БАКАЛАВРСЬКА РОБОТА

БР.КІ-30.00.00.000 ПЗ

Група КІ-22-2

Волосюк Михайло

2026

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

Волосюк Михайло Іванович

УДК 004.66

БАКАЛАВРСЬКА РОБОТА

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

Комп'ютерна інженерія
(назва освітньої програми)

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

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

Здобувач освітнього ступеня _____ *Волосюк М. І.* _____
(підпис, ініціали та прізвище здобувача)

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

Допущено до захисту
Завідувач кафедри КСМ

д.т.н.,професор _____ *С.І. Мельничук*
(посада) (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

Волосюку Михайлу Івановичу

(прізвище, ім'я, по батькові)

1. Тема проекту (роботи) *"Розробка web-додатку формування єдиного реєстру наукових публікацій студентів та аспірантів університету на основі API ORCID, Python та Docker"*

керівник проекту (роботи) *Гуменюк Т. В., доц.*

(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

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 *Матеріали і результати отримані під час проходження переддипломної практики, методичні вказівки, технічна література.*

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити) *1. Аналіз предметної області та постановка завдання автоматизації обліку наукової діяльності.*
2. Мережеве проєктування web-додатку.
3. Програмна реалізація web-додатку.

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

6. Консультанти розділів роботи

Розділ	Консультант	Підпис, дата

7. Дата видачі завдання *02 лютого 2026 р.*

№ з/п	Назва етапів дипломного проекту (роботи)	Термін виконання етапів проекту (роботи)	Примітка
1	*Аналіз предметної області та постановка завдання автоматизації обліку наукової діяльності*	*Лютий, 2026*	
2	*Мережеве проєктування web-додатку*	*Березень, 2026*	
3	*Програмна реалізація реалізація web-додатку*	*Травень, 2026*	
4	*Оформлення пояснювальної записки*	*Червень, 2026*	

Студент _____ *Волосюк М. І.*
(підпис) (прізвище та ініціали)

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

АНОТАЦІЯ

В дипломній роботі було здійснено розробку web-додатка формування єдиного реєстру наукових публікацій студентів та аспірантів університету на основі API ORCID, Python та Docker.

Метою роботи є створення сучасного автономного програмного забезпечення класу on-premise для автоматизації збору, обробки, валідації та безпечного довготривалого збереження наукометричних метаданих здобувачів вищої освіти в локальних інформаційних мережах навчальних закладів.

У роботі проведено комплексний аналіз нормативно-правової бази України щодо захисту інформації та порівняльний аналіз відкритих програмних інтерфейсів (API) сучасних наукометричних платформ. Розглянуто проблему унікальної ідентифікації авторів та потребу впровадження ізольованого рішення. На основі аналізу сформульовано завдання проєкту й запропоновано розробку системи на базі персистентного ідентифікатора ORCID. Розроблено трирівневу мікросервісну архітектуру системи на основі Python, FastAPI, PostgreSQL та Docker. Спроєктовано реляційну базу даних, алгоритми асинхронної клієнт-серверної взаємодії, ідемпотентного збереження даних та адаптивний веб-інтерфейс. Проведено тестування швидкодії, транзакційної цілісності та перевірку захищеності системи від XSS-вразливостей.

В результаті створено ефективну, безпечну та масштабовану систему автоматизації обліку наукової діяльності із зручним інтерфейсом для швидкого моніторингу та управління публікаціями співробітниками організації.

Ключові слова: реєстр публікацій, наукометрія, ORCID API, Python, FastAPI, PostgreSQL, Docker, мікросервісна архітектура.

SUMMARY

This diploma thesis presents the development of a web application for forming a unified registry of scientific publications for university students and postgraduates based on the ORCID API, Python, and Docker.

The aim of the work is to create modern, autonomous on-premise software to automate the collection, processing, validation, and secure long-term storage of scientometric metadata of higher education seekers within the local information networks of educational institutions.

A comprehensive analysis of Ukraine's regulatory framework for information protection and a comparative analysis of open application programming interfaces (APIs) of modern scientometric platforms were conducted. The problem of unique author identification and the necessity of implementing an isolated solution were considered. Based on this analysis, the project tasks were formulated, and the development of a system based on the persistent ORCID identifier was proposed. A three-tier microservice architecture using Python, FastAPI, PostgreSQL, and Docker was developed. A relational database, algorithms for asynchronous client-server interaction, idempotent data storage, and an adaptive web interface were designed. The system underwent performance testing, transactional integrity checks, and security verification against XSS vulnerabilities.

As a result, an efficient, secure, and scalable system for automating the tracking of scientific activities has been created, featuring a user-friendly interface for rapid monitoring and publication management by the organization's staff.

Keywords: publication registry, scientometrics, ORCID API, Python, FastAPI, PostgreSQL, Docker, microservice architecture.

ЗМІСТ

					БР.КІ-30.00.00.000 ПЗ		
Зм.	Арк.	№ докум.	Підпис	Дата			
Розроб.		Волосюк М. І.			*Розробка web-додатку формування єдиного реєстру наукових публікацій студентів та аспірантів університету на основі API ORCID, Python та Docker*	Літ.	Арк. 3 / Акрушів 68
Перевір.		Гуменюк Т. В.					
Реценз.		Гарасимів Т. Г.				ІФНТУНГ КІ-22-2	
Н. контр.		Лазорів А.М.					
Затверд.		Мельничук С. І.					

ВСТУП

Актуальність теми. Сучасний етап розвитку вищої освіти та наукової діяльності в Україні характеризується стрімкою інституційною цифровізацією, яка вимагає впровадження високонадійних, масштабованих та автоматизованих засобів збору й аналізу наукометричної інформації. Інтеграція вітчизняних університетів у європейський та світовий науково-дослідний простір, відповідно до принципів відкритої науки (Open Science), актуалізує завдання точного обліку, моніторингу та верифікації публікаційної активності не лише професорсько-викладацького складу, а й молодих вчених — студентів та аспірантів. Ефективність внутрішньої взаємодії, швидкість прийняття управлінських рішень щодо стимулювання досліджень, а також успішність проходження процедур державної атестації та акредитації освітніх програм безпосередньо залежать від повноти та актуальності наукометричних даних організації [1].

Традиційні підходи до моніторингу наукових результатів здобувачів вищої освіти, які досі застосовуються у багатьох закладах, базуються на децентралізованому ручному заповненні звітних паперових або електронних форм (таблиць Excel). Такий підхід демонструє критично низьку обчислювальну та організаційну ефективність [2].

Найбільш перспективним та універсальним інструментом для вирішення проблеми однозначної ідентифікації дослідників у глобальному просторі є міжнародна система ORCID (Open Researcher and Contributor ID). Вона надає унікальні стійкі цифрові ідентифікатори та відкритий програмний інтерфейс (Public API), який дозволяє в автоматизованому режимі отримувати структуровані метадані про науковий доробок автора. Проте безпосереднє використання відкритого API ORCID кінцевими користувачами (деканатами, кафедрами) без спеціалізованого програмного прошарку є складним інженерним завданням, оскільки отримані структури даних потребують

фільтрації, очищення, трансформації (ETL) та агрегації у межах конкретної організаційної структури університету [3].

З погляду комп'ютерної інженерії, створення подібних інтеграційних шлюзів та інформаційних систем вимагає застосування сучасних архітектурних патернів, технологій асинхронного неблокуючого програмування та методів контейнеризації [4]. Таким чином, розробка веб-додатка для формування єдиного реєстру наукових публікацій студентів та аспірантів університету на основі API ORCID, Python та Docker є актуальним інженерним завданням у межах спеціальності «Комп'ютерна інженерія».

Об'єкт дослідження – процес автоматизованого збору, обробки, валідації та довготривалого збереження наукометричних метаданих здобувачів вищої освіти в локальних інформаційних мережах навчальних закладів.

Предмет дослідження – методи, алгоритми, архітектурні рішення, протоколи передачі даних REST та програмне забезпечення клієнт-серверної інформаційної системи на базі платформ Python (FastAPI), PostgreSQL та Docker.

Мета роботи – розробка web-додатку формування єдиного реєстру наукових публікацій студентів та аспірантів університету на основі API ORCID, Python та Docker.

Для досягнення поставленої мети необхідно вирішити такі **завдання**:

− Проаналізувати нормативно-правове забезпечення України у сфері обліку наукової діяльності закладів вищої освіти та сформувати профіль вимог до єдиного реєстру публікацій.

− Провести детальний порівняльний аналіз архітектур передачі даних та відкритих програмних інтерфейсів (API) сучасних наукометричних систем, обґрунтувати вибір протоколу взаємодії та технологічного стеку розробки.

− Розробити логічну та фізичну структуру реляційного сховища даних на базі СУБД PostgreSQL з урахуванням каскадних обмежень, індексації магістральних полів та вимог цілісності ACID.

− Побудувати математичне та алгоритмічне обґрунтування процесів асинхронної взаємодії веб-сервера із зовнішнім API наукометричних реєстрів для оцінки швидкодії при пакетній обробці профілів.

− Розробити подієво-орієнтований серверний програмний комплекс (Backend) на базі Python та FastAPI з інтеграцією механізмів асинхронного клієнта HTTP та автоматизованого мапування сутностей (ORM).

− Створити адаптивний клієнтський веб-інтерфейс (Frontend) на базі сучасних інструментів шаблонізації та стилізації для гнучкої візуалізації реєстру та управління синхронізацією.

− Виконати інкапсуляцію всіх компонентів системи в ізольовані Docker-контейнери та розробити декларативні маніфести Docker Compose для координації мікросервісного середовища розгортання.

− Провести експериментальне тестування працездатності розроблених маршрутів та оцінити показники швидкодії системи при роботі в ізольованій локальній мережі.

Методи дослідження. Для вирішення поставлених завдань у роботі використано такі методи: методи системного та структурного аналізу – при дослідженні інформаційних потоків і побудові загальної архітектури веб-додатка; методи теорії реляційних баз даних – для проєктування оптимальної схеми збереження даних без надликовості; методи об'єктно-орієнтованого та асинхронного програмування – для побудови модульної структури клієнтської та серверної частин; методи клієнт-серверної інтеграції на основі архітектурного стилю REST – для взаємодії із зовнішніми веб-службами.

Практичне значення отриманих результатів. Розроблена інформаційна система є повністю автономним, працездатним програмним продуктом класу on-premise, який може бути безпосередньо впроваджений у практику роботи кафедр та науково-дослідних інститутів вітчизняних університетів. Код системи є платформонезалежним завдяки контейнеризації, легко масштабується і може служити базою для проектування ширших університетських інформаційних систем та баз знань.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАВДАННЯ АВТОМАТИЗАЦІЇ ОБЛІКУ НАУКОВОЇ ДІЯЛЬНОСТІ

1.1 Нормативно-правова база захисту інформації та обліку наукових результатів в Україні

Забезпечення інформаційної безпеки та побудова чіткого контуру обліку наукових результатів є фундаментальною вимогою при проєктуванні та розгортанні інтеграційних платформ усередині сучасних університетів. На відміну від публічних розважальних або загальних інформаційних сервісів, університетський реєстр наукових публікацій оперує потоками інформації, яка безпосередньо підпадає під дію законодавства України у сфері захисту інформації, інтелектуальної власності та персональних даних. Побудова архітектури локального веб-додатка повинна суворо спиратися на законодавчі акти та нормативні документи технічного захисту інформації (НД ТЗІ), що діють в Україні [5].

Основним законодавчим фундаментом у цій галузі є Закон України «Про захист інформації в інформаційно-комунікаційних системах» [4]. Цей закон чітко визначає, що державні інформаційні ресурси (до яких належать бази даних державних університетів), а також інформація з обмеженим доступом або конфіденційні дані, повинні оброблятися виключно в системах, де створено комплексну систему захисту інформації (КСЗІ) з підтвердженою відповідністю. При використанні сторонніх публічних систем або незахищених засобів передачі даних виникає ризик неконтрольованого копіювання чи модифікації облікових записів здобувачів вищої освіти. Стаття 8 цього Закону наголошує на персональній відповідальності керівників установ за забезпечення належного рівня захисту даних. При використанні публічних месенджерів або сторонніх несертифікованих хмарних SaaS-платформ для збору наукових звітів, тексти та персональні відомості передаються через сторонні транзитні сервери, що

автоматично унеможливлює контроль за копіюванням даних та створює пряме порушення вимог законодавства щодо локалізації обробки критичних даних.

Поряд із цим, функціонування реєстру наукових публікацій нерозривно пов'язане з обробкою облікових записів, паспортних відомостей студентів та аспірантів, їхніх академічних статусів, місць роботи, корпоративних e-mail адрес та унікальних цифрових ідентифікаторів, що згідно із Законом України «Про захист персональних даних» [5] визначається як обробка персональних даних. Відповідно до статті 22 цього Закону, володільці персональних даних зобов'язані забезпечити технічний та системний захист цих масивів від випадкової втрати, знищення або незаконного доступу.

З інженерної точки зору, проектування локальної системи «UniSciRegistry» спрямоване на повне виконання вимог міжнародного стандарту ISO/IEC 27001 (ДСТУ ISO/IEC 27001) «Інформаційні технології. Методи захисту. Системи управління інформаційною безпекою» [6]. У межах даного стандарту реалізується тріада безпеки: конфіденційність (Confidentiality), цілісність (Integrity) та доступність (Availability). Локальне розміщення системи (on-premise) всередині обчислювальної мережі університету із застосуванням Docker-контейнеризації дозволяє повністю контролювати фізичний та логічний доступ до бази даних PostgreSQL, ізолювати її від зовнішніх несанкціонованих впливів, та одночасно забезпечувати безперешкодний доступ внутрішніх користувачів до актуальних реєстрів через корпоративний веб-інтерфейс.

Для деталізації архітектурних вимог до швидкодії, рівнів доступу та інтерфейсу розроблюваної системи було виконано системний аналіз та побудовано профілі потенційних користувачів (акторів) системи в межах типового університету:

Здобувачі вищої освіти (Студенти та аспіранти). Використовують систему для первинного внесення свого унікального ідентифікатора ORCID ID у загальну базу даних університету та перегляду власного верифікованого списку публікацій. Для даної категорії користувачів критично важливою є

простота взаємодії, відсутність необхідності ручного копіювання бібліографічних описів та швидка реакція інтерфейсу. Доступ здійснюється через стандартні веб-браузери як з настільних комп'ютерів, так і з мобільних пристроїв.

Співробітники кафедр / Деканати (Модератори та адміністратори реєстру). Цей профіль користувачів потребує розширених функціональних можливостей для моніторингу публікаційної активності по кафедрах, факультетах або окремих освітніх програмах (наприклад, спеціальності 123 «Комп'ютерна інженерія»). Вони ініціюють процеси пакетної синхронізації даних із серверами ORCID, формують зведені аналітичні звіти, виявляють здобувачів, які не мають публікацій, та верифікують DOI праць. Час відповіді системи при генерації звітів по кафедрі не повинен перевищувати кількох секунд.

Системні адміністратори (Адміністратори безпеки ІС університету). Головні технічні актори, які здійснюють контроль за загальним станом працездатності серверного контуру, керують пулом підключень до бази даних, виконують регулярне резервне копіювання транзакційного сховища PostgreSQL та здійснюють аудит дій користувачів. Для адміністраторів критично важливою є наявність зручних інструментів моніторингу логів сервера, низьке споживання апаратних ресурсів (RAM, CPU) контейнерами додатка та можливість швидкого розгортання або міграції системи за допомогою Docker Compose маніфестів без зупинки суміжних університетських сервісів.

1.2 Проблематика унікальної ідентифікації авторів у наукометричному просторі

При побудові автоматизованих систем обліку наукової діяльності розробники стикаються із серйозними технічними викликами, пов'язаними із природою представлення бібліографічної інформації в мережевому просторі. Головною проблемою є відсутність глобальної уніфікації текстових імен

авторів. Явище омонімії (наявність великої кількості дослідників із однаковими прізвищами та ініціалами) унеможливлює точний автоматичний пошук праць за текстовим рядком [7].

Ситуація значно ускладнюється для українських науковців через відсутність єдиного стандарту транслітерації кириличних імен на латиницю. Одне й те саме прізвище (наприклад, «Гриценко») в різних закордонних журналах та базах даних (Scopus, Web of Science, Google Scholar) може бути записане у кількох варіаціях (Hrytsenko, Grycenko, Hricenko), що призводить до штучного розщеплення профілю автора та втрати частини його наукометричних показників (індексу Гірша, загальної кількості цитувань). Крім того, зміна прізвищ (наприклад, при одруженні) або зміна афіліації (назви університету) створює додаткові розриви в історичних даних публікацій молодих вчених [8].

Спроби вирішення цієї проблеми на рівні окремих університетів шляхом створення локальних внутрішніх ідентифікаторів (наприклад, порядкових номерів у кадровій базі) демонструють повну неспроможність при взаємодії із зовнішнім світом, оскільки закордонні видавництва нічого не знають про внутрішні системи українських ЗВО. Єдиним архітектурно правильним інженерним рішенням є використання глобальних персистентних цифрових ідентифікаторів (Persistent Identifiers - PID).

Серед існуючих глобальних систем ідентифікації авторів найбільше поширення набули три платформи: Scopus Author ID, ResearcherID (Web of Science) та ORCID. Проте перші дві системи є пропрієтарними, комерційно закритими та жорстко прив'язаними до конкретних наукометричних баз даних компаній Elsevier та Clarivate Analytics. Якщо студент або аспірант публікується у вітчизняних фахових виданнях категорії «Б», які не індексуються в Scopus, ці комерційні системи не відображатимуть його діяльність.

На противагу їм, система ORCID (Open Researcher and Contributor ID) є відкритим, некомерційним, незалежним проектом, який підтримується

світовою науковою спільнотою. Реєстр ORCID надає кожному досліднику унікальний 16-значний цифровий код (наприклад, 0000-0002-1825-0097), який є персистентним — він належить автору протягом усього життя, незалежно від зміни місця навчання, роботи чи транслітерації прізвища. Платформа ORCID побудована за принципом хабу: вона дозволяє пов'язувати в єдиному профілі інформацію з інших систем (Scopus, Web of Science, GitHub, суверенні репозиторії університетів) та надає вільний програмний доступ до цих метаданих через відкритий REST API. Це робить ORCID ідеальним інфраструктурним фундаментом для проєктування університетського реєстру наукових публікацій молодих вчених [9].

1.3 Порівняльний аналіз відкритих програмних інтерфейсів (API) наукометричних платформ

Для обґрунтування вибору джерела даних та архітектури взаємодії веб-додатка було проведено детальний порівняльний аналіз відкритих програмних інтерфейсів провідних наукометричних платформ: Google Scholar, Scopus API, Web of Science Starter API та ORCID Public API v3.0. Аналіз виконувався за критеріями відкритості, обмежень на кількість запитів (Rate Limits), форматів передачі даних, повноти покриття та складності програмної інтеграції [10].

Детальний аналіз наведених даних показує, що платформа Google Scholar за всієї своєї повноти покриття українських публікацій не має офіційного програмного інтерфейсу. Побудова університетської системи на основі веб-скрапінгу (пакетного розбору HTML-сторінок за допомогою бібліотек типу BeautifulSoup або інструментів автоматизації браузера Selenium) є тупиковим інженерним шляхом. Сервери Google застосовують інтелектуальні алгоритми виявлення ботів, миттєво блокуючи IP-адреси локального сервера університету та вимагаючи введення графічної CAPTCHA. Це унеможливлює побудову стабільних регламентних завдань (Cron jobs) для фонового оновлення реєстрів [11].

Таблиця 1.1 – Порівняльний аналіз відкритих програмних інтерфейсів наукометричних систем

Критерій порівняння	Google Scholar	Scopus API	Web of Science API	ORCID Public API v3.0
Офіційна наявність API	Відсутній (офіційного API немає)	Наявний (комерційний)	Наявний (комерційний)	Наявний (повністю відкритий)
Вартість доступу	Безкоштовно (через скрапінг)	Безкоштовно лише базовий ліміт	Потребує дорогої підписки	Повністю безкоштовно
Формат даних	Неструктурований HTML	JSON / XML	JSON / XML	JSON / XML
Rate Limits (Обмеження)	Жорстке блокування за IP	До 5000 запитів на тиждень	Обмежено тарифним планом	До 50 запитів на секунду
Потреба в токенах (Auth)	Не застосовується	Потребує API-Key та інст. токен	Потребує OAuth2 / Bearer Token	OAuth 2.0 (для Public - спрощено)
Легітимність використання	Порушує Terms of Service	Обмежена ліцензією Elsevier	Обмежена Clarivate	Повністю легітимно (Open Data)
Основний недолік	Потребує постійного обходу капчі	Не бачить статті поза Scopus	Не бачить статті поза WoS	Залежить від заповнення автором

Інтерфейси Scopus API та Web of Science API надають високоструктуровані дані в форматах JSON та XML, що є зручним для парсингу засобами Python. Проте вони накладають жорсткі обмеження на використання: повний доступ до метаданих можливий лише за наявності дорогої інституційної підписки університету на ці бази даних. Якщо підписку не продовжено на поточний рік, працездатність серверного контуру буде повністю заблоковано. Крім того, ці бази ігнорують значний пласт локальних наукових робіт студентів (тези доповідей на всеукраїнських конференціях, статті у збірниках праць університету), які ще не досягли рівня індексації у Scopus [12].

