МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДЕРЖАВНИЙ УНІВЕРСИТЕТ «ЖИТОМИРСЬКА ПОЛІТЕХНІКА»
ФАКУЛЬТЕТ ІНФОРМАЦІЙНО-КОМП'ЮТЕРНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

ПОЯСНЮВАЛЬНА ЗАПИСКА

до кваліфікаційної роботи освітнього ступеня «бакалавр»
за спеціальністю 121 «Інженерія програмного забезпечення»
(освітня програма «Інженерія програмного забезпечення»)
на тему:

«Система управління академічними ресурсами: серверна та клієнтська логіка»

Виконав студент групи ІПЗ-21-4
ЩОКОТОВ Сергій Сергійович

Керівник роботи:
ЄФРЕМОВ Юрій Миколайович

Рецензент:
ТОЛСТОЙ Ігор Анатолійович

Житомир – 2025

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДЕРЖАВНИЙ УНІВЕРСИТЕТ «ЖИТОМИРСЬКА ПОЛІТЕХНІКА»
ФАКУЛЬТЕТ ІНФОРМАЦІЙНО-КОМП'ЮТЕРНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

«ЗАТВЕРДЖУЮ»
Зав. кафедри інженерії
програмного забезпечення
Тетяна Вакалюк
«26» лютого 2025 р.

ЗАВДАННЯ
на кваліфікаційну роботу

Здобувач вищої освіти: **ЩОКОТОВ Сергій Сергійович**
Керівник роботи: **ЄФРЕМОВ Юрій Миколайович**
Тема роботи: **«Система управління академічними ресурсами: серверна та клієнтська логіка»**,
затверджена наказом закладу вищої освіти **№61/с** від **«14» лютого 2025р.**
Вихідні дані для роботи: **об'єктом дослідження є** процеси організації та забезпечення доступу до академічних ресурсів у закладах вищої освіти за допомогою інформаційних систем, а **предметом дослідження є** серверна та клієнтська логіка програмного забезпечення для системи управління академічними ресурсами, принципи її розробки, функціонування та інтеграції з ролями користувачів (студент, викладач, адміністратор).
Консультанти з бакалаврської кваліфікаційної роботи із зазначенням розділів, що їх стосуються:

Розділ	**Консультант**	Завдання видав	Завдання прийняв
1	Єфремов Ю. М.	27.02.2025р.	27.02.2025р.
2	Єфремов Ю. М.	27.02.2025р.	27.02.2025р.
3	Єфремов Ю. М.	27.02.2025р.	27.02.2025р.

РЕФЕРАТ

Випускна кваліфікаційна робота бакалавра складається з програмного комплексу та пояснювальної записки. Пояснювальна записка до випускної роботи містить 124 сторінки, включає 65 ілюстрацій, 19 таблиць, 3 додатки та 60 використаних літературних джерела.

Метою роботи є розробка алгоритмів функціонування та реалізація програмного комплексу – інформаційної системи управління академічними ресурсами студентів, що дозволяє підвищити ефективність організації освітнього процесу, моніторингу успішності та взаємодії між учасниками навчального середовища.

У роботі проведено аналіз предметної області, сформульовано вимоги до функціоналу системи, розроблено архітектуру з урахуванням клієнт-серверної моделі та принципів масштабованості. Для реалізації серверної частини використано сучасні технології Python (FastAPI), SQLAlchemy та PostgreSQL. Система містить серверну частину (REST API), сучасний вебінтерфейс для користувачів (React), а також інструменти для адміністрування навчального процесу.

Передбачено підтримку кількох ролей користувачів (студент, викладач, адміністратор), авторизацію та двофакторну автентифікацію, управління користувачами, електронний журнал, систему оцінювання, реєстрацію на курси, чат для спілкування й логування дій. Особливу увагу приділено безпеці, масштабованості, гнучкому розподілу доступу та захисту даних.

Система має модульну структуру: управління курсами, модулями, завданнями, оцінюванням і аналітикою успішності, а також підтримує інтеграцію із зовнішніми сервісами й гнучке адміністрування.

КЛЮЧОВІ СЛОВА: АВТОМАТИЗАЦІЯ, УПРАВЛІННЯ, ОСВІТНЯ ПЛАТФОРМА, FASTAPI, POSTGRESQL, КУРСИ, ОЦІНЮВАННЯ, АУДИТ.

ABSTRACT

The Bachelor's qualification project consists of a software suite and an explanatory report. The report includes 124 pages, 65 illustrations, 19 tables, 3 appendices, and 60 references.

The aim of the work is to develop the functioning algorithms and implement a software solution–an information system for managing students' academic resources, designed to improve the efficiency of educational process organization, performance monitoring, and interaction among all participants of the learning environment.

The project includes analysis of the subject area, formulation of system functional requirements, and the development of an architecture based on a client-server model and scalability principles. The server side is implemented using modern technologies such as Python (FastAPI), SQLAlchemy, and PostgreSQL. The system includes a backend (REST API), a modern web interface for users (React), and administrative tools for managing the educational process.

The platform supports multiple user roles (student, teacher, administrator), user authentication and two-factor verification, user management, an electronic gradebook, a grading system, course enrollment, a chat for communication, and action logging. Special attention is paid to security, scalability, flexible access management, and data protection.

The system has a modular structure: course management, educational modules and assignments, grading and performance analytics, as well as integration with external services and flexible administration.

KEYWORDS: AUTOMATION, MANAGEMENT, EDUCATIONAL PLATFORM, FASTAPI, POSTGRESQL, COURSES, GRADING, AUDIT.

ЗМІСТ

РЕФЕРАТ 3
ABSTRACT 4
ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ 6
ВСТУП 7
РОЗДІЛ 1. АНАЛІЗ ТА ПОСТАНОВКА ЗАВДАННЯ 9
1.1 Постановка завдання у предметній області та основні завдання автоматизації 9
1.2 Аналіз існуючих аналогів програмного продукту 11
1.3 Вибір архітектури для система управління академічними ресурсами 24
1.4 Обґрунтування вибору інструментальних засобів 31
1.5 Командна розробка та організація спільного процесу 35
Висновки до першого розділу 40
РОЗДІЛ 2. ПРОЄКТУВАННЯ ІНФОРМАЦІЙНОЇ СИСТЕМИ УПРАВЛІННЯ АКАДЕМІЧНИМИ РЕСУРСАМИ СТУДЕНТІВ 42
2.1 Специфікація вимог до програмного продукту 42
2.2 Визначення варіантів використання та об’єктно-орієнтованої структури системи 52
2.3 Розробка бази даних системи 59
2.4. Проектування та розробка алгоритмів роботи системи. 80
Висновки до другого розділу 92
РОЗДІЛ 3. ІНТЕРФЕЙС ТА АЛГОРИТМ РОБОТИ ТА ВИКОРИСТАННЯ СИСТЕМИ УПРАВЛІННЯ АКАДЕМІЧНИМИ РЕСУРСАМИ 94
3.1. Порядок налаштування параметрів системи управління 94
3.2. Структура та алгоритм використання інтерфейсу системи 96
3.3. Тестування роботи системи управління академічними ресурсами 115
Висновки до третього розділу 116
ВИСНОВКИ 118
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ 120
ДОДАТКИ 125

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

IS – Information System / Інформаційна система

API – Application Programming Interface / Програмний інтерфейс прикладного програмування

DB – Database / База даних

ORM – Object-Relational Mapping / Об’єктно-реляційне відображення

SQL – Structured Query Language / Мова структурованих запитів

UUID – Universally Unique Identifier / Універсальний унікальний ідентифікатор

REST – Representational State Transfer / Технологія архітектури обміну даними

CRUD – Create, Read, Update, Delete / Основні операції з даними

JWT – JSON Web Token / Веб-токен у форматі JSON

TOTP – Time-based One-Time Password / Одноразовий пароль на основі часу

2FA – Two-Factor Authentication / Двофакторна автентифікація

UI – User Interface / Користувацький інтерфейс

UX – User Experience / Користувацький досвід

CSV – Comma-Separated Values / Формат табличних даних

JSON – JavaScript Object Notation / Формат обміну даними

JWT – JSON Web Token / Веб-токен у форматі JSON

CRUD – Create, Read, Update, Delete / Операції створення, читання, оновлення, видалення

WS – WebSocket / Технологія двосторонньої комунікації в реальному часі

CI/CD – Continuous Integration / Continuous Delivery / Безперервна інтеграція та доставка

RBAC – Role-Based Access Control / Контроль доступу на основі ролей

HTTP – HyperText Transfer Protocol / Протокол передачі гіпертексту

ID – Identifier / Ідентифікатор

PK – Primary Key / Первинний ключ

FK – Foreign Key / Зовнішній ключ

ВСТУП

У сучасних умовах цифровізації освіти та зростання потреб у дистанційному та змішаному навчанні особливої актуальності набувають програмні рішення для ефективного управління академічними ресурсами. Проблема організації доступу до навчальних матеріалів, тестів, завдань, а також ефективної взаємодії між студентами, викладачами та адміністрацією закладів вищої освіти залишається відкритою. Хоча на ринку існує низка платформ, таких як Moodle, Google Classroom, Canvas тощо, жодна з них не є ідеальною – кожна має обмеження, пов'язані з функціональністю, адаптивністю до української освітньої системи, складністю налаштування або інтеграції з внутрішніми процесами вишів. В Україні такі платформи використовуються обмежено, а потреба у власному, локалізованому рішенні лише зростає.

У межах цієї роботи розробляється інформаційна система управління академічними ресурсами, яка складається з серверної та клієнтської частин. Продукт дозволяє адмініструвати курси, модулі, навчальні матеріали, тести та завдання з гнучкою роллю доступу: студенти мають обмежені функції взаємодії з курсами, викладачі – можливість створювати і редагувати власні матеріали, а адміністратори – повний контроль над системою. Система передбачає автоматизовану перевірку тестів і завдань, інтерактивні елементи та логічну структуру подання контенту, що сприяє покращенню навчального процесу.

Запропонована система має низку переваг: адаптацію під національні освітні вимоги, простий інтерфейс, зручну адміністративну панель, гнучке управління контентом, модульну архітектуру. Її розробка дозволить закладам освіти оптимізувати роботу з академічною інформацією, зменшити навантаження на викладачів та адміністрацію, а також забезпечити студентам ефективний доступ до навчальних матеріалів. У той же час розробка відповідає на виклик: створення гнучкої, сучасної альтернативи існуючим закордонним рішенням, яка краще враховує потреби української аудиторії.

Таким чином, система управління академічними ресурсами студентів є комплексним інструментом, який сприятиме модернізації навчального процесу,

підвищенню якості освіти та створенню сучасного інформаційного простору для всіх учасників освітнього процесу.

Метою випускної кваліфікаційної роботи є створення інтегрованої вебсистеми для управління академічними ресурсами студентів, яка сприятиме автоматизації навчальних процесів, підвищенню зручності доступу до інформації та оптимізації роботи як студентів, так і викладачів. Система передбачає забезпечення безпечного доступу до особистих даних, організацію ефективного управління документальними та академічними потоками, а також створення сучасного інформаційного простору для комунікації та взаємодії всіх учасників освітнього процесу.

Завдання передбачає реалізацію ряду завдань, серед яких – провести аналіз існуючих освітніх платформ і визначити їх переваги та недоліки. Сформувати вимоги до програмного забезпечення, враховуючи рольову модель користувачів. Розробити архітектуру системи з розділенням серверної та клієнтської частин. Реалізувати базу даних для зберігання інформації про користувачів, курси, матеріали, тести та завдання. Реалізувати REST API для взаємодії клієнта з сервером. Розробити інтерфейс користувача з урахуванням UX/UI принципів. Забезпечити функціональність перевірки тестів та завдань. І в результаті провести тестування системи та підготувати документацію.

Об'єктом дослідження є процеси управління академічними ресурсами в освітніх установах, що включають збір, зберігання та обробку інформації про навчальну діяльність студентів і викладачів.

Предметом дослідження є розробка інтегрованої вебсистеми, яка автоматизує ключові процеси управління академічними ресурсами, зокрема реєстрацію, авторизацію, документообіг та підтримку навчального процесу, а також принципи її розробки, функціонування та інтеграції з ролями користувачів.

РОЗДІЛ 1. АНАЛІЗ ТА ПОСТАНОВКА ЗАВДАННЯ

1.1 Постановка завдання у предметній області та основні завдання автоматизації

Сучасна вища освіта функціонує в умовах стрімкої цифрової трансформації, коли більш ніж 60 % населення України активно користуються інтернетом для здобуття знань і комунікації [1]. Відповідно до ініціативи ЮНЕСКО, підтримка та розвиток онлайн-навчання для українських студентів є одним із пріоритетів у забезпеченні безперервності освіти в умовах воєнних та посткризових викликів [2]. Водночас практичний досвід свідчить, що існуючі рішення – як-то Moodle, Google Classroom або Canvas – не повною мірою враховують національні стандарти, подекуди мають незручний інтерфейс та високі вимоги до технічного обслуговування [3]. В Україні ж локальні розробки зустрічаються рідко, а ті, що є, часто обмежені вузьким функціоналом або нестабільною підтримкою [4].

Автоматизація процесів управління академічними ресурсами включає цілий спектр задач, спрямованих на оптимізацію взаємодії всіх учасників освітнього процесу. За результатами огляду функціональних критеріїв для e-learning платформ, до ключових напрямів належать управління контентом (надання, структурування та оновлення навчальних матеріалів), адміністрування користувачів із гнучкими ролями та доступами, автоматизована перевірка тестових завдань і аналітика успішності, а також засоби комунікації й колаборації між студентами та викладачами [4]. Додатково слід врахувати вимоги до масштабованості та безпеки системи з урахуванням високої інтенсивності одночасних запитів у період семестрових контрольних заходів [5].

Однією з головних цілей розробки є забезпечення зручного та інтуїтивно зрозумілого інтерфейсу, який дозволить мінімізувати час навчання користувачів роботі з платформою та знизити потребу в постійному технічному супроводі [6]. Значну увагу слід приділити адаптивності інтерфейсу для різних пристроїв (мобільні телефони, планшети, десктопи), оскільки мобільний доступ стає все більш затребуваним серед студентів під час дистанційного та змішаного навчання

[5]. Відтак завдання з автоматизації повинні охоплювати не лише бекенд-логіку, але й клієнтську частину з підтримкою сучасних UX/UI практик.

У контексті українського освітнього середовища важливо інтегрувати систему з локальними інформаційними ресурсами університетів (єдина база даних студентів, внутрішній реєстр курсів, ERP-системи), а також передбачити можливість подальшого розширення функціоналу: наприклад, впровадження штучного інтелекту для генерації адаптивних навчальних траєкторій чи аналітики поведінки користувачів [7]. Ця інтеграція вимагатиме продуманого API, системи черг для фонового оброблення завдань і механізмів синхронізації з зовнішніми сервісами.

Розподіл завдань між командою

У межах реалізації системи управління академічними ресурсами було визначено й деталізовано низку ключових задач, які підлягали автоматизації у програмному продукті. Функціональний спектр проекту охоплював як академічні, так і адміністративні процеси, а для ефективності та швидкості розробки від початку було закладено чіткий розподіл обов'язків між членами команди.

Розробка велась у форматі командної співпраці: кожен з учасників відповідав за визначений блок функціоналу, що дозволило уникнути дублювань, пришвидшити інтеграцію та забезпечити якість на кожному етапі. Зокрема, на мені було зосереджено розробку таких підсистем, як адміністрування користувачів (admins, teachers, students, users), модуль управління чатами (chats, chat_participants, messages), навчальний контент і взаємодія з курсами (courses, course_module, student_courses, teacher_courses, lessons, grades), механізми роботи із сесіями (sessions), двофакторною автентифікацією (totp_secrets), аудитом дій користувачів (audit_logs), а також організацією конфігураційних процесів (alembic_version).

Завдання мого блоку включали:

- Адміністрування користувачів та ролей: створення, активація, керування доступом для студентів, викладачів і адміністраторів, реалізація авторизації та двофакторної автентифікації, налаштування прав і розмежування доступу.

- Розробка системи навчального контенту: створення структурованих курсів, модулів, уроків, автоматизація зарахування та обліку студентів і викладачів на курсах, забезпечення можливості ведення електронного журналу оцінювання.
- Організація комунікації: реалізація інтегрованого чату для студентів і викладачів, підтримка миттєвого обміну повідомленнями, структуроване збереження історії чатів.
- Забезпечення безпеки: впровадження сесій користувачів, двофакторної аутентифікації (TOTP), системи логування всіх дій для аудиту та контролю.
- Підтримка технічної надійності: реалізація конфігураційних механізмів, ведення міграцій структури бази даних.

Загалом, основна мета автоматизації – створити єдиний інформаційний простір, у якому адміністрація університету, викладачі та студенти можуть ефективно взаємодіяти: від керування акаунтами, проходження навчальних модулів і оцінювання – до повного контролю за ходом навчального процесу й отримання актуальної аналітики. Розподіл завдань дозволив не лише підвищити швидкість і якість розробки, а й закласти гнучку основу для масштабування та подальшого розвитку платформи.

Як результат, кожен із ключових функціональних блоків – від управління користувачами й чатами до автоматизованого ведення журналу оцінок та реалізації аудиту – буде розроблений у рамках чітко визначеної відповідальності, що забезпечило цілісність, надійність і ефективність системи загалом.

