливість створювати картки, переміщати їх між колонками-статусами (“До виконання”, “В процесі”, “Готово”), призначати відповідальних та встановлювати дедлайни. Попри простоту і наочність, Trello позбавлена рольової моделі типу “студент / керівник”: будь-який учасник дошки може змінювати статус будь-якої картки, що унеможливлює формальний механізм підтвердження виконання. Крім того, система не обчислює відсотка прогресу на основі вагових коефіцієнтів і не генерує академічних звітів у форматах CSV або Excel.

Jira — потужна платформа від Atlassian для управління проектами у стилі Scrum і Kanban [2]. Jira вирізняється розвиненою системою налаштовуваних workflow, розгалуженою звітністю та підтримкою ролей на рівні проекту. Разом з тим, впровадження Jira в освітньому середовищі пов'язане з рядом практичних обмежень: платна модель доступу для інституційного використання, висока складність початкового налаштування та значна функціональна надлишковість, нерелевантна для задач кафедрального дипломного контролю. Відсутній також механізм формування зведеного академічного звіту у звичному для навчального закладу форматі.

GitHub Projects — інтегрований в екосистему GitHub інструмент управління завданнями [3]. Він забезпечує ефективне відстеження прогресу розробки коду і підтримує налаштовувані поля та подання. Проте платформа не передбачає формального навчального контексту: відсутній механізм

					БР.КІ-31.00.00.000 ПЗ	Арк.
						8
Змн.	Арк.	№ докум.	Підпис	Дата		

підтвердження виконання етапу уповноваженою особою (керівником), а звітність обмежується вбудованими графіками без можливості експорту у необхідний академічний формат. Крім того, обов'язкова прив'язка до системи контролю версій є логічною для ІТ-проектів, але надлишковою для управління дипломним процесом загалом.

Google Classroom — освітня платформа Google, орієнтована на організацію навчальних курсів і подачу завдань [4]. Дозволяє публікувати завдання, збирати файли від студентів та виставляти оцінки. Однак платформа не підтримує ієрархічні плани виконання з ваговими коефіцієнтами, не розраховує автоматичного агрегованого прогресу студента і не реалізує специфічного workflow «подача → перевірка → підтвердження / повернення на доопрацювання», характерного для кафедрального дипломного процесу.

Moodle — відкрита система управління навчанням (LMS), широко поширена у вищій школі [5]. Moodle підтримує ролі, завдання, оцінювання і плагіні звітності. Проте архітектура системи орієнтована на масові курси з однотипними завданнями, а не на індивідуальний супровід дипломної роботи. Розробка спеціалізованих плагінів під потреби кафедри потребує глибоких знань РНР-архітектури Moodle та суттєвих витрат часу на налаштування і підтримку, що нівелює переваги готового рішення.

На основі проведеного аналізу складено порівняльну характеристику розглянутих систем за ключовими критеріями, що визначають придатність інструменту для автоматизації дипломного процесу кафедри (таблиця 1.1).

Як свідчать дані таблиці 1.1, жодна з розглянутих систем не забезпечує повного набору функцій, необхідних для організації дипломного процесу кафедри: поєднання спеціалізованої рольової моделі, автоматичного розрахунку прогресу на основі вагових коефіцієнтів та формування академічної звітності. Це підтверджує потребу у власному рішенні.

					БР.КІ-31.00.00.000 ПЗ	Арк.
						9
Змн.	Арк.	№ докум.	Підпис	Дата		

Таблиця 1.1 – Порівняльний аналіз існуючих систем управління проектами

Критерій оцінювання	Trello	Jira	GitHub Projects	Google Classroom	Розроблений додаток
Спеціалізована рольова модель (студент / керівник / адміністратор)	—	Частково	—	Частково	+
Автоматичний розрахунок % прогресу	—	—	—	—	+
Вагові коефіцієнти для кожного етапу	—	—	—	—	+
Підтвердження етапу керівником зі зміною статусу	—	Частково	—	—	+
Коментар керівника при поверненні на доопрацювання	—	Частково	—	—	+
Генерація зведеного звіту CSV / Excel	—	Частково	—	—	+
Безкоштовне self-hosted розгортання	—	—	+	+	+
Адаптовано до кафедрального процесу	—	—	—	—	+

1.2 Формування вимог до системи моніторингу кваліфікаційних робіт

Формування вимог є фундаментальним етапом проектування програмного забезпечення, що визначає межі і зміст розроблюваної системи. На основі аналізу предметної галузі, виявлених недоліків існуючих рішень та консультацій з представниками кафедри комп'ютерних систем і мереж було сформульовано функціональні та нефункціональні вимоги до системи.

Функціональні вимоги описують поведінку системи та очікуваний результат виконання конкретних дій користувача. Вони структуровані відповідно до трьох рівнів доступу, передбачених у системі.

Функціональні вимоги для ролі “Студент” охоплюють такий перелік:

					БР.КІ-31.00.00.000 ПЗ	Арк.
						10
<i>Змн.</i>	<i>Арк.</i>	<i>№ докум.</i>	<i>Підпис</i>	<i>Дата</i>		

- авторизація в системі за адресою електронної пошти та паролем;
- перегляд затвердженого індивідуального плану виконання роботи із зазначенням переліку етапів та їх вагових коефіцієнтів;
- прикріплення до кожного етапу посилання на репозиторій (GitHub / GitLab) або файлу у форматі PDF, DOCX або ZIP;
- відправка матеріалів етапу на перевірку керівнику, що супроводжується зміною статусу на “Очікує підтвердження”;
- перегляд поточного відсотка загального прогресу у вигляді прогрес-бару;
- перегляд текстових коментарів керівника при поверненні етапу на доопрацювання.

Функціональні вимоги для ролі “Керівник” передбачають:

- перегляд дашборду з переліком прикріплених студентів і їх поточним прогресом;
- перегляд матеріалів, поданих студентом до конкретного етапу (файлів та посилань);
- підтвердження виконання етапу зі зміною статусу на «Виконано» або повернення на доопрацювання з обов'язковим текстовим коментарем;
- перегляд агрегованої статистики щодо прогресу групи підопічних студентів.

Функціональні вимоги для ролі “Адміністратор” включають:

- управління базою облікових записів: додавання студентів і викладачів, активація та деактивація записів;
- створення, редагування та деактивація шаблонів етапів з визначенням назви, вагового коефіцієнта та порядкового номера;
- прив'язка студента до наукового керівника і теми кваліфікаційної роботи;
- генерація та експорт зведеного звіту по всіх студентах у форматах CSV та Excel.

					БР.КІ-31.00.00.000 ПЗ	Арк.
						11
Змн.	Арк.	№ докум.	Підпис	Дата		

Нефункціональні вимоги визначають якісні характеристики системи, не пов'язані безпосередньо з функціями, проте критичні для її практичної придатності:

- коректне відображення інтерфейсу на десктопних і мобільних пристроях (адаптивний дизайн);
- захист від CSRF-атак на всіх формах системи;
- безпечне зберігання паролів з використанням алгоритму хешування PBKDF2 (бібліотека Werkzeug);
- час відповіді сервера на стандартний HTTP-запит — не більше 2 секунд при одночасній роботі до 50 користувачів;
- розмежування доступу між ролями: маршрути кожного рівня захищені декораторами перевірки ролі;
- можливість розгортання на сервері під управлінням Linux (Debian / Ubuntu) засобами Docker Compose.

Узагальнений перелік вимог систематизовано у таблиці 1.2 за типом і категорією.

Таблиця 1.2 – Функціональні та нефункціональні вимоги до системи

Тип вимоги	Категорія	Зміст вимоги
Функціональні	Роль «Студент»	Авторизація за email/паролем; перегляд плану з ваговими коефіцієнтами; прикріплення посилань (GitHub/GitLab) та файлів (PDF, DOCX, ZIP); відправка етапу на перевірку; перегляд прогрес-бару; перегляд коментарів керівника.
Функціональні	Роль «Керівник»	Дашборд зі списком студентів і їх прогресом; перегляд матеріалів етапу; підтвердження або повернення на доопрацювання з коментарем; агрегована статистика по групі.
Функціональні	Роль «Адміністратор»	Управління обліковими записами (додавання, активація/деактивація); створення та редагування шаблонів етапів із вагою; прив'язка студент ↔ керівник ↔ тема; генерація CSV/Excel-звіту.
Нефункціональні	Безпека	Хешування паролів (Werkzeug PBKDF2); захист від CSRF-атак (Flask-WTF); заборона доступу до маршрутів іншої ролі.

Тип вимоги	Категорія	Зміст вимоги
Нефункціональні	Продуктивність	Час відповіді на HTTP-запит ≤ 2 с при стандартному навантаженні; асинхронна зміна статусу через Fetch API без перезавантаження сторінки.
Нефункціональні	Розгортання	Пакування додатку та СУБД у Docker-контейнери; конфігурація через Docker Compose; підтримка Linux (Debian/Ubuntu); зберігання конфігурації у .env-файлі.

Сформований перелік вимог є вичерпною специфікацією, що безпосередньо визначає склад сутностей бази даних, логіку маршрутизації та набір контролерів Flask, котрі реалізуються в наступних розділах роботи.

1.3 Вибір та обґрунтування технологічного стеку

Вибір технологічного стеку є стратегічним рішенням, що визначає архітектурні можливості, швидкість розробки та довгострокову підтримуваність системи. Обґрунтований вибір здійснювався на основі сформованих вимог, зрілості технологій, якості документації та сумісності компонентів між собою.

1.3.1 Мова програмування Python та мікрофреймворк Flask. Python є однією з найпоширеніших мов програмування загального призначення у сфері веброзробки та аналізу даних [6]. Для реалізації серверної частини обрано мікрофреймворк Flask версії 3.0.3. На відміну від повнофункціонального Django, Flask не нав'язує конкретної структури проекту, не включає вбудованого ORM і не потребує попередньої конфігурації адмін-панелі, що робить його оптимальним вибором для навчальних і дослідницьких проектів із чітко визначеним обсягом функціоналу.

Ключовою перевагою Flask у цьому проєкті є система Blueprint — механізм логічної декомпозиції маршрутів на незалежні модулі. Чотири Blueprint-и: `auth_bp`, `admin_bp`, `supervisor_bp` та `student_bp` — відповідають чотирьом рольовим підсистемам і дозволяють реєструвати маршрути, шаблони та статичні файли кожного модуля окремо, що суттєво спрощує навігацію по кодовій базі та знижує ризик конфліктів між модулями [7].

Як альтернативу розглядався FastAPI — сучасний асинхронний фреймворк із автоматичною генерацією OpenAPI-документації. FastAPI демонструє вищу продуктивність при інтенсивних I/O-операціях завдяки підтримці `async/await`, однак екосистема розширень для аутентифікації і рендерингу HTML-шаблонів значно менш зріла порівняно з Flask. Оскільки система орієнтована на синхронне серверне рендерення HTML через Jinja2, а не на побудову SPA з окремим фронтендом, Flask забезпечує більш органічну інтеграцію всього стеку.

					БР.КІ-31.00.00.000 ПЗ	Арк.
						14
Змн.	Арк.	№ докум.	Підпис	Дата		

1.3.2 Система управління базами даних PostgreSQL та ORM SQLAlchemy. Для зберігання даних обрано реляційну СУБД PostgreSQL версії 15 [8]. Реляційна модель є природним вибором для цієї предметної галузі, оскільки між сутностями системи (користувачі, теми, шаблони етапів, виконання етапів) існують чітко визначені зв'язки “один-до-одного” та “один-до-багатьох”, цілісність яких забезпечується зовнішніми ключами. PostgreSQL вирізняється підтримкою типу ENUM для декларативного визначення статусів і ролей, повноцінною ACID-транзакційністю та надійною інтеграцією з Python через драйвер psycopg2.

У ролі альтернативи розглядалась MySQL 8.0. Обидві СУБД задовольняють функціональні вимоги, проте PostgreSQL має кращу підтримку складних типів даних і більш строгу відповідність стандарту SQL, що знижує ймовірність неочікуваної поведінки при роботі з перерахунками і транзакціями. Саме тому PostgreSQL визнано більш доцільним вибором.

Взаємодія з СУБД здійснюється через ORM SQLAlchemy версії 2.0 у зв'язці з Flask-SQLAlchemy 3.1.1 [9]. ORM дозволяє описувати таблиці бази даних у вигляді Python-класів, що усуває необхідність написання SQL-запитів у переважній більшості сценаріїв і забезпечує перевірку типів на рівні мови програмування. Міграції схеми бази даних автоматизовано за допомогою Flask-Migrate 4.0.7 на базі Alembic: кожна зміна моделей фіксується у версіонованому скрипті міграції, що дозволяє відтворити або відкатити схему на будь-якому середовищі.

1.3.3 Технології фронтенду: Bootstrap 5 та Jinja2. Побудова інтерфейсу користувача реалізована засобами серверного рендерингу з використанням шаблонізатора Jinja2, вбудованого у Flask [10]. Jinja2 реалізує принцип DRY через механізм наслідування шаблонів: базовий шаблон base.html визначає спільну HTML-структуру (навігаційну панель, контейнер контенту, блок повідомлень), яку дочірні шаблони розширюють через блоки {% block %}. Це

					БР.КІ-31.00.00.000 ПЗ	Арк.
						15
Змн.	Арк.	№ докум.	Підпис	Дата		

унеможливилює дублювання розмітки та забезпечує єдиний стиль між усіма сторінками кожного рольового кабінету.

Для стилізації інтерфейсу обрано Bootstrap 5.3 — CSS-фреймворк із компонентно-орієнтованою архітектурою [11]. Bootstrap надає адаптивну 12-колоночну сітку, готові компоненти (картки, таблиці, форми, модальні вікна, прогрес-бари, сповіщення), а також відповідає стандартам доступності WCAG 2.1. Прогрес-бар виконання роботи студента, дашборд керівника і таблиці адміністратора реалізовані виключно засобами Bootstrap без написання власного CSS, що прискорює розробку та забезпечує узгодженість зовнішнього вигляду.

Для асинхронних запитів при зміні статусу етапу керівником застосовано нативний браузерний Fetch API (стандарт ES2020). Це дозволяє оновлювати стан сторінки без її перезавантаження і знижує кількість HTTP-запитів, які повертають повну HTML-розмітку. Рішення обрано на користь нативного API замість jQuery, оскільки завдання обмежуються кількома асинхронними викликами і не потребують підключення сторонньої бібліотеки.

1.3.4 Контейнеризація засобами Docker та Docker Compose. Для забезпечення ізолюваного та відтворюваного середовища виконання обрано технологію контейнеризації Docker версії 25 [12]. Docker вирішує класичну проблему розбіжності середовищ розробки і виробничого сервера: весь додаток разом з залежностями, конфігурацією та версіями бібліотек описується у Dockerfile і запускається як ізолюваний контейнер на будь-якому Linux-хості без ручного встановлення залежностей.

Мультиконтейнерне оточення системи налаштовується через Docker Compose версії 3.9 [13]. Файл docker-compose.yml описує три сервіси: web (Flask-додаток під Gunicorn), db (PostgreSQL 15) та adminer (веб-інтерфейс для адміністрування бази даних). Healthcheck для сервісу бази даних гарантує, що Flask-контейнер не розпочне ініціалізацію до повної готовності PostgreSQL. Конфіденційні параметри (паролі, секретні ключі) передаються через .env-файл, що унеможливилює їх потрапляння у репозиторій.

					БР.КІ-31.00.00.000 ПЗ	Арк.
						16
Змн.	Арк.	№ докум.	Підпис	Дата		

У ролі WSGI-сервера для production-оточення обрано Gunicorn 22.0.0 — синхронний попередньо розгалужений сервер, що запускає Flask-додаток у кількох воркер-процесах і забезпечує стабільну роботу під навантаженням [14]. Gunicorn повністю сумісний з Linux і не потребує додаткових залежностей, що спрощує Dockerfile.

1.3.5 Інструмент тестування pytest. Для автоматизованого тестування серверної логіки обрано фреймворк pytest версії 8.2.2 [15]. pytest підтримує фікстури для ініціалізації тестового оточення (тестова база даних, тестовий клієнт Flask), параметризацію тест-кейсів і детальний вивід помилок. Тести розміщені у каталозі **tests/** і охоплюють критичні маршрути системи: авторизацію, зміну статусу етапу та розрахунок прогресу.

Зведений перелік технологій обраного стеку з їх версіями та призначенням наведено у таблиці 1.3.

Таблиця 1.3 – Обраний технологічний стек системи

Рівень / компонент	Технологія	Обґрунтування вибору
Бекенд — мова та фреймворк	Python 3.12 / Flask 3.0.3	Мікрофреймворк; Blueprint-декомпозиція; гнучкий вибір компонентів; широка екосистема
Альтернатива (бекенд)	FastAPI 0.111	Асинхронна обробка запитів; автоматична OpenAPI-документація; менша зрілість екосистеми
Аутентифікація	Flask-Login 0.6.3	Сесійна автентифікація; декоратор @login_required; інтеграція з Flask
Захист форм	Flask-WTF 1.2.1 / WTForms 3.1.2	CSRF-токени у формах; валідація на стороні сервера
СУБД	PostgreSQL 15	ACID-транзакції; тип ENUM; надійність; інтеграція з Python
Альтернатива (СУБД)	MySQL 8.0	Широке розповсюдження; менша підтримка складних типів даних
ORM	SQLAlchemy 2.0 / Flask-SQLAlchemy 3.1.1	Декларативні моделі; підтримка міграцій; абстракція від SQL
Міграції БД	Flask-Migrate 4.0.7 / Alembic	Версіонування схеми; автогенерація скриптів міграцій
Шаблонізатор	Jinja2 (вбудований у Flask)	Успадкування шаблонів; принцип DRY; контекстне рендерення

Рівень / компонент	Технологія	Обґрунтування вибору
CSS-фреймворк	Bootstrap 5.3	Адаптивна сітка; готові компоненти; WCAG-доступність
JS (асинхронність)	Fetch API (ES2020)	Зміна статусу без перезавантаження сторінки; нативний браузерний API
Звітність	openpyxl 3.1.3	Генерація .xlsx-файлів; стилізація комірок; без зовнішніх залежностей
Контейнеризація	Docker 25 / Docker Compose 3.9	Ізольовані контейнери; єдиний YAML-файл для оточення; відтворюваність
WSGI-сервер	Gunicorn 22.0.0	Production-ready; підтримка воркерів; сумісність з Linux
Тестування	pytest 8.2.2	Фікстури; параметризація тестів; зручна звітність

Сукупність обраних технологій утворює цілісний, добре інтегрований стек: Flask організовує маршрутизацію і взаємодію з усіма компонентами, SQLAlchemy абстрагує роботу з PostgreSQL, Jinja2 і Bootstrap забезпечують рендеринг інтерфейсу, а Docker ізолює оточення для надійного розгортання на цільовому Linux-сервері.

Висновки до розділу

У першому розділі проведено аналіз п'яти систем управління проектами та навчальними процесами: Trello, Jira, GitHub Projects, Google Classroom і Moodle. Встановлено, що жодна з розглянутих платформ не реалізує повного набору функцій, необхідних для кафедрального дипломного контролю: поєднання спеціалізованої рольової моделі «студент / керівник / адміністратор», автоматичного розрахунку прогресу на основі вагових коефіцієнтів та генерації академічної звітності у форматах CSV / Excel. Це підтверджує доцільність розробки власного спеціалізованого веб-додатку.

Сформовано повний перелік функціональних і нефункціональних вимог до системи у розрізі трьох ролей доступу. Вимоги безпосередньо визначають склад

					БР.КІ-31.00.00.000 ПЗ	Арк.
						18
Змн.	Арк.	№ докум.	Підпис	Дата		

сутностей бази даних, структуру маршрутизації та логіку бізнес-правил, що реалізуються у наступних розділах.

Обрано та обґрунтовано технологічний стек: Python 3.12 і Flask 3.0.3 для серверної логіки з Blueprint-декомпозицією, PostgreSQL 15 і SQLAlchemy 2.0 для реляційного зберігання даних із підтримкою ENUM-типів, Bootstrap 5.3 і Jinja2 для адаптивного серверного рендерингу інтерфейсу, Docker 25 і Docker Compose 3.9 для контейнеризації та відтворюваного розгортання, Gunicorn для production WSGI-сервера і pytest для автоматизованого тестування. Кожна технологія обрана на основі функціональних вимог, зрілості екосистеми та узгодженості з рештою стеку.

					БР.КІ-31.00.00.000 ПЗ	Арк.
						19
Змн.	Арк.	№ докум.	Підпис	Дата		

2 ПРОЕКТУВАННЯ АРХІТЕКТУРИ СИСТЕМИ ТА БАЗИ ДАНИХ

2.1 Проектування архітектури клієнт-серверного веб-додатку

Архітектурне рішення визначає всю подальшу програмну реалізацію системи. Від його якості залежать масштабованість, підтримуваність і безпека додатку. На підставі сформованих у розділі 1 вимог обрано трирівневу (three-tier) клієнт-серверну архітектуру, яка чітко розмежовує відповідальності між рівнем представлення, рівнем бізнес-логіки та рівнем даних [16].

2.1.1 Загальна структура рівнів. Рівень представлення (Presentation tier) — браузер клієнта, що відображає HTML-сторінки, отримані від сервера. Рендеринг виконується на стороні сервера засобами Jinja2 (Server-Side Rendering, SSR): Flask-контролер формує повну HTML-відповідь і повертає її браузеру. Такий підхід забезпечує коректну індексацію контенту, спрощує захист від XSS і усуває необхідність окремого API-шару для початкового завантаження сторінок. Для операцій, що не потребують перезавантаження сторінки (підтвердження / повернення етапу керівником), застосовується асинхронна взаємодія через браузерний Fetch API.

Рівень бізнес-логіки (Application tier) — Flask-додаток, що виконується у Docker-контейнері під управлінням WSGI-сервера Gunicorn. Цей рівень відповідає за маршрутизацію HTTP-запитів, перевірку автентифікації й авторизації, виконання бізнес-правил (зміна статусу, розрахунок прогресу, генерація звітів) та формування відповідей у форматі HTML або JSON.

Рівень даних (Data tier) — PostgreSQL 15, що виконується у окремому Docker-контейнері. Взаємодія між рівнями бізнес-логіки і даних здійснюється виключно через SQLAlchemy ORM: прямі SQL-запити у коді додатку відсутні, що мінімізує ризик SQL-ін'єкцій і спрощує майбутню зміну СУБД при необхідності.

					БР.КІ-31.00.00.000 ПЗ	Арк.
						20
Змн.	Арк.	№ докум.	Підпис	Дата		

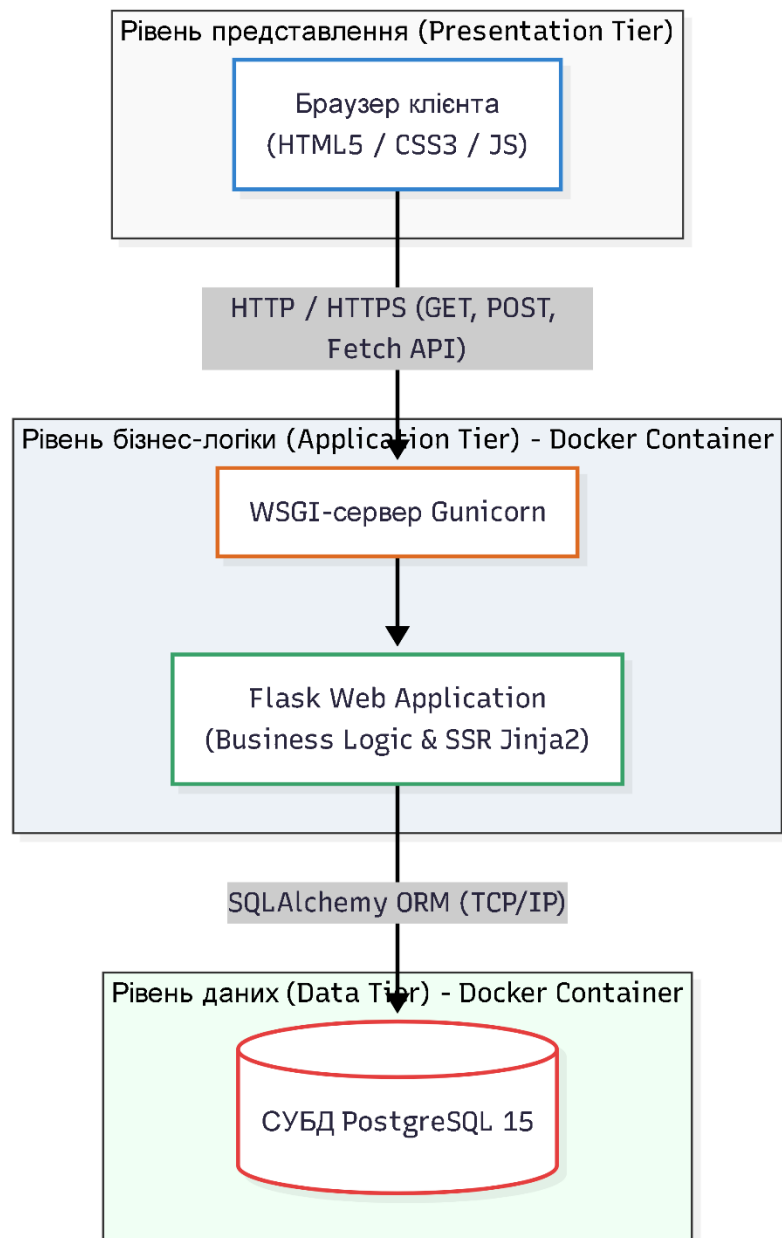


Рисунок 2.1 – Загальна архітектура клієнт-серверного веб-додатку

2.1.2 Декомпозиція серверної частини на Blueprint-модулі. Серверна частина організована за допомогою механізму Flask Blueprint, який дозволяє реєструвати маршрути, шаблони і статичні файли окремо для кожного функціонального модуля [17]. Застосування Blueprint-декомпозиції унеможливорює виникнення конфліктів між URL-просторами різних ролей та полегшує ізольоване тестування кожного модуля. Система містить чотири Blueprint-и:

- auth_bp — маршрути /auth/login і /auth/logout; відповідає за автентифікацію користувачів і управління сесіями через Flask-Login;
- admin_bp — маршрути /admin/*; реалізує управління обліковими записами, шаблонами етапів, прив'язкою студентів і генерацією звітів;
- supervisor_bp — маршрути /supervisor/*; забезпечує дашборд керівника, перегляд матеріалів студентів і операції підтвердження/повернення;
- student_bp — маршрути /student/*; реалізує перегляд плану, прикріплення матеріалів і відправку етапів на перевірку.

Маршрутизатор кореневого URL (/) реалізує ролевий редирект: автентифікований адміністратор перенаправляється на /admin/dashboard, керівник — на /supervisor/dashboard, студент — на /student/plan. Неавтентифікований користувач автоматично перенаправляється Flask-Login на /auth/login.

2.1.3 Взаємодія між компонентами та протокол обміну. Взаємодія між рівнями відбувається за протоколом HTTP/HTTPS. Браузер відправляє GET-запити для отримання сторінок і POST-запити для відправки форм. Для асинхронних операцій зміни статусу етапу керівником застосовується POST-запит через Fetch API: сервер повертає JSON-об'єкт із новим статусом і повідомленням, а JavaScript на стороні клієнта оновлює відповідний елемент DOM без перезавантаження сторінки.

Усі форми захищені CSRF-токеном, що генерується Flask-WTF і перевіряється на сервері при кожному POST-запиті. Завантаження файлів здійснюється через multipart/form-data: сервер зберігає файл у каталозі uploads/, змонтованому як Docker-volume для забезпечення персистентності між перезапусками контейнера.