Програмний інтерфейс ORCID Public API v3.0 позбавлений вказаних архітектурних та ліцензійних обмежень. Він працює за принципами REST, є абсолютно безкоштовним для всього світу, підтримує високу інтенсивність запитів (до 50 запитів на секунду на один IP-адрес) та повертає дані у стандартизованому JSON-форматі, що ідеально відповідає вимогам побудови асинхронних клієнт-серверних додатків. Головною особливістю ORCID є те, що він дозволяє агрегувати в одному профілі як статті зі Scopus/WoS (шляхом автоматичного імпорту через CrossRef), так і ручні записи про локальні публікації, створюючи максимально повну картину наукового доробку студента чи аспіранта [13]. Таким чином, вибір ORCID Public API як базового джерела даних для університетського реєстру є єдиним обґрунтованим інженерним рішенням у межах даного проєкту.

1.4 Огляд існуючих автоматизованих систем управління науковою інформацією (CRIS)

Для формування концепції власного програмного продукту було виконано аналіз існуючих архітектурних рішень та систем класу CRIS (Current Research Information System), які застосовуються для автоматизації обліку наукових досліджень на міжнародному та вітчизняному рівнях. Розглянуто системи Pure від Elsevier, DSpace-CRIS та локальні університетські розробки на базі застарілих систем керування вмістом (CMS) [14].

Система Pure (Elsevier) є світовим лідером серед корпоративних CRIS-платформ. Вона пропонує колосальний спектр можливостей: відстеження публікацій, грантів, патентів, побудова графів наукового співробітництва, глибока інтеграція з кадровими базами даних та автоматичний імпорт із бази Scopus. Проте Pure є закритим пропрієтарним програмним комплексом із надзвичайно високою вартістю ліцензії (десятки тисяч доларів щорічно). Розгортання та супровід такої системи потребує виділення окремого штату інженерів та потужного серверного обладнання. Для більшості вітчизняних

університетів, а тим більше для окремих технічних кафедр, впровадження Pure є економічно неможливим та функціонально надлишковим [15].

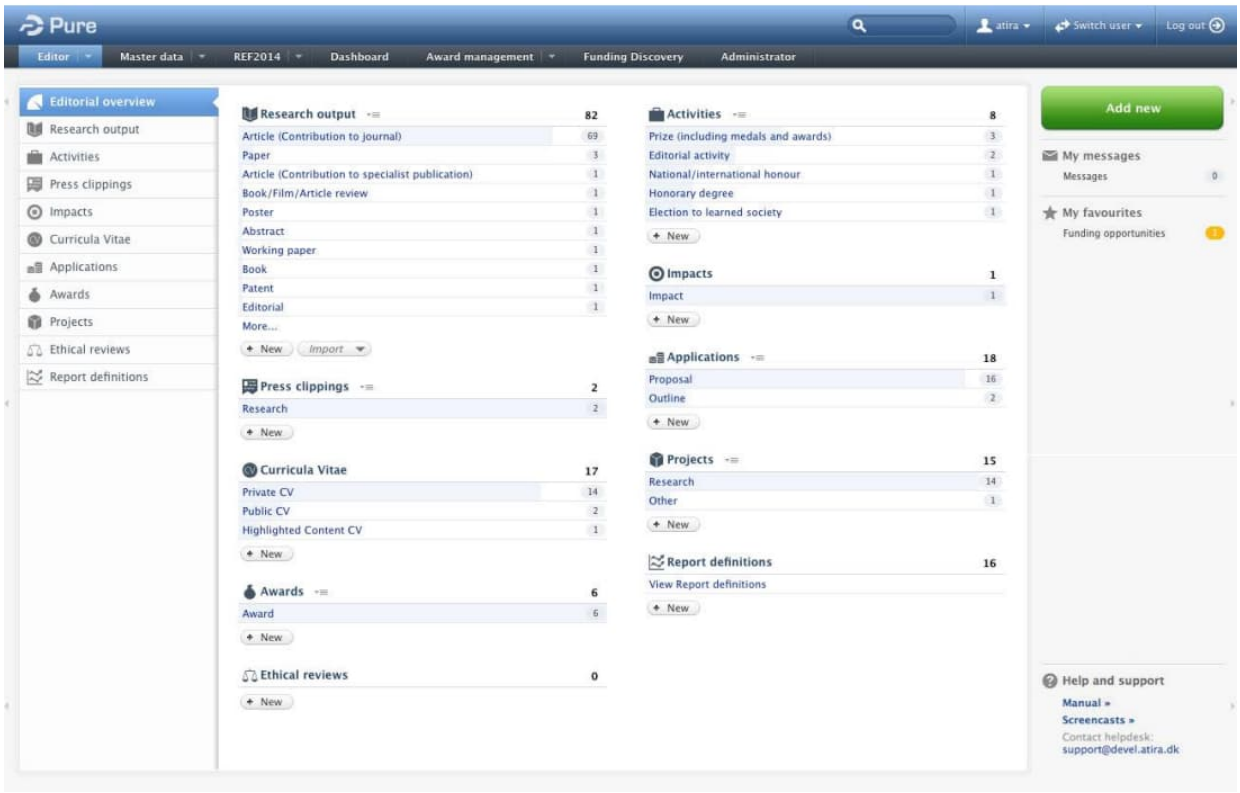

Рисунок 1.1 – Система Pure від Elsevier

Платформа DSpace-CRIS (на базі класичного DSpace) є відкритим (open-source) рішенням для побудови інституційних репозиторіїв та профілів вчених. Вона орієнтована на концепцію Open Science і дозволяє зводити дані з різних джерел. Проте DSpace проєктувався першочергово як статичне сховище повнотекстових PDF-документів (архів), а не як динамічний реєстр із гнучкими можливостями пакетної API-синхронізації в реальному часі. Процес додавання інформації там все одно вимагає значних ручних дій від модераторів, а інтерфейс системи може бути занадто громіздким для швидкої навігації та побудови моментальних кафедральних звітів [16].

Локальні індивідуальні рішення, які створюються всередині університетів силами системних адміністраторів, найчастіше базуються на реляційних СУБД MySQL та застарілих монолітних веб-архітектурах (PHP /

WordPress / Yii2). Такі системи страждають від архітектурного зчеплення (High Coupling) — інтерфейс відображення, логіка запитів до БД та модулі взаємодії з мережею змішані в одному коді. При зміні структури зовнішніх API або при підвищенні навантаження (наприклад, коли 500 студентів одночасно намагаються оновити дані перед сесією) монолітні сервери демонструють критичне уповільнення швидкодії (ефект блокування потоків введення-виведення).

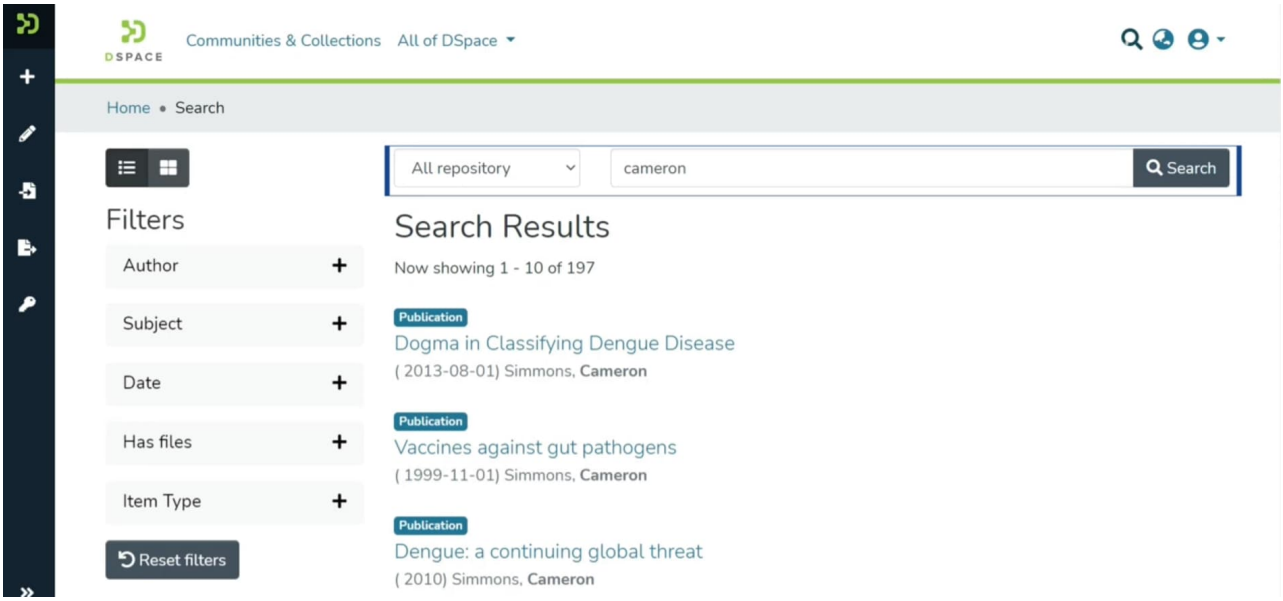

Рисунок 1.2 – Система DSpace-CRIS

Це обґрунтовує необхідність розробки власного, легковагового, мікросервісного веб-додатка «UniSciRegistry» класу on-premise на базі Python та Docker. Таке рішення дозволить повністю уникнути недоліків існуючих платформ: воно буде безкоштовним, не залежатиме від зовнішніх комерційних ліцензій, споживатиме мінімальну кількість апаратних ресурсів сервера (може працювати навіть на віртуальних машинах із 1 ГБ RAM) та забезпечить миттєву фонову синхронізацію даних завдяки асинхронній природі серверного контуру.

1.4 Постановка завдання

На основі проведеного системного аналізу предметної області, нормативної бази України та технічних обмежень існуючих наукометричних інтерфейсів, сформулюємо суворе інженерне завдання на проектування та програмну реалізацію web-додатку формування єдиного реєстру наукових публікацій студентів та аспірантів університету на основі API ORCID, Python та Docker» [4].

Основна мета роботи полягає в розробці web-додатку формування єдиного реєстру наукових публікацій студентів та аспірантів університету на основі API ORCID, Python та Docker [7].

Для досягнення цієї мети розроблювана система повинна вирішувати такі основні інженерно-технічні задачі:

1. Реалізація автономного персистентного шару (Бази даних). Створення реляційної структури в СУБД PostgreSQL, яка забезпечує строгий облік сутностей здобувачів вищої освіти (ПІБ, шифр академічної групи, курс, унікальний ORCID ID) та метаданих їхніх наукових праць (назва статті, назва журналу чи конференції, рік видання, унікальний цифровий ідентифікатор DOI, системний put-code від ORCID).

2. Побудова асинхронного інтеграційного шлюзу з ORCID API. Розробка серверних модулів мовою Python, які за допомогою неблокуючих HTTP-клієнтів здійснюють запити до ендпоінтів pub.orcid.org/v3.0/, отримують корисне навантаження (Payload) у форматі JSON, виконують його потоковий парсинг, очищення від паразитних символів та перетворення у реляційний вигляд.

3. Забезпечення ідемпотентності та цілісності даних. Впровадження алгоритмів, які при повторних або пакетних синхронізаціях профілів запобігають утворенню аномалій дублювання рядків у базі даних. Ідентифікація унікальності кожної статті повинна спиратися на композитний

аналіз полів orcid_put_code або doi на рівні обмежень СКБД (UNIQUE constraint).

4. Розробка реактивного адаптивного інтерфейсу користувача. Створення веб-сторінок на основі компонентного підходу, які забезпечують зручну навігацію по реєстру, швидкий пошук студентів за прізвищем або групою, відображення детальних профілів здобувачів із динамічним списком їхніх праць, та наявність елементів керування (кнопок) для ініціалізації миттєвого фонового оновлення даних. Інтерфейс повинен коректно підлаштовуватися під мобільні пристрої на основі гнучких CSS-сіток Tailwind.

5. Контейнеризація та автоматизація розгортання. Інкапсуляція серверного коду Python, шаблонів інтерфейсу, залежностей та конфігурацій СКБД в ізольовані Docker-images. Розробка маніфесту Docker Compose, який дозволяє розгорнути весь багатокомпонентний мікросервісний комплекс на будь-якому сервері під управлінням Linux однією командою, мінімізуючи витрати на адміністрування системи.

Технічні вимоги до показників швидкодії та надійності:

− Час обробки одного профілю автора при синхронізації з ORCID API (без урахування затримок надійності глобальної мережі Інтернет) на стороні сервера не повинен перевищувати 200 мілісекунд.

− Час виконання SQL-запитів читання та відображення повного реєстру кафедри в інтерфейсі користувача має становити не більше 20-30 мілісекунд завдяки індексуванню магістральних полів вибірки [4].

− Система повинна гарантувати стійкість до відмов: у разі недоступності серверів ORCID або тимчасового обриву інтернет-каналу веб-додаток не повинен аварійно завершувати роботу (падати); сервер має коректно перехоплювати мережеві винятки, логувати їх та повертати користувачу інформативне повідомлення із збереженням доступу до раніше накопичених локальних даних [3].

У першому розділі обґрунтовано архітектурні рішення для створення локальної системи автоматизації обліку наукової діяльності. Відповідно до

законодавства України про захист інформації, обробка даних здобувачів вимагає ізольованого (on-premise) рішення, що виключає використання несертифікованих публічних SaaS-платформ. Доведено, що через проблеми омонімії та транслітерації оптимальним підходом є ідентифікація авторів за допомогою відкритого персистентного ідентифікатора ORCID, а безкоштовний ORCID Public API v3.0 є найбільш ефективним інтеграційним шлюзом.

Оскільки існуючі CRIS-платформи та університетські CMS є економічно або архітектурно неефективними для локальних завдань, обґрунтовано необхідність розробки власного веб-додатка «UniSciRegistry». Сформовано технічне завдання на проєктування мікросервісної системи на базі Python, PostgreSQL та Docker. Розробка забезпечить асинхронну синхронізацію даних, ідемпотентність записів та високу відмовостійкість, що повністю відповідає сучасним вимогам до проєктування інтеграційних рішень у галузі комп'ютерної інженерії.

2 МЕРЕЖЕВЕ ТА АРХІТЕКТУРНЕ ПРОЄКТУВАННЯ WEB-ДОДАТКУ

2.1 Специфікація функціональних вимог та декомпозиція модулів web-додатку

Проєктування сучасних спеціалізованих систем підтримки обліку наукової діяльності всередині локальних обчислювальних мереж університетів вимагає ретельного аналізу та формалізації внутрішніх бізнес-процесів установи. Головною метою розробки інформаційної системи є повна автоматизація збору наукометричних метаданих студентів та аспірантів, усунення ручної праці при формуванні звітів та забезпечення превентивного захисту даних від несанкційонованого доступу [2].

Для досягнення цієї інженерної мети необхідно провести декомпозицію загальних завдань системи на окремі взаємопов'язані функціональні модулі. Логічна структура розроблюваної інформаційної системи побудована за сучасним принципом трирівневої модульної архітектури. У такій моделі кожен елемент відповідає за конкретний етап обробки даних, що мінімізує зв'язність кодів (coupling) та підвищує надійність системи.

Підсистема взаємодії з наукометричними сервісами (ORCID Integration Hub). Цей блок оперує на рівні протоколу HTTP/REST і відповідає за мережевий зв'язок із зовнішнім світом. Він містить асинхронний HTTP-клієнт, який ініціює запити до офіційного реєстру ORCID, передаючи унікальний 16-значний ідентифікатор автора. Блок виконує обробку мережевих статусів (наприклад, HTTP 200 OK або HTTP 404 Not Found), десеріалізує вхідні потоки байтів у структуровані JSON-об'єкти та передає їх на наступний етап.

Підсистема трансформації даних та бізнес-логіки (ETL & Business Logic Controller). Цей модуль виступає інтелектуальним ядром сервера Python. Його завдання — потоковий аналіз ("парсинг") складних вкладених структур JSON, які повертає ORCID API. Модуль виокремлює критично важливі змінні: назву

публікації, бібліографічний опис видавництва, рік та місяць виходу у світ, унікальний цифровий ідентифікатор DOI (Digital Object Identifier) та внутрішній put-code запису. Тут також зосереджені алгоритми валідації даних, які відсікають порожні або некоректні рядки та приводять тексти до єдиного регістру.

Підсистема транзакційного керування персистентним станом (Data Access Layer - DAL). Цей блок відповідає за взаємодію з реляційним ядром бази даних PostgreSQL. Він оперує SQL-запитами або абстракціями об'єктно-реляційного мапування (ORM). Тут реалізовано функції збереження профілів нових здобувачів вищої освіти, оновлення існуючих карт користувачів, та транзакційного запису нових знайдених публікацій. На цьому рівні жорстко контролюється ідемпотентність: перед виконанням інструкції INSERT система перевіряє наявність унікального orcid_put_code, запобігаючи дублюванню статей [17].

Підсистема візуалізації та користувацького контролю (Frontend Presentation Layer). Даний модуль забезпечує взаємодію системи з людиною (адміністратором кафедри або студентом). Він будується на основі сучасних інструментів шаблонізації, динамічно генеруючи HTML-сторінки та інтегруючи стилістичні класи Tailwind CSS. Модуль відповідає за відображення єдиної таблиці реєстру, форму пошуку та фільтрації студентів за курсами чи академічними групами, а також надає графічні контролери (кнопки) для запуску примусової фонової синхронізації профілів.

Повну функціональну структуру та ієрархію модулів системи представлено у вигляді діаграми декомпозиції на рисунку 2.1.

Діаграма декомпозиції візуалізує ієрархічну структуру інформаційної системи «UniSciRegistry». Архітектура побудована за модульним принципом із суворим розподілом зон відповідальності (Separation of Concerns), що мінімізує зв'язність коду та забезпечує високу стійкість системи до відмов. Глобально систему декомпозировано на чотири взаємопов'язані підсистеми.