1.2 Аналіз існуючих аналогів програмного продукту

Перш ніж розпочати власну розробку системи управління академічними ресурсами, доцільно здійснити ґрунтовний аналіз існуючих на ринку рішень. Такий аналіз дозволяє виявити сильні та слабкі сторони конкурентів, зрозуміти потреби кінцевих користувачів, а також сформулювати вимоги до майбутнього продукту з урахуванням реальних побажань аудиторії. Крім того, порівняння функціональних можливостей і архітектурних підходів аналізованих систем сприяє мінімізації

ризиків при проєктуванні власної платформи, підвищує конкурентоздатність рішення та забезпечує його високу адаптивність до специфіки українських вишів [8].

Сучасний ринок освітніх технологій пропонує низку програмних продуктів, спрямованих на автоматизацію управління академічними ресурсами. Серед найбільш поширених рішень – системи управління навчальним процесом (Learning Management Systems, LMS), електронні журнали, фінансові інформаційні системи та системи документообігу. Такі продукти дозволяють вирішувати окремі завдання, пов’язані з організацією навчального процесу, але часто мають обмежену інтеграцію між різними модулями [9].

Перевагою існуючих аналогів є їх перевірена на практиці функціональність та здатність задовольнити базові потреби користувачів. Наприклад, системи LMS забезпечують доступ до навчальних матеріалів, тестування та відстеження успішності студентів, а електронні журнали дозволяють оперативно вести облік оцінок і відвідуваності. Також багато продуктів мають інтуїтивно зрозумілі інтерфейси та можливість масштабування відповідно до розміру навчального закладу.

Проте, недоліками існуючих систем є обмежені можливості для інтеграції з іншими платформами, високі витрати на налаштування та підтримку, а також недостатня адаптивність до специфічних вимог окремих закладів. Крім того, багато з рішень не забезпечують високого рівня безпеки, що є критичним для зберігання персональних даних студентів та викладачів.

Нижче наведено огляд деяких із них із зазначенням їх функціоналу, переваг, недоліків та посилань для детальнішого ознайомлення.

Moodle – відкрита платформа з відкритим кодом, що здобула поширення у всьому світі (рис. 1.1). Вона забезпечує управління курсами, структурою модулів, підтримку плагінів, засоби для тестування та автоматичного оцінювання. Перевагами Moodle є гнучкість налаштувань, велика спільнота розробників і безкоштовність комерційного використання [10]. Однак інтерфейс системи застарілий, а процес налаштування і масштабування вимагає серйозних технічних

знань. Відсутність вбудованих інструментів аналітики вимагає додаткових модулів, що підвищує складність адміністрування.

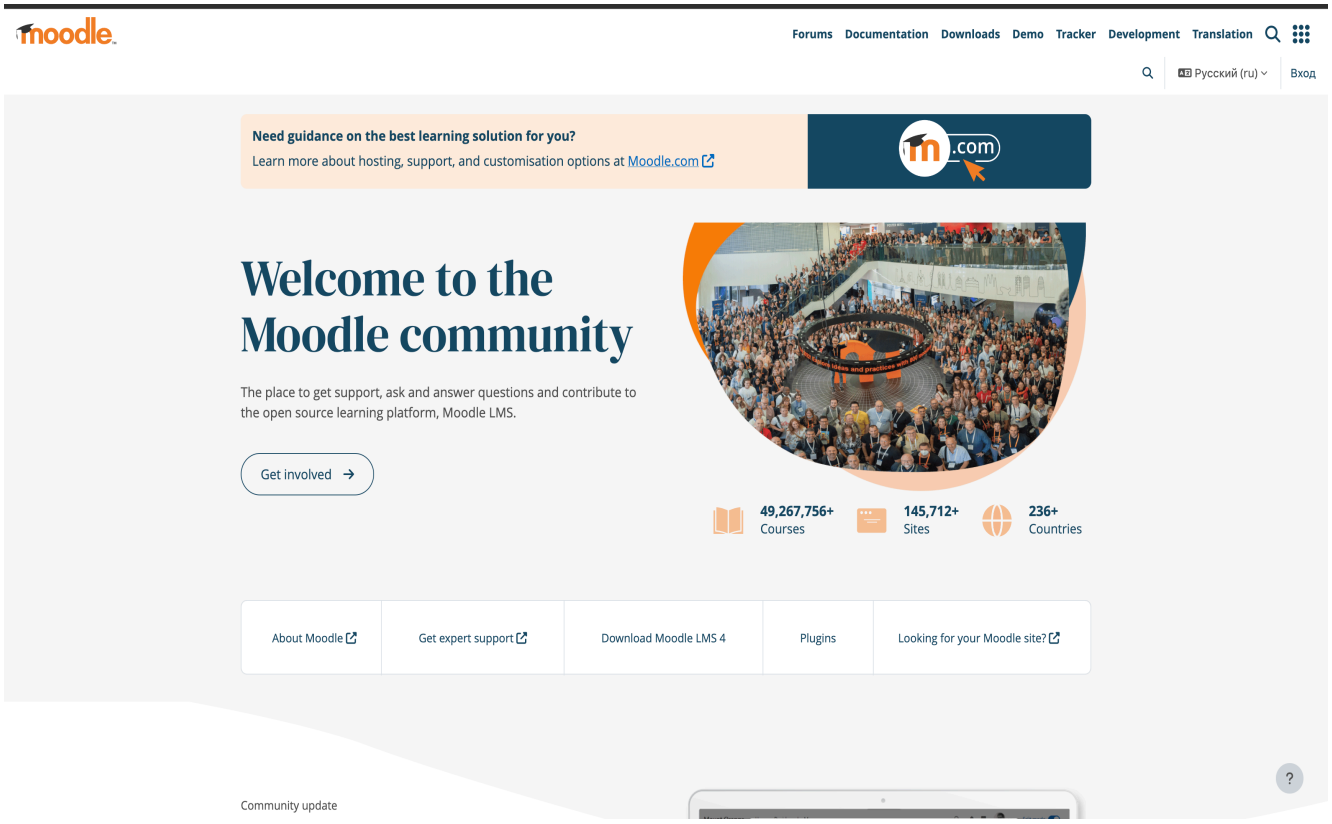


Рис. 1.1 Вигляд платформи Moodle

Moodle передусім орієнтована на управління освітніми курсами, пропонує гнучку структуру модулів, дозволяє інтегрувати різноманітні мультимедійні матеріали, забезпечує підтримку плагінів, а також має вбудовані інструменти для проведення онлайн-тестувань із можливістю їх автоматичної перевірки. Саме завдяки такій багатофункціональності Moodle часто стає першим вибором для закладів, які шукають універсальне рішення для організації навчального процесу у дистанційному або змішаному форматі.

Однією з найвагоміших переваг цієї системи є її безкоштовність для комерційного використання, що дає змогу навіть невеликим освітнім організаціям впроваджувати власні електронні курси, не витрачаючи значних коштів на ліцензії. Крім того, Moodle має дуже широку спільноту користувачів і розробників, яка постійно створює нові модулі, плагіни й інструменти для розширення функціоналу системи. Це суттєво спрощує життя адміністраторам та викладачам, оскільки практично будь-яке завдання вже має готове або хоча б частково реалізоване рішення.

Ще одним беззаперечним плюсом Moodle є його гнучкість у налаштуванні: кожен заклад може створити власну унікальну структуру курсів, змінити вигляд інтерфейсу, додати або прибрати ті чи інші елементи, підлаштувати систему під конкретні освітні завдання чи вимоги певної навчальної програми.

Проте, попри велику кількість позитивних сторін, Moodle має і низку недоліків, які не можна ігнорувати. Зокрема, досить часто користувачі нарікають на складність первинного налаштування платформи, особливо якщо мова йде про тих, хто не має ґрунтовного досвіду у сфері адміністрування серверів або веб-ресурсів. Інтерфейс системи, попри численні оновлення, все ще залишається досить архаїчним, що може створювати додаткові труднощі для студентів та викладачів, які звикли до сучасних інтуїтивно зрозумілих вебсервісів. Окремо слід згадати і про масштабування: якщо заклад планує використовувати Moodle для великої кількості користувачів, варто заздалегідь готуватися до потенційних проблем із продуктивністю, а також виділяти ресурси на технічну підтримку та супровід системи.

Ще одним аспектом, що може створювати незручності, є обмежений функціонал аналітики «з коробки». Для більшості навчальних закладів цього не вистачає, тому доводиться підключати додаткові модулі чи зовнішні сервіси, що у свою чергу ускладнює адміністрування та вимагає додаткових знань. Таким чином, Moodle, хоч і є потужним інструментом для впровадження дистанційної освіти, не позбавлений певних обмежень, і при виборі цієї системи слід уважно оцінити, наскільки її плюси та мінуси відповідають реальним потребам конкретної організації.

Google Classroom – це безкоштовна вебплатформа від Google, призначена для спрощення створення, розповсюдження та оцінювання завдань (рис 1.2). Ця система інтегрується з популярними сервісами Google (Drive, Docs, Meet) та орієнтована насамперед на спрощене взаємодію викладача і студента. Інтерфейс інтуїтивний, практично не потребує навчання, що значно знижує поріг входу для користувачів [11]. Основні недоліки полягають у відсутності розширеної ролевої моделі (мінімально відрізняються права студента та викладача) і недостатній

адаптації до стандартів вітчизняної освіти. Також залежність від екосистеми Google ставить під загрозу приватність даних та створює ризики при збої зовнішніх сервісів.

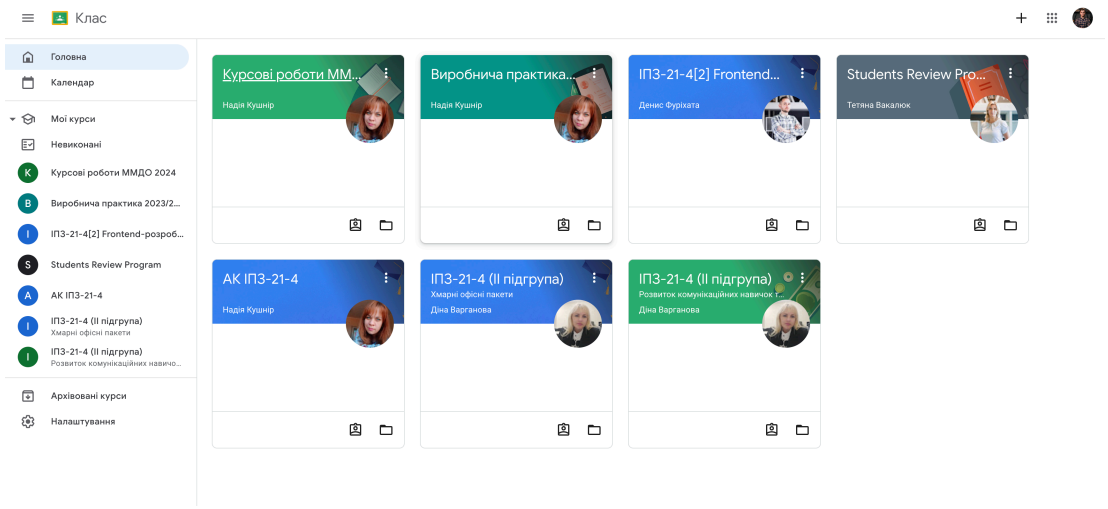


Рис. 1.2 Вигляд платформи Google Classroom

Ця система створена Google з чіткою метою – максимально спростити взаємодію між викладачем та студентом, зробити процес організації завдань і навчального матеріалу якомога зрозумілішим і доступним для всіх. По суті, Classroom позиціонується як такий собі «центральний хаб» для навчального процесу, де легко можна створити новий курс, призначити або роздати студентам завдання, організувати спільну роботу з документами, а також налагодити обмін повідомленнями між усіма учасниками освітнього процесу.

Від самого початку система орієнтована на простоту: інтерфейс тут дуже інтуїтивний, користувачам практично не потрібно нічого додатково вивчати чи витрачати час на ознайомлення з функціоналом. Багато хто відзначає, що Classroom не створює відчуття якогось складного, перевантаженого інструменту, – навпаки, усе розташовано логічно, а потрібні дії виконуються буквально у кілька кліків. Саме через це багато навчальних закладів, які не мають у своєму розпорядженні великих IT-ресурсів або технічної підтримки, вибирають цю платформу для швидкого старту роботи в онлайні.

Одна з основних переваг Google Classroom полягає у тісній інтеграції з усією екосистемою Google – це і Google Drive, і Google Docs, і Meet, і Календар. Завдяки цьому всі навчальні матеріали можна зберігати у хмарі, спільно редагувати

документи чи презентації, організовувати онлайн-зустрічі або ж отримувати сповіщення про зміни та нові завдання. Вся ця взаємодія відбувається непомітно для користувача, що ще більше підвищує зручність платформи, а також дає змогу максимально автоматизувати рутинні дії на кшталт збирання домашніх робіт чи розсилки сповіщень.

Однак варто звернути увагу на певні недоліки, з якими стикаються користувачі цієї системи. По-перше, функціонал Classroom досить обмежений, якщо порівнювати його з іншими повноцінними системами управління навчальним процесом. Тут немає можливості детально налаштовувати ролі чи права користувачів, що може бути критичним для великих університетів чи навчальних закладів зі складною ієрархією. По суті, права студента та викладача відрізняються мінімально, і це не дає змоги гнучко підлаштувати систему під специфіку певного навчального процесу або програму.

Ще однією особливістю, яка не завжди сприймається позитивно, є майже повна залежність від екосистеми Google. Для того щоб скористатися Classroom, потрібен обліковий запис Google, а це автоматично ускладнює або навіть унеможливлює використання платформи в тих випадках, коли з якихось причин доступ до сервісів Google обмежений, або є вимоги щодо приватності та захисту персональних даних. Залежність від зовнішніх сервісів несе ще один ризик: у разі збою чи недоступності серверів Google неможливо самостійно вплинути на ситуацію або оперативно відновити роботу платформи, а це може призвести до тимчасового зриву навчального процесу, що вже траплялося в окремих регіонах.

Окрім цього, Google Classroom лише частково адаптована до вітчизняних освітніх стандартів, що створює певні труднощі у процесі впровадження її в українських навчальних закладах, де часто виникає потреба у додаткових налаштуваннях або розширенні функціоналу. Тому, незважаючи на численні переваги, серед яких простота, безкоштовний доступ для освітніх установ і зручна інтеграція з популярними хмарними сервісами, Google Classroom не завжди може стати універсальним рішенням для всіх типів навчальних закладів і потребує уважного аналізу під час вибору системи управління навчальним процесом.

Canvas – це сучасна LMS, яка пропонує широкий спектр інструментів для управління навчальним процесом (рис 1.3). Позиціонується як корпоративне рішення з маркетинговою підтримкою й активною службою підтримки. Система пропонує сучасний, адаптивний інтерфейс, потужні засоби аналітики навчальних досягнень та інтеграцію з календарями й сторонніми LMS через LTI-протокол [12]. Проте вартість ліцензії для закладів вищої освіти значно перевищує середньоринкові пропозиції, що робить Canvas малодоступним для українських установ із обмеженим бюджетом.

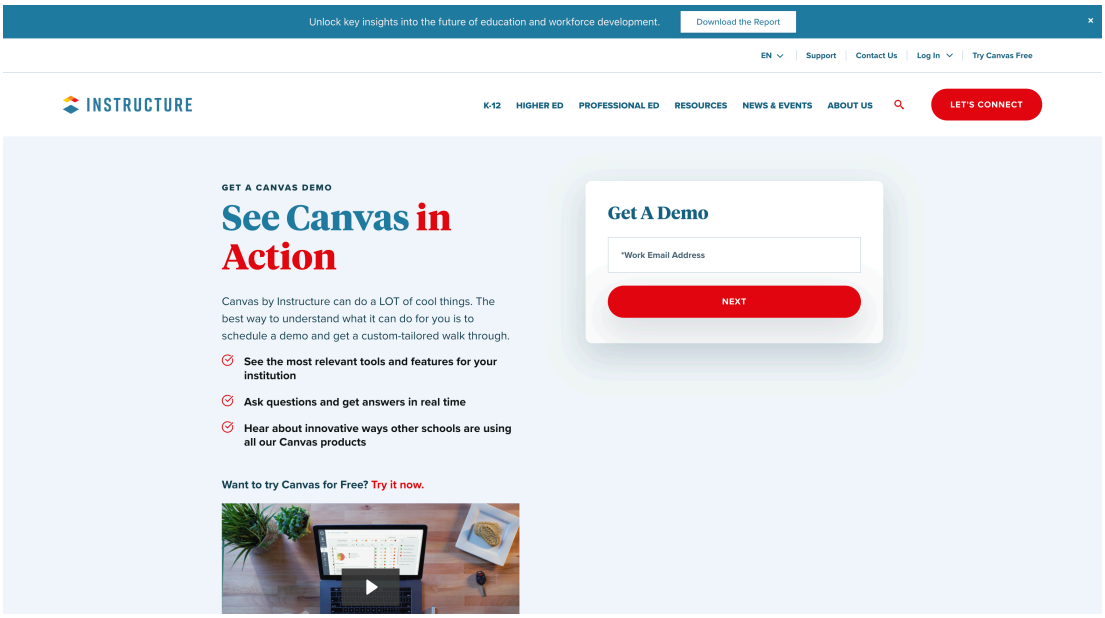


Рис. 1.3 Вигляд платформи Canvas

На перший погляд платформа вигідно вирізняється сучасним, добре продуманим інтерфейсом, який легко адаптується до різних пристроїв, що особливо зручно в умовах сучасної динаміки навчання, коли студенти часто користуються не лише комп’ютерами, а й телефонами чи планшетами.

Canvas надає дуже широкі можливості для адміністрування курсів, інтегрується з різними додатками, дозволяє викладачам не лише створювати навчальні матеріали, а й ефективно контролювати успішність студентів через розвинені аналітичні інструменти. Окремої уваги заслуговує і підтримка сучасних стандартів обміну даними між платформами (LTI-протокол), що дозволяє легко підключати зовнішні сервіси чи інші LMS, а також зручно використовувати календарі та сторонні навчальні застосунки. Не менш важливо, що для

користувачів розроблено повноцінний мобільний додаток, завдяки чому можна залишатися залученим до навчального процесу практично будь-де.