					БР.КІ-31.00.00.000 ПЗ	Арк.
						22
Змн.	Арк.	№ докум.	Підпис	Дата		

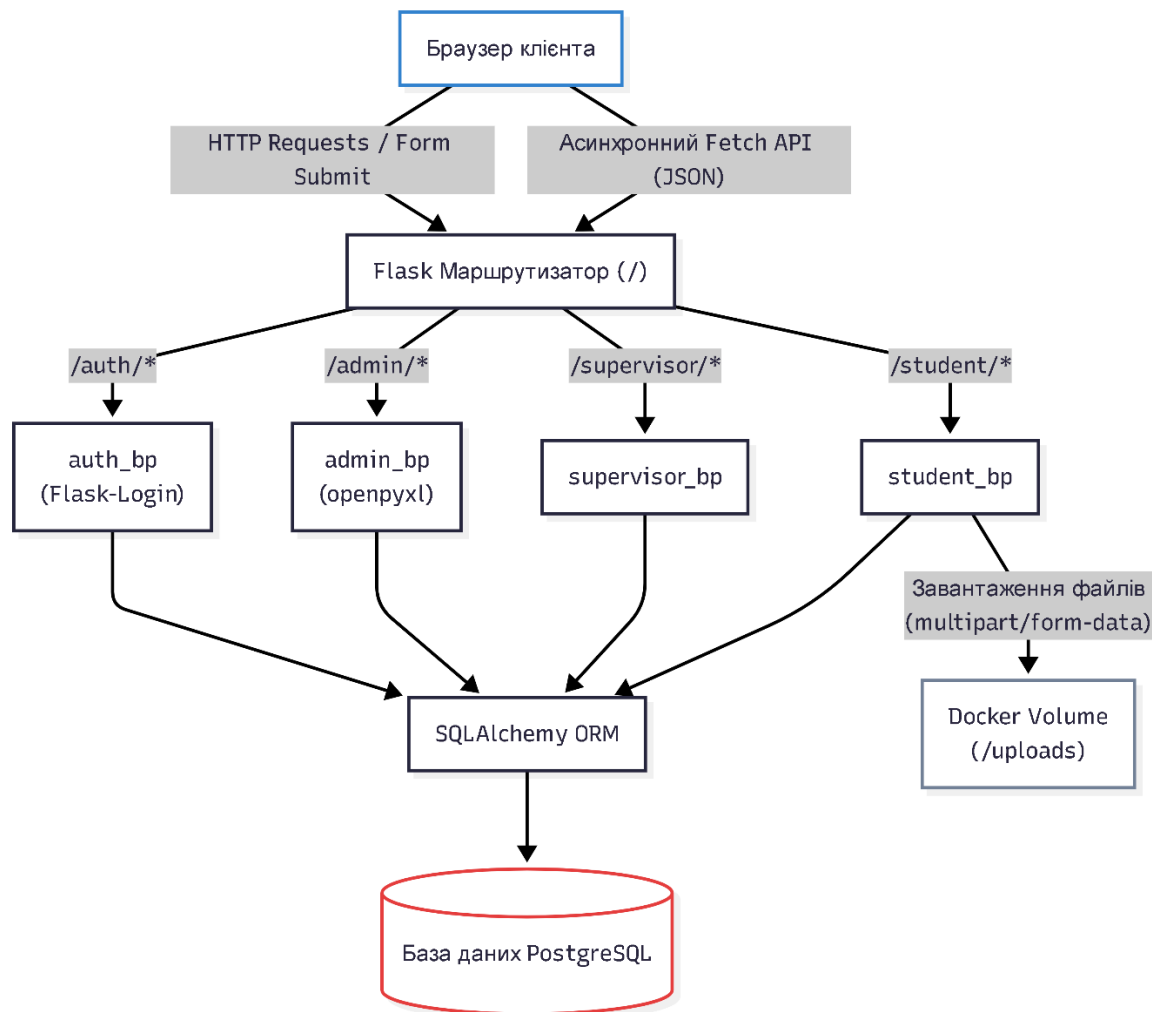


Рисунок 2.2 – Схема взаємодії між компонентами системи

2.2 Розробка моделі бази даних

Проектування схеми бази даних є ключовим етапом, що визначає структуру зберігання даних і цілісність інформаційних зв'язків між об'єктами системи. На основі аналізу предметної галузі та сформованих вимог виділено чотири основні сутності: `users`, `stage_templates`, `student_theses` та `stage_executions`. Реляційна модель обрана як найбільш природна для представлення фіксованих зв'язків між цими сутностями [18].

2.2.1 Опис сутностей та їх атрибутів. Сутність `users` реалізує єдину таблицю для всіх категорій користувачів системи — студентів, керівників і адміністраторів. Такий підхід (Single Table Inheritance на рівні атрибута `role`)

усуває необхідність дублювання полів автентифікації у кількох таблицях і спрощує реалізацію Flask-Login, який очікує одну модель користувача. Атрибут `role` визначений як тип `ENUM` з трьома значеннями: `student`, `supervisor`, `admin`. Атрибут `is_active` забезпечує м'яке видалення: деактивований користувач не може увійти до системи, але його дані та зв'язки зберігаються в базі.

Сутність `stage_templates` зберігає кафедральні шаблони етапів виконання кваліфікаційної роботи. Атрибут `weight` містить ваговий коефіцієнт конкретного етапу у відсотках; сума вагових коефіцієнтів усіх активних шаблонів повинна дорівнювати 100 % — ця інваріантність забезпечується на рівні адміністративної логіки при створенні шаблонів. Атрибут `order_num` визначає порядок відображення етапів у плані студента. Атрибут `is_active` дозволяє деактивувати шаблон без видалення пов'язаних записів виконання.

Сутність `student_theses` пов'язує конкретного студента з темою роботи та науковим керівником. Унікальне обмеження на атрибут `student_id` (`UNIQUE CONSTRAINT`) гарантує, що кожен студент може мати лише одну активну тему дипломної роботи. Зовнішній ключ `supervisor_id` посилається на таблицю `users` з фільтром на роль `supervisor`, що перевіряється на рівні бізнес-логіки Flask при прив'язці.

Сутність `stage_executions` є центральною сполучною таблицею, що реалізує зв'язок «багато-до-багатьох» між `student_theses` та `stage_templates` з додатковими атрибутами стану. Кожен запис відповідає конкретному етапу конкретного студента і зберігає поточний статус, матеріали (URL або шлях до файлу), коментар керівника та часові мітки. Складений унікальний ключ (`thesis_id`, `template_id`) запобігає дублюванню записів виконання одного й того самого етапу для однієї й тієї самої теми.

Таблиця 2.1 – Атрибути сутностей бази даних

Сутність (таблиця)	Атрибут	Тип даних	Опис
users	id	INTEGER PK	Унікальний ідентифікатор користувача
users	email	VARCHAR(120)	Адреса електронної пошти (унікальна, індексована)

users	password_hash	VARCHAR(256)	Хеш пароля (PBKDF2-SHA256, Werkzeug)
users	full_name	VARCHAR(200)	Повне ім'я користувача
users	role	ENUM	Роль: student supervisor admin
users	is_active	BOOLEAN	Ознака активності облікового запису
users	created_at	TIMESTAMP	Дата і час реєстрації (UTC)
stage_templates	id	INTEGER PK	Унікальний ідентифікатор шаблону
stage_templates	name	VARCHAR(200)	Назва етапу (наприклад, «Аналіз предметної галузі»)
stage_templates	description	TEXT	Розширений опис вимог до етапу
stage_templates	weight	INTEGER	Ваговий коефіцієнт у відсотках (1–100)
stage_templates	order_num	INTEGER	Порядковий номер відображення у плані
stage_templates	is_active	BOOLEAN	Активність шаблону (деактивованій не призначається)
student_theses	id	INTEGER PK	Унікальний ідентифікатор запису про роботу
student_theses	student_id	INTEGER FK	Посилання на users.id (студент); унікальне — один студент, одна тема
student_theses	supervisor_id	INTEGER FK	Посилання на users.id (керівник)
student_theses	topic	VARCHAR(400)	Тема кваліфікаційної роботи
student_theses	defense_date	DATE	Планова дата захисту (необов'язкове)
student_theses	created_at	TIMESTAMP	Дата призначення теми
stage_executions	id	INTEGER PK	Унікальний ідентифікатор запису
stage_executions	thesis_id	INTEGER FK	Посилання на student_theses.id
stage_executions	template_id	INTEGER FK	Посилання на stage_templates.id
stage_executions	status	ENUM	Статус: not_started in_progress pending_review done revision
stage_executions	repo_url	VARCHAR(500)	URL репозиторію GitHub / GitLab (необов'язкове)
stage_executions	file_path	VARCHAR(500)	Шлях до завантаженого файлу на сервері (необов'язкове)
stage_executions	supervisor_comment	TEXT	Коментар керівника при поверненні на доопрацювання
stage_executions	submitted_at	TIMESTAMP	Дата відправки на перевірку
stage_executions	approved_at	TIMESTAMP	Дата підтвердження керівником
stage_executions	updated_at	TIMESTAMP	Дата останнього оновлення запису

2.2.2 Зв'язки між сутностями . Між виділеними сутностями встановлено такі реляційні зв'язки:

– users (student) — student_theses: зв'язок «один-до-одного» (1:1). Один студент може мати рівно одну тему кваліфікаційної роботи; це обмеження реалізоване атрибутом UNIQUE на полі student_id.

– users (supervisor) — student_theses: зв'язок «один-до-багатьох» (1:N). Один керівник може бути прикріплений до багатьох студентів.

– student_theses — stage_executions: зв'язок «один-до-багатьох» (1:N). Одній темі відповідає набір записів виконання — по одному для кожного активного шаблону етапу.

– stage_templates — stage_executions: зв'язок «один-до-багатьох» (1:N). Один шаблон може бути пов'язаний із виконаннями різних студентів.

Наведені зв'язки відображено на ER-діаграмі (рисунок 2.3), побудованій у нотації «сутність – відношення» (crow's foot notation).

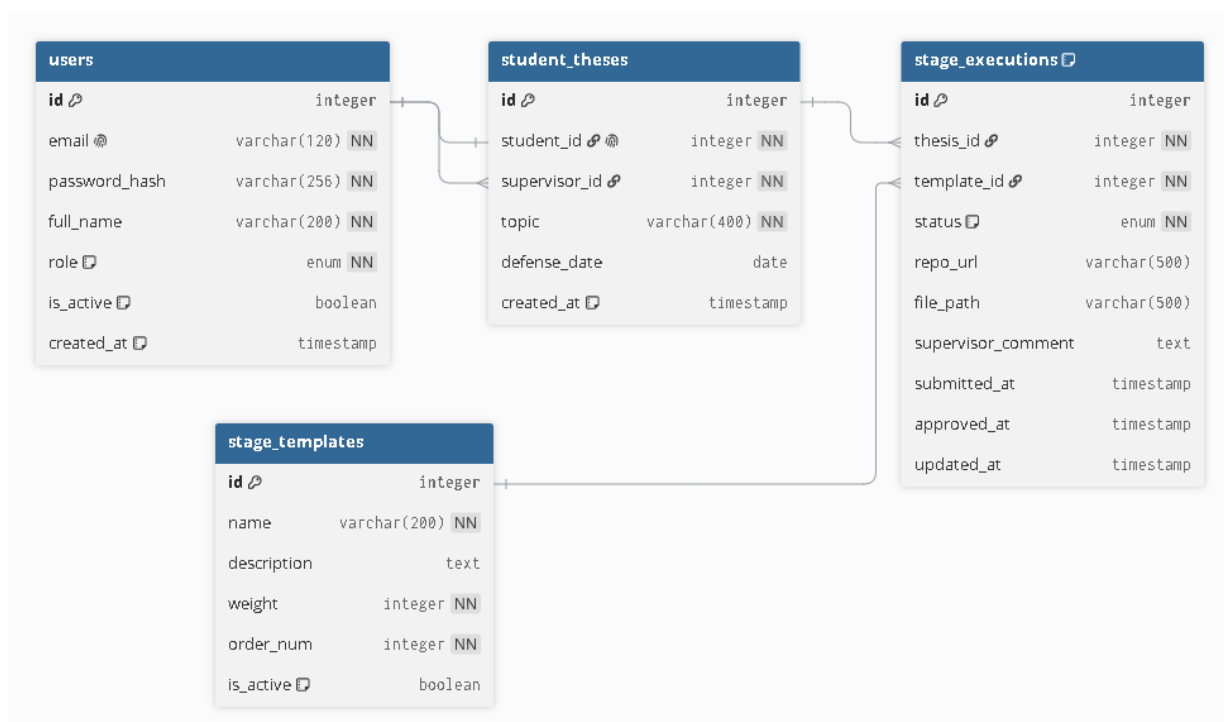


Рисунок 2.3 – ER-діаграма бази даних системи

2.2.3 Машина станів етапу виконання. Атрибут status сутності stage_executions. Перелік можливих станів, ініціатор кожного переходу та умова його виконання наведено у таблиці 2.2.

Таблиця 2.2 – Стани та переходи атрибута status сутності stage_executions

Значення ENUM	Відображувана назва	Ініціатор переходу	Умова переходу
not_started	Не розпочато	Система (авто)	Початковий стан після призначення теми
in_progress	В процесі	Студент	Студент розпочав роботу над етапом
pending_review	Очікує підтвердження	Студент	Студент подав матеріали на перевірку
done	Виконано	Керівник	Керівник підтвердив виконання; вага зараховується до прогресу
revision	На доопрацюванні	Керівник	Керівник повернув матеріали з коментарем

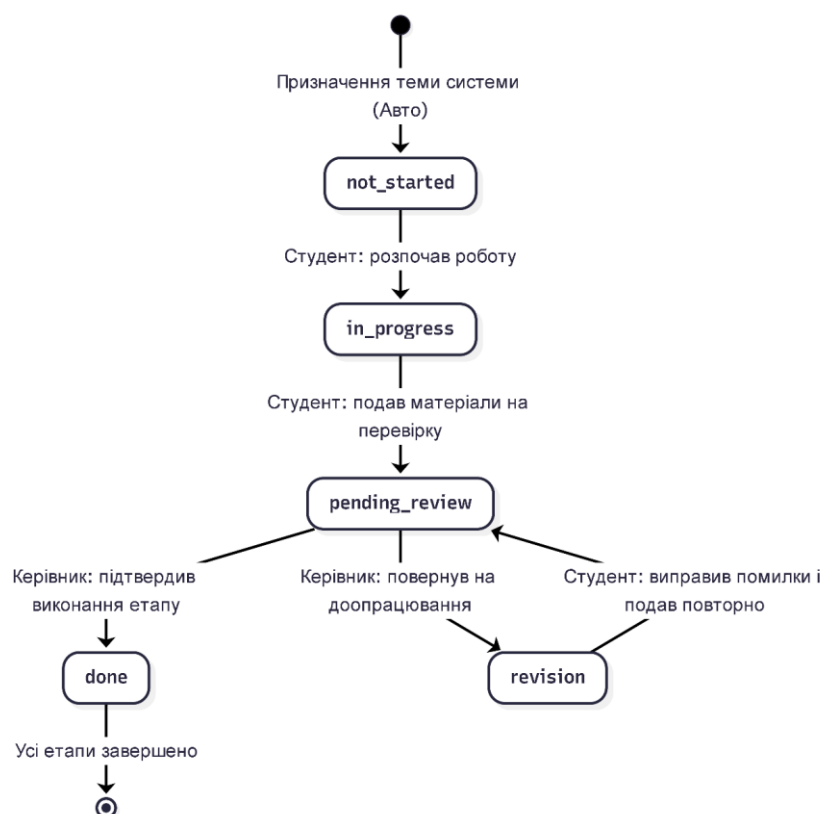


Рисунок 2.4 – Діаграма станів атрибута status сутності stage_executions

Допустимі переходи між станами обмежені бізнес-логікою Flask-контролера: студент може ініціювати лише переходи not_started → in_progress та

in_progress / revision → pending_review; керівник — лише pending_review → done або pending_review → revision. Будь-яка спроба виконати недопустимий перехід відхиляється з поверненням HTTP 403.

2.3 Алгоритм автоматичного розрахунку відсотка виконання роботи

Алгоритм розрахунку прогресу є ключовою бізнес-функцією системи, що реалізує формальну оцінку ступеня виконання кваліфікаційної роботи студентом. Алгоритм побудований на принципі вагових коефіцієнтів: кожному етапу шаблону адміністратором призначається вага у відсотках, а сума вагових коефіцієнтів усіх активних шаблонів дорівнює 100 %. Відсоток виконання зараховується лише після того, як керівник переводить відповідний запис stage_executions у статус done. Такий підхід забезпечує об'єктивну, верифіковану оцінку прогресу, що не залежить від суб'єктивних дій студента [19].

2.3.1 Формальний опис алгоритму. Нехай задано множину активних шаблонів етапів $T = \{t_1, t_2, \dots, t_n\}$, де кожному шаблону t_i відповідає ваговий коефіцієнт $w_i \in \mathbb{N}$ такий, що $\sum w_i = 100$. Для конкретного студента визначено множину записів виконання $E = \{e_1, e_2, \dots, e_n\}$, де кожен запис e_i пов'язаний із шаблоном t_i і має поточний статус $\text{status}(e_i) \in \{\text{not_started}, \text{in_progress}, \text{pending_review}, \text{done}, \text{revision}\}$. Відсоток прогресу студента P обчислюється за формулою:

$$P = \sum w_i \times \delta(\text{status}(e_i) = \text{done}), \text{ де } \delta \text{ — функція Кронекера}$$

Таким чином, ваговий коефіцієнт етапу додається до загального прогресу тоді і лише тоді, коли статус відповідного запису виконання дорівнює done. Записи зі статусами not_started, in_progress, pending_review та revision не впливають на поточне значення прогресу.

					БР.КІ-31.00.00.000 ПЗ	Арк.
						28
Змн.	Арк.	№ докум.	Підпис	Дата		

2.3.2 Словесний опис кроків алгоритму. Алгоритм розрахунку прогресу виконується у такій послідовності:

Крок 1. Отримання записів виконання. За заданим ідентифікатором теми (thesis_id) з таблиці stage_executions вибираються усі пов'язані записи. Виконується JOIN із таблицею stage_templates для отримання вагового коефіцієнта кожного запису.

Крок 2. Фільтрація підтверджених етапів. Із повного переліку записів виконання відбираються лише ті, у яких значення атрибута status дорівнює done. Записи з іншими статусами ігноруються і не впливають на результат.

Крок 3. Підсумовування вагових коефіцієнтів. Для кожного відфільтрованого запису до акумулятора додається значення атрибута weight відповідного шаблону. Після обходу всіх підтверджених записів акумулятор містить загальний відсоток прогресу.

Крок 4. Повернення результату. Обчислене значення $P \in [0; 100]$ повертається як властивість об'єкта StudentThesis і використовується для відображення прогрес-бару у кабінеті студента та дашборді керівника.

2.3.3 Реалізація алгоритму у вигляді псевдокоду. Алгоритм реалізовано у вигляді властивості-обчислення (Python @property) класу StudentThesis, що виключає необхідність явного виклику і забезпечує актуальність значення прогресу при кожному зверненні до об'єкта. Псевдокод алгоритму наведено нижче:

ВЛАСТИВІСТЬ progress(thesis):

total $\leftarrow$ 0

ДЛЯ КОЖНОГО execution В thesis.stage_executions:

ЯКЩО execution.status = 'done':

total $\leftarrow$ total + execution.template.weight

ПОВЕРНУТИ total

// Приклад (Python / SQLAlchemy):

@property

def progress(self) -> int:

					БР.КІ-31.00.00.000 ПЗ	Арк.
						29
Змн.	Арк.	№ докум.	Підпис	Дата		

```

return sum(
    ex.template.weight
    for ex in self.stage_executions
    if ex.status == 'done'
)

```

Складність алгоритму за часом — $O(n)$, де n — кількість записів виконання для даної теми. Оскільки кількість етапів у типовому кафедральному плані не перевищує 10–15, алгоритм виконується практично миттєво і не потребує кешування.

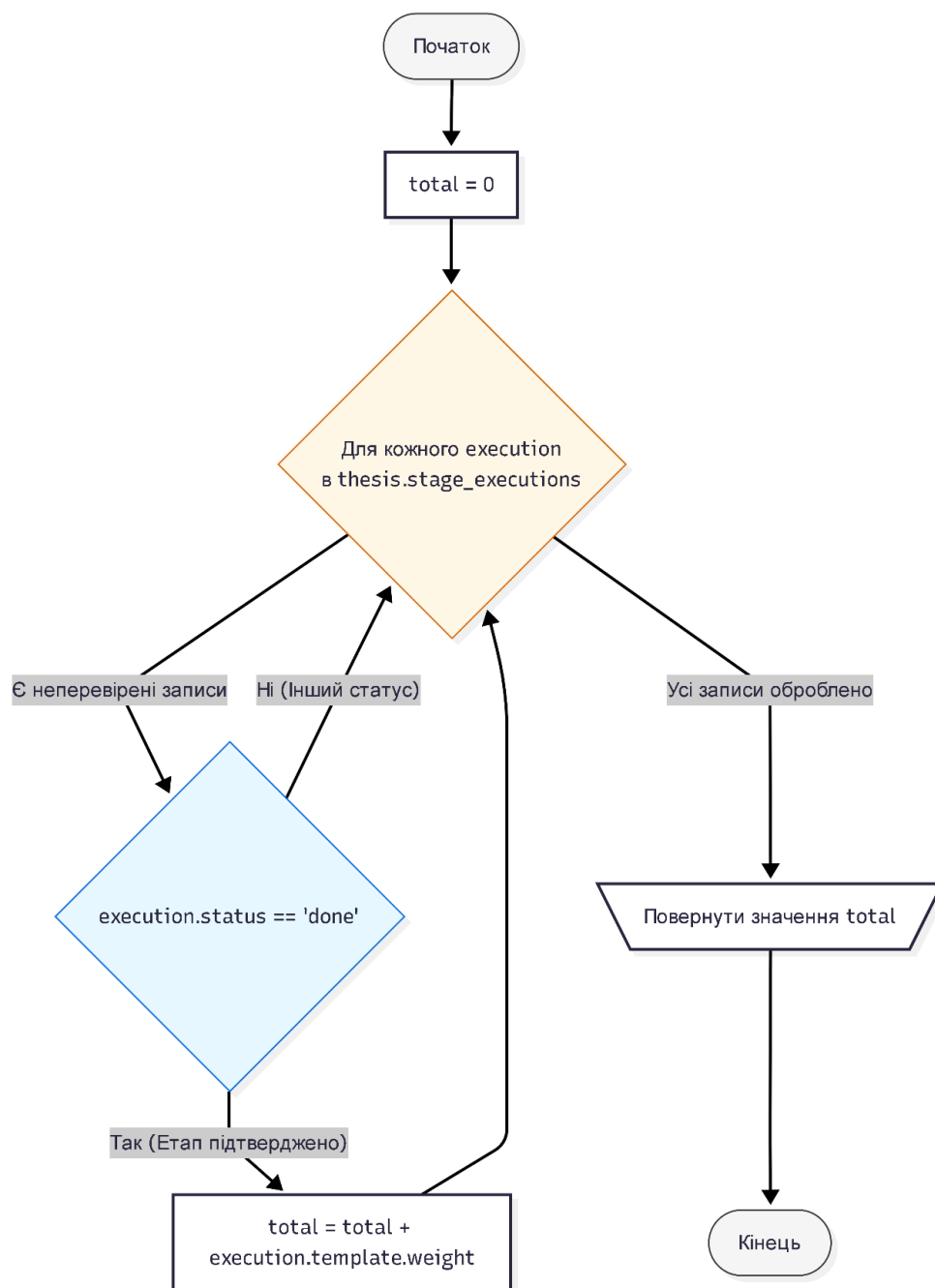


Рисунок 2.5 – Блок-схема алгоритму розрахунку відсотка виконання роботи

2.4 Проектування системи ролей та розмежування прав доступу

Розмежування прав доступу є критичною складовою безпеки системи, що гарантує неможливість виконання операцій, не передбачених для конкретної ролі користувача. Система реалізує рольову модель контролю доступу (Role-Based Access Control, RBAC), у якій кожному користувачу призначається рівно одна роль, а кожній ролі — визначений набір дозволених операцій [20].

2.4.1 Механізм автентифікації. Автентифікація реалізована через бібліотеку Flask-Login із використанням серверних сесій. При успішному вході в систему ідентифікатор користувача зберігається у підписаному cookie-файлі сесії; підпис забезпечується секретним ключем (SECRET_KEY), що зберігається у .env-файлі і не потрапляє до репозиторію. Flask-Login перевіряє наявність і дійсність сесії при кожному запиті до захищеного маршруту через декоратор @login_required.

Паролі користувачів зберігаються виключно у хешованому вигляді. Для хешування застосовується функція Werkzeug generate_password_hash(), яка використовує алгоритм PBKDF2-HMAC-SHA256 із випадковою сіллю. Перевірка пароля при вході здійснюється функцією check_password_hash(), що унеможливорює відновлення оригінального пароля навіть при компрометації бази даних.

2.4.2 Механізм авторизації на рівні маршрутів. Авторизація реалізована на двох рівнях. Перший рівень — перевірка автентифікації: декоратор @login_required на кожному захищеному маршруті перенаправляє неавтентифікованого користувача на сторінку входу. Другий рівень — перевірка ролі: спеціалізований декоратор @role_required(role) порівнює роль поточного користувача з необхідною і повертає HTTP 403 при розбіжності.

Blueprint-архітектура природно підсилює розмежування доступу: маршрути admin_bp фізично відокремлені від маршрутів student_bp і

					БР.КІ-31.00.00.000 ПЗ	Арк.
						31
Змн.	Арк.	№ докум.	Підпис	Дата		

supervisor_br, що унеможливилює випадковий або навмисний доступ між модулями без явного обходу декораторів.

2.4.3 Матриця прав доступу. Повний перелік функціональних областей системи з визначенням допустимих ролей для кожної з них наведено у таблиці 2.3. Символ “+” означає наявність дозволу, “–” — відсутність.

Таблиця 2.3 – Матриця прав доступу за ролями

Функціональна область / маршрут	Студент	Керівник	Адміністратор
Авторизація / вихід із системи	+	+	+
Перегляд власного плану (список етапів)	+	–	–
Прикріплення файлу / посилання до етапу	+	–	–
Відправка етапу на перевірку	+	–	–
Перегляд власного прогресу (%)	+	–	–
Дашборд із переліком своїх студентів	–	+	–
Перегляд матеріалів студента	–	+	–
Підтвердження / повернення етапу	–	+	–
Управління обліковими записами	–	–	+
Створення шаблонів етапів	–	–	+
Прив'язка студент ↔ керівник ↔ тема	–	–	+
Генерація CSV / Excel-звіту	–	–	+

Матриця прав доступу, наведена у таблиці 2.3, є специфікацією, яка безпосередньо визначає набір декораторів для кожного Flask-маршруту і реалізується у підрозділі 3.1.

Для наочного відображення взаємодії акторів із функціями системи на рисунку Ф.1 в додатку Ф наведено UML-діаграму варіантів використання (Use Case), що графічно узагальнює матрицю прав доступу таблиці 2.3. Діаграма ілюструє три актори — Студент, Керівник та Адміністратор — і відповідні варіанти використання в межах єдиної системної межі. Зв'язки «include»

позначають обов'язкові залежності між варіантами використання, «extend» — необов'язкові розширення основного сценарію.

Діаграма варіантів використання підтверджує повноту охоплення функціональних вимог: кожна з 12 операцій матриці прав доступу відображена у відповідному варіанті використання та закріплена за єдиним актором. Реалізація наведених варіантів використання на рівні Flask-маршрутів описана у підрозділі 3.1.

2.4.4 Захист від CSRF-атак. Усі HTML-форми, що виконують POST-запити (форма входу, форма прикріплення матеріалів, форма підтвердження/повернення), захищені від CSRF-атак (Cross-Site Request Forgery) засобами бібліотеки Flask-WTF [21]. При рендерингу форми у шаблон Jinja2 додається прихований `<input>` із унікальним CSRF-токеном, підписаним секретним ключем сесії. Flask-WTF автоматично перевіряє наявність і дійсність токена при обробці кожного POST-запиту; запит без дійсного токена відхиляється з кодом HTTP 400.

Висновок до розділу

У другому розділі виконано повне проектування архітектури системи та її ключових компонентів.