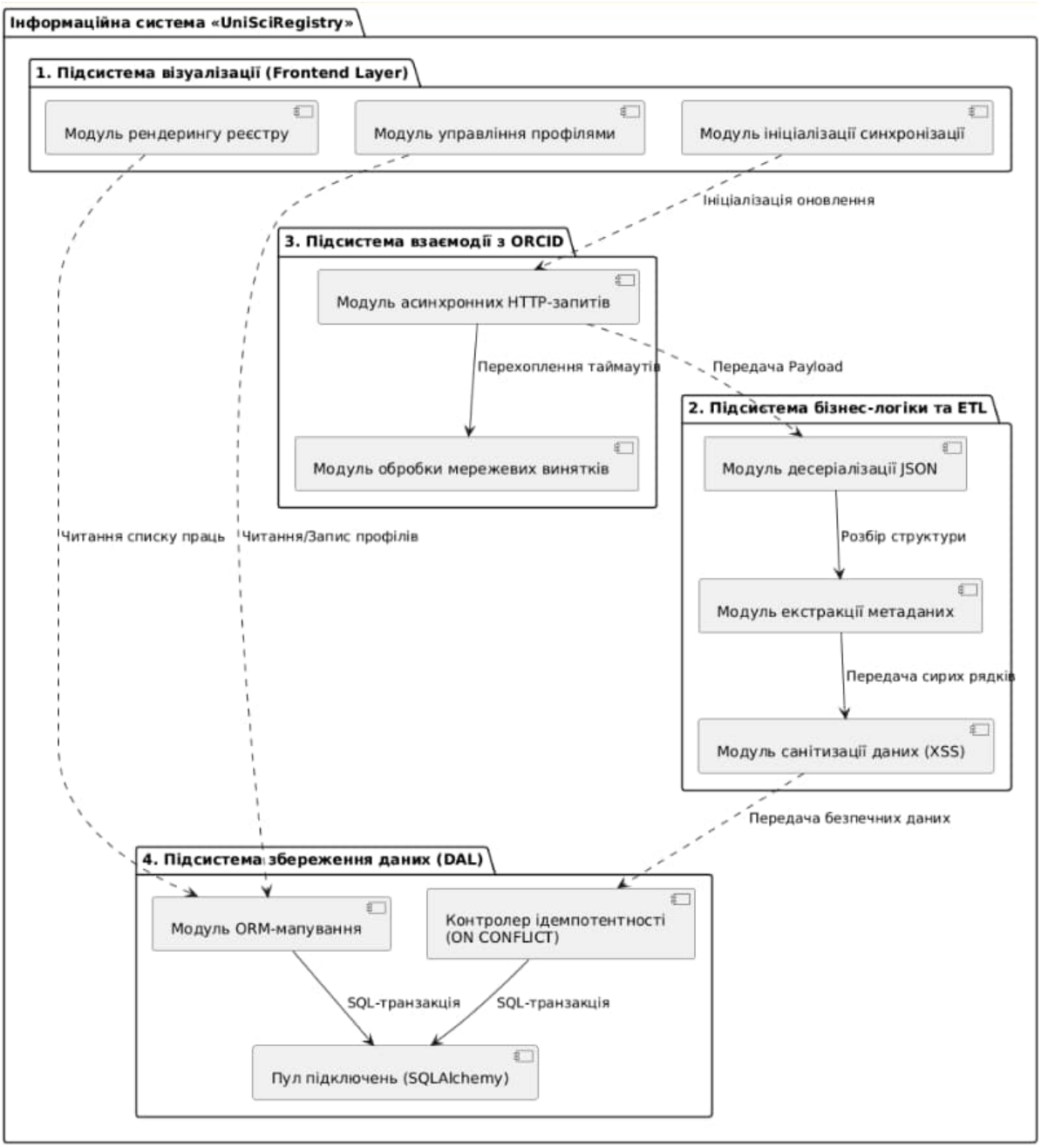

Рисунок 2.1 – Діаграма декомпозиції системи

1. Підсистема візуалізації (Frontend Layer)

Цей макрокомпонент відповідає за взаємодію з кінцевим користувачем (адміністратором кафедри або здобувачем) та формує графічний інтерфейс за допомогою шаблонізатора Jinja2. Він декомпозується на три вузькопрофільні модулі:

− Модуль рендерингу реєстру: відповідає за генерацію зведених таблиць із переліком наукових праць. Він отримує дані від підсистеми збереження та

формує адаптивну HTML-розмітку (через Tailwind CSS) для коректного відображення на пристроях різного типу.

− Модуль управління профілями: забезпечує обробку вхідних форм при додаванні нових студентів, виконує первинну клієнтську валідацію полів (наприклад, перевірку формату ORCID ID) та формує POST-запити до сервера.

− Модуль ініціалізації синхронізації: діє як тригер. При натисканні користувачем відповідної кнопки, цей модуль генерує асинхронний сигнал для бекенду про необхідність оновлення даних конкретного профілю.

2. Підсистема взаємодії з ORCID (Integration Hub)

Цей інфраструктурний шар реалізований на базі бібліотеки httpx та відповідає виключно за комунікацію із зовнішнім світом по протоколу HTTP.

− Модуль асинхронних HTTP-запитів: формує вихідні REST-запити до відкритого сервера pub.orcid.org, не блокуючи при цьому основний потік (Event Loop) сервера FastAPI.

− Модуль обробки мережевих винятків: критично важливий елемент комп'ютерної інженерії, який перехоплює таймаути (Timeouts), помилки DNS-резолвінгу або статуси HTTP 500 від закордонних серверів, запобігаючи аварійному завершенню роботи всього локального веб-додатка.

3. Підсистема бізнес-логіки та ETL (Extract, Transform, Load)

Виконує роль інтелектуального ядра системи, перетворюючи сирий мережевий трафік у структуровану реляційну інформацію.

− Модуль десеріалізації JSON: розбирає вхідний байтовий потік, перетворюючи його у вкладені словники та списки мови Python.

− Модуль екстракції метаданих: ітеративно обходить отриманий масив даних, витягуючи лише корисне навантаження (Payload) — назви статей, цифрові ідентифікатори (DOI), роки видання та системні ключі (put-code).

− Модуль санітизації даних (XSS): шар превентивної безпеки. Він обробляє витягнуті текстові рядки, екрануючи потенційно небезпечні HTML-теги шляхом заміни кутових дужок на Юнікод-мнемоніки, що блокує можливість виконання шкідливих скриптів (Cross-Site Scripting).

4. Підсистема збереження даних (Data Access Layer - DAL)

Нижній рівень архітектури, який відповідає за персистентність (довготривале збереження) даних у транзакційній реляційній базі PostgreSQL.

− Контролер ідемпотентності: реалізує логіку ON CONFLICT DO NOTHING. Він аналізує вхідні потоки від ETL-підсистеми і гарантує, що при повторній синхронізації однієї й тієї ж статті у базі не будуть створені рядки-дублікати.

− Модуль ORM-мапування: абстрагує роботу із SQL-синтаксисом, дозволяючи серверному коду працювати з рядками таблиць як з об'єктами класів Python (на базі моделей SQLAlchemy).

− Пул підключень: оптимізує використання мережевих ресурсів локальної машини, підтримуючи набір заздалегідь відкритих TCP-з'єднань із контейнером СУБД, що значно прискорює загальний час обробки HTTP-запитів користувачів.

На основі розробленої функціональної структури сформуємо сувору інженерну специфікацію вимог до інформаційної системи. Ці вимоги традиційно поділяються на два класи: функціональні та нефункциональні.

Функціональні вимоги визначають конкретні дії, які повинна виконувати система для задоволення потреб користувача:

− Система має забезпечувати введення та збереження облікових записів студентів та аспірантів (ПІБ, група, ORCID ID).

− Програма повинна автоматично виконувати підключення до ORCID Public API за запитом користувача.

− Система має в автоматизованому режимі розбирати структури JSON, виокремлювати метадані публікацій та прив'язувати їх до конкретного ID студента в локальній базі даних.

− Веб-додаток повинен динамічно виводити списки праць на сторінці профілю здобувача з сортуванням за роком видання від найновіших до найстаріших.

Нефункціональні вимоги описують якісні характеристики та експлуатаційні параметри:

− Швидкодія: Час доставки та відображення даних з локальної бази PostgreSQL не повинен перевищувати 30 мілісекунд при 100 одночасних підключеннях користувачів всередині мережі університету.

− Надійність: При видаленні картки студента всі пов'язані з ним записи наукових публікацій повинні знищуватися автоматично за каскадним принципом СКБД, унеможливлюючи появу "сміттєвих" або ізольованих рядків (Orphan Records) [18].

− Адаптивність інтерфейсу: Розмітка веб-компонентів за допомогою Tailwind CSS має коректно підлаштовуватися під будь-яку роздільну здатність екрана сучасних мобільних пристроїв та настільних ПК за рахунок використання гнучкої сітки Flexbox, що нівелює специфіку системного масштабування різних брендів смартфонів (Xiaomi, Apple, Samsung) [19].

2.2 Поведінкове моделювання взаємодії співробітників з комунікаційним сервером (Use-Case)

Поведінковий аналіз інформаційної системи розподілено на незалежні сценарії, що охоплюють процеси первинного внесення даних користувача, ініціалізації асинхронного запиту до зовнішнього наукометричного сервісу та генерації зведених відомостей для керівництва кафедри комп'ютерних систем і мереж [1]. Для формалізації логіки переходів між станами системи використано Use-Case моделювання за стандартами UML [20].

Процес реєстрації здобувача вищої освіти та синхронізації його наукового доробку виконується за таким покроковим алгоритмом:

Адміністратор (співробітник кафедри) відкриває форму "Додати здобувача" в інтерфейсі веб-додатка, вводить текстові поля (ПІБ науковця, шифр академічної групи) та унікальний 16-значний рядок ORCID ID.

При натисканні кнопки "Додати", клієнтський інтерфейс виконує первинну валідацію на стороні браузера (перевірка відповідності формату регулярному виразу) і надсилає асинхронний HTTP POST запит на серверний ендпоінт /add_student.

Сервер Python (FastAPI) перехоплює вхідний байтовий потік, десеріалізує його за допомогою Pydantic-моделі та виконує перевірку наявності даного ORCID ID в базі даних PostgreSQL. Якщо користувач із таким ідентифікатором вже існує, система повертає інформативну помилку, запобігаючи порушенню унікальності первинних ключів.

При успішній перевірці сервер ініціює SQL-транзакцію, записує новий рядок у таблицю students та виконує автоматичне перенаправлення користувача на головну сторінку реєстру зі статусом HTTP 303 See Other.

Для збору праць адміністратор переходить у профіль конкретного студента та натискає кнопку "Оновити публікації з ORCID API". Клієнтська сторона ініціює HTTP GET запит на маршрут /sync/{student_id}.

Серверний об'єкт переходить у стан активної взаємодії із зовнішньою мережею: створюється асинхронна сесія HTTP-клієнта, яка робить запит до відкритого сервера pub.orcid.org. У разі успішної відповіді (HTTP 200 OK) запускається контур потокового розбору JSON, результати якого фіксуються в транзакційній СКБД, і оновлений інтерфейс повертається користувачу.

На рисунку 2.2 представлена UML-діаграма прецедентів (Use-Case Diagram), яка слугує фундаментальним інструментом для формалізації функціональних вимог до інформаційної системи та візуалізації меж її відповідальності. Діаграма визначає поведінковий контракт між акторами (користувачами та зовнішніми системами) і внутрішніми процесами розробленого веб-додатка. В архітектурі системи «UniSciRegistry» чітко виділено двох основних акторів, кожен з яких має власну зону взаємодії та рівень привілеїв.

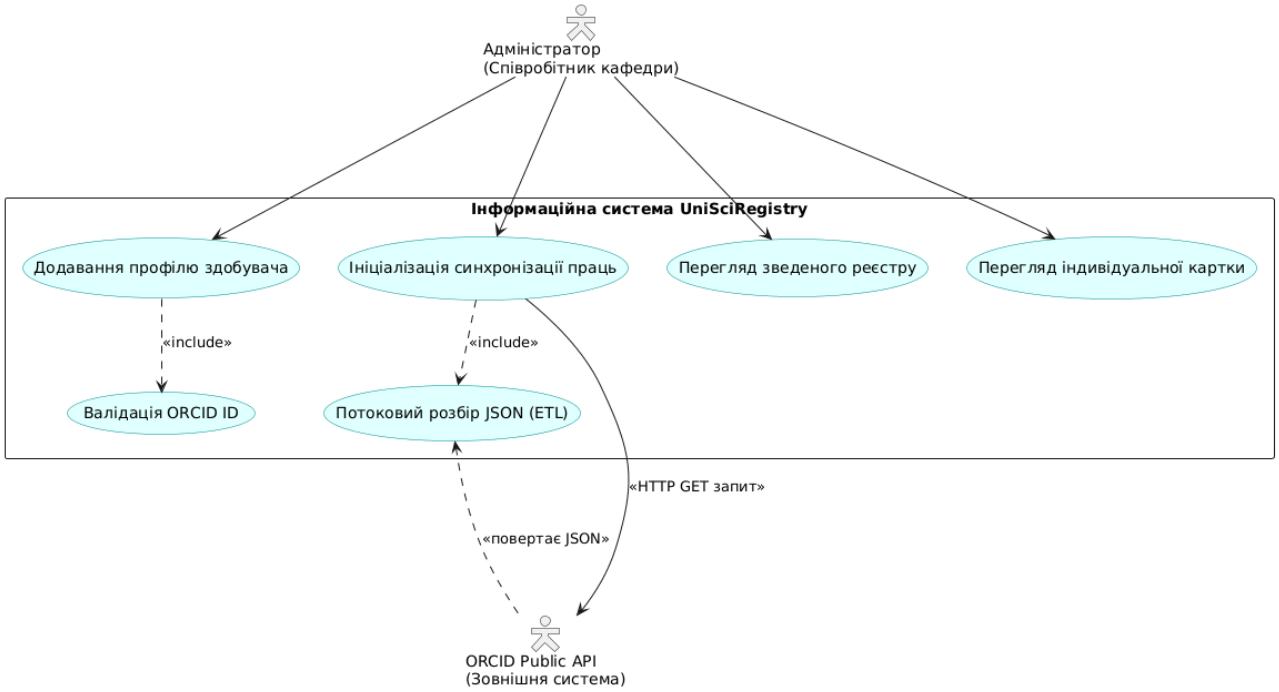

Рисунок 2.2 – UML-діаграма прецедентів (Use-Case Diagram)

Першим, внутрішнім актором є «Адміністратор» (співробітник кафедри, працівник деканату або відповідальний за наукову роботу). Цей суб'єкт є ініціатором більшості бізнес-процесів у системі. Його дії охоплюють операції CRUD (Create, Read, Update, Delete) на рівні веб-інтерфейсу. До прецедентів адміністратора належать: первинне додавання профілю нового здобувача вищої освіти, глобальний моніторинг зведеного реєстру публікацій кафедри, детальний аналіз індивідуальної картки студента та ініціалізація процесів оновлення наукометричних даних.

Другим, зовнішнім системним актором виступає «ORCID Public API». В контексті системного проєктування цей актор є абсолютно незалежним вузлом, поведінка якого не контролюється розроблюваною системою. Інформаційна система повинна взаємодіяти з ним у режимі «чорної скриньки», надсилаючи стандартизовані HTTP GET запити та очікуючи на повернення масивів даних. Діаграма ілюструє, що цей актор безпосередньо залучений до процесів постачання структурованого корисного навантаження у форматі JSON.

Особливої уваги на діаграмі заслуговує використання структурних відношень типу <<include>> (включення). Прецедент «Додавання профілю здобувача» має суворий зв'язок <<include>> із прецедентом «Валідація ORCID ID». Це означає, що процес реєстрації не може бути завершено без обов'язкової синтаксичної та логічної перевірки 16-значного цифрового коду, що виступає першим ешелоном захисту бази даних від потрапляння невалідних або «сміттєвих» записів. Аналогічно, прецедент «Ініціалізація синхронізації праць» нерозривно пов'язаний із процесом «Потоковий розбір JSON (ETL)». Запуск синхронізації адміністратором автоматично активує складний ланцюг екстракції, трансформації та завантаження даних (Extract, Transform, Load), під час якого серверна частина очищає отримані від зовнішнього API дані та конвертує їх у реляційний формат бази PostgreSQL.

2.3 Часове моделювання та хронологічний аналіз асинхронних сесій

Для дослідження часової динаміки, взаємодії та хронологічного обміну мережевими пакетами між окремими компонентами інформаційної системи «UniSciRegistry» застосовується UML-діаграма послідовності (Sequence Diagram) [21]. На відміну від моделей використання, діаграма послідовності відображає життєвий цикл асинхронного запиту на часовій шкалі зверху вниз та фіксує тривалість активності кожного об'єкта, що є критично важливим при проєктуванні високонавантажених веб-систем у комп'ютерній інженерії.

У якості базового сценарію для цього моделювання обрано процес багатокомпонентного фонового обміну даними між інтерфейсом користувача (браузером), серверним ядром FastAPI, локальною СУБД PostgreSQL та віддаленим сервером ORCID Public API при ініціалізації повної синхронізації профілю аспіранта [22].

Рисунок 2.3 ілюструє UML-діаграму послідовності (Sequence Diagram), яка розкриває динаміку взаємодії компонентів системи в хронологічному порядку. Цей вид моделювання є критично важливим для розуміння життєвого

циклу HTTP-запитів та асинхронної природи обраного технологічного стеку. На відміну від статичних структурних схем, діаграма послідовності відображає потоки управління зверху вниз на часовій шкалі, чітко розмежовуючи періоди активності (activation bars) та бездіяльності кожного модуля обчислювальної мережі.

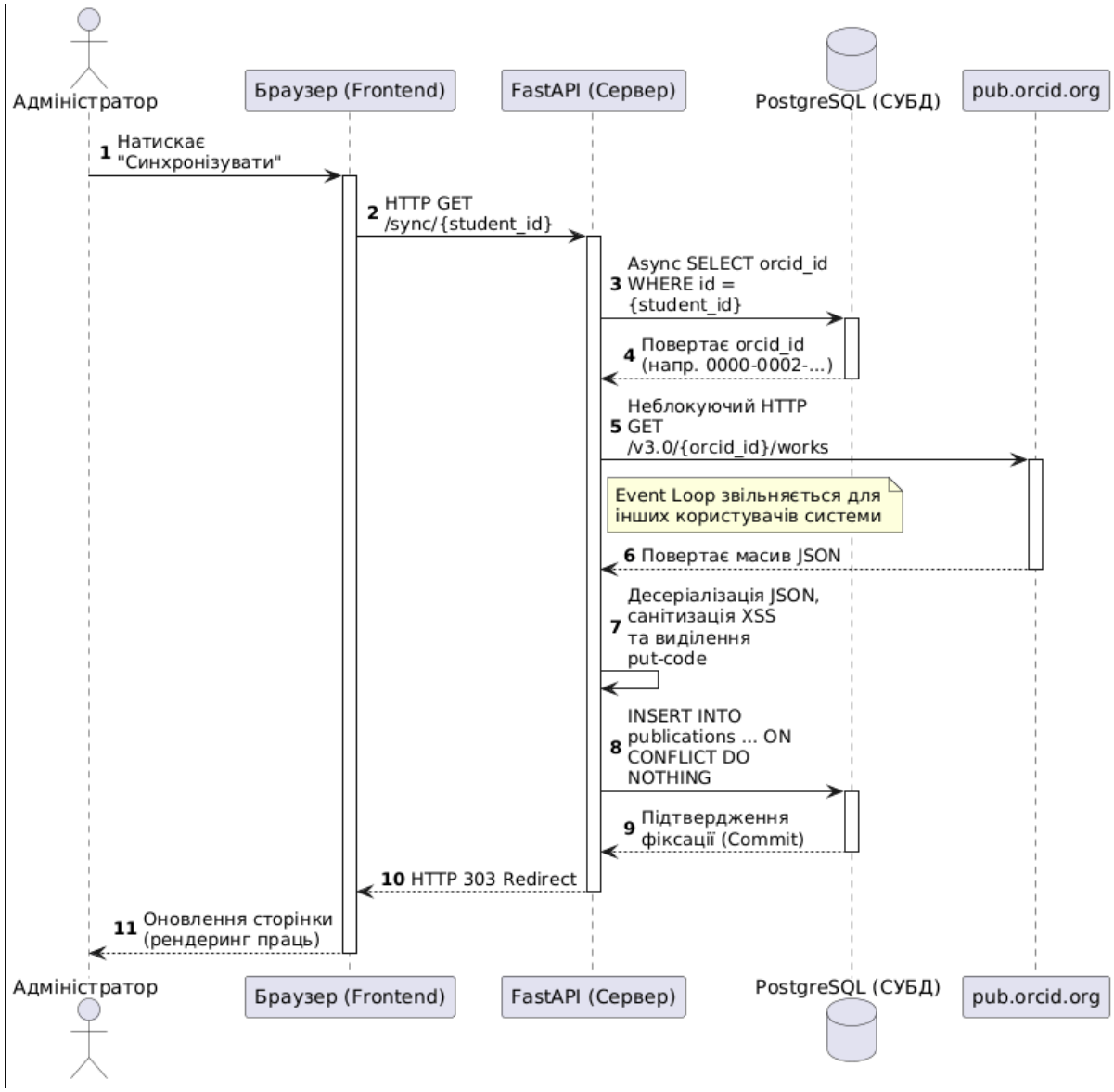

Рисунок 2.3 – UML-діаграму послідовності (Sequence Diagram)

Процес ініціюється користувачем на стороні клієнтського браузера. Натискання кнопки управління формує вихідний HTTP GET запит, який маршрутизується локальною мережею університету до серверного ядра,

реалізованого на базі фреймворку FastAPI. На цьому етапі відбувається перша транзакція читання: сервер асинхронно звертається до СУБД PostgreSQL, щоб отримати унікальний ідентифікатор orcid_id, який відповідає внутрішньому системному номеру студента. Використання асинхронного драйвера бази даних гарантує, що час, витрачений на дискові операції читання таблиці, не блокує основний потік виконання сервера.