Серед переваг цієї системи варто виділити активну спільноту, доступ до служби підтримки, а також можливість налаштовувати платформу під індивідуальні вимоги закладу. Проте високий рівень функціональності та постійний розвиток мають і зворотний бік – вартість використання Canvas для закладів вищої освіти часто виявляється досить значною, що робить це рішення менш привабливим для українських установ із обмеженим фінансуванням.

Smart-School позиціонується як комплексне рішення для шкіл і університетів (рис 1.4). Система надає можливість створення курсів, ведення журналу успішності, розкладу занять та організації дистанційних уроків. Має інтеграція з електронними щоденниками та локальними базами даних школярів/студентів, що спрощує адміністрування. Інтерфейс оформлений у стилі «корпоративного порталу», що звично для викладачів, але менш привабливо для студентів, а отже потребує адаптації UX. Автоматизована перевірка тестів реалізована у базовому вигляді (мультивибір і короткі відповіді), без можливості завантажувати практичні завдання з оцінюванням коду чи складніших форматів [13]. Архітектура системи побудована на моноліті, що ускладнює її масштабування та інтеграцію з новими сервісами.

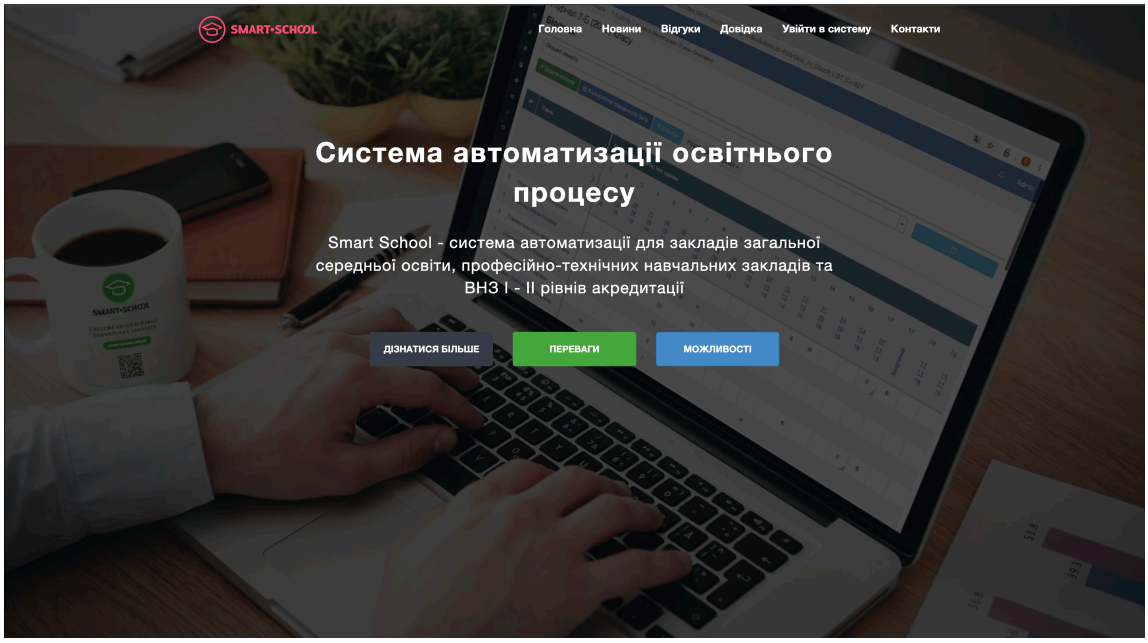


Рис. 1.4 Вигляд платформи Smart-School

Вона дає змогу створювати курси, вести електронний журнал успішності й відвідуваності, формувати розклад занять і навіть організовувати дистанційні уроки, що особливо актуально для сучасних реалій, коли навчання часто відбувається у змішаному форматі. Особливістю цієї системи є її глибока інтеграція з електронними щоденниками та локальними базами даних учнів і студентів, що значно спрощує адміністративну роботу й зменшує ризик втрати чи плутанини даних.

Інтерфейс Smart-School оформлений у стилі корпоративного порталу, до якого більшість викладачів досить швидко звикають, однак для студентів він може здаватися дещо перевантаженим і не завжди інтуїтивним. Це створює певну потребу в додатковій адаптації UX для молодших користувачів, які очікують більш легкого і сучасного дизайну. Щодо функціоналу тестування, то тут реалізовано базовий рівень автоматизованої перевірки – доступні варіанти з мультивибором і короткими відповідями, але якщо стоїть завдання приймати практичні роботи з перевіркою коду чи більш складних завдань, то таких можливостей у системі наразі немає.

Варто також звернути увагу на те, що архітектура Smart-School побудована як моноліт, а це з одного боку спрощує впровадження для невеликих шкіл чи закладів, але разом з тим суттєво ускладнює масштабування, особливо якщо згодом виникає потреба інтегрувати нові сервіси чи модулі. В результаті, попри широкий набір функцій, який дозволяє повністю закрити основні потреби навчального закладу – від ведення електронного журналу й формування звітів до організації дистанційного навчання та мобільного додатку для зручного доступу – користувачам може знадобитися певний час для освоєння всіх можливостей платформи. Ще однією особливістю є залежність від якості інтернет-з'єднання: якщо зв'язок нестабільний, це може вплинути на швидкість роботи системи, що особливо відчутно під час проведення онлайн-уроків чи масового введення даних. Водночас для вчителів і адміністрації Smart-School справді є зручним і функціональним інструментом, а служба технічної підтримки оперативно реагує на всі питання, допомагаючи вирішувати складні ситуації у процесі роботи.

Єдина школа – це українська електронна система, призначена для ведення електронних журналів та щоденників(рис 1.5). Розроблена на державному рівні для забезпечення освітнього процесу під час дистанційного навчання. Вона дозволяє проводити онлайн-уроки, опитування, завантажувати навчальні матеріали та вести сповіщення для батьків і учнів. Основні переваги – офіційна підтримка МОН, безкоштовний доступ для закладів та наявність мобільного додатку для Android і iOS. Недоліком є відсутність розвиненої рольової моделі (ролі «вчитель», «учень»; роль «адміністратор» дуже обмежена), слабка аналітика успішності та повна відсутність API для інтеграції з SIS/ERP університетів чи шкіл [14].

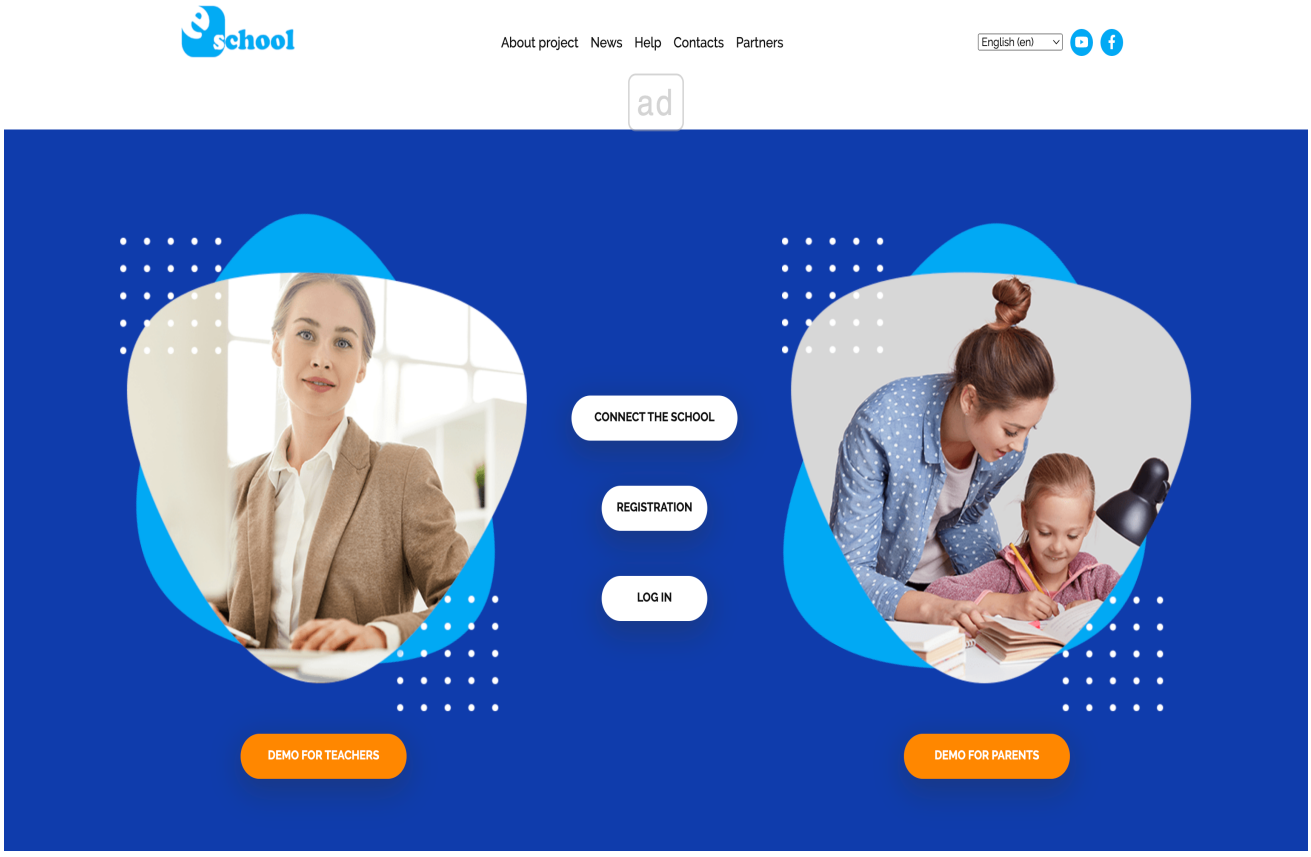


Рис. 1.5 Вигляд платформи Єдина школа

Саме тому її впровадження відбувалося на державному рівні з прямою підтримкою Міністерства освіти і науки України, і багато шкіл відчули переваги цієї централізованої платформи вже на перших етапах її використання.

Головна ідея цієї системи полягає у створенні єдиного цифрового простору, де зручно вести електронний журнал оцінок та відвідуваності, автоматично формувати розклад занять, завантажувати навчальні матеріали й організовувати

онлайн-уроки. Для учнів і їхніх батьків передбачено електронний щоденник, що дозволяє завжди бути в курсі актуальних новин, домашніх завдань та отримувати індивідуальні повідомлення від вчителів або адміністрації. Крім того, система автоматично враховує пропуски занять, що значно спрощує роботу класних керівників і дозволяє батькам своєчасно реагувати на прогули чи інші проблеми з відвідуваністю. Для адміністрації зручно, що Єдина школа може генерувати статистичні звіти та надавати елементарний аналіз успішності, хоча розвиненої системи аналітики тут все ж бракує.

Серед переваг цього рішення варто насамперед відзначити офіційну підтримку на рівні МОН, що гарантує безкоштовний доступ для всіх навчальних закладів та впевненість у довгостроковій перспективі функціонування. Окрім веб-версії, розроблено й мобільні додатки для Android та iOS, що дає можливість усім учасникам освітнього процесу – від учнів і батьків до вчителів та адміністрації – завжди залишатися на зв'язку й оперативно отримувати всю необхідну інформацію. Інтерфейс системи досить простий та зрозумілий, не викликає труднощів навіть у тих, хто не має значного досвіду користування подібними програмами.

Втім, як і будь-яка централізована державна система, Єдина школа не позбавлена своїх недоліків. Перш за все, тут відчутно обмежена рольова модель – фактично є лише ролі «вчитель» та «учень», а роль адміністратора доволі формальна і майже не дає можливостей для налаштувань чи глибокого керування. Це, звісно, створює труднощі для великих шкіл або ліцеїв, де потрібна гнучкість у розподілі прав доступу чи впровадженні додаткових адміністративних функцій. Крім цього, платформа орієнтована виключно на шкільний сегмент, і навіть попри заявлену інтеграцію з іншими освітніми системами України, можливість кастомізації під специфіку певного закладу дуже обмежена. Ще однією значною втратою є відсутність публічного API для інтеграції з внутрішніми інформаційними чи фінансовими системами вишів, що практично унеможливлює використання Єдиної школи у сфері вищої освіти або розширення функціоналу для особливих потреб.

Підсумовуючи, можна сказати, що Єдина школа вирішує базові завдання організації навчального процесу для українських шкіл, значно автоматизує рутинну

роботу викладачів та адміністрації, дозволяє оперативно інформувати учнів і батьків, але разом із тим лишається досить закритою системою з мінімальними можливостями для адаптації та інтеграції. Порівняння з іншими системами, які використовуються в Україні та за кордоном, показує, що більшість із них або фокусуються винятково на навчальному процесі, або ж на фінансово-документальній частині, а повноцінне поєднання усіх цих функцій в одній екосистемі наразі лишається радше винятком, а не правилом.

Основними перевагами сучасних LMS-систем, таких як Moodle, Google Classroom та Canvas, є доступність, широкі можливості дистанційного навчання та велика спільнота підтримки. Водночас вони мають певні недоліки, зокрема складність розгортання, обмежені можливості кастомізації та потребу в інтеграції з іншими системами.

Електронні журнали, такі як Smart-School та Єдина школа, добре вирішують питання управління академічними ресурсами в межах шкільної освіти, але мають обмежені функції для студентів вищих навчальних закладів.

Враховуючи наявні недоліки аналогічних продуктів, запропонована система управління академічними ресурсами студентів повинна об’єднувати в собі всі необхідні модулі (реєстрацію, управління навчальним процесом, документообіг, підтримку студентів та інші віажливі модулі) у єдиній екосистемі. Це дозволить забезпечити комплексний підхід до автоматизації управління академічними ресурсами та створити ефективну та зручну систему для студентів, викладачів та адміністрації.

У підсумку, ретельний аналіз існуючих LMS дозволив виокремити пріоритетні напрями для розробки локалізованого, безпечного та зручного рішення, яке відповідатиме потребам українських університетів за співвідношенням “функціональність – вартість – легкість підтримки”. Зроблений акцент на гнучкості й адаптивності системи, що дає змогу враховувати специфіку організаційної структури й освітніх процесів кожного ЗВО. Розроблене рішення дозволяє університетам оперативно впроваджувати сучасні цифрові інструменти й

покращувати якість освітнього середовища без зайвих витрат і технічних ускладнень.

Таблиця 1.1

Порівняльна таблиця існуючих програмних продуктів

Платформа	Ліцензія	Ролева модель	Автооцінювання	Аналітика	Інтеграція SIS/ERP	Мобільна підтримка	Вартість впровадження
Moodle	Open-source	Розширена (плагіни)	Плагіни	Базова (плагіни)	Через плагіни	Часткова	Низька
Google Classroom	Безкоштовно	Дво-рівнева	Обмежене	Відсутня	Лише Google API	Повна	Нульова
Canvas	Freemium / ПП	Розвинена	Вбудоване	Розвинена	LTI	Повна	Висока
Blackboard	ПП	Гнучка	Вбудоване	Розвинена	SOAP/REST	Повна	Дуже висока
Edmodo	Freemium	Мінімальна	Вбудоване	Відсутня	Відсутня	Повна	Низька
Sakai	Open-source	Розвинена	Вбудоване	Базова	REST	Часткова	Низька
Brightspace	ПП	Розвинена	AI-підсилене	Централізована	REST/LTI	Повна	Висока
Smart-School	ПП	Середня	Базове	Обмежена	Локальна БД	Часткова	Середня
Єдина школа	Держ. безкоштовно	Мінімальна	Базове	Відсутня	Відсутня	Повна	Нульова

На підставі проведеного аналізу аналогів, наша «Система управління академічними ресурсами студентів» повинна закрити низку ключових проблем, які не повною мірою вирішені існуючими рішеннями:

Перш за все, користувачу необхідно мати все в одному місці – контент, комунікацію, адміністративні інструменти та аналітику без переходу між різними сервісами. Система мусить забезпечити чітко виокремлені ролі (студент, викладач, адміністратор) з гнучким призначенням прав, можливість формування і ведення курсів, а також створення та проведення тестів і завдань різних типів із автоматизованою перевіркою.

Додатково важливо реалізувати систему, управління документами (сертифікати, довідки, документи), а також вбудовані чати для спілкування й нотифікації про дедлайни та новини. Кожен користувач має мати свій персональний кабінет зі зрозумілим інтерфейсом, де відображаються його курси, завдання, оцінки, календар подій і стрічка новин та подій.

Ще одним важливим аспектом є інтерактивність – можливість ведення вебінарів, інтеграція з мультимедійними ресурсами й офлайн-режим для слабких каналів зв'язку. Окрім того, сучасні вимоги передбачають застосування штучного інтелекту для персоналізації навчальних траєкторій, аналізу успішності та рекомендацій освітніх матеріалів.

Нарешті, система повинна включати модулі підтримки користувачів (help-desk, FAQ) та механізми технічного супроводження, щоб заклад міг оперативно реагувати на запити та забезпечувати безперебійну роботу платформи.

Задовольнивши перелічені вимоги, ми отримаємо комплексне рішення, що об'єднає всі необхідні інструменти для організації сучасного і ефективного освітнього процесу в українських закладах освіти.