Обрано трирівневу клієнт-серверну архітектуру з чітким розмежуванням рівнів представлення (браузер + Jinja2 SSR), бізнес-логіки (Flask + Gunicorn у Docker-контейнері) та даних (PostgreSQL у окремому контейнері). Серверна частина декомпонована на чотири Blueprint-модулі відповідно до рольових підсистем: `auth_bp`, `admin_bp`, `supervisor_bp`, `student_bp`.

Розроблено реляційну модель бази даних у складі чотирьох сутностей: `users` (єдина таблиця для всіх ролей із ENUM-атрибутом), `stage_templates` (шаблони етапів із ваговими коефіцієнтами), `student_theses` (прив'язка студент ↔ керівник ↔ тема) та `stage_executions` (центральна таблиця зі скінченням

					БР.КІ-31.00.00.000 ПЗ	Арк.
						33
Змн.	Арк.	№ докум.	Підпис	Дата		

автоматом статусів із п'ятьма станами). Для кожної сутності визначено атрибути, типи даних, обмеження цілісності та зовнішні ключі.

Спроековано алгоритм автоматичного розрахунку відсотка виконання роботи на основі вагових коефіцієнтів. Алгоритм реалізує лінійне підсумовування вагових коефіцієнтів лише підтверджених (done) етапів, має складність $O(n)$ і реалізований у вигляді @property Python-класу.

Спроековано систему ролей і розмежування прав доступу на основі RBAC: автентифікація — через Flask-Login із серверними сесіями і PBKDF2-хешуванням паролів; авторизація — через дворівневі декоратори @login_required та @role_required; захист від CSRF — через Flask-WTF з автоматичною перевіркою токенів. Визначено матрицю прав доступу для 12 функціональних областей системи.

					БР.КІ-31.00.00.000 ПЗ	Арк.
						34
Змн.	Арк.	№ докум.	Підпис	Дата		

3 ПРОГРАМНА РЕАЛІЗАЦІЯ ТА РОЗГОРТАННЯ ВЕБ-ДОДАТКУ

3.1 Реалізація серверної частини та налаштування взаємодії з базою даних

Серверна частина системи реалізована мовою Python з використанням мікрофреймворку Flask 3.0.3. Структура проєкту відповідає рекомендованій організації Flask-застосунку: пакет `app/` інкапсулює весь програмний код, а кожна роль користувача отримала окремий Blueprint-модуль.

3.1.1 Фабрична функція та ініціалізація розширень. Точкою входу в додаток є модуль `app/__init__.py`. У ньому визначено глобальні об'єкти розширень (`db`, `migrate`, `login_manager`, `csrf`) та реалізовано фабричну функцію `create_app()`, яка ініціалізує їх відносно переданого екземпляра Flask. Такий підхід — Application Factory Pattern — дозволяє запускати кілька екземплярів застосунку з різними конфігураціями (розробка, тестування, продуктив) без конфліктів глобального стану.

Проєкт організовано за рекомендованою структурою Flask-застосунку: весь програмний код зосереджено у пакеті `app/`, кожен функціональний модуль реалізовано в окремому Blueprint-файлі, а допоміжні засоби — міграції, тести та конфігурація розгортання — розміщено у кореневих каталогах.

Повне дерево файлів проєкту `thesis-monitor` знаходиться в Додатку У

Функція `create_app()` реєструє чотири Blueprint-и та задає маршрут кореневого URL / з перенаправленням залежно від ролі:

Реалізація фабричної функції `create_app()`. Функція приймає назву конфігурації, створює екземпляр Flask, підключає всі розширення та реєструє чотири Blueprint-модулі з відповідними URL-префіксами. Маршрут кореневого URL / реалізує логіку перенаправлення залежно від ролі авторизованого користувача.

Фабрична функція `create_app()` (`app/__init__.py`)

					БР.КІ-31.00.00.000 ПЗ	Арк.
						35
Змн.	Арк.	№ докум.	Підпис	Дата		

```

def create_app(config_name=None):
    app = Flask(__name__)
    from .config import config
    app.config.from_object(config[config_name])
    db.init_app(app)
    migrate.init_app(app, db)
    login_manager.init_app(app)
    csrf.init_app(app)
    os.makedirs(app.config['UPLOAD_FOLDER'], exist_ok=True)
    from .routes.auth import auth_bp
    from .routes.admin import admin_bp
    from .routes.supervisor import supervisor_bp
    from .routes.student import student_bp
    app.register_blueprint(auth_bp)
    app.register_blueprint(admin_bp, url_prefix='/admin')
    app.register_blueprint(supervisor_bp, url_prefix='/supervisor')
    app.register_blueprint(student_bp, url_prefix='/student')
    @app.route('/')
    def index():
        if not current_user.is_authenticated:
            return redirect(url_for('auth.login'))
        if current_user.role == 'admin':
            return redirect(url_for('admin.dashboard'))
        elif current_user.role == 'supervisor':
            return redirect(url_for('supervisor.dashboard'))
        else:
            return redirect(url_for('student.plan'))
    return app

```

3.1.2 Моделі бази даних та ORM-відображення. Усі моделі реалізовано у модулі `app/models.py` у вигляді Python-класів, успадкованих від `db.Model`. Модель `User` успадковує також `UserMixin` із `Flask-Login`, що надає методи `is_authenticated`, `is_active`, `get_id()`.

Визначення моделі `User`. Клас успадковує `db.Model` (`SQLAlchemy`) та `UserMixin` (`Flask-Login`), що забезпечує роботу механізму сесій. Поле `role` визначено як `PostgreSQL ENUM`-тип з трьома значеннями. Паролі зберігаються виключно у хешованому вигляді — метод `set_password()` хешує рядок через `Werkzeug`, а `check_password()` виконує безпечне порівняння.

Модель `User` із `ENUM`-роллю та хешуванням пароля

```

class User(UserMixin, db.Model):
    __tablename__ = 'users'
    id = db.Column(db.Integer, primary_key=True)
    email = db.Column(db.String(120), unique=True,
                      nullable=False, index=True)
    password_hash = db.Column(db.String(256), nullable=False)
    full_name = db.Column(db.String(200), nullable=False)
    role = db.Column(
        db.Enum('student', 'supervisor', 'admin',
              name='user_roles'),
        nullable=False, default='student'
    )

```

					БР.КІ-31.00.00.000 ПЗ	Арк.
						36
Змн.	Арк.	№ докум.	Підпис	Дата		

```

is_active = db.Column(db.Boolean, default=True)
created_at = db.Column(db.DateTime, default=datetime.utcnow)
def set_password(self, password):
    self.password_hash = generate_password_hash(password)
def check_password(self, password):
    return check_password_hash(self.password_hash, password)
@property
def role_label(self):
    return {'student': 'Студент', 'supervisor': 'Керівник',
            'admin': 'Адміністратор'}.get(self.role, self.role)

```

Модель StudentThesis реалізує властивість progress як обчислюваний атрибут Python, що підсумовує ваги лише підтверджених (done) етапів:

Визначення моделі StudentThesis разом із властивістю progress. Прогрес виконання обчислюється динамічно як сума ваг лише тих етапів, статус яких дорівнює done — без збереження проміжного значення у базі даних. Такий підхід гарантує актуальність показника після кожної зміни статусу.

Властивість progress класу StudentThesis

```

class StudentThesis(db.Model):
    __tablename__ = 'student_theses'
    id = db.Column(db.Integer, primary_key=True)
    student_id = db.Column(db.Integer,
                           db.ForeignKey('users.id'),
                           nullable=False, unique=True)
    supervisor_id = db.Column(db.Integer,
                              db.ForeignKey('users.id'), nullable=False)
    topic = db.Column(db.String(400), nullable=False)
    defense_date = db.Column(db.Date, nullable=True)
    stage_executions = db.relationship(
        'StageExecution', backref='thesis',
        lazy='dynamic', order_by='StageExecution.id')

    @property
    def progress(self) -> int:
        return sum(
            ex.template.weight
            for ex in self.stage_executions
            if ex.status == 'done'
        )

```

Модель StageExecution містить допоміжні @property для відображення статусів у шаблонах:

Визначення моделі StageExecution з усіма полями та допоміжними властивостями. Для зручного відображення у Jinja2-шаблонах модель надає дві властивості: status_label повертає людиночитаний рядок статусу українською мовою, а status_badge_class — відповідний CSS-клас Bootstrap для кольорового індикатора статусу.

					БР.КІ-31.00.00.000 ПЗ	Арк.
						37
Змн.	Арк.	№ докум.	Підпис	Дата		

Властивості status_label та status_badge_class

```
class StageExecution(db.Model):
    __tablename__ = 'stage_executions'
    id = db.Column(db.Integer, primary_key=True)
    thesis_id = db.Column(db.Integer,
                           db.ForeignKey('student_theses.id'),
                           nullable=False)
    template_id = db.Column(db.Integer,
                             db.ForeignKey('stage_templates.id'),
                             nullable=False)
    status = db.Column(db.Enum('not_started',
                                'in_progress', 'pending_review', 'done', 'revision',
                                name='stage_statuses'), default='not_started')
    repo_url = db.Column(db.String(500))
    file_path = db.Column(db.String(500))
    supervisor_comment = db.Column(db.Text)
    submitted_at = db.Column(db.DateTime)
    approved_at = db.Column(db.DateTime)
    @property
    def status_label(self) -> str:
        return {'not_started': 'Не розпочато',
                'in_progress': 'В процесі',
                'pending_review': 'Очікує перевірки',
                'done': 'Виконано',
                'revision': 'На доопрацюванні'}
        .get(self.status, self.status)
    @property
    def status_badge_class(self) -> str:
        return {'not_started': 'secondary',
                'in_progress': 'primary',
                'pending_review': 'warning',
                'done': 'success',
                'revision': 'danger'}
        .get(self.status, 'secondary')
```

3.1.3 Декоратор role_required та модуль автентифікації. Модуль app/routes/auth.py реалізує маршрути авторизації та функцію-декоратор role_required(), яка є фабрикою декораторів — приймає список дозволених ролей і повертає декоратор для захисту маршруту.

Реалізація декоратора role_required(). Декоратор є фабрикою: він приймає список дозволених ролей і повертає власне декоратор, що огортає функцію маршруту. При кожному запиті перевіряється факт авторизації та відповідність ролі поточного користувача; у разі порушення виконується перенаправлення з flash-повідомленням.

Декоратор role_required() (app/routes/auth.py)

```
def role_required(*roles):
    def decorator(f):
        @wraps(f)
        def decorated_function(*args, **kwargs):
```

					БР.КІ-31.00.00.000 ПЗ	Арк.
						38
Змн.	Арк.	№ докум.	Підпис	Дата		

```

        if not current_user.is_authenticated:
            return redirect(url_for('auth.login'))
        if current_user.role not in roles:
            flash('Доступ заборонено: недостатньо прав',
                  'danger')
            return redirect(url_for('index'))
        return f(*args, **kwargs)
    return decorated_function
return decorator

```

Маршрут `/auth/login` обробляє GET-запит (відображення форми) та POST-запит (перевірка облікових даних). Після успішної автентифікації виконується перенаправлення залежно від ролі:

Реалізація маршруту `/auth/login`, який обробляє як GET-запит (відображення форми входу), так і POST-запит (перевірку облікових даних). При успішній авторизації через Flask-Login користувач перенаправляється до відповідного кабінету; при невдачній — отримує flash-повідомлення про помилку без розкриття деталей.

Маршрут авторизації `/auth/login`

```

@auth_bp.route('/login', methods=['GET', 'POST'])
def login():
    if current_user.is_authenticated:
        return redirect(url_for('index'))
    if request.method == 'POST':
        email = request.form.get('email', '').strip().lower()
        password = request.form.get('password', '')
        remember = request.form.get('remember') == 'on'
        user = User.query.filter_by(email=email).first()
        if user and user.check_password(password)
            and user.is_active:
            login_user(user, remember=remember)
            flash(f'Ласкаво просимо, {user.full_name}!',
                  'success')
            return redirect(request.args.get('next')
                            or url_for('index'))
        flash('Невірна адреса або пароль', 'danger')
    return render_template('auth/login.html')

```

					БР.КІ-31.00.00.000 ПЗ	Арк.
						39
Змн.	Арк.	№ докум.	Підпис	Дата		

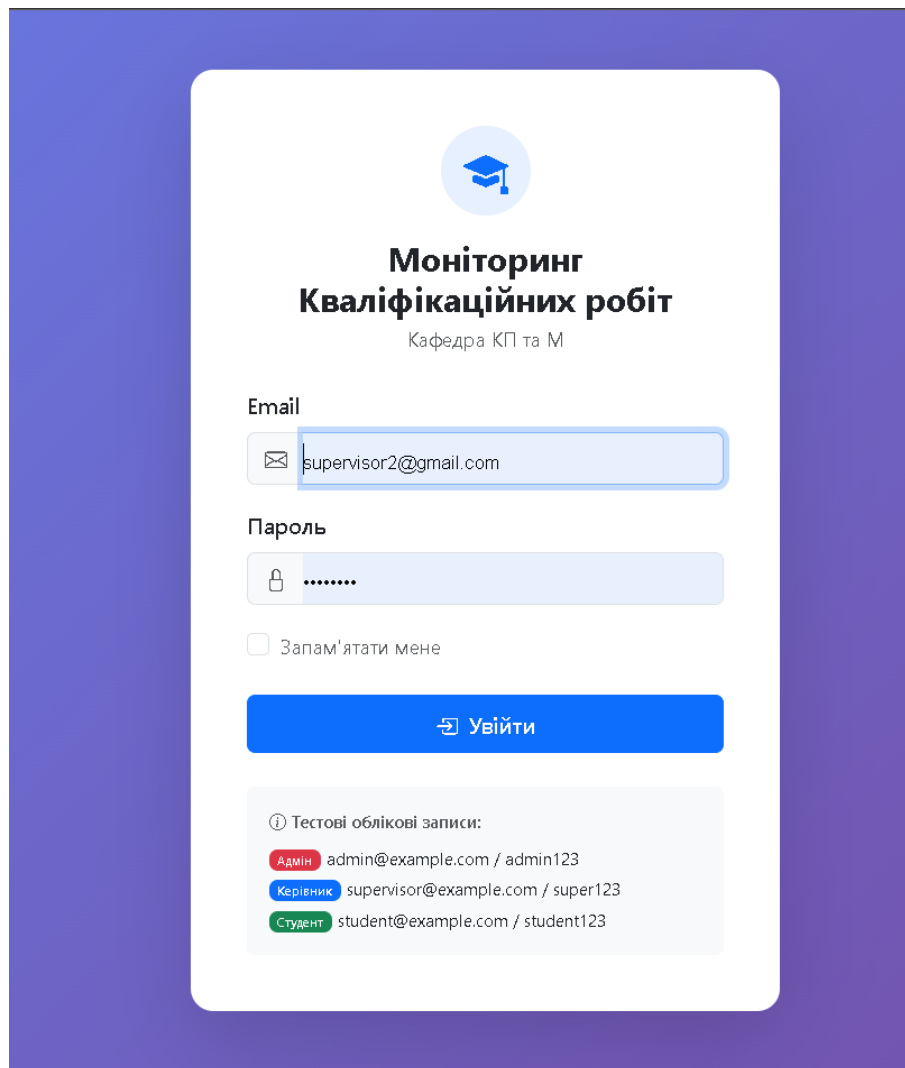


Рисунок 3.1 – Сторінка входу в систему (localhost:5000/auth/login)

На рисунку 3.1 представлено сторінку входу в систему, яка є єдиною точкою доступу для всіх ролей користувачів. Форма містить поля для введення електронної пошти та пароля з CSRF-захистом. Після успішної авторизації система автоматично перенаправляє користувача до відповідного кабінету залежно від його ролі.

3.2 Розробка інтерфейсу користувача

Інтерфейс реалізовано методом серверного рендерингу (SSR) засобами шаблонізатора Jinja2. Всі сторінки успадковують базовий шаблон base.html, де визначено sidebar з навігаційними посиланнями залежно від ролі, обробку flash-

					БР.КІ-31.00.00.000 ПЗ	Арк.
						40
Змн.	Арк.	№ докум.	Підпис	Дата		

повідомлень та підключення Bootstrap 5.3. Мета-тег csrf-token у <head> забезпечує доступ до CSRF-токена для Fetch API:

Демонстрація ключових фрагментів базового шаблону base.html. Шаблон підключає Bootstrap 5.3 через CDN, визначає мета-теги та бічну панель навігації, посилання якої формуються залежно від ролі поточного користувача через умовні конструкції Jinja2. Дочірні шаблони розширюють базовий через блоки content та title.

Базовий шаблон base.html (фрагмент <head> та sidebar)

```
<!DOCTYPE html>
<html lang="uk">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1">
  <meta name="csrf-token" content="{{ csrf_token() }}">
  <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.3/
    dist/css/bootstrap.min.css" rel="stylesheet">
  <link href="https://cdn.jsdelivr.net/npm/
    bootstrap-icons@1.11.3/font/bootstrap-icons.css"
    rel="stylesheet">
  <link rel="stylesheet" href="{{ url_for('static',
    filename='css/style.css') }}">
</head>
<body>
{% if current_user.is_authenticated %}
  <nav class="sidebar bg-dark text-white">
    {% if current_user.role == 'admin' %}
      <a href="{{ url_for('admin.dashboard') }}">Дашборд</a>
      <a href="{{ url_for('admin.users') }}">Користувачі</a>
      <a href="{{ url_for('admin.stages') }}">Шаблони</a>
      <a href="{{ url_for('admin.assign') }}">Призначення</a>
    {% elif current_user.role == 'supervisor' %}
      <a href="{{ url_for('supervisor.dashboard') }}">Дашборд</a>
    {% else %}
      <a href="{{ url_for('student.plan') }}">Мій план</a>
    {% endif %}
  </nav>
{% endif %}
<script src="{{ url_for('static', filename='js/main.js') }}">
</script>
</body></html>
```

3.2.1 Кабінет студента. Кабінет студента складається з двох сторінок. Сторінка no_thesis.html відображається, якщо студенту ще не призначено тему. Основна сторінка plan.html показує тему, керівника, дату захисту, прогрес-бар та список усіх активних етапів. Якщо керівник повернув етап (revision), студент бачить коментар у червоному алерті. Форма подачі доступна лише для статусів not_started, in_progress, revision.

					БР.КІ-31.00.00.000 ПЗ	Арк.
						41
Змн.	Арк.	№ докум.	Підпис	Дата		

Реалізація маршруту `/student/plan` — основного кабінету студента. Маршрут захищено декоратором `role_required` з роллю `student`. Він завантажує призначену студенту кваліфікаційну роботу разом із усіма етапами виконання та передає дані у шаблон для відображення плану та прогрес-бару.

Маршрут кабінету студента `/student/plan`

```
@student_bp.route('/plan')
@login_required
@role_required('student')
def plan():
    thesis = StudentThesis.query.filter_by(
        student_id=current_user.id).first()
    if not thesis:
        return render_template('student/no_thesis.html')
    executions = thesis.stage_executions\
        .order_by(StageExecution.id).all()
    return render_template('student/plan.html',
                           thesis=thesis,
                           executions=executions)
```

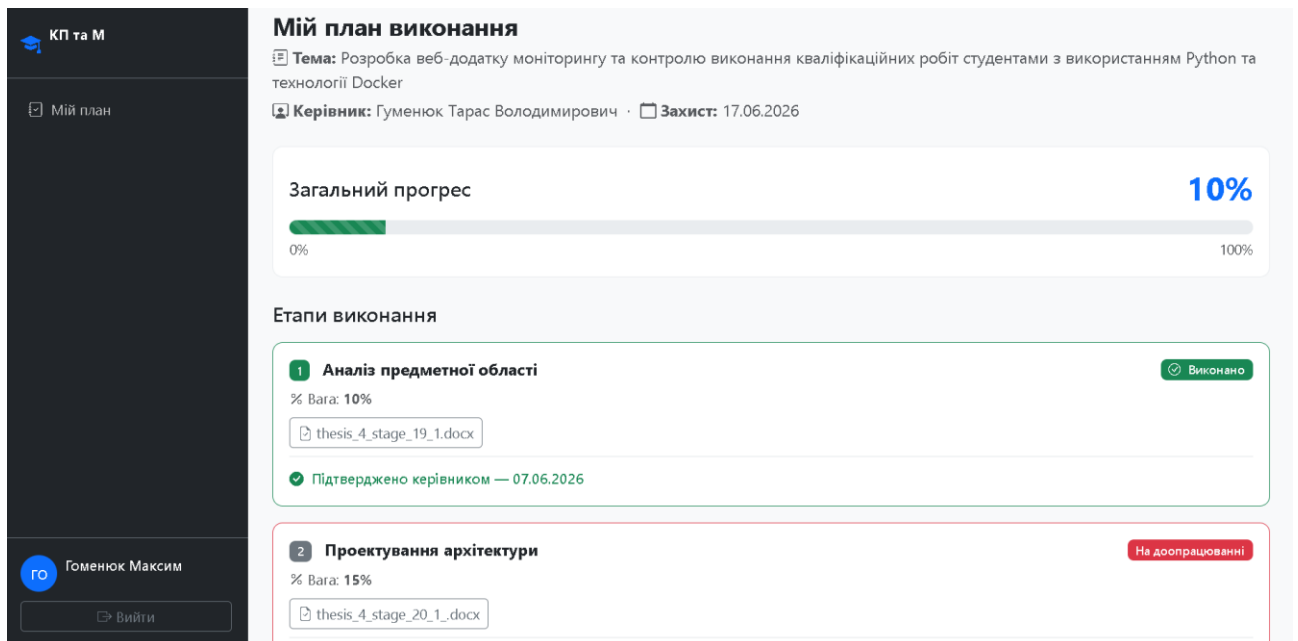


Рисунок 3.2 – Кабінет студента: план виконання з прогрес-баром та картками етапів (`localhost:5000/student/plan`)

На рисунку 3.2 зображено головну сторінку кабінету студента. У верхній частині відображається прогрес-бар із відсотком виконання роботи, що динамічно розраховується на основі підтверджених керівником етапів. Нижче розміщено картки кожного етапу з поточним статусом, можливістю подачі матеріалів та переглядом коментарів керівника.

					БР.КІ-31.00.00.000 ПЗ	Арк.
						42
Змн.	Арк.	№ докум.	Підпис	Дата		

Маршрут `/student/stage/<id>/submit` обробляє POST-запит форми. Контролер перевіряє належність запису поточному студенту, допустимість статусу, зберігає GitHub URL або файл та переводить виконання в `pending_review`:

Демонстрація обробки форми подачі матеріалів до конкретного етапу. Студент може прикріпити або посилання на репозиторій, або файл (PDF, ZIP тощо). Після успішної подачі статус відповідного `StageExecution` автоматично змінюється на `pending_review`, що сповіщає керівника про необхідність перевірки.

Маршрут подачі матеріалів студентом

```
@student_bp.route('/stage/<int:execution_id>/submit',
                 methods=['POST'])
@login_required
@role_required('student')
def submit_stage(execution_id):
    execution = StageExecution.query.get_or_404(execution_id)
    if execution.thesis.student_id != current_user.id:
        flash('Доступ заборонено', 'danger')
        return redirect(url_for('student.plan'))
    if execution.status in ('done', 'pending_review'):
        flash('Етап вже відправлено або виконано', 'warning')
        return redirect(url_for('student.plan'))
    repo_url = request.form.get('repo_url', '').strip()
    if repo_url:
        execution.repo_url = repo_url
    file = request.files.get('file')
    if file and file.filename and allowed_file(file.filename):
        filename = secure_filename(file.filename)
        safe_name = (f"thesis_{execution.thesis_id}"
                    f"_stage_{execution_id}_{filename}")
        file.save(os.path.join(
            current_app.config['UPLOAD_FOLDER'], safe_name))
        execution.file_path = safe_name
    execution.status = 'pending_review'
    execution.submitted_at = datetime.utcnow()
    execution.supervisor_comment = None
    db.session.commit()
    flash('Етап відправлено на перевірку!', 'success')
    return redirect(url_for('student.plan'))
```

3.2.2 Кабінет керівника. Дашборд керівника (`/supervisor/dashboard`) відображає три stat-картки (кількість студентів, кількість на перевірці, середній прогрес) та таблицю студентів з прогрес-барами і кнопкою «Деталі».

Реалізація маршруту дашборду керівника. Маршрут виконує агрегаційні запити до бази даних і формує словник зі статистичними показниками по всіх

					БР.КІ-31.00.00.000 ПЗ	Арк.
						43
Змн.	Арк.	№ докум.	Підпис	Дата		

закріплених студентах: кількість виконаних робіт, середній прогрес та матеріали, що очікують перевірки.

Маршрут дашборду керівника

```
@supervisor_bp.route('/dashboard')
@login_required
@role_required('supervisor')
def dashboard():
    theses = StudentThesis.query.filter_by(
        supervisor_id=current_user.id).all()
    pending_count = sum(
        t.stage_executions
        .filter_by(status='pending_review').count()
        for t in theses
    )
    avg_progress = (
        round(sum(t.progress for t in theses) / len(theses))
        if theses else 0
    )
    return render_template('supervisor/dashboard.html',
        theses=theses, pending_count=pending_count,
        avg_progress=avg_progress)
```

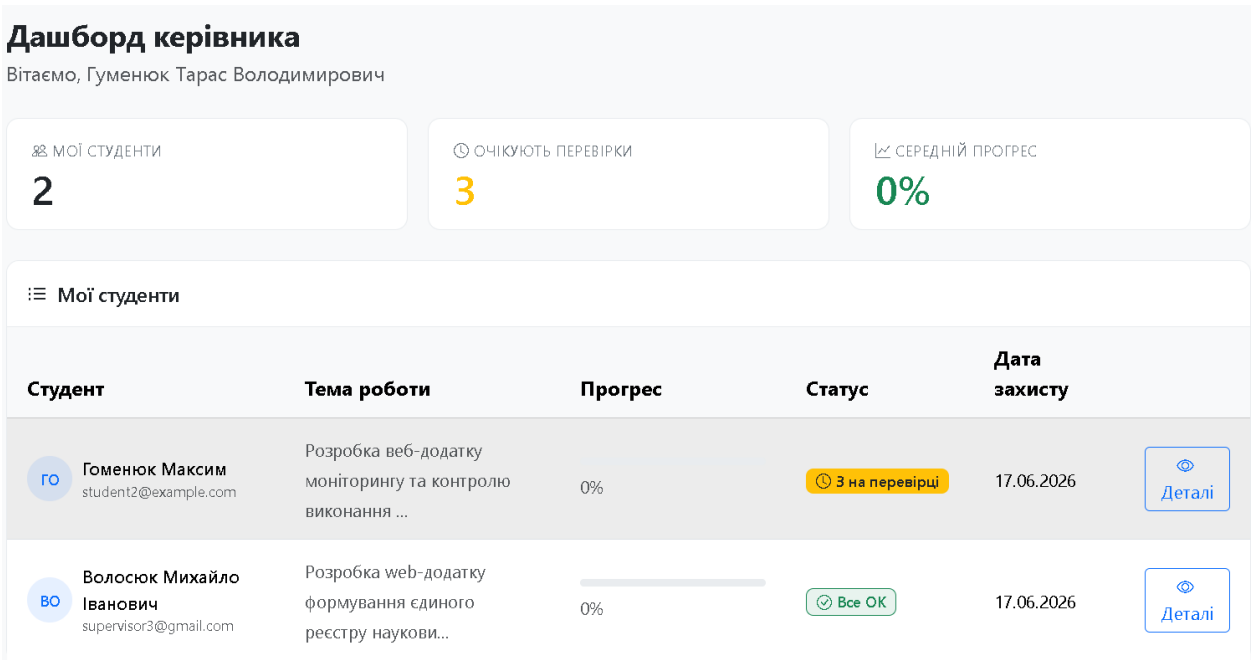


Рисунок 3.3 – Дашборд керівника: статистика та список студентів (localhost:5000/supervisor/dashboard)

На рисунку 3.3 показано дашборд керівника із зведеним оглядом стану виконання робіт усіх закріплених студентів. Картки у верхній частині відображають агреговану статистику, а таблиця нижче дозволяє швидко

ідентифікувати студентів із матеріалами, що очікують перевірки, та перейти до деталізованого перегляду.