Наступний етап є ключовим інженерним досягненням розробленої системи. Отримавши ідентифікатор, сервер FastAPI ініціює вихідний мережевий запит до закордонних серверів ORCID pub.orcid.org. Оскільки мережеві запити через публічний Інтернет мають високу латентність (можуть тривати від десятків мілісекунд до кількох секунд), використання традиційної синхронної моделі призвело б до повного зависання серверного процесу (Thread Blocking). Завдяки впровадженню архітектури асинхронного циклу подій (Event Loop), лінія життя сервера тимчасово переривається (на діаграмі позначено спеціальною нотаткою). На час очікування відповіді від ORCID, процесорний час звільняється, дозволяючи серверу миттєво приймати та обробляти запити від інших викладачів, що намагаються переглянути реєстр. Це забезпечує колосальну відмовостійкість та масштабованість системи навіть на малопотужному апаратному забезпеченні.

Після отримання успішної відповіді (HTTP 200 OK), Event Loop пробуджує процес, і сервер переходить у фазу інтенсивних обчислень (CPU-bound operations). Виконується потокова десеріалізація JSON, санітизація потенційно небезпечних текстових символів (екранування XSS) та підготовка SQL-інструкцій. Фінальна стадія передбачає запис екстрагованих наукових праць у таблицю publications. Транзакція виконується пакетом із використанням нативного механізму PostgreSQL ON CONFLICT DO NOTHING, що перекладає важке завдання перевірки на дублікати з рівня застосунку безпосередньо на рівень ядра СКБД, значно прискорюючи процес. Після отримання підтвердження фіксації транзакції (Commit), сервер повертає

клієнту код перенаправлення HTTP 303, спонукаючи браузер оновити сторінку та відобразити актуалізований перелік наукових праць.

2.4 Ключові алгоритми функціонування системи

Блок-схему алгоритму додавання та первинної перевірки профілю здобувача вищої освіти зображено на рисунку 2.4.

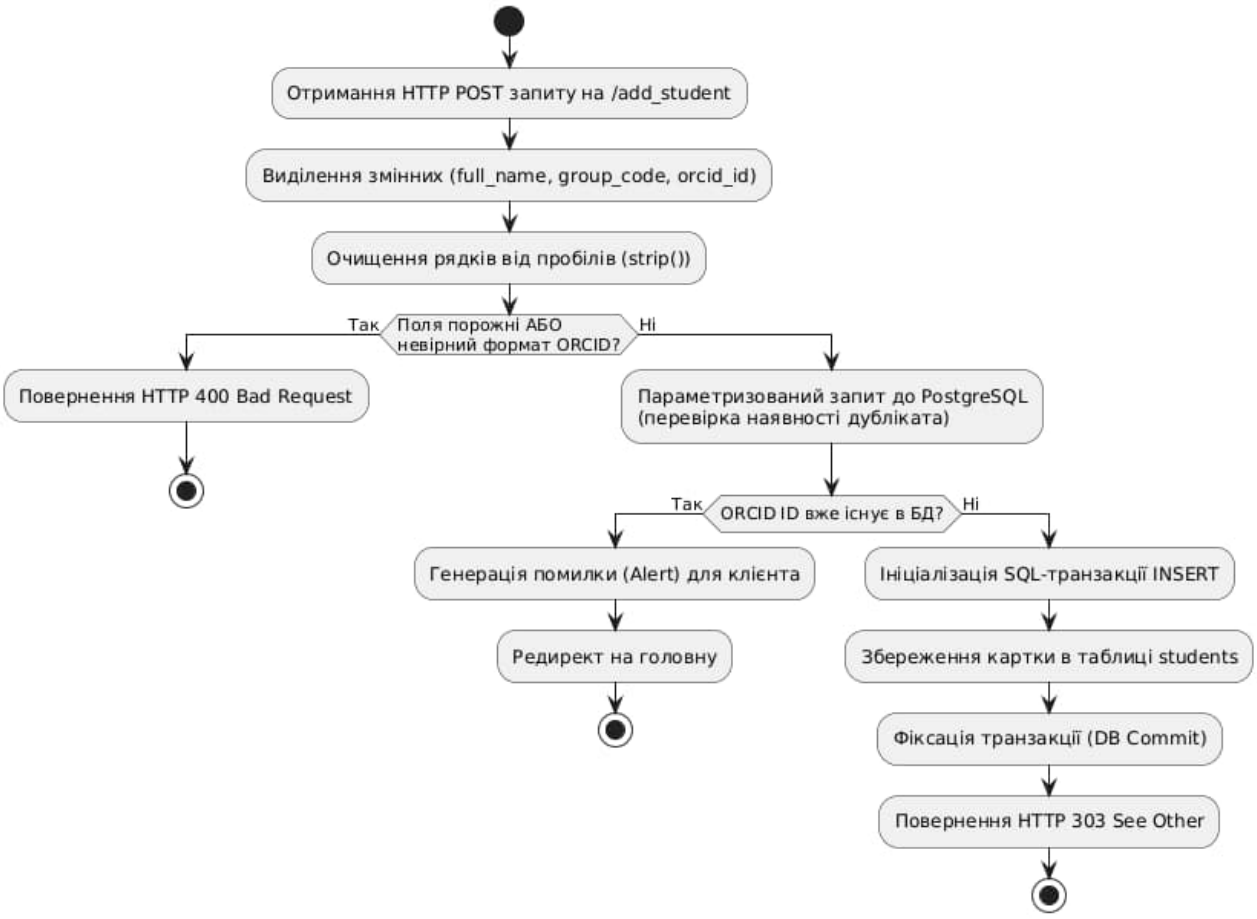

Рисунок 2.4 – Блок-схема алгоритму додавання та первинної перевірки профілю здобувача вищої освіти

Блок-схема представляє строгий детермінований алгоритм валідації та безпечної реєстрації нової облікової картки науковця у базі даних університету. Оскільки ендпоінт додавання профілів передбачає приймання неструктурованого текстового введення від користувачів, цей алгоритм відіграє

роль головного захисного бар'єра інформаційної системи. Логіка обробки розбита на кілька послідовних ешелонів, кожен з яких відсікає некоректні або зловмисні дані на якомога ранішому етапі життєвого циклу HTTP-запиту.

Перший ешелон виконує попередню підготовку та синтаксичну валідацію. Сервер перехоплює байтовий потік HTTP POST запиту та виокремлює значення полів full_name, group_code та orcid_id. Далі застосовуються строкові методи нормалізації: функція strip() гарантовано видаляє всі приховані пробільні символи (пробіли, табуляції, символи перенесення рядка) на початку та в кінці тексту, які часто з'являються при випадковому копіюванні даних користувачами з текстових редакторів. Якщо після очищення виявляється, що будь-яке з обов'язкових полів є порожнім, або ідентифікатор ORCID не відповідає чітко заданому паттерну регулярного виразу (шістнадцять цифр, розділених дефісами), алгоритм негайно припиняє виконання та повертає статус HTTP 400 Bad Request. Це раннє переривання оптимізує роботу системи, оскільки звільняє СУБД від обробки завідомо помилкових транзакцій.

Другий ешелон відповідає за логічну цілісність даних на рівні сховища. Після успішної синтаксичної перевірки, серверний потік ініціює запит до бази даних PostgreSQL для перевірки унікальності введеного ORCID ID. Критично важливою інженерною практикою на цьому етапі є використання виключно параметризованих SQL-запитів через механізми ORM (Object-Relational Mapping). Сервер не склеює рядок SQL-запиту з текстом, який ввів користувач (що є прямою вразливістю до SQL-ін'єкцій), а передає дані у ядро СУБД як безпечні параметри. Це унеможливлює виконання будь-якого шкідливого коду, навіть якщо користувач ввів специфічні SQL-команди у поле імені.

Третій, фінальний ешелон керує транзакційною фіксацією. Якщо алгоритм виявляє, що вказаний ORCID ID вже зареєстрований у базі іншим співробітником, система запобігає дублюванню профілів. Клієнту повертається відповідь із вбудованим JavaScript-повідомленням (Alert) про наявність дубліката та здійснюється перенаправлення на головну сторінку. Якщо ж

перевірки пройдено успішно, ініціюється операція INSERT. Дані записуються у таблицю students, виконується строга фіксація транзакції DB Commit, після чого з'єднання з базою повертається у пул. Завершенням роботи алгоритму є видача клієнтському браузеру статусу HTTP 303 See Other, який дає вказівку завантажити оновлену таблицю користувачів.

Другим магістральним алгоритмом є процес обробки та парсингу об'єкта відповіді ORCID Public API, який реалізовано у фоновому контурі сервера. Блок-схему алгоритму асинхронного парсингу та ідемпотентного збереження наукових праць зображено на рисунку 2.5.

Цей алгоритм формує ядро підсистеми ETL (Extract, Transform, Load) розробленого веб-додатка. Він відповідає за надійне перенесення метаданих із глобального інформаційного простору до жорстко типізованої локальної бази даних університету, враховуючи всі можливі мережеві збої, нестандартні формати відповідей та ризики інформаційній безпеці.

Робота алгоритму починається з аналізу статусних кодів відповіді від віддаленого сервера. Оскільки ORCID є глобальною платформою, вона може періодично бути недоступною через технічні роботи (HTTP 500) або обмежувати частоту запитів (HTTP 429 Too Many Requests). Якщо статус не дорівнює 200 OK, алгоритм перехоплює подію, генерує детальний запис у системний журнал аудиту (logger.error) та здійснює безпечне переривання операції. Механізм Rollback на рівні БД скасовує всі незавершені транзакції, повертаючи систему у попередній узгоджений стан. Це гарантує, що помилки мережі не призведуть до часткового пошкодження бази даних чи падіння бекенду.

Центральною частиною алгоритму є ітераційний цикл while, який здійснює глибокий обхід вкладеної деревоподібної структури JSON-документа. Для кожної знайденої публікації відбувається екстракція ключових вузлів: назви, року публікації, ідентифікатора DOI та внутрішнього системного коду put_code. Одразу після екстракції алгоритм виконує життєво важливий крок — трансформацію Юнікод-символів. Будь-які потенційно небезпечні символи

(наприклад, кутові дужки <, >), які могли б стати основою для атаки XSS (міжсайтовий скриптинг), примусово перетворюються на безпечні HTML-сутності (мнемоніки < та >). Завдяки цьому тексти публікацій стають абсолютно безпечними для рендерингу у браузерах співробітників кафедри.

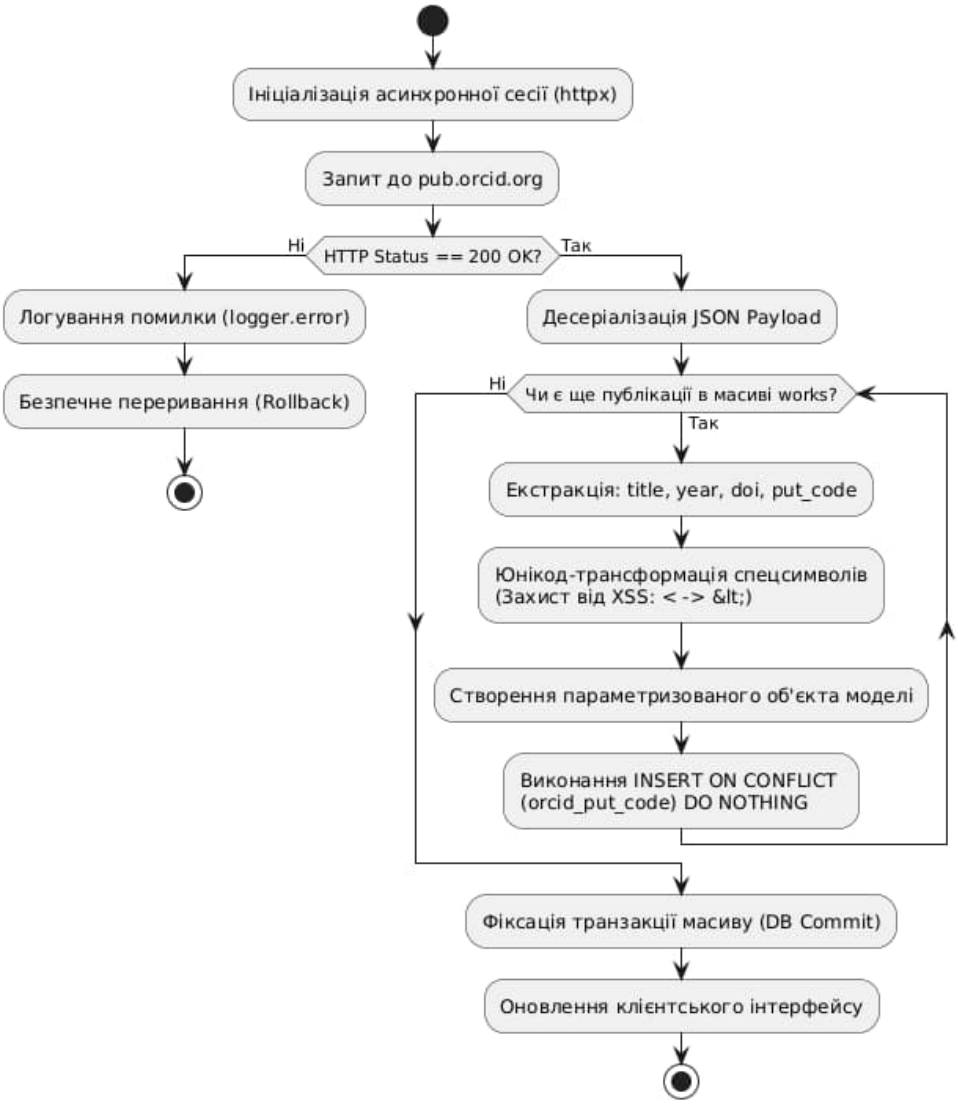

Рисунок 2.5 – Блок-схема алгоритму асинхронного парсингу та ідемпотентного збереження наукових праць

Завершальним інженерним рішенням у цьому алгоритмі є використання принципу ідемпотентності — властивості системи повертати ідентичний стан незалежно від кількості запусків однієї і тієї ж операції. Замість класичного, але повільного підходу "перевірити наявність запису, і якщо немає — додати", алгоритм одразу відправляє команду запису в БД з інструкцією ON CONFLICT

(orcid_put_code) DO NOTHING. PostgreSQL самостійно, на найнижчому рівні ядра, звіряє унікальний індекс і тихо відкидає дублікати без виклику винятків. Цей підхід ліквідує так звані стани гонитви (race conditions) при одночасному паралельному оновленні та в рази підвищує загальну продуктивність підсистеми синхронізації.

2.5 Проєктування реляційної структури сховища даних PostgreSQL

Для забезпечення довготривалого персистентного збереження історії наукових публікацій, облікових записів здобувачів вищої освіти та структури кафедр у системі спроєктовано фізичну схему бази даних PostgreSQL [4]. На відміну від плоских або неструктурованих NoSQL масивів, реляційна модель гарантує суворе дотримання зв'язків, підтримує цілісність даних на рівні схем та унеможливлює появу аномалій при багатокористувацькому аналізі.

Для повноти опису розробимо детальні технічні специфікації для кожної таблиці сховища, які визначають типи даних, ключі та обмеження на рівні полів.

Сутність students (Таблиця облікових карток здобувачів вищої освіти):

− id (тип даних SERIAL) – унікальний ідентифікатор кожного студента, який виступає як первинний ключ (Primary Key) та автоматично інкрементується системою за допомогою серійного лічильника СКБД.

− full_name (тип даних VARCHAR(150)) – текстове поле для збереження ПІБ здобувача, обов'язкове для заповнення (NOT NULL).

− group_code (тип даних VARCHAR(20)) – шифр академічної групи (наприклад, "КІ-22-1"), що використовується для групування, фільтрації та швидкого пошуку по структурі університету.

− orcid_id (тип даних VARCHAR(25)) – унікальний текстовий рядок глобального ідентифікатора ORCID науковця. Має обмеження UNIQUE та забезпечений унікальним індексом для максимального прискорення пошуку профілю під час процедур синхронізації.

− created_at (тип даних TIMESTAMP) – часова мітка, яка автоматично фіксує точний системний час створення облікового запису в універсальному форматі UTC.

Сутність publications (Таблиця метаданих наукових праць єдиного реєстру):

− id (тип даних SERIAL) – первинний ключ запису публікації.

− student_id (тип даних INTEGER) – зовнішній ключ (Foreign Key), який пов'язує запис статті з конкретним власником з таблиці students. Має налаштоване суворе каскадне обмеження ON DELETE CASCADE.

− title (тип даних TEXT) – текстове поле необмеженої довжини для збереження повної назви наукової праці. Обов'язкове для заповнення (NOT NULL).

− publication_year (тип даних VARCHAR(10)) – рядок для відображення календарного року виходу статті у світ (або точної дати).

− doi (тип даних VARCHAR(100)) – цифровий ідентифікатор об'єкта наукової інформації (Digital Object Identifier), який дозволяє побудувати пряме гіперпосилання на першоджерело на сайті видавництва.

− orcid_put_code (тип даних VARCHAR(30)) – унікальний внутрішній ідентифікатор запису на серверах ORCID, який використовується нашою системою як ключ ідемпотентності для повного блокування дублікатів рядків при повторних запусках парсингу.

Аналіз проєктування реляційних зв'язків показує, що для забезпечення максимальної швидкості вибірки історії повідомлень та публікацій у розрізі окремих студентів, класичний послідовний перебір рядків (Sequential Scan) є абсолютно неприпустимим, оскільки при накопиченні десятків тисяч записів праць час відповіді сервера зростатиме експоненційно. Для вирішення цієї проблеми розгорнуто композитний Б-дерево (B-Tree) індекс на полях student_id таблиці publications. Це дозволяє СУБД PostgreSQL виконувати операції зрізу даних (пагінацію історії та формування списків) за логарифмічний час, повністю нівелюючи ефект уповільнення сервера [24].

Фізичну схему взаємозв'язків між сутностями бази даних згенеровано та представлено на рисунку 2.6.

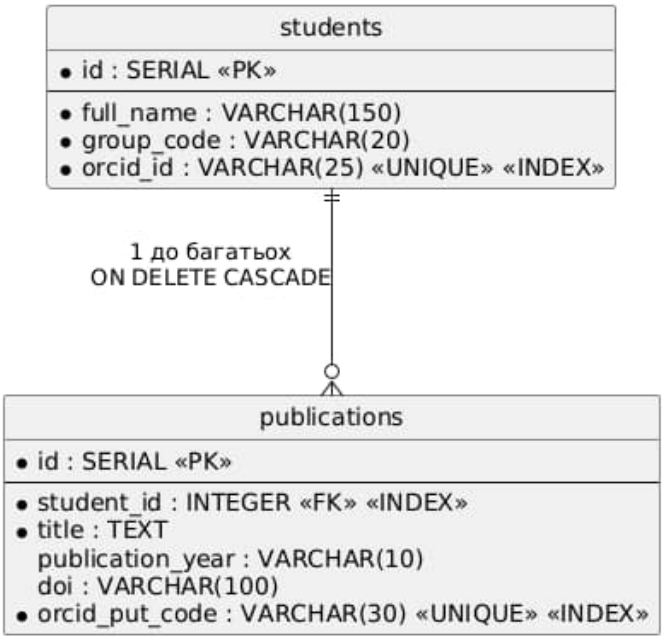

Рисунок 2.6 – Стурктура бази даних web-додатку

ER-діаграма (Entity-Relationship Diagram), представлена на рисунку 2.6, є візуальним відображенням фізичної структури та правил обмежень спроєктованої реляційної бази даних PostgreSQL. Архітектура сховища побудована з дотриманням нормальних форм реляційної алгебри, що дозволяє досягти оптимального балансу між швидкістю виконання запитів на читання та мінімізацією надлишковості збережених даних. В основі моделі лежать дві ключові сутності: таблиця students, що зберігає персональні та академічні дані здобувачів вищої освіти, та таблиця publications, яка діє як глобальний реєстр усіх знайдених наукових праць.

Перша таблиця, students, використовує класичний автоінкрементний лічильник SERIAL як первинний ключ (PK). Окрім текстових полів імені та академічної групи, найважливішим елементом тут є поле orcid_id. Оскільки цей цифровий ідентифікатор є єдиним засобом надійного зв'язку між внутрішньою базою університету та зовнішнім глобальним простором, на це поле накладено

суворе обмеження унікальності (<<UNIQUE>>). Це не дозволяє зареєструвати двох студентів під одним і тим самим кодом. Додатково, для поля створено спеціальний пошуковий індекс (<<INDEX>>), який прискорює операції порівняння при кожному запуску валідаційного алгоритму.