1.3 Вибір архітектури для система управління академічними ресурсами

Проєктування архітектури інформаційної системи є фундаментальним етапом розробки програмного забезпечення, від якого залежить гнучкість, масштабованість, безпека, ефективність обробки даних та зручність подальшої підтримки. Особливо це актуально у випадку розробки складних, багатофункціональних платформ, таких як система управління академічними ресурсами студентів. Така система повинна не лише надавати стабільний доступ до навчальних курсів, тестування, документів та персональних кабінетів, а й забезпечувати можливість взаємодії великої кількості користувачів з різними рівнями доступу, підтримувати динамічну генерацію та перевірку даних, інтегруватися з іншими сервісами та бути готовою до майбутнього масштабування.

На сучасному етапі існує низка підходів до побудови архітектурних рішень: від класичної монолітної моделі до гнучких мікросервісів і безсерверних обчислень. Кожен з підходів має як сильні сторони, так і певні обмеження, тому перед прийняттям остаточного рішення необхідно провести всебічний аналіз з урахуванням вимог проєкту, ресурсоємності реалізації, обсягу даних, очікуваної кількості користувачів та можливостей команди розробки [15].

Монолітна архітектура передбачає єдиний виконуваний модуль, який містить інтерфейс користувача, бізнес-логіку та доступ до даних в одному пакеті. Така модель спрощує початкову розробку: усі компоненти розгортаються й тестуються разом, що зменшує витрати часу на налагодження простих сценаріїв[15]. Водночас, з ростом функціональності зростають і взаємозалежності модулів, ускладнюючи підтримку, знижуючи швидкість розробки та обмежуючи можливості масштабування окремих частин системи [16].

Хоча монолітна архітектура підходить для невеликих проєктів з обмеженим набором функцій, для системи управління академічними ресурсами, що передбачає розширення та інтеграцію з різними сервісами, її жорстка структура створить значні труднощі при масштабуванні та адаптації нових модулів.

У **SOA** система розбивається на набір незалежних сервісів із чітко визначеними інтерфейсами, які взаємодіють через шину ESB (Enterprise Service Bus) [17]. Основні переваги полягають у високому рівні повторного використання модулів та полегшеному обслуговуванні: сервіси можна оновлювати без втручання в інші компоненти, а платформна незалежність дозволяє поєднувати різні технології. Проте інтеграція через ESB ускладнює інфраструктуру, знижує продуктивність через проміжний шар і може стати єдиною точкою відмови [18].

SOA доцільна в великих підприємствах із наявними legacy-системами, але для університетської платформи з обмеженими ресурсами її складність та накладні витрати на підтримку ESB роблять цей підхід малоефективним.

Мікросервіси передбачають поділ системи на невеликі, незалежні служби, кожна з яких відповідає за окрему бізнес-функцію і взаємодіє через легкі протоколи [19]. Така декомпозиція підвищує гнучкість розробки, дозволяючи командам

працювати автономно над окремими сервісами та прискорювати релізи [20]. Водночас зростає загальна складність: потрібно управляти десятками або сотнями інстансів, забезпечувати узгодженість даних між сервісами, вирішувати питання транзакцій у розподіленому середовищі й впроваджувати розвинуті DevOps-процеси [19].

Мікросервісна архітектура є потужною, проте надмірно складною для початкової реалізації проєкту з обмеженими командами та ресурсами. Вона може бути доречною на етапі масштабування, але спочатку слід обмежитися простішим підходом.

У **безсерверній** моделі бізнес-логіка реалізується у вигляді незалежних функцій (FaaS), які виконуються за подіями без необхідності керувати серверами [21]. Переваги включають, автоматичне масштабування й мінімізацію операційних зусиль з управління інфраструктурою [22]. Однак холодні старти можуть призвести до непередбачуваних затримок, розробка та відлагодження ускладнюються через відсутність прямого доступу до середовища, а залежність від провайдера створює ризики vendor lock-in [21].

Безсерверна модель підходить для точкових, краткочасних функцій, але не для комплексної освітньої платформи з великими обсягами даних і довготривалими процесами.

Клієнт-серверна модель спирається на чіткий розподіл: сервер опрацьовує запити, зберігає дані та реалізує бізнес-логіку, а клієнт відповідає за презентацію та взаємодію з користувачем через RESTful API [23]. Таке рішення забезпечує централізоване управління безпекою, резервне копіювання даних і єдину точку оновлення серверної частини, одночасно дозволяючи створювати різні клієнтські інтерфейси (веб, мобільні додатки) без модифікації бекенду [23]. Головними недоліками є потреба в надійному серверному оточенні з резервуванням та захистом від перевантажень, а також значні початкові витрати на інфраструктуру й кваліфіковану підтримку [24].

Клієнт-серверна архітектура найкраще відповідає вимогам проєкту: вона поєднує зрозумілу модель розвитку, централізоване управління безпекою та

даними, простоту інтеграції з різними клієнтами та передбачуване масштабування, що робить її оптимальною для реалізації системи управління академічними ресурсами студентів [25, 26].

Проведений аналіз дозволив виявити, що монолітна архітектура забезпечує простоту розгортання та тестування, але обмежує масштабованість й ускладнює підтримку великих систем, тоді як SOA пропонує переваги в повторному використанні сервісів та легкому обслуговуванні, проте страждає від складності ESB та продуктивності. Мікросервісний підхід гарантує високий рівень масштабованості й відділення компонентів, але створює значні DevOps-накладні витрати та проблеми синхронізації даних. Безсерверна модель мінімізує витрати на інфраструктуру і автоматично масштабується, проте вводить обмеження на тривалі процеси та ризики vendor lock-in [27]. Клієнт-серверний підхід надає чіткий розподіл обов'язків між фронтендом та бекендом, простоту інтеграції через REST і централізоване управління безпекою, що відповідає потребам розроблюваної платформи.

Отже, після всебічного аналізу можливих архітектурних підходів, врахування потреб кінцевих користувачів та функціональних вимог до системи, було прийнято рішення реалізувати платформу за класичною клієнт-серверною архітектурною моделлю з можливістю подальшого масштабування на основі мікросервісного підходу. Обрана модель забезпечує чіткий розподіл обов'язків між фронтендом і бекендом, дає змогу централізовано керувати безпекою, логікою, а також спрощує обслуговування та розширення системи.

Архітектура системи управління академічними ресурсами має бути модульною, масштабованою, розширюваною та безпечною. Основним принципом проєктування є поділ відповідальностей, що дозволяє розділити логіку додатку на незалежні частини з чітко визначеними обов'язками. Це дає змогу ефективно підтримувати код, тестувати окремі частини, а також швидко масштабувати проєкт при збільшенні навантаження.

Одним з основних підходів є принцип Separation of Concerns (SoC), що передбачає розподіл бізнес-логіки, логіки представлення та логіки зберігання даних у

різні компоненти [28]. Також застосовується принцип Single Responsibility, за яким кожен модуль або клас відповідає лише за одну задачу, що значно полегшує обслуговування та розвиток системи.

Основним патерном, який реалізується у цій системі, є Client-Server, де клієнтська частина відповідає за взаємодію з користувачем, а серверна частина – за обробку даних, логіку роботи та зберігання інформації [29]. Такий підхід дозволяє централізувати обробку даних та спростити підтримку клієнтів через універсальні API.

У межах серверної частини використовується архітектурний стиль MVC (Model-View-Controller) – хоча частково. У нашому випадку він проявляється у поділі на моделі (які взаємодіють із базою даних), сервіси (логіка), контролери (обробка запитів) та схеми представлення (через API) [30].

Додатково застосовується патерн Repository, що абстрагує доступ до бази даних, і Service Layer, що дозволяє відокремити бізнес-логіку від API-рівня. Це сприяє побудові чистої архітектури та полегшує тестування та масштабування.

Клієнтська частина сучасної інформаційної системи являє собою той рівень взаємодії, з яким безпосередньо працює користувач – будь-то через звичайний веббраузер на комп’ютері чи мобільний застосунок на смартфоні. Її головне завдання – зробити доступ до можливостей системи максимально зручним, швидким і зрозумілим, щоб кожна дія виконувалася без затримок, а всі елементи інтерфейсу були логічно розташовані й легко сприймалися навіть тими, хто вперше знайомиться з продуктом. У цьому контексті, як правило, обирають підхід SPA (Single Page Application), коли користувач завантажує основний інтерфейс один раз, а вся подальша взаємодія із сервером відбувається у фоновому режимі – шляхом динамічних запитів, без повного перезавантаження сторінок. Це суттєво підвищує відчуття «живості» застосунку та економить трафік, що особливо помітно при роботі з мобільних пристроїв [31].

Побудова сучасного фронтенду зазвичай ґрунтується на ідеї розділення на окремі, незалежні компоненти, кожен з яких відповідає за свою частину інтерфейсу чи логіки. Таке розбиття значно спрощує розробку й тестування, дозволяє різним

розробникам працювати паралельно та зменшує кількість помилок. Для централізованого управління станом застосунку часто використовуються спеціалізовані бібліотеки, такі як Redux або Context API – вони допомагають зберігати узгодженість даних на всіх рівнях додатку [32]. Окрім компонентів, у проекті чітко розділяються сторінки, утиліти, окремо виділяються файли для опису запитів до API, що у підсумку призводить до чистішої й зрозумілішої структури коду.

Вся комунікація клієнта із серверною частиною відбувається через REST API – це стандартний підхід, коли всі необхідні дані надсилаються у форматі JSON, а передача здійснюється по захищеному протоколу HTTP(S). Сервер зі свого боку приймає ці запити, перевіряє коректність вхідних даних, виконує відповідну логіку, взаємодіє з базою даних і повертає відповідь, також у вигляді JSON-структури. Для побудови такої системи на сервері зазвичай виділяють окремі частини: контролери, які відповідають за отримання та маршрутизацію запитів; сервіси, де міститься бізнес-логіка; репозиторії, що працюють безпосередньо з базою; а також спеціальні схеми для валідації даних і моделі, які описують структуру збереження інформації. Такий підхід дозволяє не тільки зберігати код впорядкованим, а й швидко масштабувати й доповнювати систему новим функціоналом, не порушуючи цілісності основної архітектури [33, 34].

Для забезпечення реєстрації, автентифікації та авторизації в системі використовуються сучасні підходи, зокрема технологія JWT-токенів (JSON Web Token) [35]. Це дозволяє не лише надійно захищати дані користувачів під час передавання між клієнтом і сервером, а й реалізовувати гнучке управління правами доступу. Кожен користувач після проходження процедури автентифікації отримує власний токен, в якому закодована інформація про його роль у системі – чи це студент, викладач, чи адміністратор. Завдяки цьому сервер під час кожного запиту може швидко ідентифікувати користувача та перевірити, чи має він відповідні повноваження для виконання тієї чи іншої дії. Це спрощує логіку контролю доступу й підвищує загальний рівень безпеки, адже всі критичні операції виконуються з урахуванням ролей, визначених у токені.

Окрім цього, для підтримки функціоналу, який потребує обміну даними в режимі реального часу – наприклад, для чату між студентами та викладачами чи для системи оперативних сповіщень про нові завдання або зміни в розкладі – доцільно використовувати WebSocket-з'єднання [36]. Завдяки бібліотекам на кшталт FastAPI WebSockets забезпечується можливість встановлення постійного двостороннього каналу між клієнтом і сервером, що дозволяє передавати повідомлення моментально, без затримок, і оперативно реагувати на події, які виникають у системі. Такий підхід особливо корисний у складних освітніх системах, де важливо підтримувати постійний зв'язок між усіма учасниками процесу та швидко інформувати їх про зміни.

Важливим архітектурним рішенням при розробці стало використання модульного підходу, коли кожен функціональний блок системи реалізується як окремий модуль із чітко визначеною відповідальністю [37]. Це не лише дозволяє розподілити роботу між різними членами команди розробників, але й спрощує подальше масштабування проєкту та полегшує підтримку – при додаванні нового функціоналу не потрібно вносити зміни у вже існуючі модулі, що знижує ризик виникнення помилок. Такий підхід сприяє дотриманню принципу інкапсуляції: внутрішня реалізація кожного модуля прихована від інших частин системи, вони взаємодіють між собою лише через чітко визначені інтерфейси. Це підвищує стійкість архітектури до змін і робить систему більш надійною в експлуатації.

У процесі розробки ключовою ідеєю є дотримання принципів Clean Architecture [38]. Це означає, що вся логіка застосунку розділена на окремі шари – інфраструктурний, логічний, презентаційний і шар зберігання даних. Кожен із цих рівнів має власну сферу відповідальності: інфраструктурний шар відповідає за взаємодію з зовнішніми сервісами, базою даних або зовнішніми API; логічний шар містить основні правила й алгоритми роботи системи; презентаційний забезпечує відображення даних та взаємодію з користувачем, а шар зберігання організовує роботу з даними та їх збереження (рис. 1.6). Таке розділення дозволяє змінювати або розширювати окремі частини системи без ризику порушити цілісність усього застосунку, що особливо цінно при розвитку складних, довготривалих проєктів. Крім

того, подібний підхід полегшує тестування, обслуговування й інтеграцію нових функцій, підвищуючи загальну надійність і якість програмного продукту.

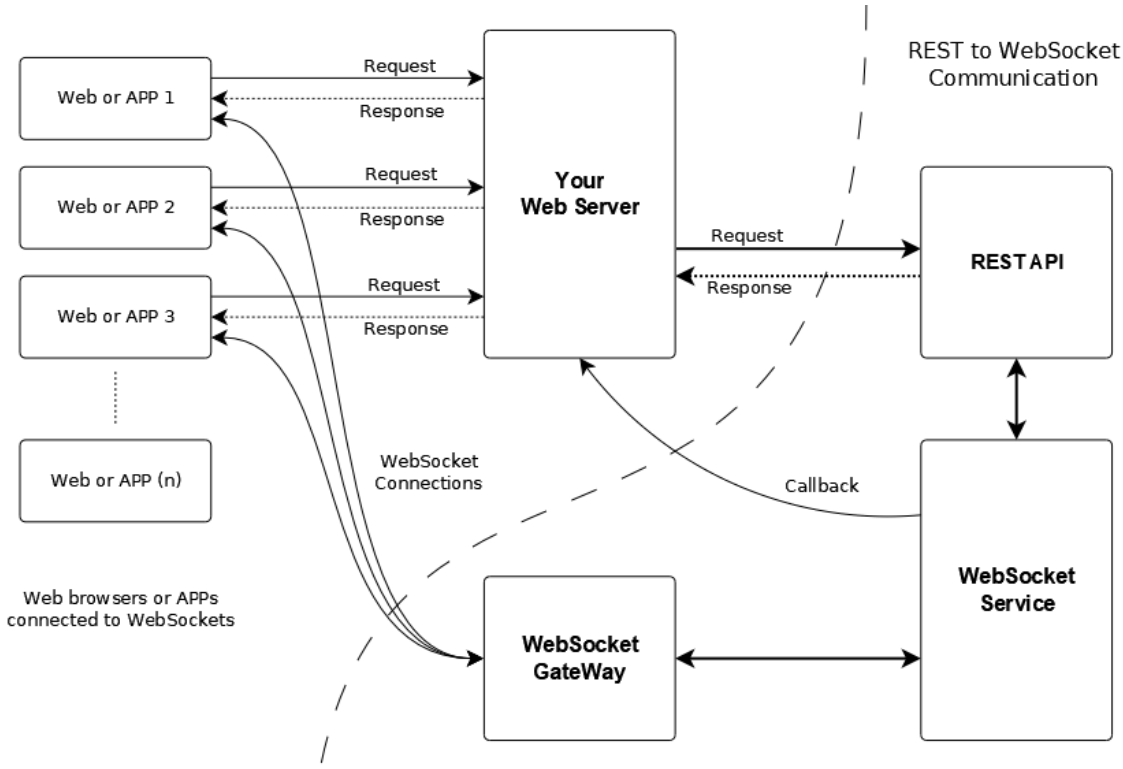


Рис. 1.6. Схема взаємодії RESTfull API + WebSocket

1.4 Обґрунтування вибору інструментальних засобів.

Побудова сучасної системи управління академічними ресурсами вимагає ретельного вибору як загальної архітектурної парадигми, так і конкретних технологій, які дозволяють досягти високої масштабованості, продуктивності, безпеки та адаптивності до майбутнього розвитку. В основі системи лежить класична клієнт-серверна модель, що демонструє свою ефективність в більшості сучасних веб-систем. Обмін між клієнтом і сервером відбувається через REST API з підтримкою WebSocket для реального часу.

У якості основної мови серверної частини обрано Python. Це рішення ґрунтується на декількох ключових перевагах. По-перше, Python має багатий набір бібліотек для роботи з вебом, обробки даних, авторизації та машинного навчання. По-друге, він є простим у вивченні й читанні, що знижує поріг входу нових учасників у команду, покращує підтримку коду й зменшує кількість помилок [39].

Хоча альтернативи – такі як Node.js (JavaScript) або Java – можуть демонструвати кращу продуктивність в деяких специфічних сценаріях, Python вирізняється гнучкістю, доступністю та активною спільнотою, що забезпечує стабільну підтримку і швидке вирішення проблем. На відміну від Java, Python дозволяє скоротити обсяг шаблонного коду, а в порівнянні з Node.js – уникнути складнощів з асинхронним кодом, завдяки природній підтримці асинхронності в FastAPI [40].

Серед численних Python-фреймворків, таких як Django, Flask, Tornado або Pyramid, обрано FastAPI, як найкращий варіант для побудови асинхронних RESTful API з високою продуктивністю. Основні переваги FastAPI включають:

- Асинхронна підтримка natively через async/await, що дозволяє обробляти тисячі одночасних запитів.

- Автоматична генерація документації Swagger та Redoc без додаткового налаштування.

- Типізація на базі Pydantic забезпечує перевірку вхідних та вихідних даних і мінімізує помилки.