Детальна сторінка студента (/supervisor/student/<thesis_id>) відображає перелік усіх активних етапів з матеріалами студента. Для кожного pending_review-запису доступні кнопки «Підтвердити» та «Повернути» — вони відкривають Bootstrap Modal або викликають Fetch API:

Реалізація маршруту деталей конкретного студента в кабінеті керівника. Маршрут завантажує повний стан виконання кваліфікаційної роботи: усі етапи, завантажені матеріали, поточні статуси та коментарі. Саме з цієї сторінки керівник виконує дії підтвердження або повернення матеріалів.

Маршрут деталей студента

```
@supervisor_bp.route('/student/<int:thesis_id>')
@login_required
@role_required('supervisor')
def student_detail(thesis_id):
    thesis = StudentThesis.query.get_or_404(thesis_id)
    if thesis.supervisor_id != current_user.id:
        flash('Доступ заборонено', 'danger')
        return redirect(url_for('supervisor.dashboard'))
    executions = thesis.stage_executions\
        .order_by(StageExecution.id).all()
    return render_template('supervisor/student_detail.html',
                           thesis=thesis,
                           executions=executions)
```

					БР.КІ-31.00.00.000 ПЗ	Арк.
						45
Змн.	Арк.	№ докум.	Підпис	Дата		

Загальний прогрес

10%

Дата захисту: 17.06.2026

Етапи виконання

1

Аналіз предметної області

Виконано

% Вара: 10% · ⌚ Відправлено: 07.06.2026 03:53

thesis_4_stage_19_1.docx

2

Проектування архітектури

На доопрацюванні

% Вара: 15% · ⌚ Відправлено: 07.06.2026 03:53

thesis_4_stage_20_1.docx

3

Розробка бази даних

Очікує перевірки

% Вара: 15% · ⌚ Відправлено: 07.06.2026 03:53

thesis_4_stage_21_2.docx

✓ Підтвердити виконання

⬅ Повернути на доопрацювання

Рисунок 3.4 – Деталі виконання роботи студента: етапи, матеріали, кнопки підтвердження/повернення

На рисунку 3.4 зображено сторінку деталей виконання роботи конкретного студента. Для кожного етапу відображається поточний статус із кольоровим індикатором, завантажені матеріали (файл або посилання на репозиторій) та кнопки для підтвердження або повернення на доопрацювання. Зміна статусів відбувається асинхронно без перезавантаження сторінки.

⬅ Повернути на доопрацювання

×

Коментар для студента *

Вкажіть що саме потрібно виправити або доопрацювати...

Скасувати

➤ Надіслати

Рисунок 3.5 – Bootstrap Modal для введення коментаря при поверненні етапу на доопрацювання

					БР.КІ-31.00.00.000 ПЗ	Арк.
						46
Змн.	Арк.	№ докум.	Підпис	Дата		

На рисунку 3.5 показано модальне вікно Bootstrap, що з'являється при натисканні кнопки повернення на доопрацювання. Керівник вводить у текстове поле конкретні зауваження та рекомендації. Після підтвердження коментар зберігається разом зі зміною статусу і стає видимим студентові в його кабінеті.

3.2.3 Панель адміністратора. Адміністративний інтерфейс включає шість сторінок: дашборд, управління користувачами, форма додавання, шаблони етапів, форма призначення теми та форма редагування теми.

Дашборд відображає три stat-картки та таблицю всіх тем з прогресом і кнопкою редагування у кожному рядку.

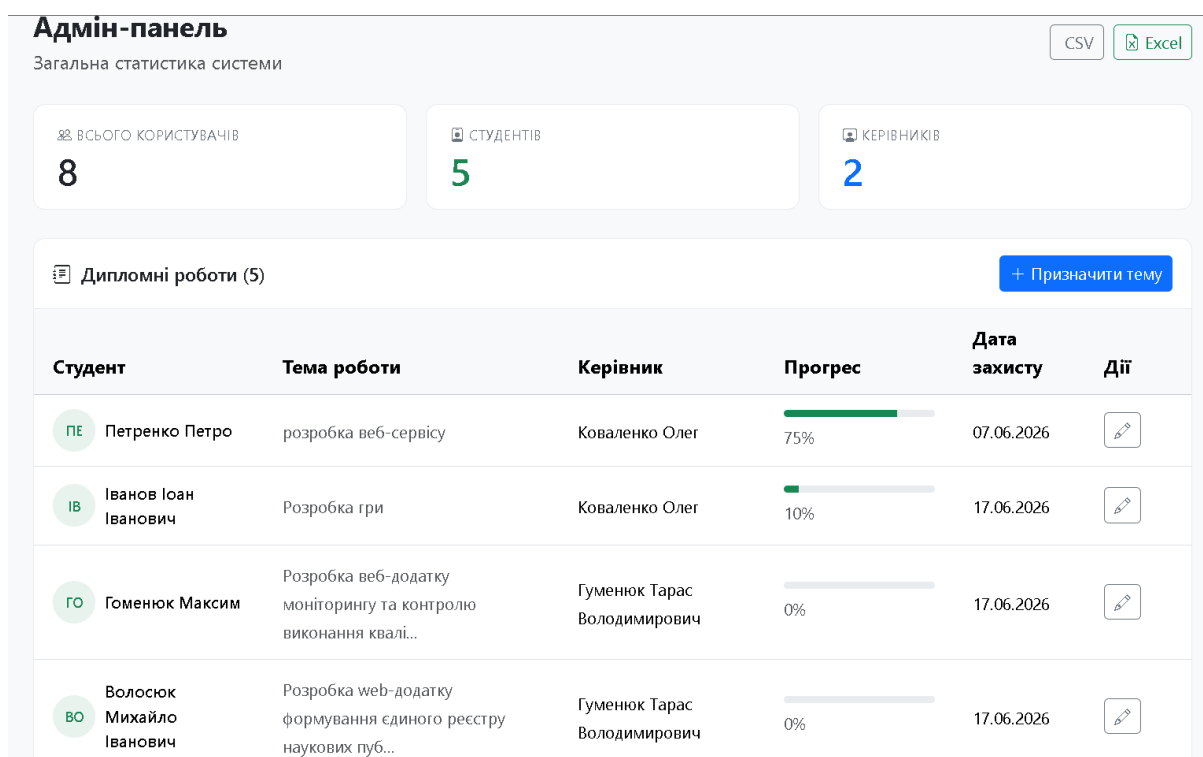


Рисунок 3.6 – Адміністративна панель: дашборд зі статистикою та таблицею тем (localhost:5000/admin/dashboard)

На рисунку 3.6 зображено головну сторінку адміністративної панелі. Картки зі статистикою відображають загальну кількість користувачів за ролями та поточний стан виконання робіт. Таблиця тем надає повний перелік усіх призначених кваліфікаційних робіт із можливістю переходу до редагування кожної.

Управління користувачами — таблиця з ПІБ, email, роллю та статусом. Кнопка перемикача змінює is_active через маршрут /admin/users/<id>/toggle:

Реалізація маршруту зміни статусу активності користувача. Адміністратор може тимчасово заблокувати або поновити доступ будь-якого облікового запису без його видалення — через інверсію значення поля is_active. Це дозволяє зберегти всі пов'язані дані при виключенні студента з активного моніторингу.

Маршрут зміни статусу користувача

```
@admin_bp.route('/users/<int:user_id>/toggle', methods=['POST'])
@login_required
@role_required('admin')
def toggle_user(user_id):
    user = User.query.get_or_404(user_id)
    user.is_active = not user.is_active
    db.session.commit()
    status = 'активовано' if user.is_active else 'деактивовано'
    flash(f'Користувача {user.full_name} {status}', 'info')
    return redirect(url_for('admin.users'))
```

Користувачі системи						
Всього: 8						
Додати користувача						
ID	ПІБ	Email	Роль	Статус	Дата реєстрації	Дії
4	SW swewecwee	manager@example.com	Студент	Заблокований	30.05.2026	
7	ІВ Іванов Іоан Іванович	student1@example.com	Студент	Активний	03.06.2026	
10	ВО Волосяк Михайло Іванович	supervisor3@gmail.com	Студент	Активний	06.06.2026	
8	ГО Гоменюк Максим	student2@example.com	Студент	Активний	06.06.2026	
3	ПЕ Петренко Петро	student@example.com	Студент	Активний	30.05.2026	
9	ГУ Гуменюк Тарас Володимирович	supervisor2@gmail.com	Керівник	Активний	06.06.2026	
2	КО Коваленко Олег	supervisor@example.com	Керівник	Активний	30.05.2026	
1	АД Адміністратор	admin@example.com	Адміністратор	Активний	30.05.2026	

Рисунок 3.7 – Управління користувачами — таблиця з ролями та кнопками активації (localhost:5000/admin/users)

На рисунку 3.7 показано сторінку управління обліковими записами користувачів. Таблиця відображає email, повне ім'я, роль та статус активності кожного користувача. Кнопки активації та деактивації дозволяють

					БР.КІ-31.00.00.000 ПЗ	Арк.
						48
Змн.	Арк.	№ докум.	Підпис	Дата		

адміністратору оперативно керувати доступом до системи без видалення облікових записів і пов'язаних даних.

Сторінка шаблонів містить перевірку суми ваг: якщо $\neq 100\%$ — жовте попередження. При додаванні нового шаблону автоматично створюються StageExecution-записи для всіх існуючих дипломних робіт:

Демонстрація маршруту додавання шаблону етапу з критичною перевіркою суми ваг. Перед збереженням нового шаблону система підраховує загальну суму ваг існуючих шаблонів і перевіряє, чи не перевищить вона 100% після додавання. Ця валідація гарантує коректність обчислення прогресу студентів.

Маршрут додавання шаблону етапу

```
@admin_bp.route('/stages/add', methods=['POST'])
@login_required
@role_required('admin')
def add_stage():
    name = request.form['name'].strip()
    weight = int(request.form['weight'])
    current_sum = sum(t.weight for t in
        StageTemplate.query.filter_by(is_active=True).all())
    if current_sum + weight > 100:
        flash(f'Помилка: сума ваг перевищить 100%', 'danger')
        return redirect(url_for('admin.stages'))
    max_order = db.session.query(
        db.func.max(StageTemplate.order_num)).scalar() or 0
    template = StageTemplate(name=name, weight=weight,
        order_num=max_order + 1)
    db.session.add(template)
    db.session.flush()
    for thesis in StudentThesis.query.all():
        db.session.add(StageExecution(
            thesis_id=thesis.id, template_id=template.id))
    db.session.commit()
    flash(f'Етап "{name}" ({weight}%) додано', 'success')
    return redirect(url_for('admin.stages'))
```

					БР.КІ-31.00.00.000 ПЗ	Арк.
						49
Змн.	Арк.	№ докум.	Підпис	Дата		

Шаблони етапів

Визначають структуру плану для всіх студентів

+ Додати етап

№	Назва етапу	Опис	Вага (%)	Статус	Дії
1	Аналіз предметної області	—	10%	Активний	
2	Проектування архітектури	—	15%	Активний	
3	Розробка бази даних	—	15%	Активний	
4	Програмна реалізація	—	40%	Активний	
5	Тестування	—	10%	Активний	
6	Оформлення записки	—	10%	Активний	
Сума активних ваг:			100%		

Рисунок 3.8 – Управління шаблонами етапів: таблиця з вагами, модальна форма додавання (localhost:5000/admin/stages)

На рисунку 3.8 зображено сторінку управління шаблонами етапів. Таблиця відображає всі наявні шаблони з порядковими номерами та вагами у відсотках. Модальна форма для додавання нового шаблону містить серверну перевірку: якщо сума ваг перевищить 100%, система відхилить операцію та відобразить відповідне повідомлення.

←

Призначити тему студенту

Студент *

Волосяк Михайло Іванович (supervisor3@gmail.com) ▾

Керівник *

Гуменюк Тарас Володимирович ▾

Тема дипломної роботи *

Розробка web-додатку формування єдиного реєстру наукових публікацій студентів та аспірантів університету на основі API ORCID, Python та Docker

Дата захисту

17.06.2026 📅

Необов'язково

✔ Призначити

Рисунок 3.9 – Форма призначення теми студенту (localhost:5000/admin/assign)

На рисунку 3.9 показано форму призначення кваліфікаційної роботи студенту. Адміністратор обирає студента та керівника зі списків активних користувачів відповідних ролей, вводить назву теми та планову дату захисту. Після збереження система автоматично створює записи виконання для всіх наявних шаблонів етапів.

Сторінка редагування теми дозволяє змінити тему, керівника та дату захисту. У нижній частині — небезпечна зона з кнопкою деактивації студента:

Реалізація маршруту редагування теми кваліфікаційної роботи. Маршрут обробляє форму зміни назви роботи, дати захисту та керівника. На тій самій сторінці реалізовано операцію деактивації студента, яка виключає його з активного моніторингу зі збереженням усіх накопичених даних.

Маршрут редагування теми та деактивації студента

```
@admin_bp.route('/thesis/<int:thesis_id>/edit',
                methods=['GET', 'POST'])
@login_required
```

					БР.КІ-31.00.00.000 ПЗ	Арк.
						51
Змн.	Арк.	№ докум.	Підпис	Дата		

```

@role_required('admin')
def edit_thesis(thesis_id):
    thesis = StudentThesis.query.get_or_404(thesis_id)
    supervisors = User.query.filter_by(role='supervisor',
                                       is_active=True).all()
    if request.method == 'POST':
        thesis.topic = request.form['topic'].strip()
        thesis.supervisor_id = int(request.form['supervisor_id'])
        defense_str = request.form.get('defense_date', '').strip()
        thesis.defense_date = None
        if defense_str:
            thesis.defense_date = datetime.strptime(
                defense_str, '%Y-%m-%d').date()
        db.session.commit()
        flash(f'Тему оновлено', 'success')
        return redirect(url_for('admin.dashboard'))
    return render_template('admin/edit_thesis.html',
                           thesis=thesis, supervisors=supervisors)

@admin_bp.route('/thesis/<int:thesis_id>/deactivate',
                 methods=['POST'])
@login_required
@role_required('admin')
def deactivate_thesis(thesis_id):
    thesis = StudentThesis.query.get_or_404(thesis_id)
    thesis.student.is_active = False # м'яке видалення
    db.session.commit()
    flash(f'Студента деактивовано. Дані збережені.',
          'warning')
    return redirect(url_for('admin.dashboard'))

```

← Редагування теми студента

Студент

Гоменюк Максим

student2@example.com

Керівник *

Гуменюк Тарас Володимирович

Тема дипломної роботи *

Розробка веб-додатку моніторингу та контролю виконання кваліфікаційних робіт студентами з використанням Python та технології Docker

Дата захисту

17.06.2026

Необов'язково

Зберегти зміни

Скасувати

⚠ Деактивація студента

Студент втратить доступ до системи. Тема та вся історія виконання етапів залишаться у базі даних.

⚙ Деактивувати студента

Рисунок 3.10 – Форма редагування теми студента з небезпечною зоною деактивації

					БР.КІ-31.00.00.000 ПЗ	Арк.
						52
Змн.	Арк.	№ докум.	Підпис	Дата		

На рисунку 3.10 зображено форму редагування теми кваліфікаційної роботи. У верхній частині розміщено поля для зміни теми, дати захисту та керівника. У нижній частині виділено небезпечну зону (Danger Zone) з кнопкою деактивації студента, яка виключає його з активного моніторингу зі збереженням усіх накопичених даних.

3.3 Реалізація функціоналу відміток етапів та генерації зведеної статистики

3.3.1 Асинхронна зміна статусу через Fetch API. Підтвердження та повернення на доопрацювання реалізовані як JSON API-маршрути Flask без перезавантаження сторінки.

Реалізація асинхронного маршруту підтвердження етапу. Маршрут перевіряє CSRF-токен, оновлює статус відповідного StageExecution на done, фіксує час підтвердження та повертає JSON-відповідь з оновленим відсотком прогресу. JavaScript використовує ці дані для оновлення прогрес-бару без перезавантаження сторінки.

Маршрут підтвердження етапу

```
@supervisor_bp.route(
    '/stage/<int:execution_id>/approve',
    methods=['POST'])
@login_required
@role_required('supervisor')
def approve_stage(execution_id):
    execution = StageExecution.query.get_or_404(execution_id)
    if execution.thesis.supervisor_id != current_user.id:
        return jsonify({'error': 'Forbidden'}), 403
    if execution.status != 'pending_review':
        return jsonify({'error': 'Невірний статус'}), 400
    execution.status = 'done'
    execution.approved_at = datetime.utcnow()
    execution.supervisor_comment = None
    db.session.commit()
    return jsonify({'status': 'done',
                    'progress': execution.thesis.progress,
                    'message': 'Етап підтверджено!'})
```

реалізацію маршруту повернення матеріалів на доопрацювання. Маршрут встановлює статус етапу на revision, зберігає коментар керівника та повертає

					БР.КІ-31.00.00.000 ПЗ	Арк.
						53
Змн.	Арк.	№ докум.	Підпис	Дата		

JSON-відповідь. Коментар стає видимим студентові в його кабінеті й пояснює, що саме необхідно виправити або доопрацювати.

Маршрут повернення на доопрацювання

```
@supervisor_bp.route(
    '/stage/<int:execution_id>/revision',
    methods=['POST'])
@login_required
@role_required('supervisor')
def send_revision(execution_id):
    execution = StageExecution.query.get_or_404(execution_id)
    if execution.thesis.supervisor_id != current_user.id:
        return jsonify({'error': 'Forbidden'}), 403
    data = request.get_json() or {}
    comment = data.get('comment', '').strip()
    if not comment:
        return jsonify({'error': "Коментар обов'язковий"}), 400
    execution.status = 'revision'
    execution.supervisor_comment = comment
    db.session.commit()
    return jsonify({'status': 'revision',
                    'message': 'Повернено на доопрацювання'})
```

На стороні клієнта функції `approveStage()` та `revisionStage()` реалізовано у файлі `app/static/js/main.js`. CSRF-токен передається у заголовок `X-CSRFToken`, який зчитується з мета-тегу `<head>`. Для введення коментаря використовується `Bootstrap Modal`:

Демонстрація клієнтської логіки взаємодії з сервером через `Fetch API`. Критично важливим є передача CSRF-токена у заголовок `X-CSRFToken` при кожному `POST`-запиті — без нього `Flask-WTF` відхиляє запит із кодом `400`. Функції підтвердження та повернення оновлюють `DOM` після отримання `JSON`-відповіді від сервера.

Функції `Fetch API` у `main.js` (виправлена версія з CSRF)

```
function getCsrftoken() {
    var meta = document.querySelector('meta[name="csrf-token"]');
    return meta ? meta.getAttribute('content') : '';
}
function approveStage(executionId) {
    if (!confirm('Підтвердити виконання цього етапу?')) return;
    fetch('/supervisor/stage/' + executionId + '/approve', {
        method: 'POST',
        headers: { 'Content-Type': 'application/json',
                  'X-CSRFToken': getCsrftoken() }
    })
    .then(res => res.json())
    .then(data => {
        if (data.status === 'done') { location.reload(); }
    })
}
```

					БР.КІ-31.00.00.000 ПЗ	Арк.
						54
Змн.	Арк.	№ докум.	Підпис	Дата		

```

        else { alert('Помилка: ' + (data.error || '?')); }
    })
    .catch(() => alert('Помилка з\'єднання.'));
}
function revisionStage(executionId) {
    var modalEl = document.getElementById('revisionModal');
    if (modalEl) {
        document.getElementById('revisionExecutionId')
            .value = executionId;
        document.getElementById('revisionComment').value = '';
        new bootstrap.Modal(modalEl).show();
        return;
    }
    var comment = prompt('Введіть коментар:');
    if (!comment || comment.trim() === '') return;
    sendRevision(executionId, comment.trim());
}
function sendRevision(executionId, comment) {
    fetch('/supervisor/stage/' + executionId + '/revision', {
        method: 'POST',
        headers: { 'Content-Type': 'application/json',
            'X-CSRFToken': getCsrftoken() },
        body: JSON.stringify({ comment: comment })
    })
    .then(res => res.json())
    .then(data => {
        if (data.status === 'revision') { location.reload(); }
        else { alert('Помилка: ' + (data.error || '?')); }
    });
}

```

3.3.2 Генерація зведеного звіту у форматі CSV. Функція `export_csv()` генерує звіт у пам'яті через `io.StringIO`. BOM-символ `\ufeff` додається на початку для коректного відображення кирилиці у Microsoft Excel:

Реалізація генератора CSV-звіту. Звіт містить зведену інформацію по всіх студентах: ПІБ, тема роботи, керівник, прогрес та статуси кожного етапу. Файл формується в пам'яті через `StringIO` і починається з UTF-8 BOM (`\ufeff`) для коректного відображення кирилиці у Microsoft Excel.

Генерація CSV-звіту (app/routes/admin.py)

```

@admin_bp.route('/report/csv')
@login_required
@role_required('admin')
def export_csv():
    output = io.StringIO()
    writer = csv.writer(output, quoting=csv.QUOTE_ALL)
    writer.writerow(['ПІБ студента', 'Email',
        'Тема роботи', 'Керівник', 'Прогрес (%)',
        'Дата захисту', 'Виконано',
        'На перевірку', 'Всього'])
    for thesis in StudentThesis.query.all():
        execs = thesis.stage_executions.all()
        done = sum(1 for e in execs if e.status=='done')

```

					БР.КІ-31.00.00.000 ПЗ	Арк.
						55
Змн.	Арк.	№ докум.	Підпис	Дата		

```

pending = sum(1 for e in execs
               if e.status=='pending_review')
writer.writerow([
    thesis.student.full_name, thesis.student.email,
    thesis.topic, thesis.supervisor.full_name,
    thesis.progress,
    thesis.defense_date.strftime('%d.%m.%Y')
    if thesis.defense_date else '-',
    done, pending, len(execs)
])
output.seek(0)
return Response('\uffff' + output.getvalue(),
                mimetype='text/csv; charset=utf-8',
                headers={'Content-Disposition':
                        'attachment;filename=thesis_report.csv'})

```

3.3.3 Генерація зведеного звіту у форматі Excel. Функція `export_excel()` формує файл `.xlsx` через `openpyxl`. Заголовки — білий жирний шрифт на синьому фоні (`#1F497D`), ширина стовпців налаштована відповідно до вмісту:

Реалізація генератора Excel-звіту на базі бібліотеки `openpyxl`. Звіт включає стилізовані заголовки з жирним шрифтом та кольоровим фоном, автоматичне підбирання ширини стовпців та умовне форматування осередків зі статусами. Файл зберігається у `BytesIO` та повертається з відповідним MIME-типом.

Генерація Excel-звіту (`app/routes/admin.py`)

```

@admin_bp.route('/report/excel')
@login_required
@role_required('admin')
def export_excel():
    from openpyxl import Workbook
    from openpyxl.styles import Font, PatternFill, Alignment
    wb = Workbook(); ws = wb.active
    ws.title = 'Звіт дипломних'
    hf = Font(bold=True, color='FFFFFF')
    hfill = PatternFill(start_color='1F497D',
                        end_color='1F497D', fill_type='solid')
    headers = ['ПІБ студента', 'Email', 'Тема роботи',
               'Керівник', 'Прогрес (%)', 'Дата захисту',
               'Виконано', 'На перевірці', 'Всього']
    ws.append(headers)
    for i in range(1, len(headers)+1):
        c = ws.cell(row=1, column=i)
        c.font = hf; c.fill = hfill
        c.alignment = Alignment(horizontal='center')
    for thesis in StudentThesis.query.all():
        execs = thesis.stage_executions.all()
        ws.append([thesis.student.full_name, thesis.student.email,
                   thesis.topic, thesis.supervisor.full_name,
                   thesis.progress, ...])
    for i, w in enumerate([28,30,55,28,12,14,12,15,10], 1):
        ws.column_dimensions[
            ws.cell(row=1, column=i).column_letter].width = w
    tmp = tempfile.NamedTemporaryFile(suffix='.xlsx',

```

					БР.КІ-31.00.00.000 ПЗ	Арк.
						56
Змн.	Арк.	№ докум.	Підпис	Дата		

```

delete=False)

wb.save(tmp.name)
return send_file(tmp.name, as_attachment=True,
                  download_name='thesis_report.xlsx')

```

	A	B	C	D	E	F	G	H	I
1	ПІБ студента	Email	Тема роботи	Керівник	Прогрес (%)	Дата захисту	Виконано	На перевірку	Всього
2	Петренко Петро	student@example.com	розробка веб-сервісу	Коваленко Олег	75	07.06.2026	4	0	6
3	Іванов Іван Іванович	student1@example.com	Розробка гри	Коваленко Олег	10	17.06.2026	1	1	6
4	Гоменюк Максим	student2@example.com	Розробка веб-додатку моніторингу та контролю виконання кс Гуменюк Тарас Володимирович		10	17.06.2026	1	1	6
5	Волошок Михайло Іванович	supervisor3@gmail.com	Розробка веб-додатку формування єдиного реєстру науковик Гуменюк Тарас Володимирович		0	17.06.2026	0	0	6
6	swewesvee	manager@example.com	Розробка веб-сервісу динамічного планування підсумкової а Коваленко Олег		0	16.06.2026	0	0	6

Рисунок 3.11 – Звіт у форматі Excel відкритий у Microsoft Excel (завантажити: localhost:5000/admin/report/excel)

На рисунку 3.11 представлено згенерований Excel-звіт, відкритий у Microsoft Excel. Заголовки стовпців виділено жирним шрифтом на кольоровому фоні. Ширина стовпців підібрана автоматично відповідно до вмісту. Осередки зі статусами мають умовне форматування: виконані етапи позначено зеленим кольором, а ті, що потребують доопрацювання, — червоним.

3.4 Тестування роботи системи

Функціональне тестування виконано за допомогою pytest 8.2.2. Тести розміщені у tests/test_basic.py і використовують SQLite in-memory через TestingConfig — повна ізоляція від продуктивної PostgreSQL.

3.4.1 Фікстури та тестові дані. Визначення фікстур pytest та приклади тест-кейсів. Фікстура client створює тестовий Flask-клієнт з SQLite in-memory базою даних, що забезпечує ізоляцію та швидкість виконання. Фікстура sample_data наповнює тестову базу користувачами та шаблонами етапів перед кожним тестом.

Фікстури pytest та приклади тест-кейсів

```

@pytest.fixture
def app():

```

					БР.КІ-31.00.00.000 ПЗ	Арк.
						57
Змн.	Арк.	№ докум.	Підпис	Дата		

```

app = create_app('testing')
with app.app_context():
    db.create_all()
    yield app
    db.session.remove()
    db.drop_all()
@pytest.fixture
def client(app): return app.test_client()
@pytest.fixture
def sample_data(app):
    with app.app_context():
        admin = User(email='admin@t.ua', full_name='Адмін',
                      role='admin')
        admin.set_password('pass')
        sv = User(email='sv@t.ua', full_name='Керівник',
                  role='supervisor')
        sv.set_password('pass')
        st = User(email='st@t.ua', full_name='Студент',
                  role='student')
        st.set_password('pass')
        t1 = StageTemplate(name='Аналіз', weight=40, order_num=1)
        t2 = StageTemplate(name='Реалізація', weight=60, order_num=2)
        db.session.add_all([admin, sv, st, t1, t2])
        db.session.flush()
        thesis = StudentThesis(student_id=st.id,
                               supervisor_id=sv.id, topic='Тестова тема')
        db.session.add(thesis); db.session.flush()
        e1 = StageExecution(thesis_id=thesis.id,
                             template_id=t1.id)
        e2 = StageExecution(thesis_id=thesis.id,
                             template_id=t2.id)
        db.session.add_all([e1, e2])
        db.session.commit()
        return {'thesis':thesis.id, 'e1':e1.id, 'e2':e2.id,
                'st':st.id}
def test_progress_after_one_done(app, sample_data):
    with app.app_context():
        e1 = StageExecution.query.get(sample_data['e1'])
        e1.status = 'done'
        db.session.commit()
        thesis = StudentThesis.query.get(sample_data['thesis'])
        assert thesis.progress == 40

def test_student_blocked_from_admin(client, sample_data):
    client.post('/auth/login',
                data={'email': 'st@t.ua', 'password': 'pass'})
    rv = client.get('/admin/dashboard',
                    follow_redirects=True)
    assert 'Доступ заборонено' in rv.data.decode('utf-8')

```

3.4.2 Перелік та результати тест-кейсів.