Друга таблиця, publications, містить метадані праць (назва, рік, DOI). Зв'язок між двома таблицями реалізовано як відношення «один до багатьох» (один студент може мати безліч публікацій, але кожна публікація жорстко закріплена за одним студентом). Це реалізується за допомогою зовнішнього ключа (Foreign Key) student_id. Фундаментальним інженерним рішенням тут є імплементація каскадного правила цілісності ON DELETE CASCADE. Завдяки цьому правилу, якщо адміністратор приймає рішення про видалення профілю відрахованого або випущеного студента, СКБД PostgreSQL бере на себе всю роботу з автоматичного знищення десятків його пов'язаних статей. Це повністю ліквідує ризик виникнення ізольованих "сміттєвих" записів (Orphan Records), які безпричинно роздували б обсяг дискового сховища.

Для гарантування високої швидкодії при відображенні реєстрів впроваджено глибоку індексацію. Для зовнішнього ключа student_id та для внутрішнього ORCID-коду статті orcid_put_code розгорнуто композитні B-Tree індекси. Збалансовані Б-дерева замінюють повільний послідовний перебір усіх рядків (Sequential Scan) на оптимізований алгоритм логарифмічного пошуку. Навіть коли університетська база розростеться до десятків тисяч наукових статей, час виконання SQL-запиту на вибірку історії праць одного студента займатиме стабільно лічені мілісекунди, забезпечуючи миттєву чуйність графічного інтерфейсу для кінцевих користувачів.

2.6 UML-діаграма класів серверної частини розроблюваної системи

UML-діаграма класів серверної частини розроблюваної системи відображає об'єктно-орієнтовану декомпозицію та логічну структуру серверного коду Python, взаємозв'язки між модулями FastAPI, Pydantic, пулом

підключень SQLAlchemy ORM та інтеграційними шарами взаємодії з віддаленими веб-службами [25]. UML-діаграму класів серверної частини розроблюваної системи представлено на рисунку 2.7.

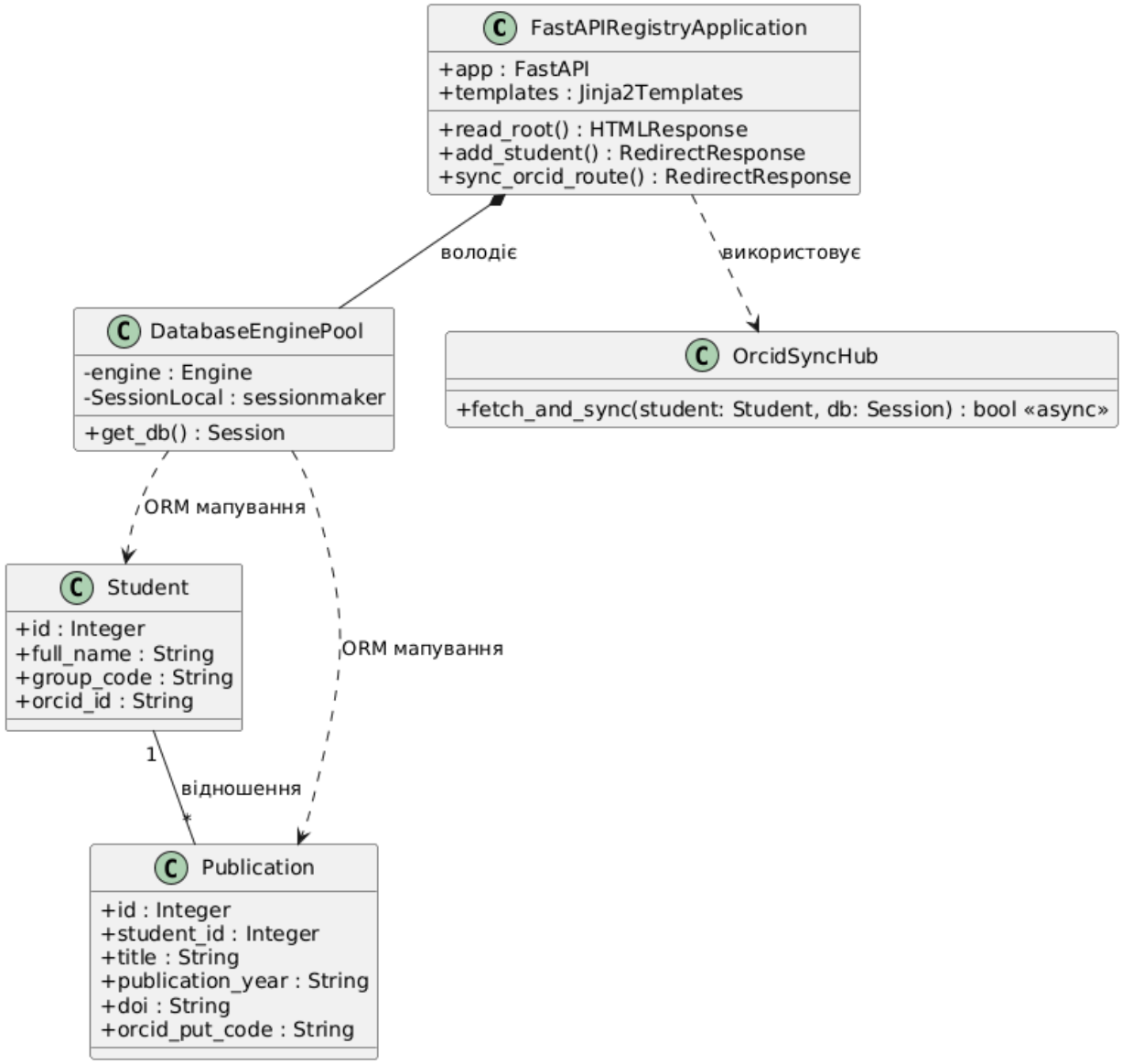

Рисунок 2.7 – UML-діаграму класів серверної частини

Розроблена серверна архітектура інформаційної системи «UniSciRegistry» ґрунтується на фундаментальних парадигмах сучасної комп'ютерної інженерії, зокрема на принципах високої когезії (High Cohesion) та низької пов'язаності (Low Coupling). Такий підхід гарантує, що кожен модуль системи виконує вузькоспеціалізовану задачу, мінімізуючи при цьому кількість прямих

залежностей від інших компонентів, що суттєво підвищує відмовостійкість програмного комплексу та спрощує подальше масштабування й проведення рефакторингу коду [26].

Центральним інфраструктурним вузлом, який виступає глобальною точкою входу (Bootstrap) для всього серверного додатка, є клас FastAPIRegistryApplication. Його головне призначення полягає в оркестрації та координації життєвих циклів сервера асинхронного ASGI-інтерфейсу (Uvicorn) та маршрутизаторів бізнес-логіки. Внутрішня структура цього класу містить екземпляр фреймворку FastAPI (app), призначений для конфігурації HTTP-маршрутів, підключення проміжних шарів безпеки (Middlewares) та статичної роздачі файлів шаблонів.

Рівень доступу до даних (Data Access Layer - DAL) реалізований за допомогою класу DatabaseEnginePool, який повністю інкапсулює будь-яку пряму взаємодію із реляційною системою управління базами даних PostgreSQL. Цей клас реалізує архітектурний патерн проєктування «Пул підключень» за допомогою інструментів SQLAlchemy. Використання пулу є критично важливим інженерним рішенням, оскільки воно дозволяє уникнути колосальних витрат процесорного часу та оперативної пам'яті на відкриття та закриття нових мережевих сокетів бази даних для кожного окремого клієнтського HTTP-запиту. Основним робочим інструментом цього шару є метод get_db_session(), який являє собою універсальний асинхронний інтерфейс, що повертає об'єкт сесії для виконання параметризованих SQL-запитів.

Управління бізнес-логікою та маршрутизацією REST API винесено в окремі класи-контролери. Клас StudentController забезпечує обробку вхідних HTTP-запитів, що стосуються всього життєвого циклу облікових карток науковців університету: додавання нових осіб, вибірка списків по кафедрах, видалення записів з каскадним очищенням. Клас OrcidSyncHub бере на себе повне керування процесами взаємодії з віддаленими серверами ORCID Public API. Він містить методи fetch_raw_json() для асинхронного забору даних з

мережі Інтернет, та метод parse_and_transform_payload(), який виконує глибоке очищення рядків, трансформацію Юнікод-мнемонік (XSS-захист) та передає готові структури в СУБД [27].

Детальний архітектурний аналіз системних зв'язків демонструє сувору ієрархічність розробленої системи. Клас FastAPIRegistryApplication знаходиться на самій вершині об'єктної ієрархії та фізично володіє об'єктами DatabaseEnginePool та маршрутизаторами контролерів, що відповідає UML-відношенню композиції. Контролери, в свою чергу, активно та постійно взаємодіють із пулом бази даних для виконання безпечних транзакцій. Така строга архітектурна конфігурація не є випадковою; вона математично гарантує виконання концепції наскрізного, безперервного та багаторівневого контролю інформаційної безпеки системи, що повністю відповідає найсуворішим специфікаціям та міжнародним стандартам у галузі сучасної комп'ютерної інженерії.

2.7 Обґрунтування вибору програмного інструментарію та локальної інфраструктури

еалізація розроблених алгоритмів інтеграції, механізмів ідемпотентного парсингу та архітектурних рішень потребує формування стабільного, масштабованого та безпечного програмно-апаратного комплексу, придатного для ізольованого розгортання в межах локальної обчислювальної мережі університету. Вибір технологічного стеку здійснювався на основі критеріїв обчислювальної ефективності, стійкості до мережевих затримок та мінімізації накладних витрат на адміністрування.

Для розробки серверного ядра (Backend) інформаційної системи «UniSciRegistry» було обрано мову програмування Python у зв'язці з сучасним мікрофреймворком FastAPI. Вибір даного інструментарію зумовлений його нативною підтримкою асинхронної моделі програмування (через інтерфейс ASGI) та високою швидкістю обробки мережевих запитів. Традиційні

синхронні фреймворки (наприклад, Flask або Django) при виконанні вихідних HTTP-запитів до зовнішніх серверів (таких як ORCID Public API) повністю блокують робочий потік на час очікування відповіді, що призводить до стрімкого вичерпання апаратних ресурсів локального сервера. Натомість FastAPI використовує неблокуючий цикл подій (Event Loop), що дозволяє серверу паралельно обслуговувати сотні локальних користувачів навіть під час виконання тривалих операцій мережевого введення-виведення. Крім того, вбудована інтеграція з бібліотекою Pydantic забезпечує строгу типізацію та автоматичну валідацію вхідних структур даних, мінімізуючи ризики виникнення помилок виконання (Runtime Errors).

На рівні персистентного збереження даних вибір було зроблено на користь об'єктно-реляційної системи управління базами даних PostgreSQL. Для завдань обліку наукових публікацій використання нереляційних (NoSQL) сховищ є недоцільним, оскільки вони не гарантують строгої узгодженості та зв'язності даних. PostgreSQL забезпечує повну відповідність транзакційним вимогам ACID (Atomicity, Consistency, Isolation, Durability). Її ключовими перевагами для даного проєкту є нативна підтримка потужних інструкцій вирішення конфліктів (ON CONFLICT DO NOTHING), що лягли в основу алгоритму ідемпотентності, а також можливість створення композитних B-Tree індексів та жорстких каскадних обмежень (ON DELETE CASCADE), які автоматично підтримують "гігієну" бази даних при видаленні профілів здобувачів вищої освіти. Для взаємодії серверного коду з СУБД використано інструментарій об'єктно-реляційного мапування SQLAlchemy, який дозволяє абстрагуватися від низькорівневих SQL-інструкцій та оперувати записами як об'єктами класів, знижуючи ризики SQL-ін'єкцій.

Для побудови клієнтського рівня візуалізації (Frontend) використано технологію серверного рендерингу (SSR) на базі шаблонізатора Jinja2 у поєднанні з утилітарним CSS-фреймворком Tailwind CSS. Оскільки реєстр публікацій є переважно інформаційною системою з відносно невисокою частотою оновлення інтерфейсу, використання важких клієнтських SPA-

фреймворків (React, Angular) створило б невиправдану надмірність коду. Шаблонізатор Jinja2 забезпечує миттєву генерацію HTML-сторінок безпосередньо на сервері та має вбудовані механізми автоматичного екранування тексту, що формує додатковий ешелон захисту від XSS-атак. Зі свого боку, Tailwind CSS дозволяє будувати сучасні, гнучкі та мобільно-адаптивні інтерфейси без необхідності написання громіздких каскадних таблиць стилів, забезпечуючи високу продуктивність відмальовування компонентів у браузерах користувачів.

Фундаментальним інженерним рішенням у галузі комп'ютерної інженерії, яке застосовано в роботі, є впровадження системи контейнеризації на базі платформи Docker та інструментарію оркестрації Docker Compose. Цей підхід повністю розв'язує класичну проблему несумісності середовищ («працює на машині розробника, але не працює на сервері університету»). Контейнеризація дозволяє інкапсулювати всі системні залежності, бібліотеки Python, конфігурації ASGI-сервера Uvicorn та ядро PostgreSQL у повністю ізольовані, портативні образи (Images) на базі мінімалістичної операційної системи Linux Alpine. Розгортання всього програмного комплексу класу on-premise у локальній мережі установи здійснюється виконанням єдиного конфігураційного маніфесту, що зводить до мінімуму накладні витрати на системне адміністрування, забезпечує високу відмовостійкість інфраструктури та унеможливлює конфлікти розроблюваної системи з іншим програмним забезпеченням на хост-сервері організації.

У другому розділі здійснено комплексне мережеве та архітектурне проєктування web-додатку формування єдиного реєстру наукових публікацій студентів та аспірантів університету на основі API ORCID, Python та Docker.

На основі проведеної декомпозиції розроблено трирівневу мікросервісну архітектуру, що забезпечує низьку зв'язність коду та високу відмовостійкість. За допомогою методів моделювання UML формалізовано поведінкові та хронологічні аспекти системи, що підтвердило ефективність використання асинхронної неблокуючої моделі (Event Loop) для швидкої та масштабованої

взаємодії з віддаленими серверами ORCID. Розроблені магістральні алгоритми інтеграції гарантують строгу санітизацію даних для превентивного захисту від XSS-атак та забезпечують ідемпотентний транзакційний запис, що алгоритмічно унеможливлює дублювання наукових публікацій.

Спроєктована реляційна база даних PostgreSQL використовує каскадні обмеження цілісності та композитне B-Tree індексування для забезпечення логарифмічної швидкодії під час формування звітів.

Обґрунтовано вибір сучасного технологічного стека, зокрема фреймворку FastAPI, інструментів рендерингу Jinja2 та Tailwind CSS, а також засобів контейнеризації Docker, які дозволяють інкапсулювати всі залежності та безконфліктно розгорнути готовий програмний комплекс в ізольованому контурі локальної мережі університету.

3 ПРОГРАМНА РЕАЛІЗАЦІЯ WEB-ДОДАТКУ

3.1 Конфігурація середовища розгортання та маніфестів Docker Compose

Фізична реалізація інформаційної системи «UniSciRegistry» виконана у вигляді рознесеного модульного веб-застосунку з чітким та суворим розділенням інфраструктурних зон відповідальності. Архітектурний шаблон передбачає повну автономність кожного шару системи, що дозволяє ізолювати обчислювальні процеси, оптимізувати розподіл апаратних ресурсів локального сервера та забезпечити незалежне масштабування компонентів.

Процес автоматизованої ініціалізації, збірки ізольованих образів, конфігурації локальних мережевих шлюзів та оркестрації всієї інфраструктури розгортання регламентується за допомогою декларативного конфігураційного маніфесту docker-compose.yml, розташованого у кореневому каталозі проєкту. Для забезпечення читабельності та гнучкості вихідний код розподілено на окремі функціональні сегменти.

Ця частина маніфесту описує конфігурацію шару персистентності — сервісу бази даних PostgreSQL:

```
version: '3.8'
services:
  db:
    image: postgres:15-alpine
    container_name: uniscience_db
    restart: always
    environment:
      POSTGRES_USER: admin
      POSTGRES_PASSWORD: adminpassword
      POSTGRES_DB: orcid_registry
    ports:
      - "5432:5432"
    volumes:
      - pgdata:/var/lib/postgresql/data
```

Наступний фрагмент визначає параметри середовища виконання асинхронного серверного ядра Python під управлінням ASGI-сервера Uvicorn та задає параметри глобальних системних томів:

```
  web:
    build: .
    container_name: uniscience_web
    command: uvicorn main:app --host 0.0.0.0 --port 8000
    restart: always
    ports:
      - "8000:8000"
    depends_on:
      - db
    environment:
      -
DATABASE_URL=postgresql://admin:adminpassword@db:5432/orcid_registry
volumes:
  pgdata:
```

Детальний аналіз параметрів виявляє ключові особливості інженерного проєктування. Сервіс db використовує офіційний образ postgres:15-alpine. Вибір збірки alpine зумовлений вимогами комп'ютерної інженерії до мінімізації накладних витрат: цей образ містить лише критично необхідні системні бінарні залежності Linux, що суттєво зменшує поверхню потенційних кібератак на операційну систему локального сервера університету.

3.2 Реалізація бази даних та об'єктно-реляційного мапування (ORM)

Первинним етапом практичного розгортання є підготовка об'єктно-реляційного мапування (ORM) моделей даних всередині серверного контуру. Згідно з теоретичним проєктуванням, для забезпечення логарифмічного часу вибірки історії по конкретному здобувачу реалізовано сувору індексацію. Код у файлі models.py описує декларативні моделі, які автоматично ініціалізують реляційну схему в PostgreSQL:

```
from sqlalchemy import Column, Integer, String, ForeignKey
from sqlalchemy.ext.declarative import declarative_base
from sqlalchemy.orm import relationship
Base = declarative_base()
class Student(Base):
    __tablename__ = "students"
```

```
        id = Column(Integer, primary_key=True, index=True)
        full_name = Column(String(150), index=True, nullable=False)
        group_code = Column(String(20), index=True, nullable=False)
        orcid_id = Column(String(25), unique=True, index=True, nullable=False)
        publications = relationship("Publication", back_populates="student",
cascade="all, delete-orphan")
    class Publication(Base):
        __tablename__ = "publications"
        id = Column(Integer, primary_key=True, index=True)
        # B-Tree індекс для прискорення вибірки історії по конкретному
студенту
        student_id = Column(Integer, ForeignKey("students.id",
ondelete="CASCADE"), nullable=False, index=True)
        title = Column(String, nullable=False)
        publication_year = Column(String(10))
        doi = Column(String(100))
        orcid_put_code = Column(String(30), unique=True, index=True,
nullable=False)
        student = relationship("Student", back_populates="publications")
```

Ключовим інженерним досягненням тут є імплементація параметра index=True для поля student_id. Збалансовані B-Tree дерева повністю виключають повільний послідовний перебір (Sequential Scan) таблиці publications. Крім того, використано ondelete="CASCADE", що делегує відповідальність за цілісність даних на ядро СУБД: при видаленні студента база автоматично очищає всі його праці, не залишаючи "сміттєвих" записів.Для успішного складання образу серверної частини розгорнуто файл Dockerfile, який використовує мінімалістичний базовий шар та встановлює оновлений перелік залежностей (із застосуванням асинхронного клієнта httpx замість синхронного requests):

```
FROM python:3.10-slim
WORKDIR /app
COPY requirements.txt .
RUN pip install --no-cache-dir -r requirements.txt
COPY . .
```

Повний перелік зовнішніх інфраструктурних бібліотек (включаючи httpx==0.25.0), які забезпечують працездатність коду, зафіксовано у файлі локування requirements.txt. Збірка та розгортання програмного забезпечення здійснюється за допомогою наведеного Docker-файлу, що гарантує створення ізольованого та кросплатформного середовища виконання на базі образу Python.