- Модульність: на відміну від більш монолітного Django, FastAPI дозволяє легко організовувати проект у вигляді окремих логічних модулів [41].

Flask, хоч і добре підходить для невеликих застосунків, вимагає значно більше ручного конфігурування при роботі з API, авторизацією, CORS, документацією та валідацією даних, у порівнянні з FastAPI.

У основі архітектури лежить розділення на клієнтську та серверну частини, які спілкуються між собою за допомогою REST API. Сервер відповідає за обробку бізнес-логіки, зберігання та захист даних, керування автентифікацією, авторизацією, логуванням і взаємодією з базою даних.

Клієнтська частина взаємодіє із сервером як споживач API, відображаючи дані користувачу та забезпечуючи інтерфейс взаємодії.

Центральною ланкою серверної частини є вебфреймворк FastAPI, який дозволяє будувати високопродуктивні веб-застосунки мовою Python. Завдяки асинхронній природі FastAPI забезпечується висока швидкодія при великій кількості

одночасних запитів. Система побудована на принципах REST-архітектури, з чітким розмежуванням маршрутів, методів запитів (GET, POST, PUT, DELETE) та структурою відповідей у форматі JSON. Це гарантує зрозумілість і масштабованість у майбутньому розвитку [41].

В якості бази даних обрано PostgreSQL – сучасну реляційну СУБД з відкритим кодом, яка забезпечує ACID-сумісність, багаторівневу транзакційність, гнучкі засоби індексації та потужні можливості для складного запиту даних [42]. PostgreSQL дозволяє ефективно працювати з відношеннями між об’єктами, що критично важливо для побудови ієрархічних структур курсів, модулів, завдань і пов’язаних сутностей у системі.

Клієнтська частина розробляється з використанням JavaScript-фреймворку React, що дозволяє реалізувати інтерфейс з високим рівнем інтерактивності, швидкою реакцією на дії користувача та сучасними UX-рішеннями. React-компоненти забезпечують повторне використання елементів інтерфейсу, що спрощує підтримку і масштабування коду. Клієнтська частина будується за патерном Flux/Redux для централізованого керування станом.

Застосунок спроектовано на основі патерну "шарова архітектура" (Layered Architecture), що передбачає поділ логіки на кілька рівнів: рівень представлення (інтерфейс), бізнес-логіка, рівень доступу до даних. Такий підхід сприяє слабкому зв’язуванню модулів, забезпечуючи їх ізольованість і зручність тестування.

Окремі модулі системи реалізуються відповідно до принципів SOLID, що підвищує читаємість і повторне використання коду. Наприклад, модуль автентифікації реалізує механізм JWT-токенів, двофакторну перевірку, обмеження доступу на основі ролей (рольова модель RBAC), а модуль курсів – CRUD-операції з валідацією, пов’язані з викладачами, студентами та структурою матеріалів.

Особлива увага приділяється використанню патерну «Депенденсі Інверсія» (Inversion of Control), що реалізується за допомогою механізму залежностей, вбудованого у FastAPI. Це дає змогу підключати різні сервіси, репозиторії та допоміжні утиліти через інтерфейси, не створюючи жорстких зв’язків між компонентами. Завдяки цьому кожен модуль можна замінити або протестувати

ізольовано від інших частин системи, що особливо важливо при масштабуванні та подальшому розвитку проєкту [43, 44]. Такий підхід знижує технічний борг, спрощує впровадження нових функцій і забезпечує стабільність архітектури при зміні бізнес-вимог.

Для ізоляції середовищ розробки, тестування та продуктивного оточення використовуються Docker-контейнери. Кожен сервіс – сервер додатку, база даних, інструменти черг повідомлень (Redis), модулі аналітики – працює у власному ізольованому середовищі, що дозволяє гнучко керувати залежностями, масштабувати окремі компоненти та швидко відновлювати систему у разі збоїв. Контейнеризація спрощує процес розгортання як на власних серверах, так і в хмарних інфраструктурах, знижуючи ризик виникнення проблем із «несумісністю оточення» [45]. Для локального запуску та розробки використовується Docker Compose, який дозволяє описати всі необхідні сервіси й налаштування в одному конфігураційному файлі. У перспективі, з ростом навантаження й кількості сервісів, передбачено інтеграцію з Kubernetes, що відкриває можливості для автоматичного масштабування, балансування навантаження, самовідновлення сервісів і централізованого моніторингу [46].

Процеси безперервної інтеграції та доставки (CI/CD) організовано за допомогою сучасних систем – GitLab CI. Кожен коміт у репозиторій автоматично ініціює процес збирання, тестування та деплойменту контейнерів у тестове або продуктивне середовище. Це мінімізує людський фактор, пришвидшує релізи, допомагає відслідковувати помилки ще на ранніх етапах та підтримує стабільність розгортання [47].

Захист персональних даних студентів, викладачів та адміністраторів є ключовим пріоритетом системи. Реалізовано автентифікацію на основі JWT, перевірку прав доступу на кожному рівні, захист від CSRF та XSS-атак, а також обмеження частоти запитів до API (rate limiting) для запобігання зловживанням і спробам brute-force. Усі спроби доступу до критичних ресурсів логуються для подальшого аналізу. Для захисту паролів використовуються сучасні алгоритми хешування – bcrypt, а передача конфіденційної інформації між клієнтом і сервером здійснюється

виключно через захищені протоколи HTTPS/TLS [48]. Додатково впроваджується можливість шифрування найчутливіших даних у базі, що підвищує загальний рівень інформаційної безпеки.

Проєкт спочатку розробляється з урахуванням можливості горизонтального масштабування. Після реалізації MVP фази заплановано поступове винесення найбільш навантажених або незалежних функціональних блоків у окремі сервіси – наприклад, підсистему тестування, чат, календар чи аналітику. Такий мікросервісний підхід дозволяє окремо масштабувати потрібні компоненти, забезпечує більшу відмовостійкість і гнучкість при зміні бізнес-логіки.

Для роботи у режимі високого навантаження реалізується кешування найчастіших запитів із використанням Redis, що дозволяє значно зменшити час відгуку та знизити навантаження на основну базу даних. Складні чи ресурсоємні завдання – наприклад, генерація PDF-документів, аналітичні розрахунки чи обробка великих файлів – відправляються на виконання у фонові воркери через систему черг Celery. Це гарантує стабільну роботу основного API та дозволяє ефективно розподіляти ресурси між різними частинами системи [49].

1.5 Командна розробка та організація спільного процесу

Ефективна командна розробка - це не лише формальний розподіл завдань, а насамперед постійний діалог, гнучка адаптація до нових викликів і створення атмосфери довіри та спільної відповідальності. На початку роботи над проєктом ми домовилися про чіткий і прозорий розподіл функціональних блоків. Це дозволило уникнути дублювання коду, чітко розподілити зони контролю, а також мінімізувати ризик появи “мертвих зон”, коли окремі важливі задачі залишаються без власника.

Практично це виглядало так: ще на етапі планування ми разом визначили, хто з учасників за що відповідає, і внесли це до системи управління задачами Jira. В подальшому це дало змогу кожному зосередитися на розвитку окремих частин системи, не витрачаючи час на хаотичне перемикання контексту.

У моїй зоні відповідальності були такі ключові напрями, як розробка системи авторизації, активації акаунтів, підтримка багатьох типів користувачів (адміністратор, студент, викладач), впровадження двофакторної автентифікації (TOTP), створення модуля чатів для внутрішньої комунікації, організація механізму активних сесій, а також реалізація викладацького журналу й системи аудиту дій користувачів. Якщо під час роботи виникали перетини в структурі спільних моделей чи схем даних, ми оперативно обговорювали деталі, спільно переглядали код і вносили необхідні корективи.

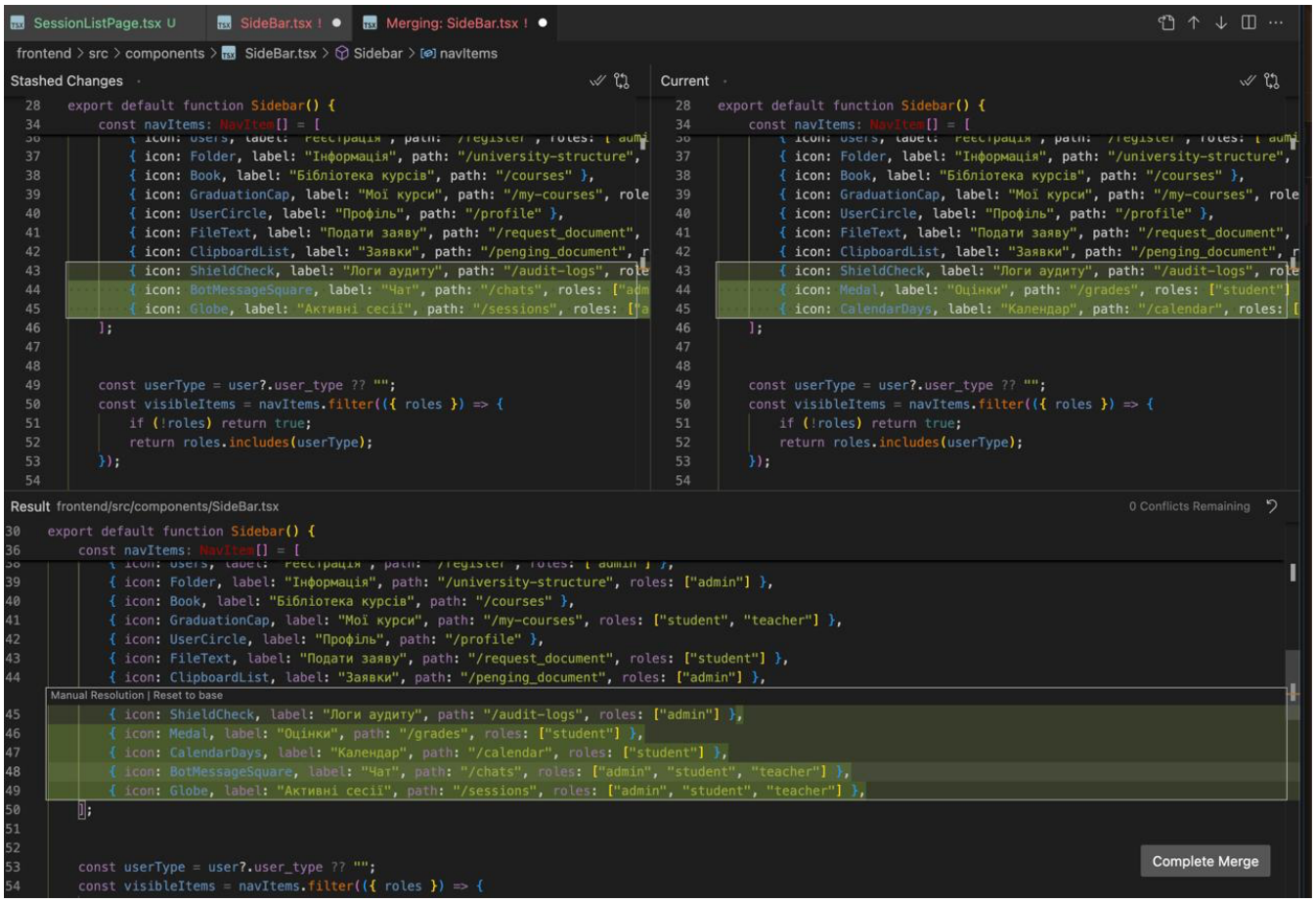


Рис. 1.7 – Приклад вирішення іншого конфлікту в IDE VSCode

Одним із ключових технічних рішень стало розгортання спільної реляційної бази даних на окремому сервері, до якої мали доступ усі члени команди. Це забезпечило інтеграцію різних компонентів у реальних умовах (не лише локально), усунуло типові проблеми з розсинхронізацією локальних баз даних і дозволило швидко масштабувати систему. Зміни у структурах моделей, схемах чи міграціях ми попередньо обговорювали всією командою - навіть якщо це стосувалося

дрібних деталей. Така дисципліна допомогла уникати технічних конфліктів, неочікуваної втрати даних і підтримувати відповідальний підхід до розробки(рис. 1.8).

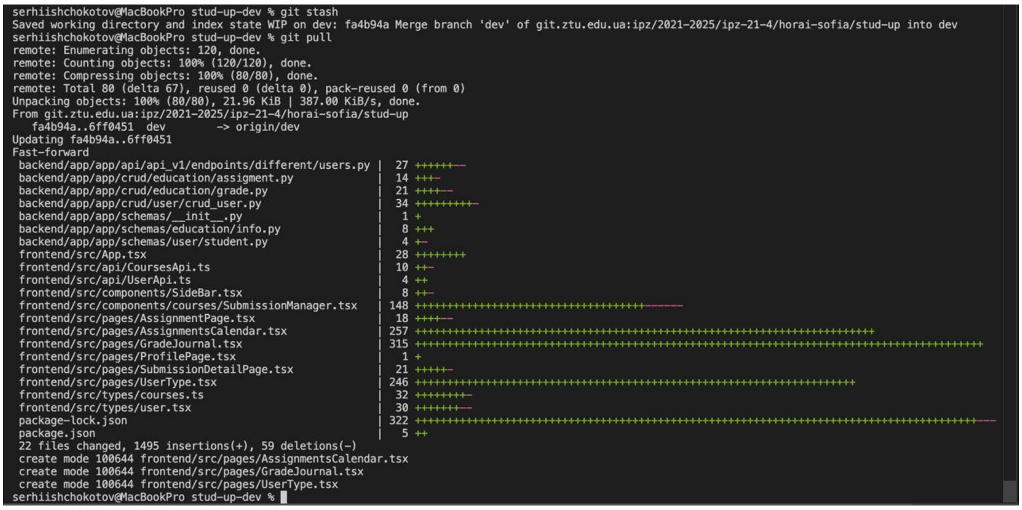


Рис. 1.8 – Результат виконання git stash та git pull команд

Не менш важливою була організація контролю версій за допомогою git, адже це - основа будь-якої сучасної командної розробки. Для кожної задачі чи великого фрагмента функціоналу ми створювали окремі гілки (feature branches), що дозволяло кожному учаснику команди незалежно реалізовувати й тестувати нові можливості, ізолювати експериментальні зміни та уникати конфліктів на початкових етапах роботи. Основна гілка проєкту (dev) завжди залишалася стабільною, містила лише перевірені й протестовані зміни - це був принцип, якого ми суворо дотримувалися для підтримки якості продукту та уникнення збоїв у роботі системи. Для фіксації проміжних змін, тестування кількох гілок одразу або уникнення конфліктів під час злиття ми активно користувалися git stash, що дозволяло тимчасово "відкласти" незавершені зміни без втрати прогресу. У разі виникнення конфліктів під час інтеграції використовували інтерактивний merge, аналізували різницю у змінених файлах (diff), спільно обговорювали альтернативи, консультувалися одне з одним щодо найкращих практик - такий підхід не тільки підвищував якість коду, а й сприяв постійному професійному зростанню кожного з розробників (рис. 1.9).

Окремо слід відзначити й культуру формування малих, логічно завершених pull request. Ми дотримувалися практики "одна задача - один pull request", намагалися не накопичувати великі об'єми змін в одному коміті, а розбивали роботу на

невеликі, ізольовані частини. Це значно спрощувало код-рев'ю: було легше побачити та зрозуміти внесені зміни, швидше знаходити і виправляти помилки, не створювати зайвого навантаження на команду під час перевірки нової функціональності. Навіть у невеликій команді з двох учасників такий підхід дав змогу уникати хаотичних злиттів, мінімізувати ризик втрати важливих напрацювань та зробити інтеграцію змін максимально прозорою, контрольованою й безпечною для всього проєкту. Сформовані принципи та звички в роботі з git залишаються актуальними і для більших команд, сприяючи загальному розвитку культури відповідальності, взаємодопомоги й технічної дисципліни.