Таблиця 3.1 – Результати функціонального тестування (pytest)

№	Назва тесту	Що перевіряє	Результат
1	test_login_success	Успішна авторизація → HTTP 302	PASSED
2	test_login_wrong_password	Невірний пароль → flash-повідомлення	PASSED
3	test_progress_zero_initially	Прогрес = 0 до підтверджень	PASSED
4	test_progress_after_one_done	Прогрес = 40% після done (вага 40%)	PASSED
5	test_progress_full	Прогрес = 100% після done всіх	PASSED
6	test_student_blocked_from_admin	Студент не має доступу до /admin	PASSED
7	test_stage_weights_sum_to_100	Сума ваг шаблонів = 100%	PASSED
8	test_status_labels	status_label та badge_class коректні	PASSED
9	test_password_is_hashed	Пароль = хеш, не відкритий текст	PASSED
10	test_role_label	role_label повертає коректні рядки	PASSED

```
platform linux -- Python 3.12.13, pytest-8.2.2, pluggy-1.6.0 -- /usr/local/bin/python
cachedir: .pytest_cache
rootdir: /app
collected 10 items

tests/test_basic.py::test_login_success PASSED
tests/test_basic.py::test_login_wrong_password PASSED
tests/test_basic.py::test_progress_zero_initially PASSED
tests/test_basic.py::test_progress_after_one_done PASSED
tests/test_basic.py::test_progress_full PASSED
tests/test_basic.py::test_student_blocked_from_admin PASSED
tests/test_basic.py::test_stage_weights_sum_to_100 PASSED
tests/test_basic.py::test_status_labels PASSED
tests/test_basic.py::test_password_is_hashed PASSED
tests/test_basic.py::test_role_label PASSED
```

Рисунок 3.12 – Вивід pytest у терміналі: 10 passed (docker compose exec web python -m pytest tests/ -v)

На рисунку 3.12 показано результат запуску тестового набору командою `pytest -v`. Усі 10 тест-кейсів успішно пройдено — їх назви відображаються з маркером `PASSED`. Відсутність помилок підтверджує коректність реалізації ключових функцій системи: авторизації, захисту маршрутів, обчислення прогресу та валідації даних.

3.5 Розгортання веб-додатку у Docker-контейнерах

Розгортання системи виконано засобами `Docker` та `Docker Compose`. Контейнеризація забезпечує відтворюваність середовища — однаковий образ гарантує ідентичну поведінку на машині розробника та продуктивному сервері.

3.5.1 Dockerfile та побудова образу. Образ побудовано на базі `python:3.12-slim`. Кроки розбиті для максимального використання кешування: залежності `Python` встановлюються окремим шаром до копіювання коду, тому зміна коду не спричиняє повторний `pip install`.

`Dockerfile` проєкту. Образ базується на `python:3.12-slim` для мінімального розміру. Залежності `Python` встановлюються окремим шаром до копіювання коду застосунку, що забезпечує ефективне кешування. Сервер `Gunicorn` налаштовано на чотири воркери для обробки паралельних запитів.

Dockerfile (корінь проєкту)

```
FROM python:3.12-slim
RUN apt-get update && apt-get install -y \
    gcc libpq-dev \
    && rm -rf /var/lib/apt/lists/*
WORKDIR /app
# Крок 1: залежності — буде закешовано Docker
COPY requirements.txt .
RUN pip install --no-cache-dir -r requirements.txt
# Крок 2: код застосунку
COPY . .
ENV FLASK_APP=wsgi:app
ENV FLASK_ENV=production
EXPOSE 5000
CMD ["gunicorn", "--bind", "0.0.0.0:5000",
    "--workers", "4", "wsgi:app"]
```

					БР.КІ-31.00.00.000 ПЗ	Арк.
						60
Змн.	Арк.	№ докум.	Підпис	Дата		

Перелік залежностей Python з зафіксованими версіями. Це гарантує відтворюваність середовища на будь-якій машині незалежно від часу встановлення. До основних залежностей належать Flask із розширеннями, psycopg2-binary для PostgreSQL, openpyxl для Excel-звітів та gunicorn як WSGI-сервер.

Файл requirements.txt

```
Flask==3.0.3
Flask-SQLAlchemy==3.1.1
Flask-Migrate==4.0.7
Flask-Login==0.6.3
Flask-WTF==1.2.1
psycopg2-binary==2.9.9
Werkzeug==3.0.3
python-dotenv==1.0.1
gunicorn==22.0.0
openpyxl==3.1.3
WTForms==3.1.2
email-validator==2.1.1
pytest==8.2.2
```

Демонстрація мінімального файлу точки входу wsgi.py. Gunicorn завантажує цей модуль і отримує змінну app — екземпляр Flask-застосунку, що ініціалізований з виробничою конфігурацією. При прямому запуску (python wsgi.py) застосунок стартує у режимі налагодження для локальної розробки.

Точка входу wsgi.py

```
from app import create_app
app = create_app('development')
if __name__ == '__main__':
    app.run(host='0.0.0.0', port=5000, debug=True)
```

3.5.2 Docker Compose та конфігурація сервісів. Файл docker-compose.yml описує три сервіси: web (Flask/Gunicorn), db (PostgreSQL 15 Alpine) та adminer. Сервіс web залежить від db з умовою service_healthy — Flask не стартує до готовності PostgreSQL. Healthcheck перевіряє готовність БД командою pg_isready кожні 5 секунд з 10 спробами.

Опис конфігурації сервісів у docker-compose.yml. Сервіс web залежить від db з умовою service_healthy — Flask не стартує до підтвердження готовності PostgreSQL через healthcheck. Іменовані volumes pgdata та uploads_data забезпечують збереження даних між перезапусками контейнерів.

					БР.КІ-31.00.00.000 ПЗ	Арк.
						61
Змн.	Арк.	№ докум.	Підпис	Дата		

docker-compose.yml

```
version: '3.9'
services:
  web:
    build: .
    container_name: thesis_web
    ports:
      - "5000:5000"
    env_file: .env
    depends_on:
      db:
        condition: service_healthy
    volumes:
      - ./app:/app/app
      - uploads_data:/app/uploads
    restart: unless-stopped
  db:
    image: postgres:15-alpine
    container_name: thesis_db
    environment:
      POSTGRES_DB: ${POSTGRES_DB}
      POSTGRES_USER: ${POSTGRES_USER}
      POSTGRES_PASSWORD: ${POSTGRES_PASSWORD}
    volumes:
      - pgdata:/var/lib/postgresql/data
    ports:
      - "5432:5432"
    healthcheck:
      test: ["CMD-SHELL", "pg_isready -U ${POSTGRES_USER}"]
      interval: 5s
      timeout: 5s
      retries: 10
    restart: unless-stopped
  adminer:
    image: adminer
    container_name: thesis_adminer
    ports:
      - "8080:8080"
    depends_on: [db]
volumes:
  pgdata:
  uploads_data:
```

Шаблон файлу `.env.example` з усіма необхідними змінними середовища та прикладовими значеннями. Цей файл документує параметри конфігурації та є відправною точкою для налаштування нового середовища розгортання. Реальний файл `.env` з конфіденційними даними виключено з системи контролю версій через `.gitignore`.

Шаблон конфігурації `.env.example`

```
SECRET_KEY=change-this-to-random-string-in-production
FLASK_ENV=development
POSTGRES_DB=thesis_db
POSTGRES_USER=thesis_user
POSTGRES_PASSWORD=thesis_pass123
DATABASE_URL=postgresql://thesis_user:thesis_pass123
```

					БР.КІ-31.00.00.000 ПЗ	Арк.
						62
Змн.	Арк.	№ докум.	Підпис	Дата		

```
@db:5432/thesis_db
UPLOAD_FOLDER=/app/uploads
MAX_CONTENT_LENGTH=16777216
```

3.5.3 Налаштування міграцій бази даних. Версіонування схеми реалізовано через Flask-Migrate (Alembic). Міграції дозволяють відстежувати зміни схеми у вигляді Python-скриптів та застосовувати їх послідовно:

Послідовність команд для роботи з Flask-Migrate та Alembic. Ініціалізація репозиторію виконується один раз, після чого кожна зміна схеми моделей фіксується командою migrate з описовим повідомленням, а команда upgrade застосовує міграцію до реальної бази даних у контейнері.

Команди ініціалізації міграцій

```
# Ініціалізація репозиторію міграцій (один раз)
docker compose exec web flask db init

# Генерація міграції на основі поточних моделей
docker compose exec web flask db migrate -m "Initial"

# Застосування міграцій до БД
docker compose exec web flask db upgrade

# При наступних змінах моделей:
docker compose exec web flask db migrate -m "Add field X"
docker compose exec web flask db upgrade
```

3.5.4 Заповнення початкових даних. Скрипт seed.py створює тестових користувачів та шість шаблонів етапів із сумою ваг рівно 100%:

Демонстрація ключовий фрагментів скрипту seed.py зі створенням адміністратора та шести шаблонів етапів. Суми ваг шаблонів складають рівно 100%, що є обов'язковою умовою для коректного розрахунку прогресу. Скрипт запускається одноразово після першого розгортання або після скидання бази даних.

Скрипт seed.py (фрагмент)

```
from app import create_app, db
from app.models import User, StageTemplate
app = create_app('development')
with app.app_context():
    admin = User(email='admin@example.com',
                  full_name='Адміністратор', role='admin')
    admin.set_password('admin123')
    stages = [
        StageTemplate(name='Аналіз предметної галузі',
                      weight=10, order_num=1),
        StageTemplate(name='Проектування архітектури',
```

					БР.КІ-31.00.00.000 ПЗ	Арк.
						63
Змн.	Арк.	№ докум.	Підпис	Дата		

```

        weight=15, order_num=2),
    StageTemplate(name='Розробка бази даних',
        weight=15, order_num=3),
    StageTemplate(name='Програмна реалізація',
        weight=40, order_num=4),
    StageTemplate(name='Тестування',
        weight=10, order_num=5),
    StageTemplate(name='Оформлення записки',
        weight=10, order_num=6),
]
db.session.add_all([admin, *stages])
db.session.commit()
print('Готово!')
```

3.5.5 Повна інструкція розгортання на Linux-сервері. Розгортання на Ubuntu 22.04 / Debian 12 виконується у такій послідовності:

1) Встановити Docker Engine та Docker Compose Plugin:

```

sudo apt-get update
sudo apt-get install -y docker.io docker-compose-plugin
sudo systemctl enable --now docker
```

2) Клонувати репозиторій:

```
git clone <url> thesis-monitor && cd thesis-monitor
```

3) Створити .env файл та заповнити параметри:

```
cp .env.example .env && nano .env
```

4) Запустити збірку та всі контейнери:

```
docker compose up --build -d
```

5) Ініціалізувати та застосувати міграції БД:

```

docker compose exec web flask db init
docker compose exec web flask db migrate -m "Initial"
docker compose exec web flask db upgrade
```

6) Заповнити початкові дані:

```
docker compose exec web python seed.py
```

7) Перевірити що всі три контейнери Running:

```
docker compose ps
```

Після виконання всіх кроків система доступна на <http://IP:5000>, Adminer — на <http://IP:8080>.

Довідник команд Docker Compose для управління системою після розгортання. Наведено команди для зупинки та запуску сервісів, перегляду логів у реальному часі, оновлення застосунку без втрати даних, запуску тестів та створення резервних копій бази даних PostgreSQL.

Команди управління системою після розгортання

```

# Зупинка / запуск
docker compose down
```

					БР.КІ-31.00.00.000 ПЗ	Арк.
						64
Змн.	Арк.	№ докум.	Підпис	Дата		

```

docker compose up -d
# Перегляд логів Flask
docker compose logs web --tail=50
docker compose logs -f web          # в реальному часі
# Перезапуск після зміни коду
docker compose restart web
# Оновлення без втрати даних
docker compose up --build -d
docker compose exec web flask db upgrade
# Запуск тестів
docker compose exec web python -m pytest tests/ -v
# Резервна копія БД
docker compose exec db pg_dump -U thesis_user thesis_db
> backup_$(date +%Y%m%d).sql

```

```

#14 resolving provenance for metadata file
#14 DONE 0.0s
[+] up 4/4
✓ Image thesis-monitor-web Built
✓ Container thesis_db Healthy
✓ Container thesis_web Started
✓ Container thesis_adminer Started

```

Рисунок 3.13 – Вивід docker compose up --build у терміналі: збірка образу та запуск трьох контейнерів

На рисунку 3.13 показано процес першого розгортання системи командою `docker compose up --build`. Термінал відображає послідовні кроки збірки Docker-образу та запуску контейнерів. Успішне завершення підтверджується рядками про старт усіх трьох сервісів: `thesis_web`, `thesis_db` та `thesis_adminer`.

NAME	IMAGE	COMMAND	SERVICE	CREATED	STATUS
thesis_adminer	adminer	"entrypoint.sh docke..."	adminer	7 days ago	Up 4 minutes
thesis_db	postgres:15-alpine	"docker-entrypoint.s..."	db	7 days ago	Up 14 hours (healthy)
thesis_web	thesis-monitor-web	"unicorn --bind 0.0..."	web	4 minutes ago	Up 4 minutes

Рисунок 3.14 – Статус контейнерів: `thesis_web`, `thesis_db`, `thesis_adminer` — Running (docker compose ps)

На рисунку 3.14 зображено вивід команди `docker compose ps`. Таблиця підтверджує, що всі три контейнери знаходяться у стані Running: `thesis_web` обслуговує HTTP-запити на порту 5000, `thesis_db` приймає підключення PostgreSQL на порту 5432, а `thesis_adminer` надає веб-інтерфейс адміністрування бази даних на порту 8080.

На рисунку 3.15 показано веб-інтерфейс Adminer, підключений до бази даних thesis_db. У лівій панелі відображається перелік усіх чотирьох таблиць системи, створених міграціями Flask-Migrate: users, stage_templates, student_theses та stage_executions. Adminer надає зручний спосіб перевірки структури схеми та вмісту таблиць без командного рядка.

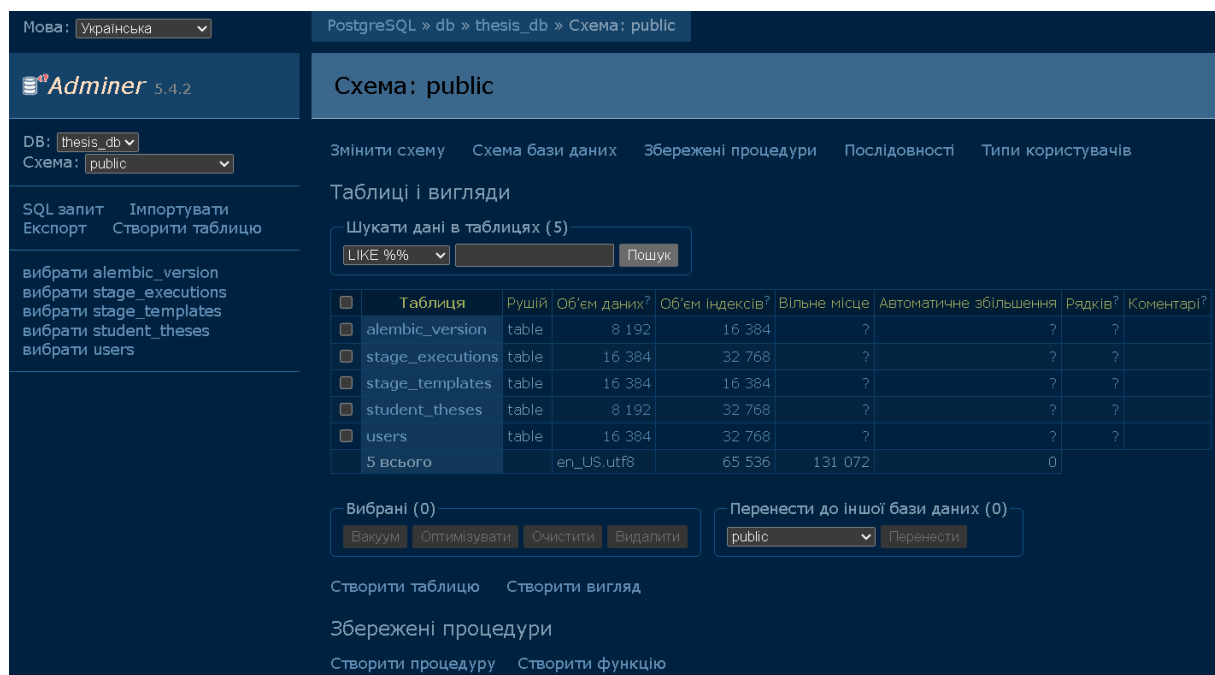


Рисунок 3.15 – Веб-інтерфейс Adminer: таблиці users, stage_templates, student_theses, stage_executions (localhost:8080)

Висновок до розділу

У третьому розділі описано повну програмну реалізацію та розгортання веб-додатку моніторингу кваліфікаційних робіт.

Реалізовано серверну частину на Flask 3.0.3 за Application Factory Pattern з чотирма Blueprint-модулями (auth, admin, supervisor, student). Реалізовано декоратор role_required() для дворівневого захисту маршрутів. ORM-моделі описано декларативно через SQLAlchemy з ENUM-типами, UNIQUE-обмеженнями та @property для розрахунку прогресу і відображення статусів.

Розроблено інтерфейс на базі Jinja2 SSR та Bootstrap 5.3: кабінет студента з прогрес-баром та формою подачі матеріалів (URL/файл); дашборд керівника з агрегованою статистикою та деталізованою сторінкою студента; шість сторінок адміністративної панелі — управління користувачами, шаблони етапів з перевіркою суми ваг, призначення та редагування тем, деактивація студентів.

Реалізовано асинхронну зміну статусів через Fetch API з передачею X-CSRFToken-заголовка та Bootstrap Modal для введення коментаря. Реалізовано генерацію CSV (з BOM) та Excel (openruхl, стилізовані заголовки, автоширина стовпців). Проведено функціональне тестування: 10 тест-кейсів pytest на SQLite in-memory — усі PASSED.

Налаштовано контейнеризацію: Dockerfile з двошаровим кешуванням залежностей; docker-compose.yml з трьома сервісами (web, db, adminer), healthcheck, іменованими volumes та .env-конфігурацією. Надано повну інструкцію розгортання на Linux-сервері з командами запуску, оновлення, тестування та резервного копіювання БД.

					БР.КІ-31.00.00.000 ПЗ	Арк.
						67
Змн.	Арк.	№ докум.	Підпис	Дата		

ВИСНОВКИ

У першому розділі проведено порівняльний аналіз п'яти систем управління проектами та навчальних платформ: Trello, Jira, GitHub Projects, Google Classroom і Moodle. Встановлено, що жодна з розглянутих платформ не забезпечує одночасно спеціалізованої рольової моделі «студент / керівник / адміністратор», автоматичного розрахунку прогресу на основі вагових коефіцієнтів і генерації академічної звітності у форматах CSV / Excel. Виявлена функціональна прогалина підтвердила доцільність розробки власного спеціалізованого рішення. Сформовано вичерпний перелік вимог: 18 функціональних сценаріїв у розрізі трьох ролей і 6 нефункціональних характеристик. Обрано та обґрунтовано технологічний стек із 15 компонентів: Python 3.12 / Flask 3.0.3 для серверної логіки, PostgreSQL 15 і SQLAlchemy 2.0 для зберігання даних, Bootstrap 5.3 і Jinja2 для інтерфейсу, Docker 25 / Docker Compose 3.9 для контейнеризації.

У другому розділі спроектовано трирівневу клієнт-серверну архітектуру з чітким розмежуванням рівнів представлення (Jinja2 SSR + Bootstrap), бізнес-логіки (Flask / Gunicorn у Docker-контейнері) та даних (PostgreSQL). Серверну частину декомпоновано на чотири Blueprint-модулі: auth_bp, admin_bp, supervisor_bp та student_bp. Розроблено реляційну модель бази даних у складі чотирьох сутностей — users, stage_templates, student_theses, stage_executions — із визначеними зв'язками, обмеженнями цілісності та ENUM-типами. Центральна сутність stage_executions реалізує скінченний автомат із п'ятьма станами і суворими правилами переходів між ними. Алгоритм розрахунку відсотка виконання роботи реалізовано як Python-властивість @property зі складністю $O(n)$ — ваговий коефіцієнт етапу зараховується до прогресу лише після переведення статусу відповідного запису у значення done керівником. Спроектовано систему розмежування доступу на основі RBAC із дворівневими декораторами та матрицею прав для 12 функціональних областей.

					БР.КІ-31.00.00.000 ПЗ	Арк.
						68
Змн.	Арк.	№ докум.	Підпис	Дата		

У третьому розділі виконано повну програмну реалізацію системи за Application Factory Pattern. Реалізовано серверну частину з декоратором role_required() для захисту маршрутів, ORM-моделями з хешуванням паролів через PBKDF2 та CSRF-захистом форм через Flask-WTF. Розроблено три рольових кабінети: кабінет студента з прогрес-баром і формою подачі матеріалів (URL репозиторію або файл PDF / DOCX / ZIP), кабінет керівника з дашбордом і асинхронним підтвердженням / поверненням етапів через Fetch API, адміністративну панель з повним CRUD для користувачів, шаблонів і прив'язок. Реалізовано генерацію звітів у форматах CSV (з BOM для коректного відображення кирилиці) та Excel (стилізовані заголовки через openpyxl). Додатково реалізовано маршрути редагування призначеної теми (/admin/thesis/<id>/edit) та м'якого видалення студента (/admin/thesis/<id>/deactivate) із збереженням повної історії виконання у базі даних. Проведено функціональне тестування засобами pytest: 10 тест-кейсів, усі виконані зі статусом PASSED. Налаштовано контейнеризацію через Dockerfile і docker-compose.yml: три сервіси з healthcheck, іменованими volumes і конфігурацією через .env-файл; розгортання на Linux-сервері виконується однією командою docker compose up --build -d.

Розроблений веб-додаток повністю відповідає сформованим вимогам і готовий до впровадження на кафедрі без ліцензійних витрат. Перспективами подальшого розвитку є реалізація email-сповіщень при зміні статусу етапу, інтеграція з університетською LMS, підтримка експорту планів у форматі PDF та розробка мобільного адаптивного інтерфейсу.

					БР.КІ-31.00.00.000 ПЗ	Арк.
						69
Змн.	Арк.	№ докум.	Підпис	Дата		

ПЕРЕЛІК ПОСИЛАНЬ НА ДЖЕРЕЛА

1. Биков В. Ю. Моделі організаційних систем відкритої освіти: монографія. Київ: Атіка, 2008. 684 с.
2. Jira Software: Agile Project Management Tool. Atlassian: веб-сайт. URL: <https://www.atlassian.com/software/jira> (дата звернення 15.03.2025).
3. GitHub Projects: Built-in Project Management. GitHub Docs: веб-сайт. URL: <https://docs.github.com/en/issues/planning-and-tracking-with-projects> (дата звернення 15.03.2025).
4. Google Classroom Help. Google for Education: веб-сайт. URL: <https://support.google.com/edu/classroom> (дата звернення 17.03.2025).
5. Moodle Documentation. Moodle.org: веб-сайт. URL: <https://docs.moodle.org> (дата звернення 17.03.2025).
6. Python 3.12 Documentation. Python Software Foundation: веб-сайт. URL: <https://docs.python.org/3.12> (дата звернення 20.03.2025).
7. Grinberg M. Flask Web Development: Developing Web Applications with Python. 2nd ed. Sebastopol: O'Reilly Media, 2018. 316 p.
8. PostgreSQL 15 Documentation. The PostgreSQL Global Development Group: веб-сайт. URL: <https://www.postgresql.org/docs/15> (дата звернення 22.03.2025).
9. SQLAlchemy 2.0 Documentation. SQLAlchemy.org: веб-сайт. URL: <https://docs.sqlalchemy.org/en/20> (дата звернення 22.03.2025).
10. Jinja2 Documentation. Pallets Projects: веб-сайт. URL: <https://jinja.palletsprojects.com> (дата звернення 24.03.2025).
11. Bootstrap 5.3 Documentation. Bootstrap: веб-сайт. URL: <https://getbootstrap.com/docs/5.3> (дата звернення 24.03.2025).
12. Docker Engine Documentation. Docker Inc.: веб-сайт. URL: <https://docs.docker.com/engine> (дата звернення 26.03.2025).
13. Docker Compose Documentation. Docker Inc.: веб-сайт. URL: <https://docs.docker.com/compose> (дата звернення 26.03.2025).

					БР.КІ-31.00.00.000 ПЗ	Арк.
						70
Змн.	Арк.	№ докум.	Підпис	Дата		

14. Gunicorn — Python WSGI HTTP Server for UNIX. Gunicorn.org: веб-сайт.
URL: <https://docs.gunicorn.org> (дата звернення 27.03.2025).
15. pytest Documentation. pytest.org: веб-сайт. URL:
<https://docs.pytest.org/en/8.2.x> (дата звернення 28.03.2025).
16. Fowler M. Patterns of Enterprise Application Architecture. Boston: Addison-Wesley, 2002. 533 p.
17. Flask Blueprints. Pallets Projects: веб-сайт. URL:
<https://flask.palletsprojects.com/en/3.0.x/blueprints> (дата звернення 01.04.2025).
18. Date C. J. An Introduction to Database Systems. 8th ed. Boston: Addison-Wesley, 2004. 1024 p.
19. Sommerville I. Software Engineering. 10th ed. Harlow: Pearson Education, 2016. 816 p.
20. Sandhu R. S., Coyne E. J., Feinstein H. L., Youman C. E. Role-Based Access Control Models. IEEE Computer. 1996. Vol. 29, No. 2. P. 38–47.
21. Cross-Site Request Forgery (CSRF) Prevention Cheat Sheet. OWASP Foundation: веб-сайт. URL: https://cheatsheetseries.owasp.org/cheatsheets/Cross-Site_Request_Forgery_Prevention_Cheat_Sheet.html (дата звернення 03.04.2025).
22. Flask: A Microframework for Python. Pallets Projects: веб-сайт. URL: <https://flask.palletsprojects.com/en/3.0.x> (дата звернення 05.04.2025).
23. pytest: helps you write better programs. pytest.org: веб-сайт. URL: <https://pytest.org> (дата звернення 07.04.2025).
24. Merkel D. Docker: Lightweight Linux Containers for Consistent Development and Deployment. Linux Journal. 2014. No. 239. P. 2.