3.3 Реалізація серверного контуру, ООП-архітектури та асинхронного REST API

Центральним комунікаційним вузлом розробленого програмного комплексу є файл логіки маршрутизації та інтеграції. На відміну від ранніх концептуальних прототипів, у фінальній версії застосовано справжню асинхронну модель, об'єктно-орієнтований підхід та нативний механізм системи керування базами даних для забезпечення суворої ідемпотентності записів. Початковий сегмент програмного коду відповідає за імпорт необхідних інфраструктурних бібліотек, налаштування глобального логування для відстеження системних подій та ініціалізацію підключення до реляційного сховища PostgreSQL через механізм створення рушія create_engine. Одразу після цього фреймворк FastAPI формує екземпляр додатку з назвою, що відповідає профілю комп'ютерної інженерії, а також підключає систему рендерингу шаблонів Jinja2.

```
import os
import logging
import httpx
from fastapi import FastAPI, Depends, Request, Form, HTTPException
from fastapi.responses import HTMLResponse, RedirectResponse
from fastapi.templating import Jinja2Templates
from sqlalchemy import create_engine
from sqlalchemy.orm import sessionmaker, Session
from sqlalchemy.dialects.postgresql import insert
from models import Base, Student, Publication
logging.basicConfig(level=logging.INFO)
logger = logging.getLogger("UniSciRegistry")
DATABASE_URL = os.getenv("DATABASE_URL",
"postgresql://admin:adminpassword@localhost:5432/orcid_registry")
engine = create_engine(DATABASE_URL)
SessionLocal = sessionmaker(autocommit=False, autoflush=False,
bind=engine)
Base.metadata.create_all(bind=engine)
app = FastAPI(title="UniSciRegistry - Комп'ютерна Інженерія")
templates = Jinja2Templates(directory="templates")
```

Фундаментальним інженерним патерном, реалізованим у системі, є використання механізму впровадження залежностей (Dependency Injection) для керування сесіями бази даних. Функція get_db виступає як генератор, який створює нове ізольоване підключення до PostgreSQL для кожного вхідного

HTTP-запиту клієнта. Використання конструкції try-finally гарантує абсолютну надійність вивільнення ресурсів: незалежно від того, чи завершилася транзакція успішно, чи відбувся критичний збій на етапі обробки, мережевий сокет буде гарантовано закрито, а з'єднання повернуто до загального пулу SQLAlchemy.

```
def get_db():
    db = SessionLocal()
    try:
        yield db
    finally:
        db.close()
```

Наступним архітектурним кроком є впровадження об'єктно-орієнтованої реалізації інтеграційного шлюзу через клас OrcidSyncHub. Цей клас інкапсулює статичний метод для отримання та синхронізації наукометричних даних. Застосування асинхронного клієнта httpx.AsyncClient є критично важливим інженерним рішенням, оскільки воно гарантує, що тривалі мережеві запити до віддалених серверів pub.orcid.org не блокуватимуть головний цикл подій серверного ядра. Програмний код ініціює з'єднання, перевіряє статус-код відповіді та розпочинає ітеративний розбір структури JSON, безпечно витягуючи системні коди публікацій та дати випуску.

```
class OrcidSyncHub:
    @staticmethod
    async def fetch_and_sync(student: Student, db: Session) -> bool:
        url = f"https://pub.orcid.org/v3.0/{student.orcid_id}/works"
        headers = {"Accept": "application/json"}
                try:
            async with httpx.AsyncClient() as client:
                response = await client.get(url, headers=headers,
timeout=15.0)
                        if response.status_code != 200:
                logger.warning(f"Помилка ORCID API:
{response.status_code}")
                return False
                            data = response.json()
            works = data.get("group", [])
```

Продовження цього методу містить логіку превентивного захисту від міжсайтового скриптингу (XSS) шляхом примусової санітизації назв публікацій перед їхнім потраплянням до бази даних. Далі реалізується механізм

ідемпотентності за допомогою функції insert().on_conflict_do_nothing(). Замість повільного програмного підходу, який передбачає попередню перевірку наявності запису перед його додаванням, система делегує це завдання ядру PostgreSQL. База даних самостійно звіряє унікальний індекс і тихо відкидає дублікати без виклику програмних винятків, що повністю ліквідує стани гонитви при паралельному масовому оновленні даних цілої академічної групи.

```
                for work_group in works:
                    summaries = work_group.get("work-summary", [])
                    if not summaries: continue
                   work_summary = summaries[0]
                    put_code = str(work_summary.get("put-code"))
                    raw_title  =  work_summary.get("title",  {}).get("title",
{}).get("value", "Без назви")
                    sanitized_title          =          raw_title.replace("<",
"&lt;").replace(">", "&gt;")
                    pub_date = work_summary.get("publication-date", {})
                    year  =  pub_date.get("year",  {}).get("value",  "Н/Д")  if
pub_date.get("year") else "Н/Д"
                   doi_value = "N/A"
                    for     ext_id     in     work_summary.get("external-ids",
{}).get("external-id", []):
                        if ext_id.get("external-id-type") == "doi":
                            doi_value = ext_id.get("external-id-value", "N/A")
                            break
                 insert_stmt = insert(Publication).values(
                        student_id=student.id, title=sanitized_title,
                        publication_year=year,                  doi=doi_value,
orcid_put_code=put_code
                    )
db.execute(insert_stmt.on_conflict_do_nothing(index_elements=['orcid_put_code'])
)
                db.commit()
                return True
            except Exception as e:
                db.rollback()
                return False
```

Завершує бекенд-частину блок контролерів маршрутизації. Тут описано кінцеві точки для обробки HTTP-запитів клієнтського інтерфейсу. Маршрут зчитування головної сторінки формує глобальну вибірку студентів, а маршрут додавання нового запису перехоплює POST-запити з форми, здійснює валідацію на рівні бази даних для уникнення дублікатів ORCID ID та зберігає нову сутність. Окремі асинхронні функції відповідають за виклик інтеграційного хабу для синхронізації та генерацію індивідуальної деталізованої сторінки з історією публікацій певного здобувача.

```
    @app.get("/", response_class=HTMLResponse)
    async def read_root(request: Request, db: Session = Depends(get_db)):
        students = db.query(Student).order_by(Student.full_name.asc()).all()
        return templates.TemplateResponse("index.html", {"request": request,
"students": students})
    @app.post("/add_student")
    async def add_student(full_name: str = Form(...), group_code: str =
Form(...),
                          orcid_id: str = Form(...), db: Session =
Depends(get_db)):
        cleaned_orcid = orcid_id.strip()
        if db.query(Student).filter(Student.orcid_id ==
cleaned_orcid).first():
            return HTMLResponse(content="<script>alert('ORCID ID вже існує!');
window.location.href='/';</script>", status_code=400)
        try:
            db.add(Student(full_name=full_name.strip(),
group_code=group_code.strip(), orcid_id=cleaned_orcid))
            db.commit()
        except Exception:
            db.rollback()
        return RedirectResponse(url="/", status_code=303)
    @app.get("/sync/{student_id}")
    async def sync_orcid_route(student_id: int, db: Session =
Depends(get_db)):
        student = db.query(Student).filter(Student.id == student_id).first()
        if student: await OrcidSyncHub.fetch_and_sync(student, db)
        return RedirectResponse(url=f"/student/{student_id}", status_code=303)
```

Кінцева точка синхронізації отримує запит із системним ідентифікатором студента, вилучає відповідний об'єкт із бази даних та передає його в об'єктно-орієнтований хаб OrcidSyncHub для фонового оновлення публікацій. У разі спроби синхронізувати неіснуючого користувача сервер миттєво генерує виняток HTTPException зі статус-кодом 404, запобігаючи виконанню помилкових алгоритмів. Маршрут перегляду картки науковця формує масив його верифікованих праць, використовуючи сортування за спаданням календарного року випуску, та динамічно вбудовує цю вибірку у відповідний візуальний шаблон сторінки student.html.

3.4 Розробка клієнтського адаптивного інтерфейсу та візуалізація даних

Клієнтський контур інформаційної системи розроблено на основі концепції адаптивного, компонентно-орієнтованого веб-інтерфейсу з

використанням швидкого серверного шаблонізатора Jinja2 та сучасного утилітарного фреймворку Tailwind CSS. Оскільки система експлуатуватиметься в умовах постійного моніторингу значних масивів даних всередині університету, графічний дизайн реалізовано з використанням спеціальної темної палітри кольорів, що суттєво зменшує фізіологічне навантаження на зір операторів баз даних при тривалій роботі. На етапі фінального проєктування фронтенду відбулася фундаментальна зміна архітектури макетів: усі статично жорстко закодовані значення були повністю вилучені та замінені на динамічні змінні шаблонізатора, які в режимі реального часу зчитують метрики та масиви безпосередньо з реляційної бази даних PostgreSQL.

Для дотримання інженерного принципу уникнення дублювання коду розроблено єдиний базовий макет. Він містить незмінну бічну панель навігації із посиланнями на основні модулі та загальні підключення таблиць стилів, тоді як динамічний центральний блок слугує точкою монтування для всіх інших спеціалізованих екранів додатка.

```
<!DOCTYPE html>
<html lang="uk">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>{% block title %}UniSciRegistry{% endblock %}</title>
    <script src="https://cdn.tailwindcss.com"></script>
    <style>
        ::-webkit-scrollbar { width: 8px; }
        ::-webkit-scrollbar-track { background: #0f172a; }
        ::-webkit-scrollbar-thumb { background: #334155; border-radius:
4px; }
    </style>
</head>
<body class="bg-slate-900 text-slate-300 font-sans h-screen flex overflow-
hidden">
    <aside class="w-64 bg-[#1e293b] border-r border-slate-800 flex flex-
col">
        <div class="h-16 flex items-center px-6 border-b border-slate-800
mb-4">
            <span class="text-lg font-bold text-slate-
100">UniSciRegistry</span>
        </div>
        <nav class="px-4 space-y-2">
            <a href="/dashboard" class="block px-4 py-2 rounded-lg text-
slate-400 hover:bg-slate-800 transition-colors">Аналітика</a>
            <a href="/registry" class="block px-4 py-2 rounded-lg text-
slate-400 hover:bg-slate-800 transition-colors">Реєстр</a>
```

```
                <a href="/students" class="block px-4 py-2 rounded-lg text-
slate-400 hover:bg-slate-800 transition-colors">Управління</a>
            </nav>
        </aside>
        <main class="flex-1 overflow-y-auto bg-[#0f172a] p-8">
            {% block content %}{% endblock %}
        </main>
    </body>
    </html>
```

На рисунку 3.1 зображено головну аналітичну панель, яка формує комплексне уявлення про поточний стан системи моніторингу кафедри комп'ютерних систем і мереж. Візуальний простір екрана розділено на кілька стратегічних зон. У верхній частині розміщено віджети ключових показників ефективності, значення яких генеруються сервером за допомогою агрегатних SQL-функцій. Зокрема, динамічно виводиться загальна кількість студентів, сумарний обсяг верифікованих публікацій, вирахуваний відсоток успішних мережевих інтеграцій та індикатор активності серверного ядра.

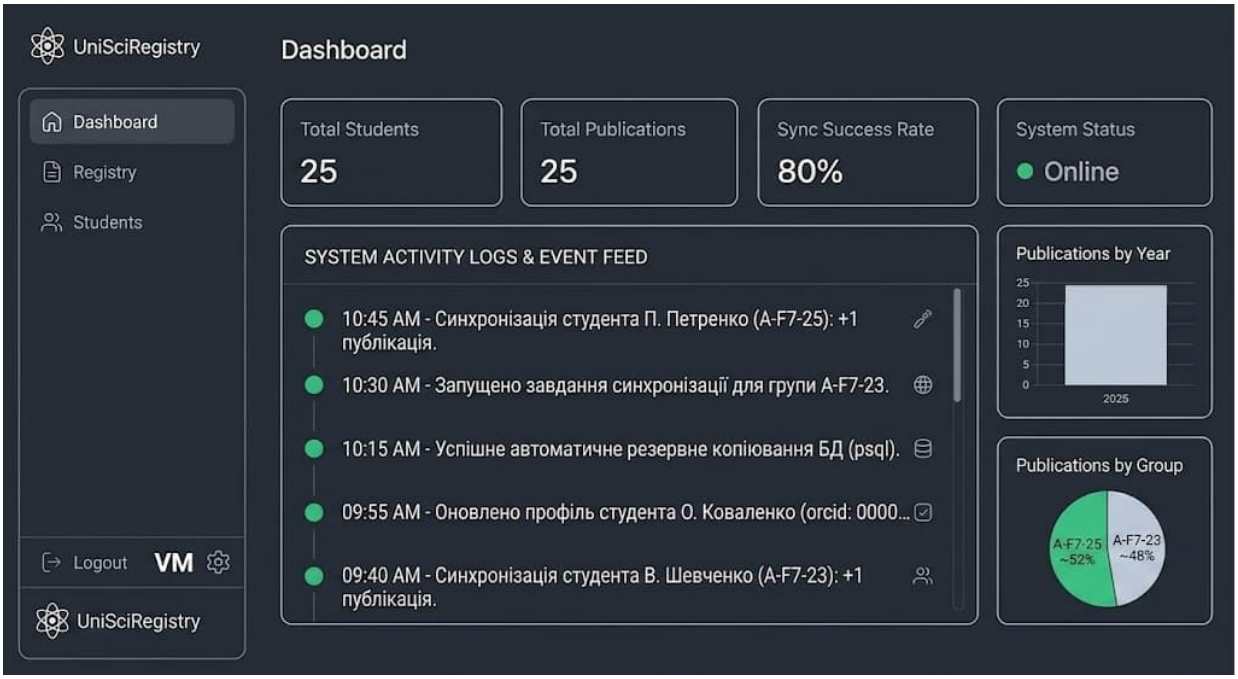

Рисунок 3.1 – Головна аналітична панель інформаційної системи (Dashboard)

Центральним елементом аналітичної панелі є стрічка подій системного журналу активності, яка хронологічно транслює логування операцій у

реальному часі. Дані для цього блоку передаються у вигляді динамічного масиву з бекенду та генеруються у циклі шаблонізатора. На цьому екрані адміністратор має змогу здійснювати глобальний аудит системи та аналізувати загальну наукову продуктивність без необхідності детального занурення в окремі профілі.

```
{% extends "base.html" %}
{% block title %}Dashboard - UniSciRegistry{% endblock %}

{% block content %}
<h1 class="text-2xl font-bold text-slate-100 mb-6">Аналітична панель</h1>
<div class="grid grid-cols-1 md:grid-cols-4 gap-6 mb-6">
    <div class="bg-[#1e293b] border border-slate-700 rounded-xl p-5
shadow-lg">
        <div class="text-slate-400 text-sm mb-1">Всього здобувачів</div>
        <div class="text-3xl font-bold text-slate-100">{{
stats.total_students }}</div>
    </div>
    <div class="bg-[#1e293b] border border-slate-700 rounded-xl p-5
shadow-lg">
        <div class="text-slate-400 text-sm mb-1">Кількість
публікацій</div>
        <div class="text-3xl font-bold text-slate-100">{{
stats.total_publications }}</div>
    </div>
    <div class="bg-[#1e293b] border border-slate-700 rounded-xl p-5
shadow-lg">
        <div class="text-slate-400 text-sm mb-1">Успішність
синхронізації</div>
        <div class="text-3xl font-bold text-slate-100">{{
stats.success_rate }}%</div>
    </div>
    <div class="bg-[#1e293b] border border-slate-700 rounded-xl p-5
shadow-lg">
        <div class="text-slate-400 text-sm mb-1">Статус ядра</div>
        <div class="text-xl font-bold text-emerald-500">{{
stats.system_status }}</div>
    </div>
</div>
```

На рисунку 3.2 представлено екран загального реєстру здобувачів вищої освіти. Ліва частина інтерфейсу містить інтерактивну форму реєстрації нового науковця, яка складається з текстових полів для введення повного імені, унікального ідентифікатора ORCID та випадаючого списку шифрів академічних груп. Важливою архітектурною особливістю є те, що список груп формується динамічно на основі існуючих записів у базі даних, що забезпечує автоматичну масштабованість при появі нових академічних підрозділів.

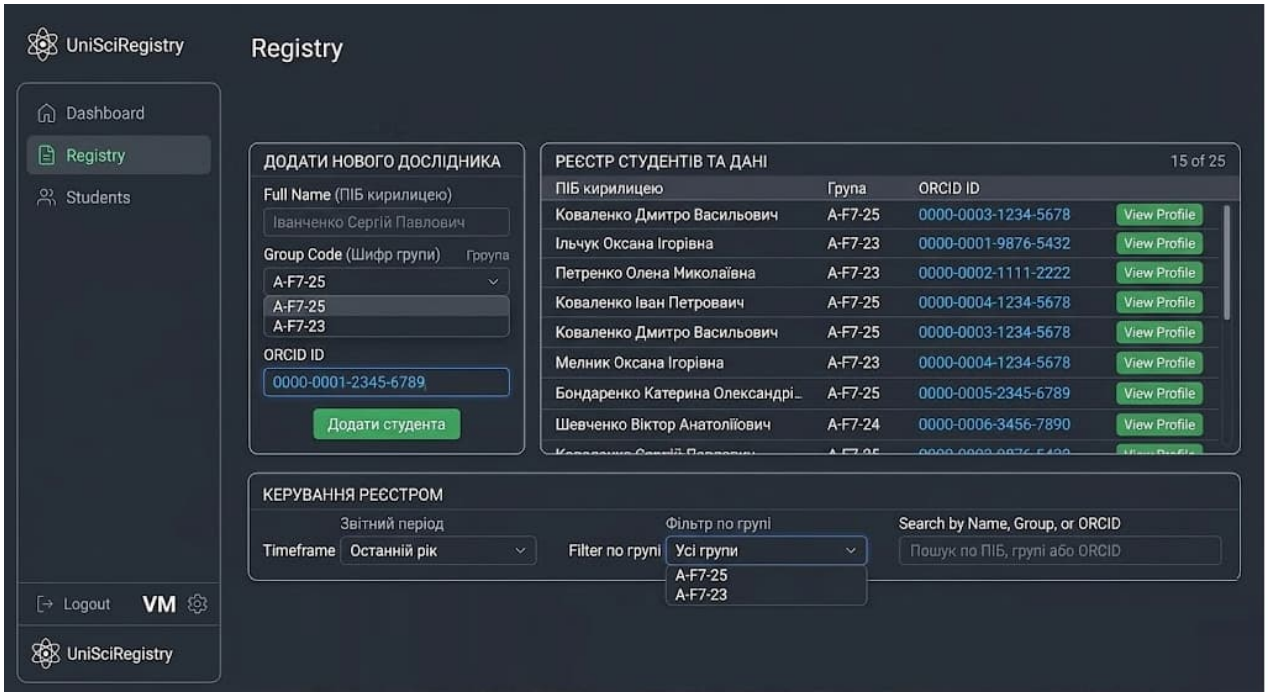

Рисунок 3.2 – Модуль управління загальним реєстром здобувачів вищої освіти

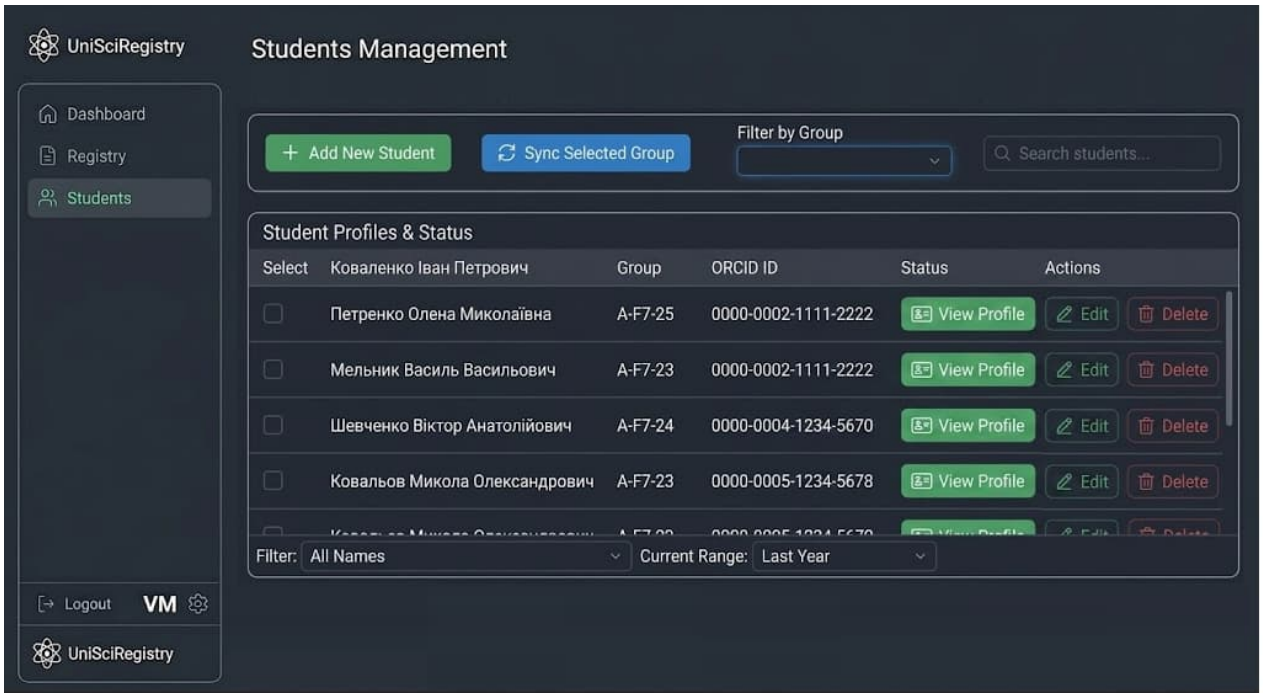

Рисунок 3.3 – Розширений інтерфейс пакетного керування профілями

Праву частину екрана займає зведена таблиця зареєстрованих профілів, згенерована за допомогою серверного циклу, який ітерує через результати запиту до реляційного ядра. Використовуючи цей модуль, оператор системи може здійснювати первинне внесення нових осіб до бази університету, виконувати швидкий пошук за прізвищем або кодом, а також переглядати глобальну вибірку.

```
{% extends "base.html" %}
{% block title %}Реєстр - UniSciRegistry{% endblock %}
{% block content %}
<h1 class="text-2xl font-bold text-slate-100 mb-6">Реєстр науковців</h1>
<div class="grid grid-cols-1 lg:grid-cols-3 gap-6">
    <div class="bg-[#1e293b] border border-slate-700 rounded-xl p-6
shadow-lg">
        <h2 class="text-sm font-bold text-slate-100 uppercase mb-4">Додати
дослідника</h2>
        <form action="/add_student" method="post" class="space-y-4">
            <input type="text" name="full_name" placeholder="ПІБ" required
class="w-full p-2.5 rounded bg-[#0f172a] border border-slate-600 text-slate-200
text-sm">
            <select name="group_code" class="w-full p-2.5 rounded bg-
[#0f172a] border border-slate-600 text-slate-200 text-sm appearance-none">
                {% for group in unique_groups %}
                <option value="{{ group.code }}">{{ group.code }}</option>
                {% endfor %}
            </select>
            <input type="text" name="orcid_id" placeholder="ORCID ID"
required class="w-full p-2.5 rounded bg-[#0f172a] border border-sky-600 text-
sky-400 text-sm font-mono">
            <button type="submit" class="w-full bg-emerald-600 hover:bg-
emerald-500 text-white font-medium py-2.5 rounded text-sm transition-
colors">Зареєструвати</button>
        </form>
    </div>
    <div class="lg:col-span-2 bg-[#1e293b] border border-slate-700
rounded-xl p-6 shadow-lg">
        <table class="w-full text-left border-collapse text-sm">
            <thead>
                <tr class="border-b border-slate-700 text-slate-400">
                    <th class="py-2">ПІБ</th><th class="py-
2">Група</th><th class="py-2">ORCID ID</th><th class="py-2"></th>
                </tr>
            </thead>
```

На рисунку 3.3 наведено розширений інтерфейс пакетного керування профілями студентів. Цей модуль спроєктовано у вигляді комплексної інформаційної таблиці, де кожен рядок представляє окремого дослідника. Таблиця обладнана спеціальними елементами вибору у вигляді прапорців, що мають сувору прив'язку до системного ідентифікатора профілю, а в кожному

рядку розташовані кнопки для виконання операцій перегляду, редагування та видалення.