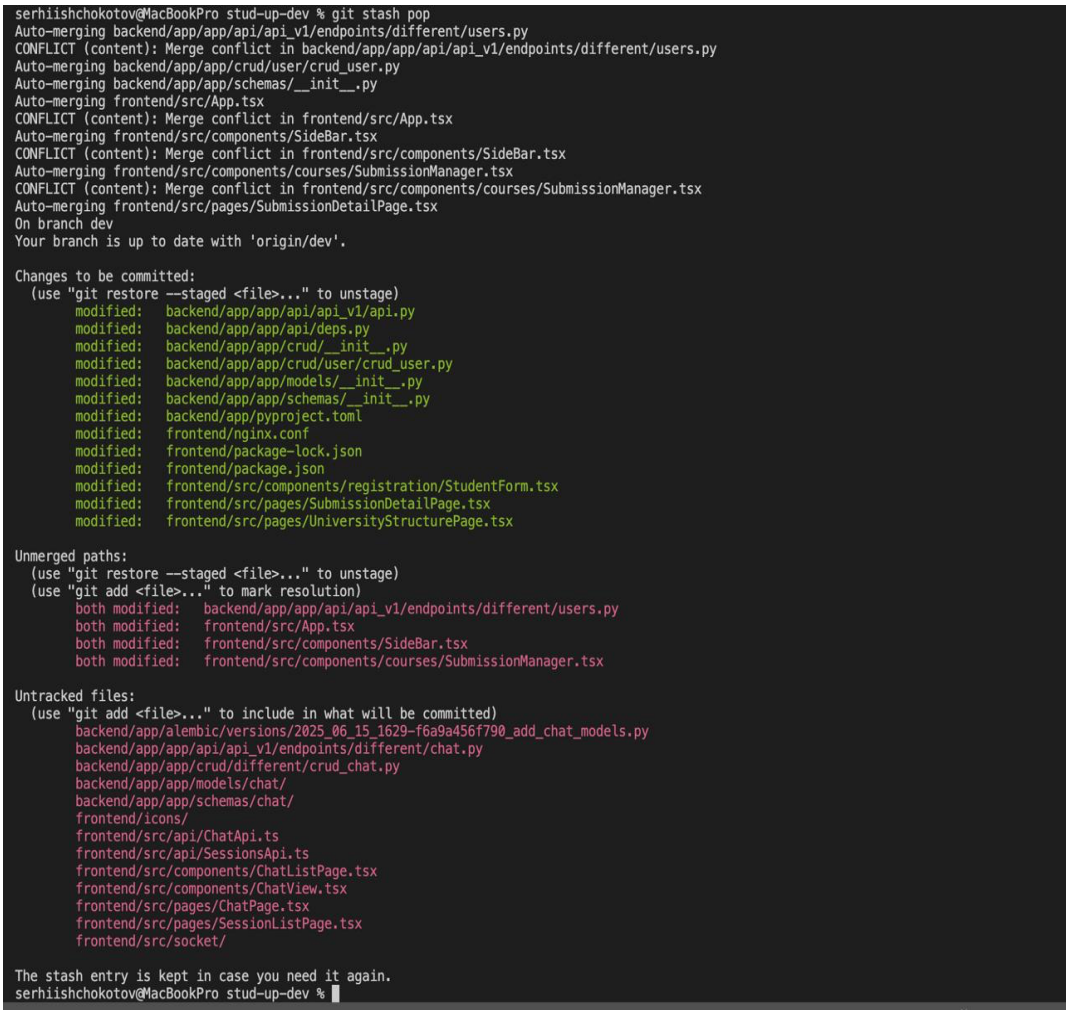


Рис. 1.9 – Результат виконання git stash pop команди та поява конфліктів

Систему планування та трекінгу задач ми організували через Jira (рис. 1.10). Кожна фіча, баг чи навіть дрібне поліпшення оформлювалися як окрема задача із чітким зазначенням відповідального, дедлайну, статусу виконання й, за потреби, коментарями. Це допомогло структурувати роботу за пріоритетами, контролювати

прогрес, уникати ефекту “загубленого завдання”, а також забезпечити прозорість для кожного члена команди. Jira також слугувала інструментом для планування спринтів, проведення стендапів, ретроспектив і аналізу витрат часу на різні етапи.

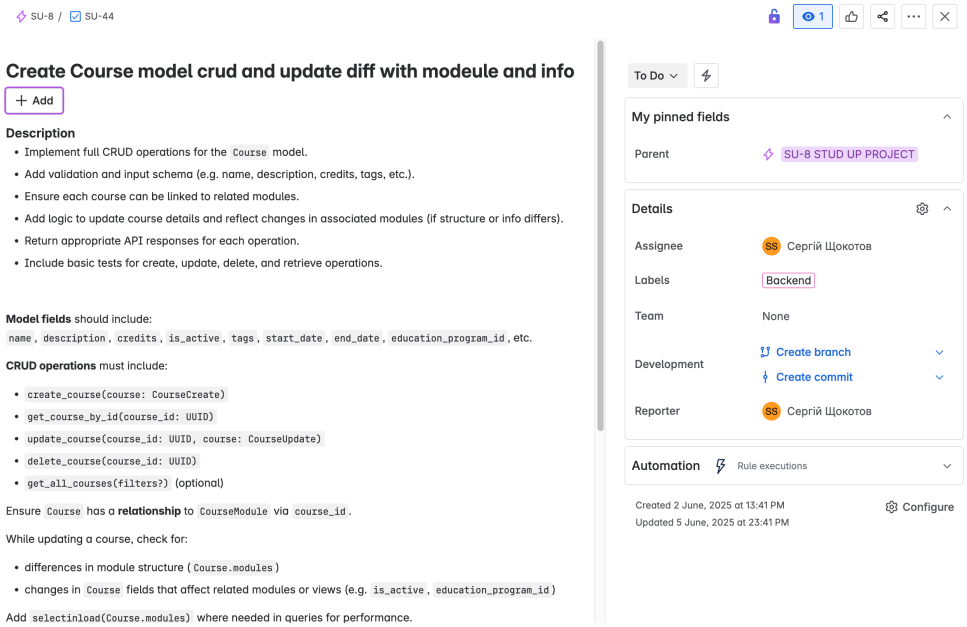


Рис. 1.10 – Вигляд одного із завдань

Регулярні обговорення перед стартом роботи, брейншторми, розбиття великих цілей на менші підзадачі й оцінка ризиків сприяли швидкій реакції на зміну вимог та пріоритетів. Крім того, досвід Jira показав, що навіть у невеликій команді варто фіксувати ключові рішення та історію обговорень прямо в задачах - це допомагає зберігати цілісність знань і швидко адаптуватися до змін(рис. 1.11).

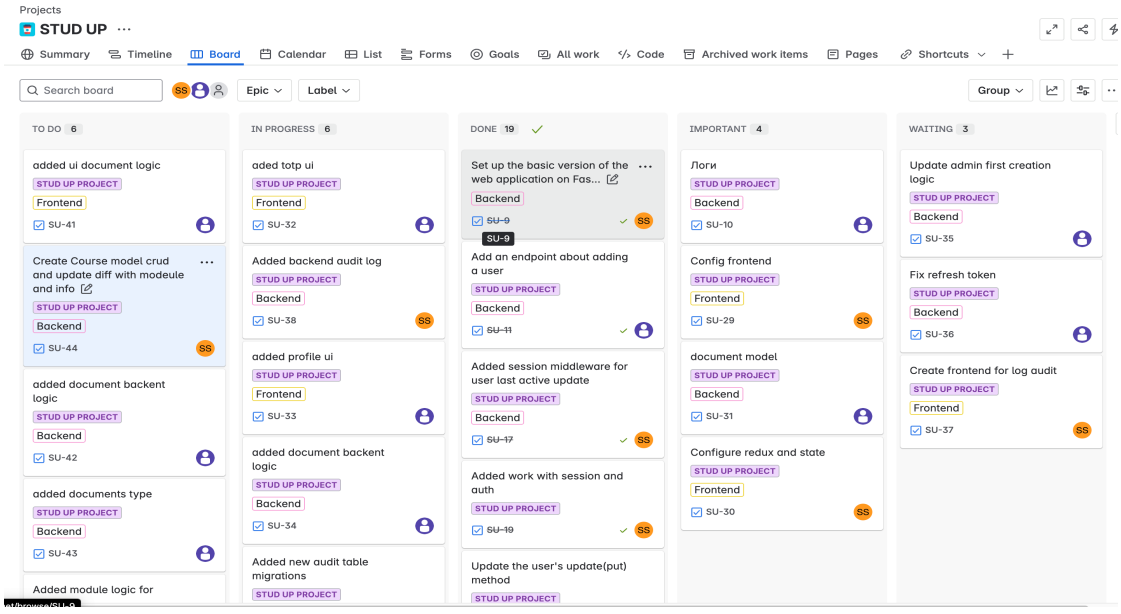


Рис. 1.11 – Загальний вигляд дошки в процесі розробки

Сукупність усіх описаних підходів і сучасних інструментів забезпечила нам не лише якісне виконання поточних задач, а й створила міцний фундамент для масштабування та розвитку проєкту. Досвід спільної розробки сприяв формуванню відповідальності, довіри, взаємоповаги та технічної грамотності, що є основою для сучасних команд і розвитку складних програмних продуктів.

Командна розробка, побудована на чіткому розподілі зон відповідальності, регулярній комунікації, прозорому плануванні та сучасних інструментах (git, Jira, віддалена база даних), дозволила досягти високої якості продукту, мінімізувати ризики та забезпечити гнучкість і масштабованість системи. Досвід, отриманий під час проєкту, є універсальним і може бути використаний у майбутніх розробках різного масштабу.

Висновки до першого розділу

Проведений аналіз предметної області, функціональних вимог і існуючих інструментальних засобів показав, що розробка сучасної системи управління академічними ресурсами повинна спиратися на перевірені практики програмної інженерії та новітні технології. Основними пріоритетами стали гнучкість, масштабованість, безпека та легка інтеграція з іншими сервісами, що визначило вибір як архітектурної моделі, так і стека технологій.

Застосування клієнт-серверної архітектури на основі REST API дозволило чітко розмежувати функціональність фронтенду й бекенду, забезпечити незалежний розвиток модулів системи, спростити масштабування та оновлення функціоналу, а також підготувати платформу до інтеграції з мобільними застосунками або зовнішніми сервісами. Вибір мови Python та фреймворку FastAPI обумовлений їхньою продуктивністю, підтримкою сучасних підходів до типізації, асинхронної обробки запитів і зручністю генерації документації. PostgreSQL було обрано як надійну СУБД із широкими можливостями для складних реляційних і транзакційних задач. На стороні клієнта використано React, що забезпечує створення динамічних, адаптивних і масштабованих інтерфейсів.

Архітектурні рішення ґрунтувалися на багаторівневій організації, принципах SOLID і впровадженні Docker-контейнерів для спрощення розгортання та CI/CD для автоматизації процесів розробки й підтримки якості продукту. Особливу увагу приділено питанням інформаційної безпеки - впроваджено багаторівневу аутентифікацію (JWT, 2FA), контроль ролей (RBAC), захист від типових веб-загроз, кешування за допомогою Redis.

Важливою складовою успішної реалізації стала організація командної роботи. Чітке розділення обов'язків, дисципліна у використанні git (окремі гілки для задач, малі pull request), спільна робота з віддаленою базою даних, а також застосування сучасних інструментів планування (Jira) дозволили не лише ефективно координувати процес розробки, але й забезпечити гнучкість, взаємозамінність і швидке реагування на зміни у вимогах. Постійна комунікація, обговорення складних рішень і взаємна підтримка стали основою командної культури, що значно підвищила якість кінцевого продукту.

Усі обрані технології та підходи довели свою доцільність і ефективність, заклавши надійний фундамент для подальшого розвитку системи, розширення функціоналу та якісного виконання задач як сьогодні, так і в майбутньому.

РОЗДІЛ 2. ПРОЄКТУВАННЯ ІНФОРМАЦІЙНОЇ СИСТЕМИ УПРАВЛІННЯ АКАДЕМІЧНИМИ РЕСУРСАМИ СТУДЕНТІВ

2.1 Специфікація вимог до програмного продукту

Проєктування інформаційної системи неможливе без ретельного формулювання вимог, які визначають її поведінку, функціональні можливості, якість, обмеження та архітектурні передумови. Специфікація вимог є критичним етапом життєвого циклу розробки програмного забезпечення, адже саме на цьому етапі закладається основа для створення функціонально повноцінного, надійного й масштабованого рішення. Помилки або нечіткість у формуванні вимог часто призводять до суттєвих витрат на подальших стадіях розробки, збоїв у роботі продукту або невдоволення кінцевих користувачів.

У межах даного підрозділу подано структурований аналіз вимог до інформаційної системи управління академічними ресурсами студентів. Вимоги згруповано за загальноприйнятим принципом поділу, а саме бізнес-вимоги, вимоги користувачів за ролями, функціональні, нефункціональні, а також вимоги до апаратного забезпечення. Така класифікація забезпечує повне охоплення усіх аспектів майбутнього програмного продукту та сприяє уніфікації підходів до його розробки.

Бізнес вимоги полягають у тому що інформаційна система управління академічними ресурсами студентів має відповідати стратегічним завданням навчального закладу, орієнтованим на цифрову трансформацію освітнього процесу. Її розробка зумовлена потребою у модернізації управлінських та навчальних процесів, підвищенні якості освітніх послуг та зменшенні витрат на адміністрування.

Основними бізнес-вимогами до інформаційної системи управління академічними ресурсами студентів є забезпечення централізованого цифрового простору для ефективної взаємодії всіх учасників освітнього процесу. Платформа має інтегрувати всі ключові компоненти – курси, навчальні матеріали, оцінки, завдання, комунікації та сервіси – в єдиній екосистемі. Особлива увага приділяється автоматизації освітніх і адміністративних процесів, що дозволяє значно скоротити ручну працю, оптимізувати формування розкладів, автоматичну перевірку тестів,

генерацію звітів, обробку заявок, а також моніторинг успішності й аналітику активності студентів.

Для підвищення прозорості та об’єктивності оцінювання система повинна мінімізувати суб’єктивний вплив, забезпечувати збереження історії оцінок, автоматичний підрахунок балів і відкритий доступ студентів до своїх результатів у реальному часі. Не менш важливим є поліпшення комунікації та зворотного зв’язку. Платформа повинна підтримувати сучасні засоби обміну повідомленнями, сповіщення, організацію обговорень у рамках курсів та швидкий зворотний зв’язок між студентами й викладачами.

Інтеграція сучасних підходів до навчання передбачає підтримку інноваційних освітніх технологій, таких як гейміфікація (бонусні бали, рейтинги), адаптивне навчання, можливість самостійного темпування, використання мультимедійного контенту та застосування елементів штучного інтелекту для аналізу прогресу студентів. Завдяки автоматизації й централізації передбачається скорочення витрат часу й ресурсів на рутинні операції адміністрації, зменшення потреби у використанні стороннього ПЗ та підвищення ефективності прийняття управлінських рішень.

Крім того, система повинна бути масштабованою й легко підтримуватися, з можливістю розширення функціоналу та масштабуванням на рівні всієї освітньої установи або мережі закладів. Такий підхід забезпечить довготривалу актуальність і ефективність впровадженої платформи.

Вимоги студентів полягають у тому що інформаційна система повинна забезпечити повноцінну цифрову підтримку навчального процесу для студентів, які є основними користувачами системи. Ключовими очікуваннями з боку студентів є зручність, прозорість, інтерактивність і доступність освітніх матеріалів та адміністративних сервісів у режимі 24/7. Основні вимоги студентів можна згрупувати за певними напрямками, описавши нижче і офромвишви діагарму (рис. 2.1).

Студенти повинні мати змогу переглядати каталог доступних курсів, реєструватися або подавати заявку на приєднання до них. Після зарахування на

курс, користувач має отримати доступ до відповідних модулів, тем, навчальних матеріалів (текстів, відео, презентацій, зовнішніх посилань тощо) та контрольних заходів.

Проходження тестів і виконання завдань, система повинна надавати інтерфейс для виконання тестових завдань з автоматичною або ручною перевіркою результатів. Передбачається підтримка завдань різного типу, а саме вибір однієї/декількох правильних відповідей, відкриті запитання, завантаження файлів. Також має бути можливість перегляду критеріїв оцінювання, дедлайнів і отримання фідбеку від викладача.

Студент повинен мати особистий профіль, у якому зберігатиметься інформація про навчальний прогрес, рейтинг, виконані завдання, результати тестування, нараховані бали. Важливо забезпечити зручний візуальний інтерфейс для аналітики – графіки, діаграми, таблиці.

Інтерфейс повинен включати візуалізований календар із подіями (дедлайни, консультації, онлайн-заняття), розкладом занять, важливими датами для кожного курсу. Синхронізація з мобільними календарями (Google, iOS) є бажаною функціональністю.

Важливі події (оновлення курсу, наближення дедлайну, нове повідомлення, зміна розкладу) повинні супроводжуватись push- або email-сповіщеннями. Студент повинен мати змогу налаштувати тип і частоту нотифікацій.

Необхідна підтримка внутрішньої системи обміну повідомленнями або чатів. Це сприятиме швидкому зворотному зв’язку, проведенню обговорень, уточненню умов завдань та підтримці колективної взаємодії.

Студенти мають потребу в інструментах подачі заяв, запитів і генерації документів (довідка про навчання, академічна довідка, залікова відомість тощо). Такі дії повинні бути максимально автоматизовані і доступні в декілька кліків без потреби особистого звернення.

Очікується адаптивність інтерфейсу до мобільних пристроїв, планшетів, десктопів. Можливість працювати через браузер або мобільний додаток забезпечить максимальне охоплення користувачів.

Кожен студент повинен мати унікальні облікові дані. Доступ до особистих даних, оцінок, переписок має бути захищений, а авторизація реалізована із застосуванням сучасних механізмів (рис. 2.1).