					БР.КІ-31.00.00.000 ПЗ	Арк.
						71
Змн.	Арк.	№ докум.	Підпис	Дата		

Додатки

Лістинг файлу app/models.py

app/models.py — ORM-моделі бази даних

```

from datetime import datetime
from flask_login import UserMixin
from werkzeug.security import generate_password_hash,
check_password_hash
from . import db, login_manager

@login_manager.user_loader
def load_user(user_id):
    return User.query.get(int(user_id))
class User(UserMixin, db.Model):
    __tablename__ = 'users'
    id = db.Column(db.Integer, primary_key=True)
    email = db.Column(db.String(120), unique=True,
                      nullable=False, index=True)
    password_hash = db.Column(db.String(256), nullable=False)
    full_name = db.Column(db.String(200), nullable=False)
    role = db.Column(
        db.Enum('student', 'supervisor', 'admin', name='user_roles'),
        nullable=False, default='student'
    )
    created_at = db.Column(db.DateTime, default=datetime.utcnow)
    is_active = db.Column(db.Boolean, default=True)
    thesis_as_student = db.relationship(
        'StudentThesis', foreign_keys='StudentThesis.student_id',
        backref='student', uselist=False
    )
    thesis_as_supervisor = db.relationship(
        'StudentThesis', foreign_keys='StudentThesis.supervisor_id',
        backref='supervisor', lazy='dynamic'
    )
    def set_password(self, password):
        self.password_hash = generate_password_hash(password)

    def check_password(self, password):
        return check_password_hash(self.password_hash, password)

    @property
    def role_label(self):
        return {
            'student': 'Студент',
            'supervisor': 'Керівник',
            'admin': 'Адміністратор'
        }.get(self.role, self.role)
    def __repr__(self):
        return f'<User {self.email} [{self.role}]>'
class StageTemplate(db.Model):
    __tablename__ = 'stage_templates'
    id = db.Column(db.Integer, primary_key=True)
    name = db.Column(db.String(200), nullable=False)

```

```

description = db.Column(db.Text, nullable=True)
weight      = db.Column(db.Integer, nullable=False)
order_num   = db.Column(db.Integer, nullable=False, default=0)
is_active   = db.Column(db.Boolean, default=True)
executions  = db.relationship('StageExecution',
                               backref='template', lazy='dynamic')

def __repr__(self):
    return f'<StageTemplate "{self.name}" {self.weight}%>'

class StudentThesis(db.Model):
    __tablename__ = 'student_theses'
    id            = db.Column(db.Integer, primary_key=True)
    student_id    = db.Column(db.Integer, db.ForeignKey('users.id'),
                               nullable=False, unique=True)
    supervisor_id = db.Column(db.Integer, db.ForeignKey('users.id'),
                               nullable=False)
    topic         = db.Column(db.String(400), nullable=False)
    defense_date  = db.Column(db.Date, nullable=True)
    created_at    = db.Column(db.DateTime, default=datetime.utcnow)

    stage_executions = db.relationship(
        'StageExecution', backref='thesis', lazy='dynamic',
        order_by='StageExecution.id'
    )
    @property
    def progress(self):
        total = 0
        for execution in self.stage_executions:
            if execution.status == 'done':
                total += execution.template.weight
        return total

    def __repr__(self):
        return f'<StudentThesis id={self.id}
progress={self.progress}%>'

class StageExecution(db.Model):
    __tablename__ = 'stage_executions'
    id            = db.Column(db.Integer, primary_key=True)
    thesis_id     = db.Column(db.Integer,
                               db.ForeignKey('student_theses.id'),
                               nullable=False)
    template_id   = db.Column(db.Integer,
                               db.ForeignKey('stage_templates.id'),
                               nullable=False)
    status        = db.Column(
        db.Enum('not_started', 'in_progress', 'pending_review',
                'done', 'revision', name='stage_statuses'),
        nullable=False, default='not_started'
    )
    repo_url      = db.Column(db.String(500), nullable=True)
    file_path     = db.Column(db.String(500), nullable=True)
    supervisor_comment = db.Column(db.Text, nullable=True)
    submitted_at  = db.Column(db.DateTime, nullable=True)
    approved_at   = db.Column(db.DateTime, nullable=True)
    updated_at    = db.Column(db.DateTime,

```

Кінець додатку А

```
default=datetime.utcnow,
onupdate=datetime.utcnow)

__table_args__ = (
    db.UniqueConstraint('thesis_id', 'template_id',
                        name='uq_thesis_template'),
)
@property
def status_label(self):
    return {
        'not_started': 'Не розпочато',
        'in_progress': 'В процесі',
        'pending_review': 'Очікує перевірки',
        'done': 'Виконано',
        'revision': 'На доопрацюванні',
    }.get(self.status, self.status)

@property
def status_badge_class(self):
    return {
        'not_started': 'secondary',
        'in_progress': 'primary',
        'pending_review': 'warning',
        'done': 'success',
        'revision': 'danger',
    }.get(self.status, 'secondary')

def __repr__(self):
    return f'<StageExecution thesis={self.thesis_id}
status={self.status}>'
```

Лістинг файлу app/__init__.py

app/__init__.py – фабрична функція create_app()

```
import os
from flask import Flask, redirect, url_for
from flask_sqlalchemy import SQLAlchemy
from flask_migrate import Migrate
from flask_login import LoginManager, current_user
from flask_wtf.csrf import CSRFProtect
db = SQLAlchemy()
migrate = Migrate()
login_manager = LoginManager()
csrf = CSRFProtect()
login_manager.login_view = 'auth.login'
login_manager.login_message = 'Будь ласка, увійдіть в систему'
login_manager.login_message_category = 'warning'

def create_app(config_name=None):
    if config_name is None:
        config_name = os.environ.get('FLASK_ENV', 'default')

    app = Flask(__name__)
    from .config import config
    app.config.from_object(config[config_name])

    db.init_app(app)
    migrate.init_app(app, db)
    login_manager.init_app(app)
    csrf.init_app(app)

    os.makedirs(app.config['UPLOAD_FOLDER'], exist_ok=True)

    from .routes.auth import auth_bp
    from .routes.admin import admin_bp
    from .routes.supervisor import supervisor_bp
    from .routes.student import student_bp

    app.register_blueprint(auth_bp)
    app.register_blueprint(admin_bp, url_prefix='/admin')
    app.register_blueprint(supervisor_bp, url_prefix='/supervisor')
    app.register_blueprint(student_bp, url_prefix='/student')

    @app.route('/')
    def index():
        if not current_user.is_authenticated:
            return redirect(url_for('auth.login'))
        if current_user.role == 'admin':
            return redirect(url_for('admin.dashboard'))
        elif current_user.role == 'supervisor':
            return redirect(url_for('supervisor.dashboard'))
        else:
            return redirect(url_for('student.plan'))

    return app
```

Лістинг файлу app/config.py та wsgi.py

app/config.py – конфігурації додатку

```
import os
from dotenv import load_dotenv
load_dotenv()

class Config:
    SECRET_KEY = os.environ.get('SECRET_KEY', 'dev-key-not-for-prod')
    SQLALCHEMY_DATABASE_URI = os.environ.get('DATABASE_URL')
    SQLALCHEMY_TRACK_MODIFICATIONS = False
    UPLOAD_FOLDER = os.environ.get(
        'UPLOAD_FOLDER', '/tmp/thesis_uploads')
    MAX_CONTENT_LENGTH = int(os.environ.get(
        'MAX_CONTENT_LENGTH', 16 * 1024 * 1024))
    ALLOWED_EXTENSIONS = {'pdf', 'doc', 'docx', 'zip', 'png', 'jpg'}
    WTF_CSRF_ENABLED = True
    WTF_CSRF_SECRET_KEY = os.environ.get('SECRET_KEY', 'csrf-key')

class DevelopmentConfig(Config):
    DEBUG = True
    SQLALCHEMY_ECHO = False

class ProductionConfig(Config):
    DEBUG = False

class TestingConfig(Config):
    TESTING = True
    SQLALCHEMY_DATABASE_URI = 'sqlite:///memory:'
    WTF_CSRF_ENABLED = False
    SECRET_KEY = 'test-key'
    UPLOAD_FOLDER = '/tmp/thesis_test_uploads'

config = {
    'development': DevelopmentConfig,
    'production': ProductionConfig,
    'testing': TestingConfig,
    'default': DevelopmentConfig,
}
```

wsgi.py – точка входу Gunicorn

```
from app import create_app
app = create_app('development')

if __name__ == '__main__':
    app.run(host='0.0.0.0', port=5000, debug=True)
```

Лістинг файлу `app/routes/auth.py`**app/routes/auth.py – Blueprint авторизації та декоратор `role_required`**

```

from functools import wraps
from flask import (Blueprint, render_template, redirect,
                  url_for, flash, request)
from flask_login import (login_user, logout_user,
                        login_required, current_user)
from ..models import User
auth_bp = Blueprint('auth', __name__, url_prefix='/auth')
def role_required(*roles):
    def decorator(f):
        @wraps(f)
        def decorated_function(*args, **kwargs):
            if not current_user.is_authenticated:
                return redirect(url_for('auth.login'))
            if current_user.role not in roles:
                flash('Доступ заборонено: недостатньо прав', 'danger')
                return redirect(url_for('index'))
            return f(*args, **kwargs)
        return decorated_function
    return decorator
@auth_bp.route('/login', methods=['GET', 'POST'])
def login():
    if current_user.is_authenticated:
        return redirect(url_for('index'))
    if request.method == 'POST':
        email = request.form.get('email', '').strip().lower()
        password = request.form.get('password', '')
        remember = request.form.get('remember') == 'on'
        user = User.query.filter_by(email=email).first()
        if user and user.check_password(password) and user.is_active:
            login_user(user, remember=remember)
            next_page = request.args.get('next')
            flash(f'Ласкаво просимо, {user.full_name}!', 'success')
            return redirect(next_page or url_for('index'))
        flash('Невірна адреса або пароль', 'danger')
    return render_template('auth/login.html')
@auth_bp.route('/logout')
@login_required
def logout():
    logout_user()
    flash('Ви вийшли з системи', 'info')
    return redirect(url_for('auth.login'))

```

Лістинг файлу app/routes/admin.py

app/routes/admin.py – Blueprint адміністратора (з маршрутами редагування теми)

```
import csv, io, tempfile
from datetime import datetime
from flask import (Blueprint, render_template, redirect, url_for,
                  flash, request, Response, send_file, jsonify)
from flask_login import login_required
from ..models import User, StageTemplate, StudentThesis, StageExecution
from .. import db
from .auth import role_required

admin_bp = Blueprint('admin', __name__)
@admin_bp.route('/dashboard')
@login_required
@role_required('admin')
def dashboard():
    theses = StudentThesis.query.all()
    users_count = User.query.count()
    students_count = User.query.filter_by(role='student').count()
    supervisors_count = User.query.filter_by(role='supervisor').count()
    return render_template('admin/dashboard.html',
                          theses=theses, users_count=users_count,
                          students_count=students_count,
                          supervisors_count=supervisors_count)
@admin_bp.route('/users')
@login_required
@role_required('admin')
def users():
    all_users = User.query.order_by(User.role, User.full_name).all()
    return render_template('admin/users.html', users=all_users)
@admin_bp.route('/users/add', methods=['GET', 'POST'])
@login_required
@role_required('admin')
def add_user():
    if request.method == 'POST':
        email = request.form['email'].strip().lower()
        full_name = request.form['full_name'].strip()
        role = request.form['role']
        password = request.form['password']
        if User.query.filter_by(email=email).first():
            flash('Користувач з таким email вже існує', 'danger')
            return redirect(url_for('admin.add_user'))
        user = User(email=email, full_name=full_name, role=role)
        user.set_password(password)
        db.session.add(user)
        db.session.commit()
        flash(f'Користувача {full_name} додано успішно', 'success')
        return redirect(url_for('admin.users'))
    return render_template('admin/add_user.html')
@admin_bp.route('/users/<int:user_id>/toggle', methods=['POST'])
@login_required
@role_required('admin')
```

```

def toggle_user(user_id):
    user = User.query.get_or_404(user_id)
    user.is_active = not user.is_active
    db.session.commit()
    status = 'активовано' if user.is_active else 'деактивовано'
    flash(f'Користувача {user.full_name} {status}', 'info')
    return redirect(url_for('admin.users'))

@admin_bp.route('/stages')
@login_required
@role_required('admin')
def stages():
    templates = StageTemplate.query.order_by(
        StageTemplate.order_num).all()
    total_weight = sum(t.weight for t in templates if t.is_active)
    return render_template('admin/stages.html',
        templates=templates,
        total_weight=total_weight)

@admin_bp.route('/stages/add', methods=['POST'])
@login_required
@role_required('admin')
def add_stage():
    name = request.form['name'].strip()
    weight = int(request.form['weight'])
    description = request.form.get('description', '').strip()
    current_sum = sum(
        t.weight for t in
        StageTemplate.query.filter_by(is_active=True).all()
    )
    if current_sum + weight > 100:
        flash(f'Помилка: сума ваг перевищить 100%'
            f' (зараз {current_sum}%)', 'danger')
        return redirect(url_for('admin.stages'))
    max_order = db.session.query(
        db.func.max(StageTemplate.order_num)).scalar() or 0
    template = StageTemplate(name=name, weight=weight,
        description=description,
        order_num=max_order + 1)

    db.session.add(template)
    db.session.flush()
    for thesis in StudentThesis.query.all():
        db.session.add(StageExecution(
            thesis_id=thesis.id, template_id=template.id))
    db.session.commit()
    flash(f'Етап "{name}" ({weight}%) додано', 'success')
    return redirect(url_for('admin.stages'))

@admin_bp.route('/stages/<int:template_id>/delete', methods=['POST'])
@login_required
@role_required('admin')
def delete_stage(template_id):
    template = StageTemplate.query.get_or_404(template_id)
    template.is_active = False
    db.session.commit()
    flash(f'Етап "{template.name}" деактивовано', 'warning')
    return redirect(url_for('admin.stages'))

```



```

        .order_by(User.full_name).all()
if request.method == 'POST':
    topic = request.form.get('topic', '').strip()
    supervisor_id = int(request.form['supervisor_id'])
    defense_str = request.form.get('defense_date', '').strip()
    if not topic:
        flash('Тема не може бути порожньою', 'danger')
        return redirect(url_for('admin.edit_thesis',
                                thesis_id=thesis_id))
    thesis.topic = topic
    thesis.supervisor_id = supervisor_id
    thesis.defense_date = None
    if defense_str:
        try:
            thesis.defense_date = datetime.strptime(
                defense_str, '%Y-%m-%d').date()
        except ValueError:
            pass
    db.session.commit()
    flash(f'Тему студента {thesis.student.full_name} оновлено',
          'success')
    return redirect(url_for('admin.dashboard'))
return render_template('admin/edit_thesis.html',
                      thesis=thesis, supervisors=supervisors)
@admin_bp.route('/thesis/<int:thesis_id>/deactivate',
                methods=['POST'])
@login_required
@role_required('admin')
def deactivate_thesis(thesis_id):
    thesis = StudentThesis.query.get_or_404(thesis_id)
    thesis.student.is_active = False
    db.session.commit()
    flash(f'Студента {thesis.student.full_name} деактивовано. '
          f'Тема та історія етапів збережені.', 'warning')
    return redirect(url_for('admin.dashboard'))
@admin_bp.route('/report/csv')
@login_required
@role_required('admin')
def export_csv():
    output = io.StringIO()
    writer = csv.writer(output, quoting=csv.QUOTE_ALL)
    writer.writerow(['ПІБ студента', 'Email студента',
                    'Тема роботи', 'Керівник', 'Прогрес (%)',
                    'Дата захисту', 'Виконано', 'На перевірці',
                    'Всього'])
    for thesis in StudentThesis.query.all():
        execs = thesis.stage_executions.all()
        done = sum(1 for e in execs if e.status == 'done')
        pending = sum(1 for e in execs
                      if e.status == 'pending_review')
        writer.writerow([
            thesis.student.full_name, thesis.student.email,
            thesis.topic, thesis.supervisor.full_name,
            thesis.progress,

```

```

thesis.defense_date.strftime('%d.%m.%Y')
    if thesis.defense_date else '-',
done, pending, len(execs)
    ])
output.seek(0)
return Response('\uffeff' + output.getvalue(),
    mimetype='text/csv; charset=utf-8',
    headers={'Content-Disposition':
        'attachment;filename=thesis_report.csv'})
@admin_bp.route('/report/excel')
@login_required
@role_required('admin')
def export_excel():
    from openpyxl import Workbook
    from openpyxl.styles import Font, PatternFill, Alignment
    wb = Workbook()
    ws = wb.active
    ws.title = 'Звіт дипломних'
    hf = Font(bold=True, color='FFFFFF')
    hfill = PatternFill(start_color='1F497D',
        end_color='1F497D', fill_type='solid')
    center = Alignment(horizontal='center')
    headers = ['ПІБ студента', 'Email', 'Тема роботи',
        'Керівник', 'Прогрес (%)', 'Дата захисту',
        'Виконано', 'На перевірці', 'Всього']
    ws.append(headers)
    for i in range(1, len(headers) + 1):
        c = ws.cell(row=1, column=i)
        c.font = hf; c.fill = hfill; c.alignment = center
    for thesis in StudentThesis.query.all():
        execs = thesis.stage_executions.all()
        done = sum(1 for e in execs if e.status == 'done')
        pending = sum(1 for e in execs
            if e.status == 'pending_review')
        ws.append([thesis.student.full_name, thesis.student.email,
            thesis.topic, thesis.supervisor.full_name,
            thesis.progress,
            thesis.defense_date.strftime('%d.%m.%Y')
                if thesis.defense_date else '-',
            done, pending, len(execs)])
    for i, w in enumerate([28,30,55,28,12,14,12,15,10], 1):
        ws.column_dimensions[
            ws.cell(row=1, column=i).column_letter].width = w
    tmp = tempfile.NamedTemporaryFile(suffix='.xlsx', delete=False)
    wb.save(tmp.name); tmp.seek(0)
    return send_file(tmp.name, as_attachment=True,
        download_name='thesis_report.xlsx',
        mimetype='application/vnd.openxmlformats-
            'officedocument.spreadsheetml.sheet')

```

Лістинг файлу `app/routes/supervisor.py``app/routes/supervisor.py` – Blueprint керівника

```

from datetime import datetime
from flask import (Blueprint, render_template, redirect,
                  url_for, flash, request, jsonify)
from flask_login import login_required, current_user
from ..models import StudentThesis, StageExecution
from .. import db
from .auth import role_required

supervisor_bp = Blueprint('supervisor', __name__)
@supervisor_bp.route('/dashboard')
@login_required
@role_required('supervisor')
def dashboard():
    theses = StudentThesis.query.filter_by(
        supervisor_id=current_user.id).all()
    pending_count = sum(
        t.stage_executions.filter_by(status='pending_review').count()
        for t in theses
    )
    avg_progress = (
        round(sum(t.progress for t in theses) / len(theses))
        if theses else 0
    )
    return render_template('supervisor/dashboard.html',
        theses=theses, pending_count=pending_count,
        avg_progress=avg_progress)
@supervisor_bp.route('/student/<int:thesis_id>')
@login_required
@role_required('supervisor')
def student_detail(thesis_id):
    thesis = StudentThesis.query.get_or_404(thesis_id)
    if thesis.supervisor_id != current_user.id:
        flash('Доступ заборонено', 'danger')
        return redirect(url_for('supervisor.dashboard'))
    executions = thesis.stage_executions\
        .order_by(StageExecution.id).all()
    return render_template('supervisor/student_detail.html',
        thesis=thesis, executions=executions)
@supervisor_bp.route('/stage/<int:execution_id>/approve',
    methods=['POST'])
@login_required
@role_required('supervisor')
def approve_stage(execution_id):
    execution = StageExecution.query.get_or_404(execution_id)
    if execution.thesis.supervisor_id != current_user.id:

```

```

        return jsonify({'error': 'Forbidden'}), 403
    if execution.status != 'pending_review':
        return jsonify({'error': 'Невірний статус'}), 400
    execution.status = 'done'
    execution.approved_at = datetime.utcnow()
    execution.supervisor_comment = None
    db.session.commit()
    return jsonify({
        'status': 'done',
        'progress': execution.thesis.progress,
        'message': 'Етап підтверджено!'
    })
@supervisor_bp.route('/stage/<int:execution_id>/revision',
                    methods=['POST'])
@login_required
@role_required('supervisor')
def send_revision(execution_id):
    execution = StageExecution.query.get_or_404(execution_id)
    if execution.thesis.supervisor_id != current_user.id:
        return jsonify({'error': 'Forbidden'}), 403
    data = request.get_json() or {}
    comment = data.get('comment', '').strip()
    if not comment:
        return jsonify({'error': "Коментар обов'язковий"}), 400
    execution.status = 'revision'
    execution.supervisor_comment = comment
    db.session.commit()
    return jsonify({
        'status': 'revision',
        'message': 'Повернено на доопрацювання'
    })

```

Лістинг файлу app/routes/student.py

app/routes/student.py – Blueprint студента

```
import os
from datetime import datetime
from flask import (Blueprint, render_template, redirect,
                  url_for, flash, request, current_app)
from flask_login import login_required, current_user
from werkzeug.utils import secure_filename
from ..models import StudentThesis, StageExecution
from .. import db
from .auth import role_required

student_bp = Blueprint('student', __name__)
def allowed_file(filename):
    return ('.' in filename and
            filename.rsplit('.', 1)[1].lower()
            in current_app.config['ALLOWED_EXTENSIONS'])
@student_bp.route('/plan')
@login_required
@role_required('student')
def plan():
    thesis = StudentThesis.query.filter_by(
        student_id=current_user.id).first()
    if not thesis:
        flash('Вашу тему ще не призначено. '
              'Зверніться до адміністратора.', 'warning')
        return render_template('student/no_thesis.html')
    executions = thesis.stage_executions\
        .order_by(StageExecution.id).all()
    return render_template('student/plan.html',
                          thesis=thesis, executions=executions)
@student_bp.route('/stage/<int:execution_id>/submit',
                  methods=['POST'])

@login_required
@role_required('student')
def submit_stage(execution_id):
    execution = StageExecution.query.get_or_404(execution_id)
    if execution.thesis.student_id != current_user.id:
        flash('Доступ заборонено', 'danger')
        return redirect(url_for('student.plan'))
    if execution.status in ('done', 'pending_review'):
        flash('Цей етап вже відправлено або виконано', 'warning')
        return redirect(url_for('student.plan'))
    repo_url = request.form.get('repo_url', '').strip()
    if repo_url:
        execution.repo_url = repo_url
```

```
file = request.files.get('file')
if file and file.filename and allowed_file(file.filename):
    filename = secure_filename(file.filename)
    safe_name = (f"thesis_{execution.thesis_id}"
                 f"_stage_{execution_id}_{filename}")
    file.save(os.path.join(
        current_app.config['UPLOAD_FOLDER'], safe_name))
    execution.file_path = safe_name
if not repo_url and not (file and file.filename):
    flash('Прикріпіть репозиторій або файл', 'warning')
    return redirect(url_for('student.plan'))
execution.status = 'pending_review'
execution.submitted_at = datetime.utcnow()
execution.supervisor_comment = None
db.session.commit()
flash('Етап відправлено на перевірку керівнику!', 'success')
return redirect(url_for('student.plan'))
```

Лістинг файлу app/static/js/main.js

app/static/js/main.js – Fetch API з CSRF-токеном (виправлена версія)

```
// — Автоматичне зникнення flash-повідомлень —————
document.addEventListener('DOMContentLoaded', function () {
  document.querySelectorAll('.alert:not(.alert-permanent)')
    .forEach(function (alert) {
      setTimeout(function () {
        alert.style.transition = 'opacity 0.5s';
        alert.style.opacity = '0';
        setTimeout(function () { alert.remove(); }, 500);
      }, 5000);
    });
});

// — Зчитати CSRF-токен з мета-тегу —————
function getCsrftoken() {
  var meta = document.querySelector('meta[name="csrf-token"]');
  return meta ? meta.getAttribute('content') : '';
}

// — Підтвердити виконання етапу (AJAX) —————
function approveStage(executionId) {
  if (!confirm('Підтвердити виконання цього етапу?')) return;
  fetch('/supervisor/stage/' + executionId + '/approve', {
    method: 'POST',
    headers: {
      'Content-Type': 'application/json',
      'X-CSRFToken': getCsrftoken()
    }
  })
  .then(function (res) { return res.json(); })
  .then(function (data) {
    if (data.status === 'done') { location.reload(); }
    else { alert('Помилка: ' + (data.error || '?')); }
  })
  .catch(function () {
    alert('Помилка з\'єднання. Спробуйте ще раз.');
```

```
function sendRevision(executionId, comment) {
  fetch('/supervisor/stage/' + executionId + '/revision', {
    method: 'POST',
    headers: {
      'Content-Type': 'application/json',
      'X-CSRFToken': getCsrftoken()
    },
    body: JSON.stringify({ comment: comment })
  })
  .then(function (res) { return res.json(); })
  .then(function (data) {
    if (data.status === 'revision') { location.reload(); }
    else { alert('Помилка: ' + (data.error || '?')); }
  })
  .catch(function () {
    alert('Помилка з\'єднання. Спробуйте ще раз.');
```

Лістинги Dockerfile та docker-compose.yml

Dockerfile

```
FROM python:3.12-slim

RUN apt-get update && apt-get install -y \
    gcc libpq-dev \
    && rm -rf /var/lib/apt/lists/*

WORKDIR /app

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

COPY . .

ENV FLASK_APP=wsgi:app
ENV FLASK_ENV=production

EXPOSE 5000

CMD ["gunicorn", "--bind", "0.0.0.0:5000",
    "--workers", "4", "wsgi:app"]
```

docker-compose.yml

```
version: '3.9'

services:
  web:
    build: .
    container_name: thesis_web
    ports:
      - "5000:5000"
    env_file:
      - .env
    depends_on:
      db:
        condition: service_healthy
    volumes:
      - ./app:/app/app
      - uploads_data:/app/uploads
    restart: unless-stopped

  db:
    image: postgres:15-alpine
    container_name: thesis_db
    environment:
      POSTGRES_DB: ${POSTGRES_DB}
      POSTGRES_USER: ${POSTGRES_USER}
```

```
    POSTGRES_PASSWORD: ${POSTGRES_PASSWORD}
  volumes:
    - pgdata:/var/lib/postgresql/data
  healthcheck:
    test: ["CMD-SHELL",
           "pg_isready -U ${POSTGRES_USER}"]
    interval: 5s
    timeout: 5s
    retries: 10
  restart: unless-stopped

adminer:
  image: adminer
  container_name: thesis_adminer
  ports:
    - "8080:8080"
  depends_on: [db]

volumes:
  pgdata:
  uploads_data:
```

Лістинг файлу tests/test_basic.py

tests/test_basic.py – функціональні тести (pytest)

```
import pytest
from app import create_app, db
from app.models import (User, StageTemplate,
                        StudentThesis, StageExecution)

@pytest.fixture
def app():
    app = create_app('testing')
    with app.app_context():
        db.create_all()
        yield app
        db.session.remove()
        db.drop_all()

@pytest.fixture
def client(app):
    return app.test_client()

@pytest.fixture
def sample_data(app):
    with app.app_context():
        admin = User(email='admin@test.ua',
                     full_name='Адмін Тест', role='admin')
        admin.set_password('pass')
        supervisor = User(email='sv@test.ua',
                          full_name='Керівник Тест',
                          role='supervisor')
        supervisor.set_password('pass')
        student = User(email='st@test.ua',
                       full_name='Студент Тест', role='student')
        student.set_password('pass')
        t1 = StageTemplate(name='Аналіз', weight=40, order_num=1)
        t2 = StageTemplate(name='Реалізація', weight=60, order_num=2)
        db.session.add_all([admin, supervisor, student, t1, t2])
        db.session.flush()
        thesis = StudentThesis(
            student_id=student.id,
            supervisor_id=supervisor.id,
            topic='Тестова тема дипломної роботи'
        )
        db.session.add(thesis)
        db.session.flush()
        e1 = StageExecution(
            thesis_id=thesis.id, template_id=t1.id)
        e2 = StageExecution(
            thesis_id=thesis.id, template_id=t2.id)
        db.session.add_all([e1, e2])
        db.session.commit()
    return {
        'admin_id': admin.id,
        'supervisor_id': supervisor.id,
```

```

        'student_id': student.id,
        'thesis_id': thesis.id,
        'exec1_id': e1.id,
        'exec2_id': e2.id,
        't1_id': t1.id, 't2_id': t2.id,
    }

def test_login_success(client, sample_data):
    rv = client.post('/auth/login',
        data={'email': 'sv@test.ua', 'password': 'pass'},
        follow_redirects=False)
    assert rv.status_code == 302

def test_login_wrong_password(client, sample_data):
    rv = client.post('/auth/login',
        data={'email': 'sv@test.ua', 'password': 'WRONG'},
        follow_redirects=True)
    assert 'Невірна' in rv.data.decode('utf-8')

def test_progress_zero_initially(app, sample_data):
    with app.app_context():
        thesis = StudentThesis.query.get(sample_data['thesis_id'])
        assert thesis.progress == 0

def test_progress_after_one_done(app, sample_data):
    with app.app_context():
        e1 = StageExecution.query.get(sample_data['exec1_id'])
        e1.status = 'done'
        db.session.commit()
        thesis = StudentThesis.query.get(sample_data['thesis_id'])
        assert thesis.progress == 40

def test_progress_full(app, sample_data):
    with app.app_context():
        for eid in [sample_data['exec1_id'],
                    sample_data['exec2_id']]:
            e = StageExecution.query.get(eid)
            e.status = 'done'
            db.session.commit()
        thesis = StudentThesis.query.get(sample_data['thesis_id'])
        assert thesis.progress == 100

def test_student_blocked_from_admin(client, sample_data):
    client.post('/auth/login',
        data={'email': 'st@test.ua', 'password': 'pass'})
    rv = client.get('/admin/dashboard', follow_redirects=True)
    assert 'Доступ заборонено' in rv.data.decode('utf-8')

def test_stage_weights_sum_to_100(app, sample_data):
    with app.app_context():
        templates = StageTemplate.query.filter_by(
            is_active=True).all()
        assert sum(t.weight for t in templates) == 100