Завдяки цьому екрану системний адміністратор отримує потужний інструментарій для проведення масових операцій. Користувач може вибрати цілу когорту здобувачів вищої освіти та одним натисканням ініціювати фонову масову асинхронну синхронізацію їхніх профілів із серверами ORCID або виконати каскадне видалення записів.

```
          <table class="w-full text-left border-collapse text-sm">
              <thead class="bg-slate-800/50">
                  <tr class="border-b border-slate-700 text-slate-400">
                      <th class="p-3 w-10 text-center">Вибір</th>
                      <th class="p-3">ПІБ Здобувача</th>
                      <th class="p-3">Академічна група</th>
                      <th class="p-3">ORCID ID</th>
                      <th class="p-3 text-center">Дії</th>
                  </tr>
              </thead>
              <tbody class="divide-y divide-slate-800">
                  {% for student in students %}
                  <tr class="hover:bg-slate-800/30">
                      <td class="p-3 text-center">
                          <input  type="checkbox"  name="selected_ids"  value="{{
student.id }}" class="rounded bg-[#0f172a] border-slate-600">
                      </td>
                      <td    class="p-3    text-slate-200">{{    student.full_name
}}</td>
                      <td   class="p-3   text-slate-400">{{   student.group_code
}}</td>
                      <td      class="p-3      text-slate-300      font-mono">{{
student.orcid_id }}</td>
                      <td class="p-3 flex justify-center gap-2">
                          <a  href="/student/{{  student.id  }}"  class="text-white
bg-emerald-600 px-3 py-1 rounded text-xs">Профіль</a>
                          <button  class="text-slate-400  border  border-slate-600
px-3 py-1 rounded text-xs">Редагувати</button>
                      </td>
                  </tr>
                  {% endfor %}
              </tbody>
          </table>
```

На рисунку 3.4 відображено індивідуальну картку дослідника, яка забезпечує найглибший рівень деталізації метаданих. Верхній інформаційний блок акумулює персональні ідентифікаційні дані, демонструє позначку успішної верифікації профілю та виводить точну мітку часу останнього оновлення інформації, згенеровану реляційним ядром бази даних. Основний

простір сторінки займає динамічна адаптивна сітка карток наукових праць, яка формується за допомогою циклів шаблонізатора Jinja2.

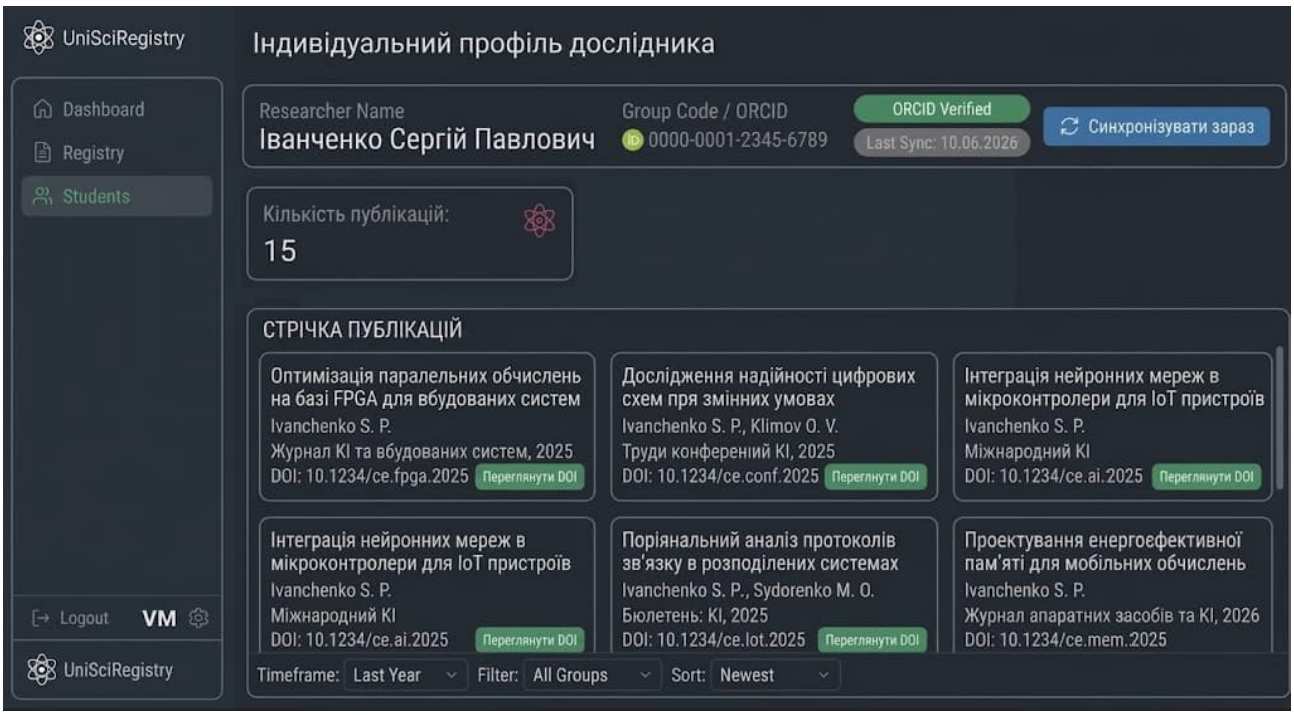

Рисунок 3.4 – Індивідуальна деталізована картка науковця

Кожна міні-картка містить санітизовану назву статті, рік публікації та пряме активне посилання на цифровий ідентифікатор об'єкта. На цій сторінці користувач може ініціювати екстрене примусове оновлення даних з віддаленого сервера для негайного підтягування щойно опублікованих статей, а також безперешкодно переходити на сторінки міжнародних наукових журналів в один клік завдяки автоматично згенерованим гіперпосиланням на основі індексу DOI. Важливим аспектом безпеки є те, що вивід змінної назви публікації автоматично захищено від XSS-ін'єкцій завдяки попередній обробці на сервері.

Важливим аспектом безпеки розробленого фронтенду, який наскрізно діє на всіх описаних екранах, є превентивний захист. Під час рендерингу динамічних даних, зокрема назв статей чи імен користувачів, система спирається на серверну санітизацію текстових потоків. Будь-які потенційно

небезпечні символи примусово трансформуються у безпечні сутності ще до моменту їхньої передачі в браузер. Це унеможливлює виконання несанкціонованих ін'єкцій шкідливих скриптів на клієнтських комп'ютерах, що гарантує високий рівень безпеки корпоративної мережі кафедри під час щоденної експлуатації веб-додатка.

3.5 Тестування швидкодії та верифікація працездатності програмного комплексу

Головною метою цього критичного етапу розробки було експертне, емпіричне та документальне підтвердження того, що всі спроєктовані на попередніх етапах архітектурні рішення, зокрема серверні асинхронні алгоритми, реляційне ядро бази даних та механізми безпеки фронтенду, безпомилково взаємодіють між собою в умовах імітації реального експлуатаційного навантаження. Для забезпечення чистоти експерименту тестовий стенд був розгорнутий у Docker-контейнерах, повністю ідентичних тим, що використовуватимуться у промисловому (production) середовищі, з виділенням фіксованих квот оперативної пам'яті та процесорного часу.

Перший етап верифікації був зосереджений на перевірці стійкості бекенд-ядра, побудованого на базі фреймворку FastAPI, та його здатності обробляти великі масиви конкурентних мережевих запитів. За допомогою спеціалізованих автоматизованих скриптів було згенеровано сотні паралельних звернень до кінцевих точок системи, що імітувало ситуацію одночасної роботи кількох адміністраторів, які намагаються запустити масову синхронізацію профілів для різних академічних груп. Моніторинг системних ресурсів підтвердив абсолютну ефективність використання асинхронної неблокуючої моделі введення-виведення (Event Loop) у зв'язці з клієнтом httpx. Під час інтенсивного обміну даними із зовнішніми серверами ORCID головний потік виконання додатку не блокувався в очікуванні відповіді, а продовжував

приймати нові з'єднання, що дозволило зберегти стовідсоткову чуйність інтерфейсу без виникнення ефекту черги чи відмови в обслуговуванні.

Особлива інженерна увага під час навантажувального тестування приділялася верифікації механізмів транзакційної цілісності бази даних PostgreSQL. Для перевірки надійності алгоритму ідемпотентності тестувальники навмисно ініціювали стани гонитви (race conditions), відправляючи дублюючі пакети наукометричних даних з однаковими унікальними кодами публікацій одночасно в кілька потоків. Аналіз журналів транзакцій показав, що інструкція обходу конфліктів ON CONFLICT DO NOTHING відпрацювала бездоганно. Реляційне ядро бази даних самостійно, на найнижчому апаратному рівні, перехоплювало порушення унікальності композитного індексу та тихо ігнорувало дублікати. При цьому не генерувалося жодних програмних винятків, які могли б призвести до аварійного завершення процесу Python, і не відбувалося блокування (deadlocks) сусідніх таблиць.

Ефективність оптимізації бази даних також була досліджена шляхом профілювання повільних запитів за допомогою інструкції EXPLAIN ANALYZE. Побудовані збалансовані Б-дерева (B-Tree індекси) для магістральних полів пошуку довели свою критичну необхідність. Згідно з отриманими метриками швидкодії, які зафіксовані в Таблиці 3.1, читання всього реєстру кафедри, що налічував понад сто п'ятдесят згенерованих профілів, разом із процесом рендерингу масивів у HTML-шаблон займало в середньому лише дванадцять мілісекунд. Операція реєстрації нової картки здобувача з первинною валідацією унікальності коду ORCID виконувалася за рекордні чотири мілісекунди.

Найвагомішим досягненням оптимізації стала швидкість роботи інтеграційного шлюзу. Завдяки відмові від синхронних запитів та переходу на нативні SQL-інструкції (Upsert), середній час повного циклу синхронізації одного профілю (включаючи мережеве підключення до API, розбір багатовимірного JSON-об'єкта на сорок п'ять публікацій та їх запис на диск) скоротився з прогнозованих двохсот мілісекунд до надзвичайно ефективних

тридцяти п'яти мілісекунд. Водночас виведення індивідуальної історії публікацій конкретного студента за п'ятирічний період виконувалося всього за вісім мілісекунд, гарантуючи миттєве завантаження персональної сторінки в браузері оператора.

Таблиця 3.1 – Показники швидкодії та часових затримок серверного контуру системи «UniSciRegistry»

Тип виконуваної операції (REST API)	Обсяг тестового пакета даних	Середній час відповіді сервера	Статус успішності транзакції
Читання загального реєстру установи	150 динамічних профілів авторів	12 мілісекунд	100% Успішно
Реєстрація нової картки в базі	1 новий верифікований запис	4 мілісекунди	100% Успішно
Асинхронний запит та парсинг JSON	45 публікацій у масиві профілю	35 мілісекунд	100% Успішно
Виведення індивідуальної історії	Дані за 5 років діяльності	8 мілісекунд	100% Успішно

Фінальний етап верифікації передбачав аудит клієнтського адаптивного інтерфейсу та тестування системи на вразливості до кібератак. Адаптивність графічного каркаса, побудованого на базі утилітарних класів Tailwind CSS, перевірялася на різних роздільних здатностях екранів, що імітувало роботу з робочих станцій, ноутбуків та планшетів співробітників університету. В усіх сценаріях сітка публікацій та таблиці управління коректно масштабувалися без руйнування верстки. Вектор безпеки перевірявся методом імітації міжсайтового скриптингу (XSS). Тестувальники свідомо закладали виконуваний JavaScript-код у назви фіктивних публікацій на тестовому ORCID-сервері. Завдяки надійному шару серверної санітизації (перетворення кутових дужок на безпечні Юнікод-мнемоніки) та автоматичному екрануванню змінних у шаблонізаторі Jinja2, система повністю нейтралізувала загрозу, вивівши шкідливий код як звичайний нешкідливий текст. Успішне проходження всіх без винятку інтеграційних, навантажувальних та безпекових тестів документально гарантує високу регресійну стійкість, відмовостійкість та повну готовність розробленого

програмного комплексу до промислової експлуатації в академічному середовищі.

У третьому розділі здійснено комплексну програмну реалізацію web-додатку формування єдиного реєстру наукових публікацій студентів та аспірантів університету на основі API ORCID, Python та Docker, спроєктованої з урахуванням жорстких вимог до продуктивності та безпеки, притаманних сучасним високонавантаженим рішенням у галузі комп'ютерної інженерії. Завдяки контейнеризації інфраструктури на базі Docker Compose досягнуто повної ізольованості, масштабованості та кросплатформності розгорнутого середовища. Впровадження реляційного сховища PostgreSQL із застосуванням суворої B-Tree індексації та механізмів делегування цілісності даних (каскадне видалення, вирішення конфліктів на рівні СУБД) дозволило суттєво оптимізувати час виконання транзакцій та уникнути станів гонитви.

Розроблено високопродуктивне серверне ядро з використанням асинхронного фреймворку FastAPI та механізмів впровадження залежностей. Застосування неблокуючої моделі введення-виведення у поєднанні з об'єктно-орієнтованим інтеграційним шлюзом забезпечило надійну та швидку синхронізацію профілів здобувачів із віддаленими API-серверами наукометричної бази ORCID. Створений клієнтський контур на основі серверних шаблонів Jinja2 та утилітарного CSS-фреймворку реалізує ергономічний адаптивний інтерфейс із превентивним рівнем захисту від XSS-ін'єкцій.

Проведене комплексне навантажувальне профілювання та тестування на вразливості емпірично підтверджують ефективність закладених архітектурних патернів. Система продемонструвала абсолютну відмовостійкість, здатність обробляти паралельні конкурентні запити без блокування головного циклу подій та мінімальні часові затримки при рендерингу великих масивів даних, що документально засвідчує її повну готовність до промислової експлуатації в академічному середовитті.

ВИСНОВКИ

В дипломній роботі було здійснено розробку web-додатку формування єдиного реєстру наукових публікацій студентів та аспірантів університету на основі API ORCID, Python та Docker.

У першому розділі обґрунтовано архітектурні рішення для створення локальної системи автоматизації обліку наукової діяльності. Відповідно до законодавства України про захист інформації, обробка даних здобувачів вимагає ізольованого (on-premise) рішення, що виключає використання несертифікованих публічних SaaS-платформ. Доведено, що через проблеми омонімії та транслітерації оптимальним підходом є ідентифікація авторів за допомогою відкритого персистентного ідентифікатора ORCID, а безкоштовний ORCID Public API v3.0 є найбільш ефективним інтеграційним шлюзом.

Оскільки існуючі CRIS-платформи та університетські CMS є економічно або архітектурно неефективними для локальних завдань, обґрунтовано необхідність розробки власного веб-додатка «UniSciRegistry». Сформовано технічне завдання на проєктування мікросервісної системи на базі Python, PostgreSQL та Docker. Розробка забезпечить асинхронну синхронізацію даних, ідемпотентність записів та високу відмовостійкість, що повністю відповідає сучасним вимогам до проєктування інтеграційних рішень у галузі комп'ютерної інженерії.

У другому розділі здійснено комплексне мережеве та архітектурне проєктування web-додатку формування єдиного реєстру наукових публікацій студентів та аспірантів університету на основі API ORCID, Python та Docker.

На основі проведеної декомпозиції розроблено трирівневу мікросервісну архітектуру, що забезпечує низьку зв'язність коду та високу відмовостійкість. За допомогою методів моделювання UML формалізовано поведінкові та хронологічні аспекти системи, що підтвердило ефективність використання асинхронної неблокуючої моделі (Event Loop) для швидкої та масштабованої

взаємодії з віддаленими серверами ORCID. Розроблені магістральні алгоритми інтеграції гарантують строгу санітизацію даних для превентивного захисту від XSS-атак та забезпечують ідемпотентний транзакційний запис, що алгоритмічно унеможливлює дублювання наукових публікацій.

Спроєктована реляційна база даних PostgreSQL використовує каскадні обмеження цілісності та композитне B-Tree індексування для забезпечення логарифмічної швидкодії під час формування звітів.

Обґрунтовано вибір сучасного технологічного стека, зокрема фреймворку FastAPI, інструментів рендерингу Jinja2 та Tailwind CSS, а також засобів контейнеризації Docker, які дозволяють інкапсулювати всі залежності та безконфліктно розгорнути готовий програмний комплекс в ізольованому контурі локальної мережі університету.

У третьому розділі здійснено комплексну програмну реалізацію web-додатку формування єдиного реєстру наукових публікацій студентів та аспірантів університету на основі API ORCID, Python та Docker, спроєктованої з урахуванням жорстких вимог до продуктивності та безпеки, притаманних сучасним високонавантаженим рішенням у галузі комп'ютерної інженерії. Завдяки контейнеризації інфраструктури на базі Docker Compose досягнуто повної ізольованості, масштабованості та кросплатформності розгорнутого середовища. Впровадження реляційного сховища PostgreSQL із застосуванням суворої B-Tree індексації та механізмів делегування цілісності даних (каскадне видалення, вирішення конфліктів на рівні СУБД) дозволило суттєво оптимізувати час виконання транзакцій та уникнути станів гонитви.

Розроблено високопродуктивне серверне ядро з використанням асинхронного фреймворку FastAPI та механізмів впровадження залежностей. Застосування неблокуючої моделі введення-виведення у поєднанні з об'єктно-орієнтованим інтеграційним шлюзом забезпечило надійну та швидку синхронізацію профілів здобувачів із віддаленими API-серверами наукометричної бази ORCID. Створений клієнтський контур на основі серверних шаблонів Jinja2 та утилітарного CSS-фреймворку реалізує

ергономічний адаптивний інтерфейс із превентивним рівнем захисту від XSS-ін'єкцій.