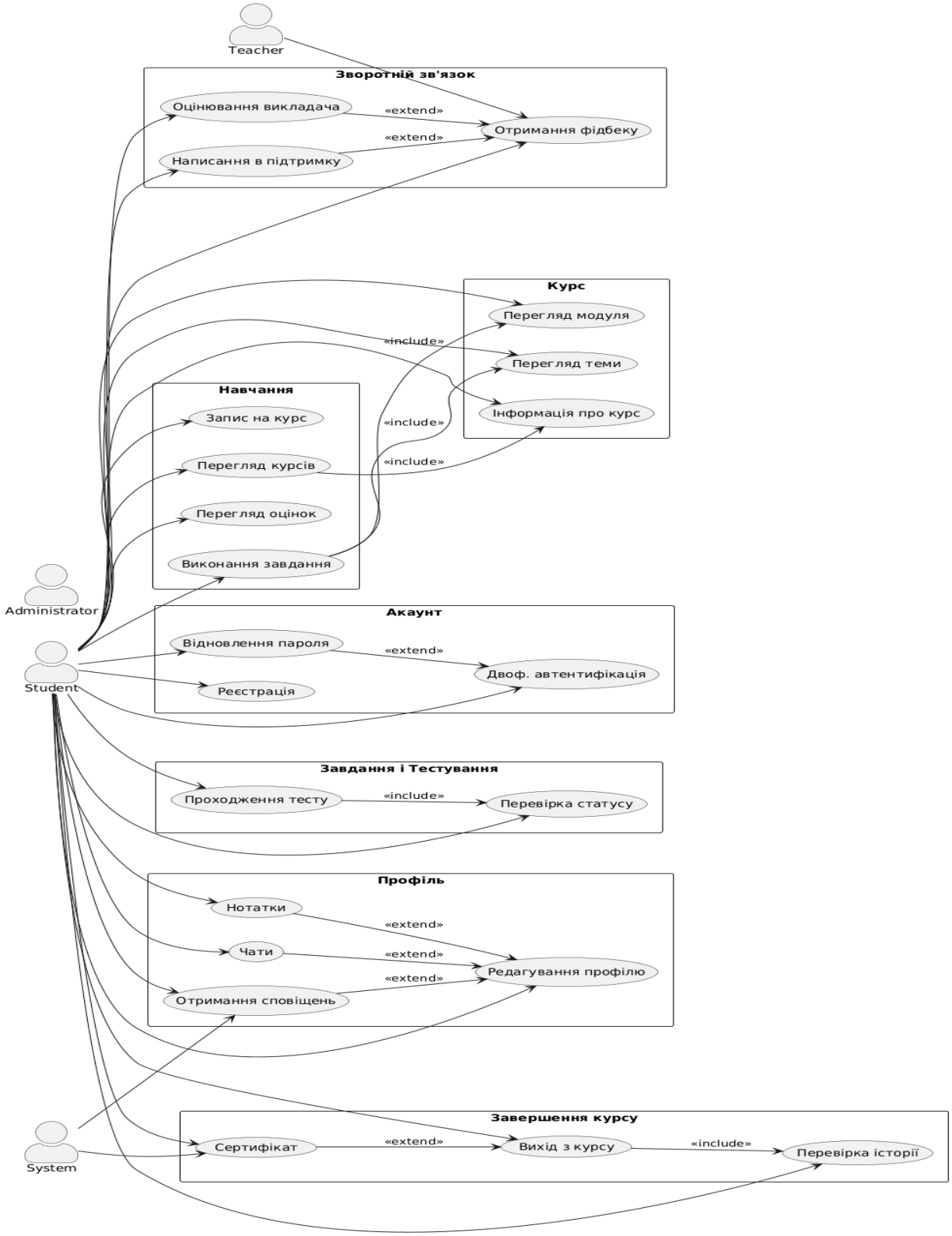


Рис. 2.1 Use-case діаграма для студентів

Вимоги викладачів є ключовими бо вони є основними користувачами системи з точки зору формування та організації освітнього процесу. Інформаційна система повинна надати викладачам широкий спектр інструментів для керування навчальним контентом, контролю за успішністю студентів і здійснення аналітики навчального процесу. Функціональність має сприяти підвищенню ефективності педагогічної діяльності та зниженню адміністративного навантаження. Основні вимоги для викладачів можна згрупувати декількома наступними чинами (рис. 2.2).

Система має забезпечити викладачам інтуїтивно зрозумілий інтерфейс для створення нових курсів, визначення їхніх назв, описів, мети, вимог, цільової аудиторії. Передбачається можливість керування доступом до курсу, призначення співавторів, визначення обмежень на зарахування, а також архівації завершених курсів.

Кожен курс складається з окремих модулів, що включають теми, навчальні матеріали, тести та завдання. Викладач повинен мати змогу додавати, редагувати, впорядковувати та публікувати елементи модуля. Підтримуються такі типи контенту, як текстові блоки, PDF-файли, відео (локальні або з YouTube), зовнішні посилання, презентації, а також вбудовані інтерактивні елементи.

Важливо передбачити гнучку систему оцінювання – автоматичну для тестів і напівавтоматичну або ручну для завдань. Викладач повинен мати змогу залишати коментарі до робіт студентів, вказувати на помилки, ставити оцінки за власними шкалами (наприклад, 100-бальна або ECTS), а також переглядати історію спроб та завантажені файли.

Система повинна генерувати докладну аналітику по кожному курсу – кількість активних студентів, середній бал, відсоток завершення модулів, динаміка прогресу, успішність по завданнях і тестах. Такі звіти мають бути представлені у вигляді графіків, таблиць або з можливістю експорту у формати CSV, PDF для подальшого аналізу.

Необхідна можливість надсилати оголошення для всієї групи, вести особисте листування зі студентами, організовувати обговорення в межах курсів, а також відповідати на питання в системі коментарів (рис. 2.2).

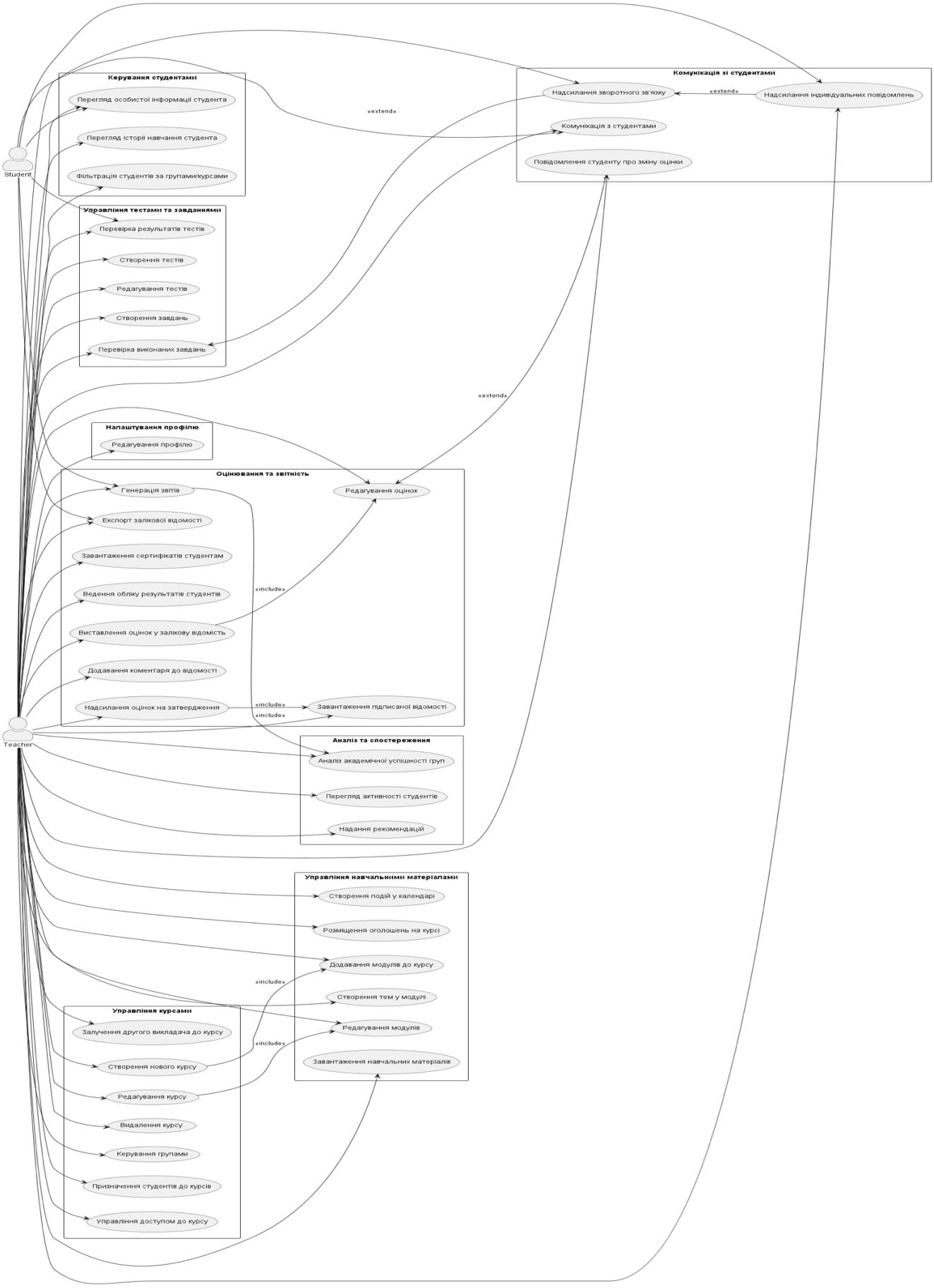


Рис. 2.2 Use-case діаграма для викладачів

Також підтримується можливість проведення синхронних онлайн-сесій (через інтеграцію з відеоплатформами) або створення форумів для асинхронного спілкування.

Викладач повинен мати змогу задавати строки виконання завдань, вмикати автоматичні нотифікації про їх наближення, налаштовувати політику прийняття запізнілих робіт. Також може бути реалізована можливість повторної спроби з обмеженнями.

Для зворотного зв'язку або поточної оцінки навчального процесу викладачі можуть створювати опитування та анкети з підтримкою різних типів запитань, а саме відкритих, закритих, шкал оцінювання. Результати мають бути доступні у вигляді агрегованої статистики.

У курсах із великою кількістю студентів важливо мати можливість фільтрації користувачів за активністю, прізвищем, рівнем виконання, а також призначення асистентів чи модераторів, які допомагатимуть у перевірці робіт і комунікації.

Система має гарантувати, що викладачі можуть редагувати лише власні курси, а також отримувати доступ лише до тієї інформації, яка належить до їх навчального навантаження. Права мають бути чітко обмежені відповідно до ролі в системі (викладач, модератор, співавтор курсу).

Адміністратор у системі управління академічними ресурсами виконує ключову роль, фактично виступаючи гарантом безперервної й безпечної роботи всієї інформаційної інфраструктури. На нього покладається як технічна, так і організаційна відповідальність, і саме від його дій багато в чому залежить стабільність та ефективність функціонування всієї системи. Обсяг його повноважень охоплює майже всі аспекти підтримки платформи – від налаштування користувацьких акаунтів і управління ролями до адміністрування безпеки, моніторингу подій і реагування на потенційні загрози (рис. 2.3).

Щоденна діяльність адміністратора включає керування всіма обліковими записами користувачів, контроль за їх активністю, активацію, блокування або видалення акаунтів у разі потреби. Він відповідає за призначення ролей, може змінювати їхній склад або скасовувати окремі права доступу, регулюючи таким

чином рівень доступу кожного користувача до ресурсів системи. У його обов'язки також входить відстеження всіх важливих дій користувачів у системі, перегляд журналів подій і проведення аудиту для виявлення потенційних проблем або порушень політик безпеки.

Крім того, адміністратор опікується технічною стороною функціонування системи, він здійснює резервне копіювання даних, слідкує за своєчасним оновленням програмного забезпечення, налаштовує політики безпеки, такі як складність паролів, термін їхньої дії чи обов'язковість використання двофакторної автентифікації, а також впроваджує сучасні засоби захисту від кіберзагроз. До його обов'язків належить моніторинг журналів аудиту для виявлення підозрілої активності, контроль за стабільністю серверної інфраструктури та підтримка безперебійної роботи системи. При потребі адміністратор має змогу інтегрувати систему з іншими сервісами, зокрема зовнішніми базами користувачів, навчальними платформами, системами електронної пошти чи месенджерами, організовувати імпорт і експорт навчальних матеріалів, підтримувати взаємодію з іншими освітніми або аналітичними ресурсами.

Велике значення мають і організаційно-управлінські функції: адміністратор формує статистичні й аналітичні звіти для керівництва або викладачів, стежить за дотриманням ліцензійної політики та юридичних вимог, забезпечує коректне відображення платних послуг (якщо такі передбачені системою) та контролює правомірність їхнього використання. Додатково адміністратор бере участь у розробці та впровадженні нових функцій, тестуванні оновлень та міграції даних при зміні інфраструктури.

Завдяки такому широкому спектру обов'язків і повноважень адміністратор виступає центральною фігурою, яка поєднує технічну підтримку, інформаційну безпеку, організаційний менеджмент і розвиток функціоналу системи. Це забезпечує цілісність, стабільність та відповідність сучасним вимогам у сфері управління академічними ресурсами, дозволяє адаптувати систему під нові виклики та гарантує її ефективне й безпечне функціонування в освітньому середовищі (рис. 2.3).

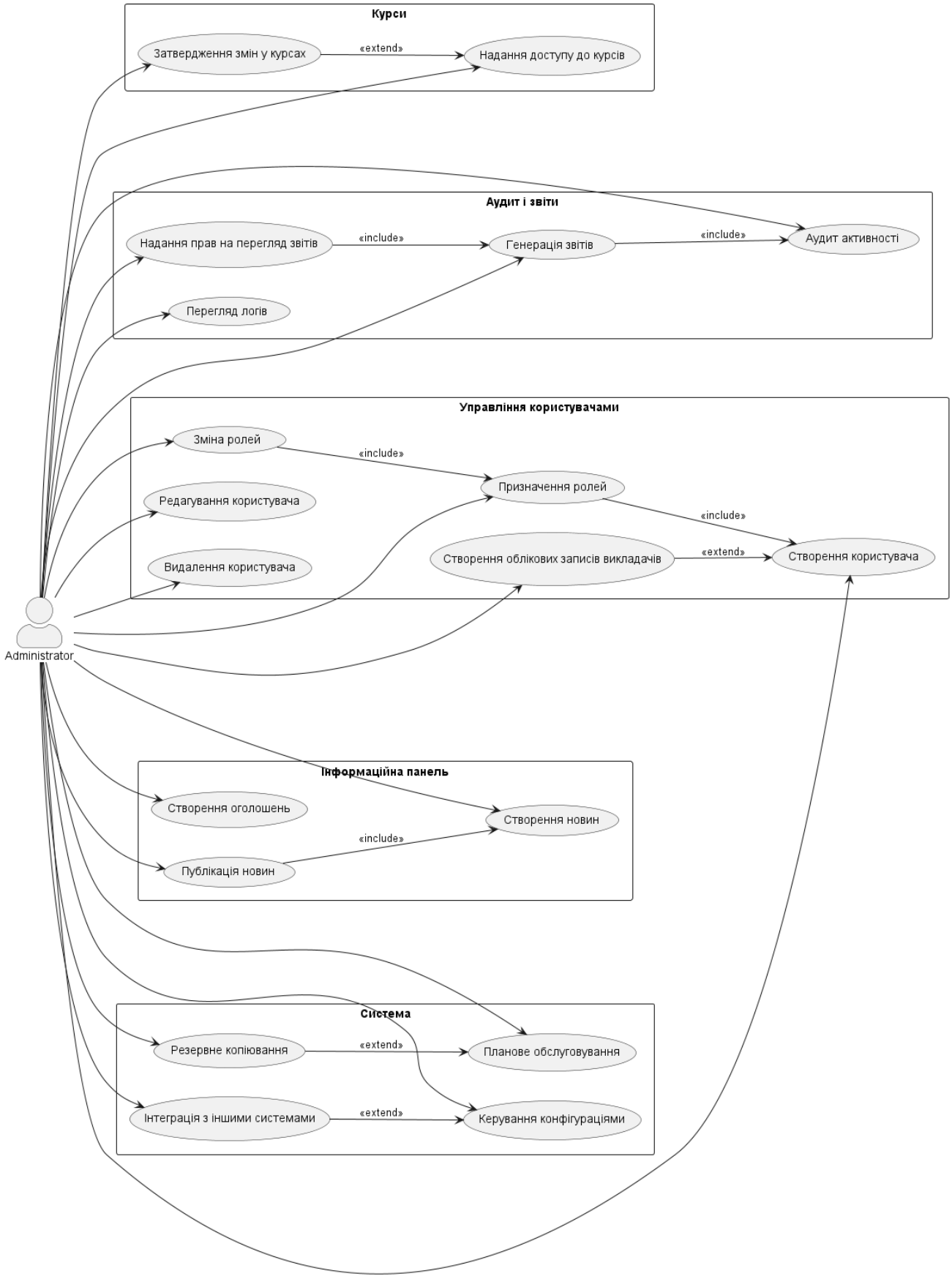


Рис. 2.3 Use-case діаграма для адміністраторів

Об'єктом комп’ютеризації виступає взаємодія студентів, викладачів і адміністративного персоналу в межах навчального процесу. Вона включає всі аспекти, від планування курсів і розкладів до оцінювання результатів та адміністративного супроводу. Уся інформація в системі повинна бути структурована за курсами, модулями, темами, а кожен користувач повинен мати персоналізований доступ із урахуванням прав та обов’язків. Комп’ютеризація передбачає автоматизацію всіх інформаційних потоків, які раніше оброблялися вручну або в розрізнених системах.

Функціональні вимоги

1. Реєстрація та автентифікація користувачів, включаючи двофакторну перевірку та підтримку OAuth.
2. Рольова модель доступу – студент, викладач, адміністратор.
3. Можливість створення, редагування, видалення курсів, модулів, тем, тестів, завдань.
4. Автоматична та ручна перевірка завдань.
5. Створення та проходження тестів, збереження результатів.
6. Чати між студентами та викладачами, групові чати.
7. Календар подій, інтеграція з персональними розкладами.
8. Механізм нотифікацій – системні повідомлення, нагадування, push-сповіщення.
9. Генерація аналітичних звітів для викладачів і адміністрації.
10. Генерація довідок, сертифікатів, залікових книжок у форматі PDF.

Нефункціональні вимоги

1. Система має бути масштабованою – передбачена підтримка зростання кількості користувачів і курсів без втрати продуктивності.
2. Система має бути доступною – веб-інтерфейс доступний через браузер з будь-якого пристрою.
3. Система має бути високопродуктивною – обробка запитів має відбуватись із мінімальними затримками, навіть при великій кількості паралельних сесій.

4. Система має бути захищеною – реалізація сучасних протоколів безпеки, збереження чутливих даних у зашифрованому вигляді.
5. Система має бути надійною – підтримка резервного копіювання, журналювання змін, відновлення після збоїв.
6. Система має бути інтернаціоналізованою – з можливістю локалізації інтерфейсу.
7. Система має бути тестованою – наявність автоматизованого тестування на рівні модулів, інтеграції, API.