```

```
def test_status_labels(app, sample_data):
    with app.app_context():
        e = StageExecution.query.get(sample_data['exec1_id'])
        assert e.status_label == 'Не розпочато'
        assert e.status_badge_class == 'secondary'
        e.status = 'done'
        db.session.commit()
        e = StageExecution.query.get(sample_data['exec1_id'])
        assert e.status_label == 'Виконано'
        assert e.status_badge_class == 'success'

def test_password_is_hashed(app, sample_data):
    with app.app_context():
        user = User.query.get(sample_data['student_id'])
        assert user.password_hash != 'pass'
        assert user.check_password('pass') is True
        assert user.check_password('wrong') is False

def test_role_label(app, sample_data):
    with app.app_context():
        admin = User.query.get(sample_data['admin_id'])
        sv = User.query.get(sample_data['supervisor_id'])
        st = User.query.get(sample_data['student_id'])
        assert admin.role_label == 'Адміністратор'
        assert sv.role_label == 'Керівник'
        assert st.role_label == 'Студент'
```

Лістинг файлу app/templates/base.html

app/templates/base.html – Базовий шаблон (sidebar, flash-повідомлення, block content)

Лістинг Й.1 – Базовий HTML-шаблон (app/templates/base.html)

```
<!DOCTYPE html>
<html lang="uk">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1">
  <meta name="csrf-token" content="{{ csrf_token() }}">
  <title>{% block title %}Моніторинг дипломних{% endblock %}</title>
  <link
href="https://cdn.jsdelivrivr.net/npm/bootstrap@5.3.3/dist/css/bootstrap.
min.css" rel="stylesheet">
  <link href="https://cdn.jsdelivrivr.net/npm/bootstrap-
icons@1.11.3/font/bootstrap-icons.css" rel="stylesheet">
  <link rel="stylesheet" href="{{ url_for('static',
filename='css/style.css') }}">
</head>
<body>
{% if current_user.is_authenticated %}
<div class="d-flex" id="wrapper">
  <div class="sidebar bg-dark text-white flex-shrink-0">
    <div class="p-3 border-bottom border-secondary">
      <div class="d-flex align-items-center gap-2">
        <i class="bi bi-mortarboard-fill fs-5 text-primary"></i>
        <div>
          <div class="fw-bold small text-white">КП та М</div>
          <div class="text-muted" style="font-size:11px;">
            {% if current_user.role == 'admin' %}Адміністратор
            {% elif current_user.role == 'supervisor' %}Керівник
            {% else %}Студент{% endif %}
          </div>
        </div>
      </div>
    </div>
    <div>
      <nav class="sidebar-nav py-2 flex-grow-1">
        {% if current_user.role == 'admin' %}
          <a href="{{ url_for('admin.dashboard') }}" class="nav-link {%
if request.endpoint == 'admin.dashboard' %}active{% endif %}">
            <i class="bi bi-speedometer2 me-2"></i>Дашборд
          </a>
          <a href="{{ url_for('admin.users') }}" class="nav-link {% if
'users' in (request.endpoint or '') %}active{% endif %}">
            <i class="bi bi-people me-2"></i>Користувачі
          </a>
          <a href="{{ url_for('admin.stages') }}" class="nav-link {% if
'stages' in (request.endpoint or '') %}active{% endif %}">
```

```

        <i class="bi bi-list-check me-2"></i>Шаблони етапів
    </a>
    <a href="{{ url_for('admin.assign') }}" class="nav-link {% if
'assign' in (request.endpoint or '') %}active{% endif %}">
        <i class="bi bi-person-plus me-2"></i>Призначення
    </a>
    <div class="border-top border-secondary my-2 mx-3"></div>
    <a href="{{ url_for('admin.export_csv') }}" class="nav-link">
        <i class="bi bi-file-earmark-csv me-2"></i>Звіт CSV
    </a>
    <a href="{{ url_for('admin.export_excel') }}" class="nav-
link">
        <i class="bi bi-file-earmark-excel me-2"></i>Звіт Excel
    </a>
    {% elif current_user.role == 'supervisor' %}
    <a href="{{ url_for('supervisor.dashboard') }}" class="nav-
link">
        <i class="bi bi-speedometer2 me-2"></i>Дашборд
    </a>
    {% else %}
    <a href="{{ url_for('student.plan') }}" class="nav-link">
        <i class="bi bi-journal-check me-2"></i>Мій план
    </a>
    {% endif %}
</nav>
<div class="sidebar-footer p-3 border-top border-secondary">
    <div class="d-flex align-items-center gap-2 mb-2">
        <div class="avatar-circle bg-primary text-white">
            {{ current_user.full_name[:2].upper() }}
        </div>
        <div style="min-width:0;">
            <div class="small fw-500 text-white text-
truncate">{{ current_user.full_name }}</div>
            <div class="text-muted text-truncate" style="font-
size:11px;">{{ current_user.email }}</div>
        </div>
    </div>
    <a href="{{ url_for('auth.logout') }}" class="btn btn-outline-
secondary btn-sm w-100">
        <i class="bi bi-box-arrow-right me-1"></i>Вийти
    </a>
</div>
</div>
<div class="main-content flex-grow-1">
    <div class="p-3 pb-0">
        {% with messages = get_flashed_messages(with_categories=true) %}
        {% if messages %}
            {% for category, message in messages %}
                <div class="alert alert-{{ category }} alert-dismissible
fade show" role="alert">
                    {{ message }}

```

```

        <button type="button" class="btn-close" data-bs-
dismiss="alert"></button>
    </div>
    {% endfor %}
{% endif %}
{% endwith %}
</div>
<div class="p-4">
    {% block content %}{% endblock %}
</div>
</div>

{% else %}

<div class="p-3 pb-0">
    {% with messages = get_flashed_messages(with_categories=true) %}
    {% if messages %}
        {% for category, message in messages %}
            <div class="alert alert-{{ category }} alert-dismissible fade
show" role="alert">
                {{ message }}
                <button type="button" class="btn-close" data-bs-
dismiss="alert"></button>
            </div>
        {% endfor %}
    {% endif %}
    {% endwith %}
</div>

{{ self.content() }}

{% endif %}

<script
src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.3/dist/js/bootstrap.bu
ndle.min.js"></script>
<script src="{{ url_for('static', filename='js/main.js') }}"></script>
</body>
</html>

```

Лістинги HTML-шаблонів авторизації та студента

app/templates/auth/login.html — Сторінка входу в систему

Лістинг К.1 – Шаблон сторінки авторизації (app/templates/auth/login.html)

```

{% extends "base.html" %}
{% block title %}Вхід в систему{% endblock %}

{% block content %}
<div class="login-wrapper">
  <div class="login-card">
    <div class="text-center mb-4">
      <div class="bg-primary bg-opacity-10 rounded-circle d-inline-
flex align-items-center justify-content-center mb-3"
        style="width:64px;height:64px;">
        <i class="bi bi-mortarboard-fill fs-2 text-primary"></i>
      </div>
      <h4 class="fw-bold mb-1">Моніторинг Кваліфікаційних робіт</h4>
      <p class="text-muted small mb-0">Кафедра КП та М</p>
    </div>

    <form method="POST" action="{{ url_for('auth.login') }}">
      <input type="hidden" name="csrf_token"
value="{{ csrf_token() }}">

      <div class="mb-3">
        <label class="form-label fw-500">Email</label>
        <div class="input-group">
          <span class="input-group-text bg-light border-end-0">
            <i class="bi bi-envelope text-muted"></i>
          </span>
          <input type="email" name="email" class="form-control border-
start-0 ps-0"
            placeholder="your@email.com" required autofocus>
        </div>
      </div>

      <div class="mb-3">
        <label class="form-label fw-500">Пароль</label>
        <div class="input-group">
          <span class="input-group-text bg-light border-end-0">
            <i class="bi bi-lock text-muted"></i>
          </span>
          <input type="password" name="password" class="form-control
border-start-0 ps-0"
            placeholder="....." required>
        </div>
      </div>

      <div class="mb-4 form-check">

```

Продовження додатку П

```


```

app/templates/student/plan.html – Кабінет студента (план виконання)

Лістинг К.2 – Шаблон кабінету студента (app/templates/student/plan.html)

[illegible]

```

<div class="card mb-4">
  <div class="card-body">
    <div class="d-flex justify-content-between align-items-center mb-2">
      <span class="fw-500 fs-5">Загальний прогрес</span>
      <span class="fw-bold text-primary" style="font-size:2rem;">{{ thesis.progress }}%</span>
    </div>
    <div class="progress" style="height:14px;">
      <div class="progress-bar progress-bar-striped progress-bar-animated bg-success" style="width:{{ thesis.progress }}%" role="progressbar" aria-valuenow="{{ thesis.progress }}" aria-valuemin="0" aria-valuemax="100">
      </div>
    </div>
    <div class="d-flex justify-content-between mt-1">
      <small class="text-muted">0%</small>
      <small class="text-muted">100%</small>
    </div>
  </div>
</div>
<!-- Список етапів -->
<h5 class="fw-500 mb-3">Етапи виконання</h5>

<div class="d-flex flex-column gap-3">
  {% for execution in executions %}
  {% if execution.template.is_active %}
  <div class="card {% if execution.status == 'pending_review' %}border-warning {% elif execution.status == 'done' %}border-success {% elif execution.status == 'revision' %}border-danger{% endif %}">
    <div class="card-body">

      <!-- Заголовок -->
      <div class="d-flex justify-content-between align-items-start mb-2">
        <div>
          <h6 class="fw-bold mb-1">
            <span class="badge bg-{{ 'success' if execution.status == 'done' else 'secondary' }} me-2">
              {{ loop.index }}
            </span>
            {{ execution.template.name }}
          </h6>
          <small class="text-muted">
            <i class="bi bi-percent me-1"></i>Варі:
          <strong>{{ execution.template.weight }}%</strong>
          </small>
        </div>
        <span class="badge bg-{{ execution.status_badge_class }}">

```

```

        {% if execution.status == 'done' %}<i class="bi bi-check-
circle me-1"></i>{% endif %}
        {{ execution.status_label }}
    </span>
</div>

    <!-- Коментар керівника при поверненні -->
    {% if execution.status == 'revision' and
execution.supervisor_comment %}
    <div class="alert alert-danger py-2 mb-2">
        <small>
            <i class="bi bi-chat-text me-1"></i>
            <strong>Коментар керівника:</strong>
            {{ execution.supervisor_comment }}
        </small>
    </div>
    {% endif %}

    <!-- Вже прикріплені матеріали -->
    {% if execution.repo_url or execution.file_path %}
    <div class="mb-2 d-flex flex-wrap gap-2">
        {% if execution.repo_url %}
        <a href="{{ execution.repo_url }}" target="_blank" class="btn
btn-sm btn-outline-dark">
            <i class="bi bi-github me-1"></i>Мій репозиторій
        </a>
        {% endif %}
        {% if execution.file_path %}
        <span class="btn btn-sm btn-outline-secondary"
style="cursor:default;">
            <i class="bi bi-file-earmark-check me-
1"></i>{{ execution.file_path }}
        </span>
        {% endif %}
    </div>
    {% endif %}

    <!-- Форма відправки (тільки якщо статус дозволяє) -->
    {% if execution.status in ('not_started', 'in_progress',
'revision') %}
    <form method="POST"
        action="{{ url_for('student.submit_stage',
execution_id=execution.id) }}"
        enctype="multipart/form-data"
        class="border-top pt-3 mt-2">
        <input type="hidden" name="csrf_token"
value="{{ csrf_token() }}">

        <div class="row g-2 align-items-end">
            <div class="col-md-5">
                <label class="form-label small text-muted mb-1">
                    <i class="bi bi-github me-1"></i>GitHub репозиторій

```

```

        </label>
        <input type="url" name="repo_url" class="form-control
form-control-sm"
            placeholder="https://github.com/username/repo"
            value="{{ execution.repo_url or '' }}">
    </div>
    <div class="col-md-1 text-center text-muted small py-
2">або</div>
    <div class="col-md-4">
        <label class="form-label small text-muted mb-1">
            <i class="bi bi-file-earmark-arrow-up me-1"></i>Файл
(.pdf, .doc, .zip)
        </label>
        <input type="file" name="file" class="form-control form-
control-sm"
            accept=".pdf,.doc,.docx,.zip,.png,.jpg">
    </div>
    <div class="col-md-2">
        <button type="submit" class="btn btn-primary btn-sm w-
100">
            <i class="bi bi-send me-1"></i>Відправити
        </button>
    </div>
</div>
</form>
{% elif execution.status == 'done' %}
<div class="border-top pt-2 mt-2">
    <small class="text-success fw-500">
        <i class="bi bi-check-circle-fill me-1"></i>
        Підтверджено керівником
        {% if execution.approved_at %}-
        {{ execution.approved_at.strftime('%d.%m.%Y') }}{% endif %}
    </small>
    </div>
    {% elif execution.status == 'pending_review' %}
    <div class="border-top pt-2 mt-2">
        <small class="text-warning fw-500">
            <i class="bi bi-hourglass-split me-1"></i>
            Очікує перевірки керівника
        </small>
    </div>
    {% endif %}
</div>
</div>
{% endif %}
{% endfor %}
</div>
{% endblock %}

```

Лістинг К.3 – Шаблон сторінки відсутньої теми (app/templates/student/no_thesis.html)

```
{% extends "base.html" %}
{% block title %}Тему не призначено{% endblock %}

{% block content %}
<div class="text-center py-5">
  <div class="mb-4">
    <i class="bi bi-journal-x display-1 text-muted"></i>
  </div>
  <h4 class="fw-bold mb-2">Тему дипломної роботи ще не призначено</h4>
  <p class="text-muted mb-4">
    Зверніться до адміністратора або вашого наукового керівника.<br>
    Після призначення теми ваш план з'явиться тут автоматично.
  </p>
  <div class="alert alert-info d-inline-flex align-items-center gap-2
px-4">
    <i class="bi bi-info-circle fs-5"></i>
    <span>Увійдіть пізніше або оновіть сторінку</span>
  </div>
</div>
{% endblock %}
```

Лістинги HTML-шаблонів керівника

app/templates/supervisor/dashboard.html – Дашборд керівника

Лістинг Л.1 – Дашборд керівника (app/templates/supervisor/dashboard.html)

```

{% extends "base.html" %}
{% block title %}Дашборд керівника{% endblock %}

{% block content %}
<h4 class="fw-bold mb-1">Дашборд керівника</h4>
<p class="text-muted mb-4">Вітаємо, {{ current_user.full_name }}</p>

<div class="row g-3 mb-4">
  <div class="col-md-4">
    <div class="stat-card">
      <div class="stat-label"><i class="bi bi-people me-1"></i>Мої
студенти</div>
      <div class="stat-value">{{ theses|length }}</div>
    </div>
  </div>
  <div class="col-md-4">
    <div class="stat-card">
      <div class="stat-label"><i class="bi bi-clock me-1"></i>Очікують
перевірки</div>
      <div class="stat-value text-warning">{{ pending_count }}</div>
    </div>
  </div>
  <div class="col-md-4">
    <div class="stat-card">
      <div class="stat-label"><i class="bi bi-graph-up me-
1"></i>Середній прогрес</div>
      <div class="stat-value text-success">{{ avg_progress }}%</div>
    </div>
  </div>
</div>

<div class="card">
  <div class="card-header">
    <i class="bi bi-list-ul me-2"></i>Мої студенти
  </div>
  <div class="card-body p-0">
    {% if theses %}
    <div class="table-responsive">
      <table class="table table-hover mb-0 align-middle">
        <thead class="table-light">
          <tr>
            <th>Студент</th>
            <th>Тема роботи</th>
            <th style="width:180px;">Прогрес</th>
            <th style="width:150px;">Статус</th>
          </tr>
        </thead>
        <tbody>
          <tr>
            <td>{{ theses.0.student }}</td>
            <td>{{ theses.0.topic }}</td>
            <td>{{ theses.0.progress }}%</td>
            <td>{{ theses.0.status }}</td>
          </tr>
        </tbody>
      </table>
    </div>
    </div>
  </div>
</div>

```

```

        <th style="width:120px;">Дата захисту</th>
        <th style="width:100px;"></th>
    </tr>
</thead>
<tbody>
    {% for thesis in theses %}
    {% set pending =
thesis.stage_executions.filter_by(status='pending_review').count() %}
    <tr>
        <td>
            <div class="d-flex align-items-center gap-2">
                <div class="avatar-circle bg-primary bg-opacity-10
text-primary" style="font-size:12px;">
                    {{ thesis.student.full_name[:2].upper() }}
                </div>
                <div>
                    <div class="fw-500
small">{{ thesis.student.full_name }}</div>
                    <div class="text-muted" style="font-
size:11px;">{{ thesis.student.email }}</div>
                </div>
            </td>
            <td><small class="text-muted">{{ thesis.topic[:55] }}{% if
thesis.topic|length > 55 %}...{% endif %}</small></td>
            <td>
                <div class="progress mb-1" style="height:6px;">
                    <div class="progress-bar bg-success"
style="width:{{ thesis.progress }}%"></div>
                </div>
                <small class="text-muted">{{ thesis.progress }}%</small>
            </td>
            <td>
                {% if pending > 0 %}
                <span class="badge bg-warning text-dark">
                    <i class="bi bi-clock me-1"></i>{{ pending }} на
перевірці
                </span>
                {% else %}
                <span class="badge bg-success bg-opacity-10 text-
success border border-success">
                    <i class="bi bi-check-circle me-1"></i>Все ОК
                </span>
                {% endif %}
            </td>
            <td><small>{{ thesis.defense_date.strftime('%d.%m.%Y') if
thesis.defense_date else '-' }}</small></td>
            <td>
                <a href="{{ url_for('supervisor.student_detail',
thesis_id=thesis.id) }}"
                    class="btn btn-sm btn-outline-primary">
                    <i class="bi bi-eye me-1"></i>Деталі

```

```

        </a>
      </td>
    </tr>
  {% endfor %}
</tbody>
</table>
</div>
{% else %}
<div class="text-center text-muted py-5">
  <i class="bi bi-people display-4 d-block mb-3"></i>
  До вас ще не прив'язані студенти.<br>
  <small>Зверніться до адміністратора.</small>
</div>
{% endif %}
</div>
</div>
{% endblock %}

```

app/templates/supervisor/student_detail.html – Деталі студента з кнопками підтвердження

Лістинг Л.2 – Деталі студента (app/templates/supervisor/student_detail.html)

```

{% extends "base.html" %}
{% block title %}{{ thesis.student.full_name }} – Деталі{% endblock %}

{% block content %}
<!-- Шапка -->
<div class="mb-4">
  <a href="{% url_for('supervisor.dashboard') %}" class="btn btn-sm
  btn-outline-secondary mb-3">
    <i class="bi bi-arrow-left me-1"></i>Назад до дашборду
  </a>
  <div class="d-flex align-items-start gap-3">
    <div class="avatar-circle bg-primary bg-opacity-10 text-primary"
      style="width:52px;height:52px;font-size:18px;flex-shrink:0;">
      {{ thesis.student.full_name[:2].upper() }}
    </div>
    <div>
      <h4 class="fw-bold mb-1">{{ thesis.student.full_name }}</h4>
      <p class="text-muted mb-1"><i class="bi bi-envelope me-1"></i>{{ thesis.student.email }}</p>
      <p class="mb-0"><i class="bi bi-journal-text me-1 text-muted"></i>{{ thesis.topic }}</p>
    </div>
  </div>
</div>
<!-- Профес бар -->
<div class="card mb-4">
  <div class="card-body">

```

```
<div class="d-flex justify-content-between align-items-center mb-2">  
    <span class="fw-500">Загальний прогрес</span>  
    <span class="fs-3 fw-bold text-primary"  
id="progressValue">{{ thesis.progress }}%</span>  
</div>  
<div class="progress" style="height:12px;">  
    <div class="progress-bar bg-success progress-bar-striped"  
        id="progressBar" style="width:{{ thesis.progress }}%"  
        role="progressbar"></div>  
</div>  
{% if thesis.defense_date %}  
<small class="text-muted mt-1 d-block">  
    <i class="bi bi-calendar me-1"></i>Дата захисту:  
{{ thesis.defense_date.strftime('%d.%m.%Y') }}  
</small>  
{% endif %}  
</div>  
</div>  
<!-- Етапи -->  
<h5 class="fw-500 mb-3">Етапи виконання</h5>  
<div class="d-flex flex-column gap-3">  
    {% for execution in executions %}  
        {% if execution.template.is_active %}  
            <div class="card border-{{ execution.status_badge_class }}"  
id="stage-card-{{ execution.id }}">  
                <div class="card-body">  
                    <!-- Заголовок етапу -->  
                    <div class="d-flex justify-content-between align-items-start mb-3">  
  
                        <div>  
                            <h6 class="fw-bold mb-1">  
                                <span class="badge bg-secondary me-2">{{ loop.index }}</span>  
                                {{ execution.template.name }}  
                            </h6>  
                            <small class="text-muted">  
                                <i class="bi bi-percent me-1"></i>Вара:  
                                {{ execution.template.weight }}%  
                                {% if execution.submitted_at %}  
                                    &nbsp;·&nbsp;&nbsp;   
                                <i class="bi bi-clock me-1"></i>Відправлено:  
                                {{ execution.submitted_at.strftime('%d.%m.%Y %H:%M') }}  
                                {% endif %}  
                            </small>  
                        </div>  
                        <span class="badge bg-{{ execution.status_badge_class }}"  
id="status-{{ execution.id }}">  
                            {{ execution.status_label }}  
                        </span>  
                    </div>  
                    <!-- Матеріали студента -->
```

```

    {% if execution.repo_url or execution.file_path %}
    <div class="mb-3 d-flex gap-2 flex-wrap">
      {% if execution.repo_url %}
      <a href="{{ execution.repo_url }}" target="_blank" class="btn
btn-sm btn-outline-dark">
        <i class="bi bi-github me-1"></i>Відкрити репозиторій
      </a>
      {% endif %}
      {% if execution.file_path %}
      <span class="btn btn-sm btn-outline-secondary">
        <i class="bi bi-file-earmark me-
1"></i>{{ execution.file_path }}
      </span>
      {% endif %}
    </div>
    {% endif %}
    <!-- Попередній коментар -->
    {% if execution.supervisor_comment %}
    <div class="alert alert-info py-2 mb-3">
      <small><i class="bi bi-chat-text me-1"></i><strong>Ваш
коментар:</strong> {{ execution.supervisor_comment }}</small>
    </div>
    {% endif %}
    <!-- Кнопки дій (тільки якщо pending_review) -->
    {% if execution.status == 'pending_review' %}
    <div class="border-top pt-3 d-flex gap-2">
      <button class="btn btn-success btn-sm"
        onclick="approveStage({{ execution.id }})">
        <i class="bi bi-check-lg me-1"></i>Підтвердити виконання
      </button>
      <button class="btn btn-outline-danger btn-sm"
        onclick="revisionStage({{ execution.id }})">
        <i class="bi bi-arrow-return-left me-1"></i>Повернути на
доопрацювання
      </button>
    </div>
    {% endif %}
  </div>
</div>
{% endif %}
{% endfor %}
</div>
<!-- == Модальне вікно коментаря == -->
<div class="modal fade" id="revisionModal" tabindex="-1">
  <div class="modal-dialog">
    <div class="modal-content">
      <div class="modal-header">
        <h5 class="modal-title">
          <i class="bi bi-arrow-return-left me-2 text-
danger"></i>Повернути на доопрацювання
        </h5>

```

```

        <button type="button" class="btn-close" data-bs-
dismiss="modal"></button>
    </div>
    <div class="modal-body">
        <input type="hidden" id="revisionExecutionId">
        <label class="form-label fw-500">
            Коментар для студента <span class="text-danger">*</span>
        </label>
        <textarea id="revisionComment" class="form-control" rows="4"
placeholder="Вкажіть що саме потрібно виправити або
доопрацювати..."></textarea>
        <div id="revisionError" class="text-danger small mt-1"
style="display:none;">
            <i class="bi bi-exclamation-circle me-1"></i>Коментар
обов'язковий
        </div>
    </div>
    <div class="modal-footer">
        <button type="button" class="btn btn-secondary" data-bs-
dismiss="modal">Скасувати</button>
        <button type="button" class="btn btn-danger"
id="confirmRevision">
            <i class="bi bi-send me-1"></i>Надіслати
        </button>
    </div>
</div>
</div>
{% endblock %}

```

Лістинги HTML-шаблонів адміністратора

app/templates/admin/dashboard.html – Адмін-панель дашборд

Лістинг М.1 – Адмін-панель (app/templates/admin/dashboard.html)

```
{% extends "base.html" %}
{% block title %}Адмін-панель{% endblock %}

{% block content %}
<div class="d-flex justify-content-between align-items-center mb-4">
  <div>
    <h4 class="fw-bold mb-1">Адмін-панель</h4>
    <p class="text-muted mb-0">Загальна статистика системи</p>
  </div>
  <div class="d-flex gap-2">
    <a href="{% url_for('admin.export_csv') %}" class="btn btn-outline-secondary btn-sm">
      <i class="bi bi-file-earmark-csv me-1"></i>CSV
    </a>
    <a href="{% url_for('admin.export_excel') %}" class="btn btn-outline-success btn-sm">
      <i class="bi bi-file-earmark-excel me-1"></i>Excel
    </a>
  </div>
</div>

<div class="row g-3 mb-4">
  <div class="col-md-4">
    <div class="stat-card">
      <div class="stat-label"><i class="bi bi-people me-1"></i>Всього користувачів</div>
      <div class="stat-value">{{ users_count }}</div>
    </div>
  </div>
  <div class="col-md-4">
    <div class="stat-card">
      <div class="stat-label"><i class="bi bi-person-badge me-1"></i>Студентів</div>
      <div class="stat-value text-success">{{ students_count }}</div>
    </div>
  </div>
  <div class="col-md-4">
    <div class="stat-card">
      <div class="stat-label"><i class="bi bi-person-workspace me-1"></i>Керівників</div>
      <div class="stat-value text-primary">{{ supervisors_count }}</div>
    </div>
  </div>
</div>
```

```

<div class="card">
  <div class="card-header d-flex justify-content-between align-items-
center">
    <span><i class="bi bi-journal-text me-2"></i>Дипломні роботи
    ({{ theses|length }})</span>
    <a href="{{ url_for('admin.assign') }}" class="btn btn-primary
btn-sm">
      <i class="bi bi-plus-lg me-1"></i>Призначити тему
    </a>
  </div>
  <div class="card-body p-0">
    {% if theses %}
    <div class="table-responsive">
      <table class="table table-hover mb-0 align-middle">
        <thead class="table-light">
          <tr>
            <th>Студент</th>
            <th>Тема роботи</th>
            <th>Керівник</th>
            <th style="width:160px;">Прогрес</th>
            <th>Дата захисту</th>
            <th style="width:90px;">Дії</th>
          </tr>
        </thead>
        <tbody>
          {% for thesis in theses %}
          <tr>
            <td>
              <div class="d-flex align-items-center gap-2">
                <div class="avatar-circle bg-success bg-opacity-10
text-success" style="font-size:12px;">
                  {{ thesis.student.full_name[:2].upper() }}
                </div>
                <span class="small fw-
500">{{ thesis.student.full_name }}</span>
              </div>
            </td>
            <td><span class="small text-
muted">{{ thesis.topic[:60] }}{% if thesis.topic|length > 60 %}...{%
endif %}</span></td>
            <td><span
class="small">{{ thesis.supervisor.full_name }}</span></td>
            <td>
              <div class="progress mb-1" style="height:6px;">
                <div class="progress-bar bg-success"
style="width:{{ thesis.progress }}%"></div>
              </div>
              <small class="text-muted">{{ thesis.progress }}%</small>
            </td>
            <td><small>{{ thesis.defense_date.strftime('%d.%m.%Y') if
thesis.defense_date else '-' }}</small></td>
          </tr>
        </tbody>
      </table>
    </div>
  </div>
</div>