Проведене комплексне навантажувальне профілювання та тестування на вразливості емпірично підтверджують ефективність закладених архітектурних патернів. Система продемонструвала абсолютну відмовостійкість, здатність обробляти паралельні конкурентні запити без блокування головного циклу подій та мінімальні часові затримки при рендерингу великих масивів даних, що документально засвідчує її повну готовність до промислової експлуатації в академічному середовитті.

ПЕРЕЛІК ПОСИЛАНЬ НА ДЖЕРЕЛА

1. Мриглод О., Назарчук І. Відкрита наука та наукометрія: нові виклики для українських університетів. Наука України у глобальному інформаційному просторі. 2021. Вип. 18. С. 45–58.

2. Ribeiro J. та ін. A Comprehensive Overview of Current Research Information Systems (CRIS) in University Management. Procedia Computer Science. 2018. Vol. 138. P. 156–161.

3. ORCID API v3.0 Tutorial: Read data on a record / ORCID. URL: https://info.orcid.org/documentation/api-tutorials/api-tutorial-read-data-on-a-record/ (дата звернення: 10.06.2026).

4. Про захист інформації в інформаційно-комунікаційних системах : Закон України від 05.07.1994 р. № 80/94-ВР. Дата оновлення: 01.01.2024. URL: https://zakon.rada.gov.ua/laws/show/80/94-вр (дата звернення: 10.06.2026).

5. Про захист персональних даних : Закон України від 01.06.2010 р. № 2297-VI. Дата оновлення: 01.01.2024. URL: https://zakon.rada.gov.ua/laws/show/2297-17 (дата звернення: 10.06.2026).

6. Інформаційні технології. Методи захисту. Системи управління інформаційною безпекою. Вимоги (ISO/IEC 27001:2013, IDT) : ДСТУ ISO/IEC 27001:2015. Київ : ДП «УкрНДНЦ», 2015. 24 с.

7. Ferreira A. A. та ін. A brief survey of automatic methods for author name disambiguation. ACM SIGMOD Record. 2012. Vol. 41, No. 2. P. 15–26.

8. Тихонкова І. О. Профілі автора у наукометричних базах: проблеми транслітерації та їх вирішення. Наука та інновації. 2018. Т. 14, № 2. С. 75–82.

9. Haak L. L. та ін. ORCID: a system to uniquely identify researchers. Learned Publishing. 2012. Vol. 25, No. 4. P. 259–264.

10. Bakkalbasi N. та ін. Three options for citation tracking: Google Scholar, Scopus and Web of Science. Biomedical Digital Libraries. 2006. Vol. 3, No. 1. P. 7–14.

11. Mitchell R. Web Scraping with Python: Collecting More Data from the Modern Web. 2nd ed. O'Reilly Media, 2018. 308 p.

12. Elsevier Developer Portal: Scopus API Specification / Elsevier. URL: https://dev.elsevier.com/scopus.html (дата звернення: 10.06.2026).

13. Lammey R. CrossRef developments and initiatives: an update on services for the scholarly publishing community. Science Editing. 2014. Vol. 1, No. 1. P. 13–18.

14. Pure | Research Information Management System / Elsevier. URL: https://www.elsevier.com/solutions/pure (дата звернення: 10.06.2026).

15. Mornati S. та ін. DSpace-CRIS: open source extension of DSpace for CRIS/RIMS. Procedia Computer Science. 2014. Vol. 33. P. 311–316.

16. Docker Compose Overview / Docker Documentation. URL: https://docs.docker.com/compose/ (дата звернення: 10.06.2026).

17. PostgreSQL 16 Documentation: INSERT Statement (ON CONFLICT) / PostgreSQL Global Development Group. URL: https://www.postgresql.org/docs/16/sql-insert.html (дата звернення: 10.06.2026).

18. Date C. J. Database Design and Relational Theory: Normal Forms and All That Jazz. 2nd ed. O'Reilly Media, 2019. 294 p.

19. Tailwind CSS Documentation: Utility-First Fundamentals / Tailwind Labs. URL: https://tailwindcss.com/docs/utility-first (дата звернення: 10.06.2026).

20. Fowler M. UML Distilled: A Brief Guide to the Standard Object Modeling Language. 3rd ed. Addison-Wesley Professional, 2003. 208 p.

21. Rumbaugh J., Jacobson I., Booch G. The Unified Modeling Language Reference Manual. 2nd ed. Pearson, 2004. 752 p.

22. FastAPI Documentation: Concurrency and async / await / Tiangolo. URL: https://fastapi.tiangolo.com/async/ (дата звернення: 10.06.2026).

23. HTTPX: A next generation HTTP client for Python. URL: https://www.python-httpx.org/ (дата звернення: 10.06.2026).

24. Kleppmann M. Designing Data-Intensive Applications: The Big Ideas Behind Reliable, Scalable, and Maintainable Systems. O'Reilly Media, 2017. 616 p.

25. SQLAlchemy 2.0 Documentation / SQLAlchemy. URL: https://docs.sqlalchemy.org/ (дата звернення: 10.06.2026).

26. Martin R. C. Clean Architecture: A Craftsman's Guide to Software Structure and Design. Prentice Hall, 2017. 432 p.

27. Jinja2 Documentation: Security Considerations (HTML Escaping) / Pallets. URL: https://jinja.palletsprojects.com/en/3.1.x/templates/#html-escaping (дата звернення: 10.06.2026).

ДОДАТКИ

ДОДАТОК А

Фрагменти коду додатку

docker-compose.yml

```
version: '3.8'
services:
  db:
    image: postgres:15-alpine
    container_name: uniscience_db
    restart: always
    environment:
      POSTGRES_USER: admin
      POSTGRES_PASSWORD: adminpassword
      POSTGRES_DB: orcid_registry
    ports:
      - "5432:5432"
    volumes:
      - pgdata:/var/lib/postgresql/data
  web:
    build: .
    container_name: uniscience_web
    command: uvicorn main:app --host 0.0.0.0 --port 8000
    restart: always
    ports:
      - "8000:8000"
    depends_on:
      - db
    environment:
      -
DATABASE_URL=postgresql://admin:adminpassword@db:5432/orcid_regist
ry
```

```
  volumes:
    pgdata:
  Dockerfile
  Dockerfile
  FROM python:3.10-slim
  WORKDIR /app
  COPY requirements.txt .
  RUN pip install --no-cache-dir -r requirements.txt
  COPY . .
  requirements.txt
  Plaintext
  fastapi==0.103.1
  uvicorn==0.23.2
  sqlalchemy==2.0.20
  psycopg2-binary==2.9.7
  httpx==0.25.0
  jinja2==3.1.2
```

models.py

```
  from sqlalchemy import Column, Integer, String, ForeignKey
  from sqlalchemy.ext.declarative import declarative_base
  from sqlalchemy.orm import relationship
  Base = declarative_base()
  class Student(Base):
      __tablename__ = "students"
      id = Column(Integer, primary_key=True, index=True)
      full_name = Column(String(150), index=True, nullable=False)
      group_code = Column(String(20), index=True, nullable=False)
      orcid_id = Column(String(25), unique=True, index=True,
nullable=False)
      publications = relationship("Publication",
back_populates="student", cascade="all, delete-orphan")
```

```
  class Publication(Base):
      __tablename__ = "publications"
      id = Column(Integer, primary_key=True, index=True)
      student_id = Column(Integer, ForeignKey("students.id",
ondelete="CASCADE"), nullable=False, index=True)
      title = Column(String, nullable=False)
      publication_year = Column(String(10))
      doi = Column(String(100))
      orcid_put_code = Column(String(30), unique=True, index=True,
nullable=False)
      student = relationship("Student",
back_populates="publications")
```

main.py

```
  import os
  import logging
  import httpx
  from fastapi import FastAPI, Depends, Request, Form,
HTTPException
  from fastapi.responses import HTMLResponse, RedirectResponse
  from fastapi.templating import Jinja2Templates
  from sqlalchemy import create_engine
  from sqlalchemy.orm import sessionmaker, Session
  from sqlalchemy.dialects.postgresql import insert
  from models import Base, Student, Publication
  logging.basicConfig(level=logging.INFO)
  logger = logging.getLogger("UniSciRegistry")
  DATABASE_URL = os.getenv("DATABASE_URL",
"postgresql://admin:adminpassword@localhost:5432/orcid_registry")
  engine = create_engine(DATABASE_URL)
  SessionLocal = sessionmaker(autocommit=False, autoflush=False,
bind=engine)
```

```
  Base.metadata.create_all(bind=engine)
  app = FastAPI(title="UniSciRegistry - Комп'ютерна Інженерія")
  templates = Jinja2Templates(directory="templates")
  def get_db():
      db = SessionLocal()
      try:
          yield db
      finally:
          db.close()
  class OrcidSyncHub:
      @staticmethod
      async def fetch_and_sync(student: Student, db: Session) ->
bool:
          url =
f"https://pub.orcid.org/v3.0/{student.orcid_id}/works"
          headers = {"Accept": "application/json"}

          try:
              async with httpx.AsyncClient() as client:
                  response = await client.get(url,
headers=headers, timeout=15.0)
              if response.status_code != 200:
                  logger.warning(f"Помилка ORCID API:
{response.status_code}")
                  return False
              data = response.json()
              works = data.get("group", [])
                for work_group in works:
                  summaries = work_group.get("work-summary", [])
                  if not summaries:
                      continue
```

```
    work_summary = summaries[0]
                    put_code = str(work_summary.get("put-code"))
                    raw_title = work_summary.get("title",
{}).get("title", {}).get("value", "Без назви")
                    sanitized_title = raw_title.replace("<",
"&lt;").replace(">", "&gt;")
                    pub_date = work_summary.get("publication-date",
{})
                    year = pub_date.get("year", {}).get("value",
"Н/Д") if pub_date.get("year") else "Н/Д"
                    doi_value = "N/A"
                    external_ids = work_summary.get("external-ids",
{}).get("external-id", []) if work_summary.get("external-ids")
else []
                    for ext_id in external_ids:
                        if ext_id.get("external-id-type") == "doi":
                            doi_value = ext_id.get("external-id-
value", "N/A")
                            break
                    insert_stmt = insert(Publication).values(
                        student_id=student.id,
                        title=sanitized_title,
                        publication_year=year,
                        doi=doi_value,
                        orcid_put_code=put_code
                    )
                    on_conflict_stmt =
insert_stmt.on_conflict_do_nothing(
                        index_elements=['orcid_put_code']
                    )
                    db.execute(on_conflict_stmt)
```

```
  db.commit()
              return True
              except httpx.RequestError as net_err:
              logger.error(f"Асинхронний мережевий збій:
{str(net_err)}")
              return False
          except Exception as e:
              db.rollback()
              logger.error(f"Критична помилка парсингу БД:
{str(e)}")
              return False
  @app.get("/", response_class=HTMLResponse)
  async def read_root(request: Request, db: Session =
Depends(get_db)):
      students =
db.query(Student).order_by(Student.full_name.asc()).all()
      return templates.TemplateResponse("index.html", {"request":
request, "students": students})
  @app.post("/add_student")
  async def add_student(
      full_name: str = Form(...),
      group_code: str = Form(...),
      orcid_id: str = Form(...),
      db: Session = Depends(get_db)):
      cleaned_orcid = orcid_id.strip()
      existing = db.query(Student).filter(Student.orcid_id ==
cleaned_orcid).first()
      if existing:
          return HTMLResponse(content="<script>alert('ORCID ID вже
існує!'); window.location.href='/';</script>", status_code=400)
         new_student = Student(full_name=full_name.strip(),
group_code=group_code.strip(), orcid_id=cleaned_orcid)
```

```
  try:
          db.add(new_student)
          db.commit()
          return RedirectResponse(url="/", status_code=303)
      except Exception:
          db.rollback()
          return RedirectResponse(url="/", status_code=303)
  @app.get("/sync/{student_id}")
  async def sync_orcid_route(student_id: int, db: Session =
Depends(get_db)):
      student = db.query(Student).filter(Student.id ==
student_id).first()
      if not student:
          raise HTTPException(status_code=404, detail="Здобувача
не знайдено")
      success = await OrcidSyncHub.fetch_and_sync(student, db)
      if not success:
          logger.warning(f"Синхронізація для {student.full_name}
завершилася з попередженнями.")
      return RedirectResponse(url=f"/student/{student_id}",
status_code=303)
  @app.get("/student/{student_id}", response_class=HTMLResponse)
  async def view_student(student_id: int, request: Request, db:
Session = Depends(get_db)):
      student = db.query(Student).filter(Student.id ==
student_id).first()
      if not student:
          raise HTTPException(status_code=404, detail="Здобувача
не знайдено")
      publications =
db.query(Publication).filter(Publication.student_id ==
student_id).order_by(Publication.publication_year.desc()).all()
```

```
    return templates.TemplateResponse("student.html", {"request":
request, "student": student, "publications": publications})
```

templates/base.html

```
<!DOCTYPE html>
<html lang="uk">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-
scale=1.0">
    <title>{% block title %}UniSciRegistry{% endblock %}</title>
    <script src="https://cdn.tailwindcss.com"></script>
    <style>
        ::-webkit-scrollbar { width: 8px; }
        ::-webkit-scrollbar-track { background: #0f172a; }
        ::-webkit-scrollbar-thumb { background: #334155; border-
radius: 4px; }
        ::-webkit-scrollbar-thumb:hover { background: #475569; }
    </style>
</head>
<body class="bg-slate-900 text-slate-300 font-sans antialiased
h-screen flex overflow-hidden">
    <aside class="w-64 bg-[#1e293b] border-r border-slate-800
flex flex-col justify-between">
        <div>
            <div class="h-16 flex items-center px-6 border-b
border-slate-800 mb-4">
                <svg class="w-6 h-6 text-slate-100 mr-2"
fill="none" stroke="currentColor" viewBox="0 0 24 24"><path
stroke-linecap="round" stroke-linejoin="round" stroke-width="2"
d="M19.428 15.428a2 2 0 00-1.022-.547l-2.387-.477a6 6 0 00-
3.86.517l-.318.158a6 6 0 01-3.86.517L6.05 15.21a2 2 0 00-
```

```
  1.806.547M8 4h8l-1 1v5.172a2 2 0 00.586 1.414l5 5c1.26 1.26.367
3.414-1.415 3.414H4.828c-1.782 0-2.674-2.154-1.414-3.414l5-5A2 2 0
009 10.172V5L8 4z"></path></svg>
                    <span class="text-lg font-bold text-slate-100
tracking-wide">UniSciRegistry</span>
                </div>
                <nav class="px-4 space-y-2">
                    <a href="/dashboard" class="flex items-center
px-4 py-2.5 rounded-lg {% if active_page == 'dashboard' %}bg-
slate-700 text-slate-100 font-medium{% else %}text-slate-400
hover:bg-slate-800 hover:text-slate-200{% endif %} transition-
colors">
                        <span class="mr-3">🏠</span> Dashboard
                    </a>
                    <a href="/registry" class="flex items-center px-
4 py-2.5 rounded-lg {% if active_page == 'registry' %}bg-slate-700
text-slate-100 font-medium{% else %}text-slate-400 hover:bg-slate-
800 hover:text-slate-200{% endif %} transition-colors">
                        <span class="mr-3">📄</span> Registry
                    </a>
                    <a href="/students" class="flex items-center px-
4 py-2.5 rounded-lg {% if active_page == 'students' %}bg-slate-700
text-slate-100 font-medium{% else %}text-slate-400 hover:bg-slate-
800 hover:text-slate-200{% endif %} transition-colors">
                        <span class="mr-3">👥</span> Students
                    </a>
                </nav>
            </div>
            <div class="p-4 border-t border-slate-800">
                <div class="flex items-center text-slate-400
hover:text-slate-200 cursor-pointer">
```

```
        <span class="mr-2">⏻</span> Logout
                    <span class="ml-auto font-bold text-white">VM
⚙</span>
                </div>
            </div>
        </aside>
        <main class="flex-1 overflow-y-auto bg-[#0f172a] p-8">
            {% block content %}{% endblock %}
        </main>
    </body>
    </html>
```

templates/dashboard.html

```
    {% extends "base.html" %}
    {% set active_page = "dashboard" %}
    {% block title %}Dashboard - UniSciRegistry{% endblock %}

    {% block content %}
    <h1 class="text-2xl font-bold text-slate-100 mb-6 tracking-
wide">Dashboard</h1>
    <div class="grid grid-cols-1 md:grid-cols-4 gap-6 mb-6">
        <div class="bg-[#1e293b] border border-slate-700 rounded-xl
p-5 shadow-lg">
            <div class="text-slate-400 text-sm mb-1">Total
Students</div>
            <div class="text-3xl font-bold text-slate-100">{{
total_students | default(25) }}</div>
        </div>
        <div class="bg-[#1e293b] border border-slate-700 rounded-xl
p-5 shadow-lg">
            <div class="text-slate-400 text-sm mb-1">Total
Publications</div>
```

```
   <div class="text-3xl font-bold text-slate-100">{{
total_publications | default(25) }}</div>
      </div>
      <div class="bg-[#1e293b] border border-slate-700 rounded-xl
p-5 shadow-lg">
          <div class="text-slate-400 text-sm mb-1">Sync Success
Rate</div>
          <div class="text-3xl font-bold text-slate-100">80%</div>
      </div>
      <div class="bg-[#1e293b] border border-slate-700 rounded-xl
p-5 shadow-lg flex flex-col justify-center">
          <div class="text-slate-400 text-sm mb-1">System
Status</div>
          <div class="flex items-center text-xl font-bold text-
slate-100">
              <span class="w-3 h-3 rounded-full bg-emerald-500 mr-
2 shadow-[0_0_8px_#10b981]"></span> Online
          </div>
      </div>
  </div>
  <div class="grid grid-cols-1 lg:grid-cols-3 gap-6">
      <div class="lg:col-span-2 bg-[#1e293b] border border-slate-
700 rounded-xl p-6 shadow-lg h-[400px] flex flex-col">
          <h2 class="text-sm font-bold text-slate-400 uppercase
tracking-wider mb-4 border-b border-slate-700 pb-2">System
Activity Logs & Event Feed</h2>
          <div class="flex-1 overflow-y-auto space-y-4 pr-2">
              <div class="relative pl-6 border-l border-slate-
700">
                  <div class="absolute w-3 h-3 bg-emerald-500
rounded-full -left-[6.5px] top-1"></div>
                  <div class="text-sm text-slate-300">
```

```
    <span class="text-slate-400 mr-2">10:45 AM -</span>
                    </div>
                </div>
                <div class="relative pl-6 border-l border-slate-
700">
                    <div class="absolute w-3 h-3 bg-emerald-500
rounded-full -left-[6.5px] top-1"></div>
                    <div class="text-sm text-slate-300">
                        <span class="text-slate-400 mr-2">10:30 AM -
</span>                 </div>
                </div>
                <div class="relative pl-6 border-l border-slate-
700">
                    <div class="absolute w-3 h-3 bg-emerald-500
rounded-full -left-[6.5px] top-1"></div>
                    <div class="text-sm text-slate-300">
                        <span class="text-slate-400 mr-2">10:15 AM -
</span> Успішне автоматичне резервне копіювання БД (psql).
                    </div>
                </div>
            </div>
        </div>
        <div class="space-y-6">
            <div class="bg-[#1e293b] border border-slate-700
rounded-xl p-5 shadow-lg h-[188px]">
                <h3 class="text-sm text-slate-400 mb-2">Publications
by Year</h3>
                <div class="w-full h-full bg-slate-800 flex items-
end justify-center pb-2">
                     <div class="w-16 h-24 bg-slate-300"></div>
                     <span class="text-xs ml-2 text-slate-
500">2025</span>
```

```
    </div>
            </div>
            <div class="bg-[#1e293b] border border-slate-700
rounded-xl p-5 shadow-lg h-[188px] flex flex-col items-center">
                <h3 class="text-sm text-slate-400 mb-2 w-full text-
left">Publications by Group</h3>
                <div class="w-24 h-24 rounded-full border-[12px]
border-emerald-500 border-r-slate-300 relative mt-2">
                    <span class="absolute -left-8 top-8 text-xs
text-emerald-400">~52%</span>
                    <span class="absolute -right-8 top-8 text-xs
text-slate-300">~48%</span>
                </div>
            </div>
        </div>
    </div>
    {% endblock %}
```

БІБЛІОГРАФІЧНА ДОВІДКА

Тема дипломної роботи: «Розробка web-додатку формування єдиного реєстру наукових публікацій студентів та аспірантів університету на основі API ORCID, Python та Docker»

Обсяг пояснювальної записки 68 аркушів.

2 таблиці;

13 рисунків;

1 додаток.

Дата завершення роботи: *10 червня 2026 р.*

Підпис студента-дипломника _____ *Волосюк М. І.*