```

```

        <a href="{{ url_for('admin.edit_thesis',
thesis_id=thesis.id) }}"
            class="btn btn-outline-secondary btn-sm"
            title="Редагувати тему">
            <i class="bi bi-pencil"></i>
        </a>
    </td>
</tr>
{% endfor %}
</tbody>
</table>
</div>
{% else %}
<div class="text-center text-muted py-5">
    <i class="bi bi-journal-x display-4 d-block mb-2"></i>
    Ще немає дипломних робіт.
    <a href="{{ url_for('admin.assign') }}">Призначити першу</a>
</div>
{% endif %}
</div>
</div>
{% endblock %}

```

app/templates/admin/users.html – Список користувачів

Лістинг М.2 – Список користувачів (app/templates/admin/users.html)

```

{% extends "base.html" %}
{% block title %}Користувачі{% endblock %}

{% block content %}
<div class="d-flex justify-content-between align-items-center mb-4">
    <div>
        <h4 class="fw-bold mb-1">Користувачі системи</h4>
        <p class="text-muted mb-0">Всього: {{ users|length }}</p>
    </div>
    <a href="{{ url_for('admin.add_user') }}" class="btn btn-primary">
        <i class="bi bi-person-plus me-1"></i>Додати користувача
    </a>
</div>

<div class="card">
    <div class="card-body p-0">
        <div class="table-responsive">
            <table class="table table-hover mb-0 align-middle">
                <thead class="table-light">
                    <tr>

<th>ID</th><th>ПІБ</th><th>Email</th><th>Роль</th><th>Статус</th><th>Д
ата реєстрації</th><th>Дії</th>

```

```

    </tr>
</thead>
<tbody>
    {% for user in users %}
    <tr>
        <td><small class="text-muted">{{ user.id }}</small></td>
        <td>
            <div class="d-flex align-items-center gap-2">
                <div class="avatar-circle
                    {% if user.role == 'admin' %}bg-danger bg-opacity-10
text-danger
                    {% elif user.role == 'supervisor' %}bg-primary bg-
opacity-10 text-primary
                    {% else %}bg-success bg-opacity-10 text-success{%
endif %}"
                    style="font-size:12px;width:32px;height:32px;">
                    {{ user.full_name[:2].upper() }}
                </div>
                <span class="fw-500 small">{{ user.full_name }}</span>
            </div>
        </td>
        <td><small class="text-
muted">{{ user.email }}</small></td>
        <td>
            {% if user.role == 'admin' %}
                <span class="badge bg-danger">Адміністратор</span>
            {% elif user.role == 'supervisor' %}
                <span class="badge bg-primary">Керівник</span>
            {% else %}
                <span class="badge bg-success">Студент</span>
            {% endif %}
        </td>
        <td>
            {% if user.is_active %}
                <span class="badge bg-success bg-opacity-10 text-
success border border-success">
                    <i class="bi bi-check-circle me-1"></i>Активний
                </span>
            {% else %}
                <span class="badge bg-secondary bg-opacity-10 text-
secondary border border-secondary">
                    <i class="bi bi-slash-circle me-1"></i>Заблокований
                </span>
            {% endif %}
        </td>
        <td><small class="text-
muted">{{ user.created_at.strftime('%d.%m.%Y') if user.created_at else
'-' }}</small></td>
        <td>
            <form method="POST"
action="{{ url_for('admin.toggle_user', user_id=user.id) }}"
style="display:inline">

```

```

        <input type="hidden" name="csrf_token"
value="{{ csrf_token() }}">
        <button type="submit" class="btn btn-sm btn-outline-
secondary"
            title="{{ 'Деактивувати' if user.is_active
else 'Активувати' }}"
            onclick="return confirm('Змінити статус
користувача {{ user.full_name }}?')">
            <i class="bi bi-toggle-{{ 'on text-success' if
user.is_active else 'off' }}"></i>
        </button>
    </form>
</td>
</tr>
{% endfor %}
</tbody>
</table>
</div>
</div>
</div>
{% endblock %}

```

app/templates/admin/add_user.html – Форма додавання користувача

Лістинг М.3 – Форма додавання користувача (app/templates/admin/add_user.html)

```

{% extends "base.html" %}
{% block title %}Додати користувача{% endblock %}

{% block content %}
<div class="row justify-content-center">
    <div class="col-md-6 col-lg-5">
        <div class="d-flex align-items-center gap-2 mb-4">
            <a href="{{ url_for('admin.users') }}" class="btn btn-sm btn-
outline-secondary">
                <i class="bi bi-arrow-left"></i>
            </a>
            <h4 class="fw-bold mb-0">Новий користувач</h4>
        </div>

        <div class="card">
            <div class="card-body p-4">
                <form method="POST" action="{{ url_for('admin.add_user') }}">
                    <input type="hidden" name="csrf_token"
value="{{ csrf_token() }}">

                    <div class="mb-3">
                        <label class="form-label fw-500">ПІБ <span class="text-
danger">*</span></label>
                        <input type="text" name="full_name" class="form-control"

```

```

        placeholder="Іванов Іван Іванович" required>
    </div>
    <div class="mb-3">
        <label class="form-label fw-500">Email <span class="text-
danger">*</span></label>
        <input type="email" name="email" class="form-control"
placeholder="user@university.ua" required>
    </div>
    <div class="mb-3">
        <label class="form-label fw-500">Роль <span class="text-
danger">*</span></label>
        <select name="role" class="form-select" required>
            <option value="student">Студент</option>
            <option value="supervisor">Керівник</option>
            <option value="admin">Адміністратор</option>
        </select>
    </div>
    <div class="mb-4">
        <label class="form-label fw-500">Пароль <span class="text-
danger">*</span></label>
        <input type="password" name="password" class="form-
control"
placeholder="Мінімум 6 символів" required
minlength="6">
    </div>

    <button type="submit" class="btn btn-primary w-100 py-2">
        <i class="bi bi-person-check me-2"></i>Створити
користувача
    </button>
</form>
</div>
</div>
{% endblock %}

```

app/templates/admin/assign.html – Форма призначення теми

Лістинг М.4 – Форма призначення теми (app/templates/admin/assign.html)

```
{% extends "base.html" %}
{% block title %}Призначення теми{% endblock %}

{% block content %}
<div class="row justify-content-center">
  <div class="col-md-8 col-lg-7">
    <div class="d-flex align-items-center gap-2 mb-4">
      <a href="{% url_for('admin.dashboard') %}" class="btn btn-sm
btn-outline-secondary">
        <i class="bi bi-arrow-left"></i>
      </a>
    </div>
  </div>
</div>
{% endblock %}
```

```

<h4 class="fw-bold mb-0">Призначити тему студенту</h4>
</div>
{% if not students %}
<div class="alert alert-warning">
  <i class="bi bi-info-circle me-2"></i>
  Немає студентів без призначеної теми. Спочатку <a
href="{{ url_for('admin.add_user') }}">додайте студента</a>.
</div>
{% else %}
<div class="card">
  <div class="card-body p-4">
    <form method="POST" action="{{ url_for('admin.assign') }}">
      <input type="hidden" name="csrf_token"
value="{{ csrf_token() }}">
      <div class="mb-3">
        <label class="form-label fw-500">Студент <span
class="text-danger">*</span></label>
        <select name="student_id" class="form-select" required>
          <option value="">— Оберіть студента —</option>
          {% for student in students %}
            <option value="{{ student.id }}">{{ student.full_name }}
({{ student.email }})</option>
          {% endfor %}
        </select>
      </div>
      <div class="mb-3">
        <label class="form-label fw-500">Керівник <span
class="text-danger">*</span></label>
        <select name="supervisor_id" class="form-select" required>
          <option value="">— Оберіть керівника —</option>
          {% for supervisor in supervisors %}
            <option
value="{{ supervisor.id }}">{{ supervisor.full_name }}</option>
          {% endfor %}
        </select>
      </div>
      <div class="mb-3">
        <label class="form-label fw-500">Тема дипломної роботи
<span class="text-danger">*</span></label>
        <textarea name="topic" class="form-control" rows="3"
placeholder="Розробка веб-додатку..."
required></textarea>
      </div>
      <div class="mb-4">
        <label class="form-label fw-500">Дата захисту</label>
        <input type="date" name="defense_date" class="form-
control">
        <small class="text-muted">Необов'язково</small>
      </div>

      <button type="submit" class="btn btn-primary w-100 py-2">
        <i class="bi bi-check2-circle me-2"></i>Призначити

```

```

        </button>
    </form>
</div>
</div>
{% endif %}
</div>
</div>
{% endblock %}

```

app/templates/admin/stages.html – Шаблони етапів

Лістинг М.5 – Шаблони етапів (app/templates/admin/stages.html)

```

{% extends "base.html" %}
{% block title %}Шаблони етапів{% endblock %}

{% block content %}
<div class="d-flex justify-content-between align-items-center mb-4">
    <div>
        <h4 class="fw-bold mb-1">Шаблони етапів</h4>
        <p class="text-muted mb-0">Визначають структуру плану для всіх студентів</p>
    </div>
    <button type="button" class="btn btn-primary" data-bs-toggle="modal" data-bs-target="#addStageModal">
        <i class="bi bi-plus-lg me-1"></i>Додати етап
    </button>
</div>

{% if total_weight != 100 %}
<div class="alert alert-warning">
    <i class="bi bi-exclamation-triangle me-2"></i>
    Поточна сума вар: <strong>{{ total_weight }}%</strong>.
    Сума активних етапів повинна дорівнювати <strong>100%</strong>.
    Залишилось: <strong>{{ 100 - total_weight }}%</strong>
</div>
{% else %}
<div class="alert alert-success">
    <i class="bi bi-check-circle me-2"></i>
    Сума вар: <strong>100%</strong> – правильно!
</div>
{% endif %}

<div class="card">
    <div class="card-body p-0">
        <table class="table table-hover mb-0 align-middle">
            <thead class="table-light">
                <tr><th style="width:50px;">№</th><th>Назва етапу</th><th>Опис</th><th style="width:100px;">Вар (%)</th><th style="width:80px;">Статус</th><th style="width:80px;">Дії</th></tr>
            </thead>

```

```

<tbody>
  {% for template in templates %}
    <tr {% if not template.is_active %}class="text-muted"{%
endif %}>
      <td><span class="badge bg-
secondary">{{ template.order_num }}</span></td>
      <td class="fw-500">{{ template.name }}</td>
      <td><small class="text-muted">{{ template.description or '-'
' }}</small></td>
      <td><span class="badge bg-info text-dark fs-
6">{{ template.weight }}%</span></td>
      <td>
        {% if template.is_active %}
          <span class="badge bg-success">Активний</span>
        {% else %}
          <span class="badge bg-secondary">Вимкнено</span>
        {% endif %}
      </td>
      <td>
        {% if template.is_active %}
          <form method="POST"
action="{{ url_for('admin.delete_stage', template_id=template.id) }}"
          style="display:inline">
            <input type="hidden" name="csrf_token"
value="{{ csrf_token() }}">
            <button type="submit" class="btn btn-sm btn-outline-
danger"
              onclick="return confirm('Деактивувати етап
\'{{ template.name }}\'?') "
              title="Деактивувати">
              <i class="bi bi-trash"></i>
            </button>
          </form>
        {% endif %}
      </td>
    </tr>
  {% endfor %}
</tbody>
<tfoot class="table-light">
  <tr>
    <td colspan="3" class="text-end fw-500">Сума активних
var:</td>
    <td><span class="badge {{ 'bg-success' if total_weight ==
100 else 'bg-warning text-dark' }}" fs-
6">{{ total_weight }}%</span></td>
    <td colspan="2"></td>
  </tr>
</tfoot>
</table>
</div>
</div>

```

```

<!-- Модальне вікно: додати етап -->
<div class="modal fade" id="addStageModal" tabindex="-1">
  <div class="modal-dialog">
    <div class="modal-content">
      <form method="POST" action="{{ url_for('admin.add_stage') }}">
        <input type="hidden" name="csrf_token"
value="{{ csrf_token() }}">
        <div class="modal-header">
          <h5 class="modal-title"><i class="bi bi-plus-circle me-
2"></i>Новий етап</h5>
          <button type="button" class="btn-close" data-bs-
dismiss="modal"></button>
        </div>
        <div class="modal-body">
          <div class="mb-3">
            <label class="form-label fw-500">Назва етапу <span
class="text-danger">*</span></label>
            <input type="text" name="name" class="form-control"
placeholder="Програмна реалізація" required>
          </div>
          <div class="mb-3">
            <label class="form-label fw-500">Вара (%) <span
class="text-danger">*</span></label>
            <input type="number" name="weight" class="form-control"
min="1" max="{{ 100 - total_weight }}" required>
            <small class="text-muted">Залишилось: {{ 100 -
total_weight }}%</small>
          </div>
          <div class="mb-3">
            <label class="form-label fw-500">Опис</label>
            <textarea name="description" class="form-control" rows="2"
placeholder="Короткий опис що входить до цього
етапу..."></textarea>
          </div>
        </div>
        <div class="modal-footer">
          <button type="button" class="btn btn-secondary" data-bs-
dismiss="modal">Скасувати</button>
          <button type="submit" class="btn btn-primary">
            <i class="bi bi-plus-lg me-1"></i>Додати
          </button>
        </div>
      </form>
    </div>
  </div>
</div>
{% endblock %}

```

Лістинг М.6 – Редагування теми (app/templates/admin/edit_thesis.html)

```
{% extends "base.html" %}
{% block title %}Редагування теми{% endblock %}

{% block content %}
<div class="row justify-content-center">
  <div class="col-md-8 col-lg-7">

    <div class="d-flex align-items-center gap-2 mb-4">
      <a href="{% url_for('admin.dashboard') %}" class="btn btn-sm
btn-outline-secondary">
        <i class="bi bi-arrow-left"></i>
      </a>
      <h4 class="fw-bold mb-0">Редагування теми студента</h4>
    </div>
    <div class="card mb-3">
      <div class="card-body p-4">
        <p class="text-muted mb-1">Студент</p>
        <p class="fw-bold mb-0">{{ thesis.student.full_name }}</p>
        <small class="text-muted">{{ thesis.student.email }}</small>
      </div>
    </div>
    <div class="card">
      <div class="card-body p-4">
        <form method="POST" action="{% url_for('admin.edit_thesis',
thesis_id=thesis.id) %}">
          <input type="hidden" name="csrf_token"
value="{% csrf_token() %}">

          <div class="mb-3">
            <label class="form-label fw-500">
              Керівник <span class="text-danger">*</span>
            </label>
            <select name="supervisor_id" class="form-select" required>
              {% for s in supervisors %}
                <option value="{% s.id %}"
                  {% if s.id == thesis.supervisor_id %}>selected{%
endif %}>
                  {{ s.full_name }}
                </option>
              {% endfor %}
            </select>
          </div>
          <div class="mb-3">
            <label class="form-label fw-500">
              Тема дипломної роботи <span class="text-danger">*</span>
            </label>
            <textarea name="topic" class="form-control" rows="3"
              required>{{ thesis.topic }}</textarea>
          </div>
          <div class="mb-4">

```

```

        <label class="form-label fw-500">Дата захисту</label>
        <input type="date" name="defense_date" class="form-
control"
            value="{{ thesis.defense_date.strftime('%Y-%m-%d')
                if thesis.defense_date else '' }}">
        <small class="text-muted">Необов'язково</small>
    </div>
    <div class="d-flex gap-2">
        <button type="submit" class="btn btn-primary flex-grow-1
py-2">
            <i class="bi bi-check2-circle me-2"></i>Зберегти зміни
        </button>
        <a href="{{ url_for('admin.dashboard') }}"
            class="btn btn-outline-secondary py-2">
            Скасувати
        </a>
    </div>
</form>
</div>
</div>
<!-- Небезпечна зона -->
<div class="card border-danger mt-4">
    <div class="card-body p-4">
        <h6 class="text-danger fw-bold mb-1">
            <i class="bi bi-exclamation-triangle me-1"></i>
            Деактивація студента
        </h6>
        <p class="text-muted small mb-3">
            Студент втратить доступ до системи. Тема та вся історія
            виконання етапів залишаться у базі даних.
        </p>
        <form method="POST"
            action="{{ url_for('admin.deactivate_thesis',
                thesis_id=thesis.id) }}"
            onsubmit="return confirm(
                'Деактивувати студента
                {{ thesis.student.full_name }}?')">
            <input type="hidden" name="csrf_token"
value="{{ csrf_token() }}">
            <button type="submit" class="btn btn-outline-danger btn-sm">
                <i class="bi bi-person-x me-1"></i>Деактивувати студента
            </button>
        </form>
    </div>
</div>
</div>
</div>
{% endblock %}

```

Лістинг файлу app/static/css/style.css та допоміжних файлів

app/static/css/style.css – Власні стилі (Sidebar, Stat-cards, Login, Misc)

Лістинг Н.1 – Файл стилів (app/static/css/style.css)

```
/* — Reset ————— */
* {
  box-sizing: border-box;
}

html, body {
  height: 100%;
  margin: 0;
  padding: 0;
}

/* — Wrapper ————— */
#wrapper {
  min-height: 100vh;
  height: 100%;
}

/* — Sidebar ————— */
.sidebar {
  width: 240px;
  min-height: 100vh;
  height: 100%;
  display: flex;
  flex-direction: column;
  position: sticky;
  top: 0;
  overflow-y: auto;
  flex-shrink: 0;
}

.sidebar-nav {
  flex: 1;
}

.sidebar-nav .nav-link {
  color: rgba(255, 255, 255, .7);
  padding: .6rem 1rem;
  border-radius: 6px;
  margin: 2px 8px;
  font-size: 14px;
  transition: all .15s;
  display: flex;
  align-items: center;
```

```
.sidebar-nav .nav-link:hover,
.sidebar-nav .nav-link.active {
  color: #fff;
  background: rgba(255, 255, 255, .12);
}

.sidebar-nav .nav-link.active {
  border-left: 3px solid #0d6efd;
}

.sidebar-footer {
  margin-top: auto;
}

/* — Main content ————— */
.main-content {
  flex: 1;
  min-height: 100vh;
  background: #f8f9fa;
  overflow-y: auto;
}

/* — Stat cards ————— */
.stat-card {
  background: #fff;
  border: 1px solid #e9ecef;
  border-radius: 10px;
  padding: 1rem 1.25rem;
  height: 100%;
}

.stat-label {
  font-size: 12px;
  color: #6c757d;
  margin-bottom: 6px;
  text-transform: uppercase;
  letter-spacing: .3px;
}

.stat-value {
  font-size: 2rem;
  font-weight: 600;
  line-height: 1;
}

/* — Avatar circle ————— */
.avatar-circle {
  width: 36px;
  height: 36px;
  border-radius: 50%;
  display: flex;
```

```

    align-items: center;
    justify-content: center;
    font-weight: 600;
    font-size: 13px;
    flex-shrink: 0;
}

/* — Cards ————— */
.card {
    border-radius: 10px;
    border: 1px solid #e9ecef;
}
.card-header {
    background: #fff;
    border-bottom: 1px solid #e9ecef;
    border-radius: 10px 10px 0 0 !important;
    padding: .875rem 1rem;
    font-weight: 500;
}

/* — Login page ————— */
.login-wrapper {
    min-height: 100vh;
    width: 100%;
    display: flex;
    align-items: center;
    justify-content: center;
    background: linear-gradient(135deg, #667eea 0%, #764ba2 100%);
}

.login-card {
    width: 100%;
    max-width: 420px;
    background: #fff;
    border-radius: 16px;
    padding: 2.5rem;
    box-shadow: 0 20px 60px rgba(0, 0, 0, .15);
}

/* — Misc ————— */
.progress {
    border-radius: 10px;
}

.badge {
    padding: .35em .65em;
    font-weight: 500;
}

.table > :not(caption) > * > * {
    padding: .75rem 1rem;
}

```

```
.fw-500 {
    font-weight: 500;
}
```

wsgi.py – Точка входу WSGI для Gunicorn

Лістинг Н.2 – Точка входу WSGI (wsgi.py)

```
from app import create_app
app = create_app('development')

if __name__ == '__main__':
    app.run(host='0.0.0.0', port=5000, debug=True)
```

seed.py – Скрипт заповнення початкових даних БД

Лістинг Н.3 – Скрипт seed.py для початкових даних

```
from app import create_app, db
from app.models import User, StageTemplate

app = create_app('development')
with app.app_context():
    admin = User(email='admin@example.com', full_name='Адміністратор',
role='admin')
    admin.set_password('admin123')

    sv = User(email='supervisor@example.com', full_name='Коваленко
Олер', role='supervisor')
    sv.set_password('super123')

    st = User(email='student@example.com', full_name='Петренко Петро',
role='student')
    st.set_password('student123')

    for s in [
        StageTemplate(name='Аналіз предметної області', weight=10,
order_num=1),
        StageTemplate(name='Проектування архітектури', weight=15,
order_num=2),
        StageTemplate(name='Розробка бази даних', weight=15,
order_num=3),
        StageTemplate(name='Програмна реалізація', weight=40,
order_num=4),
        StageTemplate(name='Тестування', weight=10, order_num=5),
        StageTemplate(name='Оформлення записки', weight=10,
order_num=6),
    ]:
        db.session.add(s)

    db.session.add_all([admin, sv, st])
    db.session.commit()
    print('Готово!')
```

requirements.txt – Залежності Python

Лістинг Н.4 – Файл залежностей (requirements.txt)

```
Flask==3.0.3
Flask-SQLAlchemy==3.1.1
Flask-Migrate==4.0.7
Flask-Login==0.6.3
Flask-WTF==1.2.1
psycopg2-binary==2.9.9
Werkzeug==3.0.3
python-dotenv==1.0.1
gunicorn==22.0.0
openpyxl==3.1.3
WTForms==3.1.2
email-validator==2.1.1
pytest==8.2.2
```

.env.example – Шаблон конфігурації середовища

Лістинг Н.5 – Шаблон .env.example

```
SECRET_KEY=change-this-to-random-string-in-production
FLASK_ENV=development
POSTGRES_DB=thesis_db
POSTGRES_USER=thesis_user
POSTGRES_PASSWORD=thesis_pass123
DATABASE_URL=postgresql://thesis_user:thesis_pass123@db:5432/thesis_db
UPLOAD_FOLDER=/app/uploads
MAX_CONTENT_LENGTH=16777216
```

.gitignore – Перелік файлів що не зберігаються в репозиторії

Лістинг Н.6 – Файл .gitignore

```
.env
__pycache__/
*.pyc
*.pyo
uploads/
migrations/
instance/
.DS_Store
*.db
*.sqlite
.pytest_cache/
*.egg-info/
```

```
THEESIS-MONITOR
├── app
│   ├── routes
│   │   ├── __pycache__
│   │   ├── __init__.py
│   │   ├── admin.py
│   │   ├── auth.py
│   │   ├── student.py
│   │   └── supervisor.py
│   ├── static
│   │   ├── css
│   │   │   └── style.css
│   │   └── js
│   │       └── main.js
│   └── templates
│       ├── admin
│       │   ├── add_user.html
│       │   ├── assign.html
│       │   ├── dashboard.html
│       │   ├── edit_thesis.html
│       │   ├── stages.html
│       │   └── users.html
│       ├── auth
│       │   └── login.html
│       ├── student
│       │   ├── no_thesis.html
│       │   └── plan.html
│       ├── supervisor
│       │   ├── dashboard.html
│       │   ├── student_detail.html
│       │   └── base.html
│       ├── __init__.py
│       ├── config.py
│       └── models.py
├── tests
├── .env
├── .env.example
├── .gitignore
├── docker-compose.yml
├── Dockerfile
├── requirements.txt
├── seed.py
└── wsgi.py
```

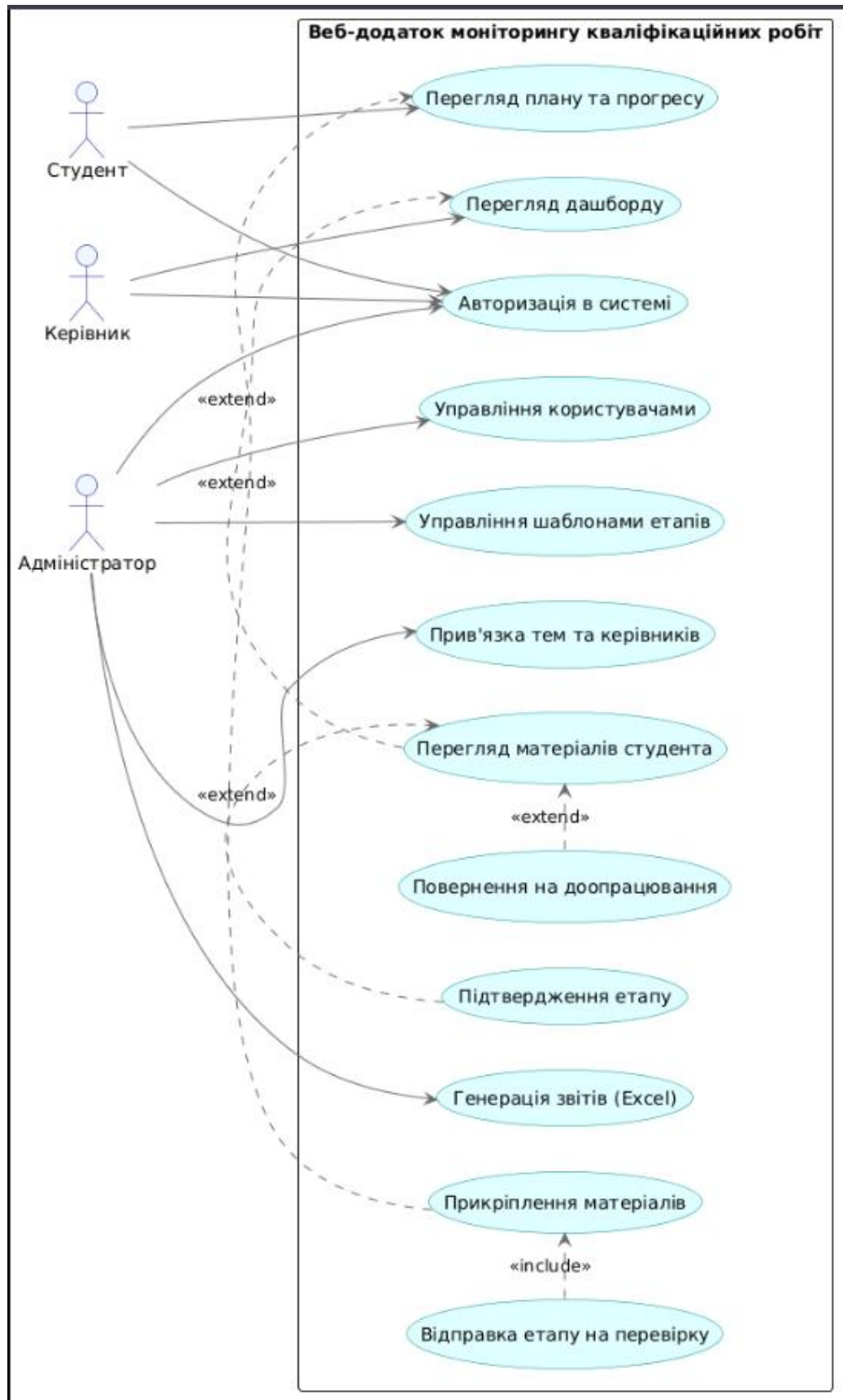


Рисунок Ф.1

БІБЛІОГРАФІЧНА ДОВІДКА

Тема бакалаврської роботи: *Розробка веб-додатку моніторингу та контролю виконання кваліфікаційних робіт студентами на базі Python та технології контейнеризації*

Обсяг пояснювальної записки 69 аркуші.

<u>7</u>	таблиць;
<u>23</u>	рисунка;
<u>17</u>	додатків.

Дата завершення роботи: *10 червня 2026 р.*

Підпис студента-дипломника _____ Гоменюк М.В.