Серверна частина системи може розгортатися на віртуальних серверах або в хмарному середовищі з такими мінімальними характеристиками: CPU від 4 ядер. RAM від 8 GB. Дисковий простір від 100 GB SSD. Сумісність з Docker та підтримка HTTPS (через сертифікати).

Для клієнтської частини користувачу достатньо наявності сучасного браузера (Chrome, Firefox, Edge), стабільного інтернет-з'єднання та екрану з роздільною здатністю від 1024x768. Додаткове програмне забезпечення не потребується.

2.2 Визначення варіантів використання та об’єктно-орієнтованої структури системи

Завдяки визначенню варіантів використання (use cases) на початковому етапі проєктування вдалося не лише уникнути дублювання функціоналу чи суперечливих вимог, а й забезпечити прозорість і передбачуваність процесу розробки для всіх зацікавлених сторін - від адміністрації до кінцевих користувачів. Документування сценаріїв взаємодії дозволило детально описати типові та нетипові дії кожної ролі (студент, викладач, адміністратор), врахувати альтернативні шляхи, граничні ситуації, винятки й специфіку бізнес-процесів навчального закладу. Це значно полегшило подальшу постановку задач для розробників, дозволило стандартизувати підхід до тестування й забезпечити повторюваність у роботі над різними модулями системи.

Всі варіанти використання були погоджені із замовником та узгоджені між учасниками команди, що дозволило уникнути «сліпих зон» - функцій, які могли б

залишитись без належної уваги чи реалізації. Такий підхід також сприяв гнучкості: поява нових вимог чи змін у процесах вже не вимагала радикальної переробки системи, а лише доповнення чи корекції відповідних сценаріїв та класів. Значно зменшилися ризики появи неочікуваних помилок на стадії інтеграції, оскільки всі залежності й взаємозв'язки між компонентами системи були враховані ще до початку програмування.

Об'єктно-орієнтована структура застосунку забезпечила можливість точного моделювання реальних освітніх процесів через відповідні класи та об'єкти. Кожна ключова сутність - студент, викладач, адміністратор, курс, модуль, завдання - отримала своє відображення у системі з визначеним набором атрибутів та методів. Це дало змогу впровадити спадкування, поліморфізм і повторне використання коду, що особливо важливо для масштабування й розвитку додатку. Наприклад, додавання нової ролі або типу навчального контенту не вимагає переписування всієї архітектури, а лише розширення існуючих класів.

До того ж, об'єктно-орієнтований підхід дозволяє підтримувати високу гнучкість і адаптивність – зміни в законодавстві, організаційній структурі чи освітніх програмах можна легко інтегрувати в систему за рахунок модульності та чіткої структури взаємодії між компонентами. Наприклад, додавання нових ролей користувачів, змін у структурі факультетів або оновлення освітніх планів не потребують повного перероблення всієї архітектури, а лише внесення локальних змін у відповідні класи чи модулі.

Загалом, комбінування сценарного аналізу (use cases) з грамотною реалізацією об'єктно-орієнтованої моделі є надійним фундаментом для створення інформаційних систем, які не лише повністю відповідають сучасним вимогам освіти, а й залишаються зручними, зрозумілими й масштабованими у довгостроковій перспективі. Це особливо важливо в освітній сфері, де процеси постійно вдосконалюються, а гнучкість і швидкість адаптації стають вирішальними для успіху інформаційної системи в майбутньому.

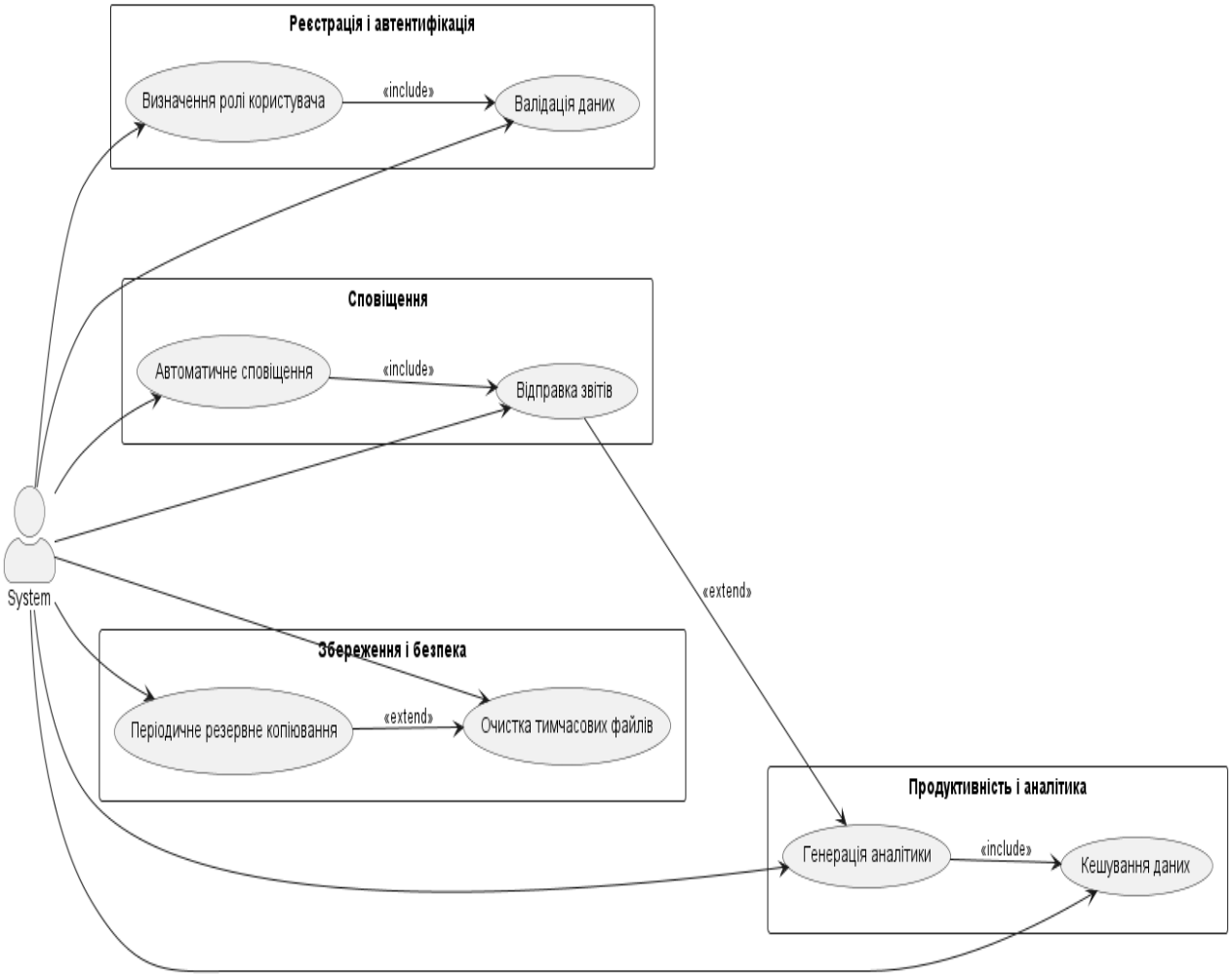


Рис. 2.4 Use-case діаграма для системи

Таблиця 2.1

Опис варіантів використання

Роль	Варіант використання	Короткий опис варіанту використання
Студент	Перегляд доступних курсів	Отримання списку курсів, до яких студент може долучитися або які вже доступні для проходження.
	Запис на курс	Можливість приєднатися до вибраного курсу після підтвердження викладачем або автоматично.
	Перегляд інформації про курс	Ознайомлення зі структурою курсу, описом, цілями та вимогами до навчання.
	Перегляд модуля	Перегляд змісту окремих модулів, що входять до складу курсу.

Продовження таблиці 2.1

Студент	Перегляд теми	Отримання доступу до навчального матеріалу в межах обраної теми модуля.
	Перегляд оцінок	Ознайомлення з результатами тестів, завдань, курсів у табличному або графічному форматі.
	Ведення власних нотаток	Створення та збереження особистих нотаток у контексті навчального процесу.
	Отримання сповіщень	Оповіщення про оновлення курсу, дедлайни, оцінки, нові повідомлення.
	Спілкування через чати	Комунікація з викладачем та іншими студентами в межах наявного курсу.
Викладач	Вихід з курсу	Відмова від участі у курсі з можливістю відновлення або остаточного виходу.
	Перевірка історії навчання	Огляд пройдених курсів, тестів, оцінок, отриманих сертифікатів.
	Реєстрація акаунта	Створення нового облікового запису студента з підтвердженням email-адреси.
	Налаштування двоф. автентифікації	Підвищення безпеки облікового запису шляхом увімкнення TOTP або інших методів 2FA.
	Надсилання звернення до техпідтримки	Форма зворотного зв’язку або чат для вирішення технічних чи навчальних питань.
	Створення нового курсу	Ініціація нового навчального курсу з базовими параметрами (назва, опис, доступність тощо).
	Редагування курсу	Модифікація існуючого курсу: оновлення опису, структури, доступу тощо.

Продовження таблиці 2.1

Викладач	Видалення курсу	Можливість видалити курс, якщо він більше неактуальний або був створений помилково.
	Додавання модулів до курсу	Створення логічно структурованих розділів у межах курсу.
	Редагування модулів	Оновлення тематики модуля, додавання чи зміна порядку елементів.
	Надсилання зворотного зв’язку	Можливість залишити коментар до виконаного завдання чи тесту студента.
	Ведення обліку результатів студентів	Перегляд та управління індивідуальними навчальними успіхами кожного студента.
	Фільтрація студентів за групами/курсами	Перегляд окремих груп студентів залежно від курсу або потоку.
	Комунікація зі студентами	Надсилання повідомлень, організація обговорень або відповіді на запитання в чаті.
	Перегляд активності студентів	Моніторинг часу входу, проходження модулів, активності в курсі.
	Надання рекомендацій	Формування індивідуальних або загальних порад для покращення результатів студентів.
	Генерація звітів	Створення аналітичних звітів щодо прогресу студентів, завершення курсів, статистики оцінок.
	Управління доступом до курсу	Обмеження чи розширення прав доступу для окремих груп або студентів.
	Надсилання індивідуальних повідомлень	Можливість особистого контакту зі студентом через вбудовану систему повідомлень.
	Розміщення оголошень на курсі	Публікація важливих повідомлень для всіх студентів курсу.

Продовження таблиці 2.1

Викладач	Виставлення оцінок у залікову відомість	Занесення підсумкових балів за виконані студентом завдання, тести та курси до електронної відомості.
	Редагування оцінок	Можливість змінити вже виставлені оцінки за результатами повторної перевірки або за апеляцією.
	Експорт залікової відомості	Завантаження результатів курсу у вигляді таблиці (PDF/Excel) для подальшої звітності.
	Перегляд особистої інформації студента	Доступ до академічного профілю студента (контакти, група, статус, попередні результати).
	Перегляд історії навчання студента	Ознайомлення з прогресом студента по іншим модулям чи курсам для повнішого розуміння динаміки.
	Надсилання оцінок на затвердження	Передача заповненої відомості на затвердження куратору чи адміністратору.
	Повідомлення студенту про зміну оцінки	Система автоматично або вручну інформує студента про зміну балу або зарахування курсу.
Адміністратор	Створення, редагування та видалення користувачів	Управління акаунтами студентів, викладачів, новдоданих адміністраторів системи.
	Призначення та зміна ролей користувачів	Визначення рівня доступу до функцій системи (студент, викладач, адміністратор).
	Надання доступу до курсів	Вручну або автоматично додавати/виключати користувачів з певних курсів або модулів.
	Аудит активності користувачів	Моніторинг дій користувачів у системі (вхід, редагування, доступ до матеріалів).

Продовження таблиці 2.1

Адміністратор	Перегляд системних логів	Аналіз логів доступу, помилок, змін у даних.
	Генерація статистичних звітів	Створення загальних або індивідуальних звітів про успішність, відвідуваність, активність.
	Створення оголошень	Надсилання сповіщень із критичними повідомленнями або технічними оновленнями.
	Планове технічне обслуговування	Закриття доступу до системи або її частин на час обслуговування.
	Контроль резервного копіювання даних	Перевірка цілісності бекапів, ініціація створення резервних копій, відновлення після збою.
	Керування конфігураціями системи	Зміна системних параметрів, оновлення середовища, безпеки, інтеграційних налаштувань.
	Надання прав для перевірки звітів	Доступ до перегляду та затвердження електронних відомостей з оцінками.
	Затвердження змін у структурі курсів	Контроль і погодження нових курсів, модулів, тем, які додаються викладачами.
Система	Автоматичне сповіщення користувачів	Надсилання email або push-сповіщень про оновлення курсів, дедлайни, нові повідомлення тощо.
	Періодичне резервне копіювання	Автоматичне створення бекапів бази даних і файлів за розкладом.
	Валідація даних при реєстрації	Перевірка унікальності email, формату введення пароля, TOTP-підтвердження при створенні акаунту.
	Визначення ролі користувача при вході	Автоматичне призначення відповідних прав і доступу після автентифікації.

Продовження таблиці 2.1

Система	Генерація аналітики	Підрахунок статистики: відвідуваність курсів, успішність студентів, рейтинг активності викладачів.
	Кешування часто використовуваних даних	Збереження у Redis даних курсів, сторінок, профілів для прискорення доступу.
	Очистка тимчасових файлів	Періодичне видалення застарілих або неактуальних тимчасових ресурсів.

Таблиця 2.2

Опис акторів

Актор	Опис
Студент	Користувач системи, який отримує доступ до курсів, матеріалів, завдань, здає тести і переглядає.
Викладач	Користувач, який створює та редагує курси, модулі, завдання, перевіряє роботи студентів, виставляє оцінки, спілкується зі студентами.
Адміністратор	Користувач із розширеними правами для управління всією системою: додає нових користувачів, керує ролями, курсовими процесами, переглядає аналітику.
Система	Програмна частина, що автоматично виконує фонові або тригерні операції: сповіщення, резервне копіювання, кешування, аналітика тощо.

2.3 Розробка бази даних системи

У системі управління академічними ресурсами студентів використовується реляційна система керування базами даних PostgreSQL. Цей вибір обґрунтовано її високою надійністю, підтримкою складних зв'язків між таблицями, повною сумісністю з SQL-стандартом та широкими можливостями масштабування. Розробка бази даних здійснювалася із застосуванням ORM-бібліотеки SQLAlchemy, яка дозволяє описувати структуру таблиць у вигляді Python-класів та автоматизує генерацію запитів до БД.

Бази даних є невід'ємною частиною будь-якої сучасної інформаційної системи. Саме вони виступають надійним підґрунтям для організації, зберігання й обробки інформації, без якої неможливо уявити ефективну роботу масштабних онлайн-сервісів, освітніх платформ чи корпоративних продуктів. У практиці розробки розрізняють дві основні категорії баз даних – реляційні та нереляційні (NoSQL). Реляційні БД на кшталт PostgreSQL відзначаються структурованістю: дані у них організовано у вигляді таблиць, між якими встановлюються чіткі зв'язки (foreign key), що дозволяє легко відображати складні багаторівневі взаємини реального світу – наприклад, співвідношення між студентами, викладачами, курсами, оцінками чи групами. Високий рівень цілісності даних, підтримка транзакцій (ACID) і можливість формувати запити практично будь-якої складності – усе це визначає вибір реляційних систем для складних освітніх чи бізнес-застосунків, де особливо цінуються точність, надійність та передбачуваність роботи.

Роль бази даних у такій системі важко переоцінити: вона не лише зберігає масиви персональних даних, структуру курсів та навчальних програм, але й дозволяє контролювати доступ до функціоналу, фіксувати історію дій, захищати критичні дані й інтегруватися з зовнішніми сервісами. Кожна таблиця у базі є своєрідним «гравцем» у великій команді: вони разом формують єдиний інформаційний простір, де кожен запис важливий, а всі зв'язки – продумані до дрібниць. Щоб краще зрозуміти архітектуру такої системи, розглянемо призначення кожної ключової моделі, яка реалізована у базі даних:

Таблиця admins створена для зберігання профілів адміністраторів системи. Вона містить посилання на користувача, що отримав статус адміністратора, а також додаткову службову інформацію: до якого факультету чи кафедри він належить, який рівень доступу має, які функції виконує в межах організації. Це дозволяє гнучко керувати правами адміністративного доступу та делегувати повноваження в масштабі всієї освітньої установи.

teachers – ще одна важлива складова системи, що містить повний перелік викладачів із деталями про їхню академічну кваліфікацію, зв'язок із кафедрою,

наукові ступені, посаду й інші індивідуальні характеристики. Наявність цієї таблиці забезпечує автоматизацію багатьох процесів у навчальному процесі: від призначення ролей до формування навчальних планів.

У таблиці students зберігається вся ключова інформація про студентів: їх групи, факультети, спеціальності, програми навчання, а також детальні зв'язки з освітніми рівнями, формами навчання та іншими параметрами. Це дозволяє системі легко формувати персональні освітні траєкторії, стежити за академічними успіхами та організовувати комунікацію між різними структурними підрозділами.

users – це центральна таблиця, через яку здійснюється автентифікація, авторизація, управління профілями користувачів та зв'язок із ролями, сесіями, токенами чи двофакторною автентифікацією. Тут зберігається персональна інформація, контактні дані, тип користувача, email, хешований пароль, а також посилання на роль у системі. Саме звідси починається більшість дій у системі, адже практично кожен запит пов'язаний із певним користувачем.

Окремий пласт складає модуль комунікації – chats, який фіксує інформацію про чати (як приватні, так і групові), їхню назву, тип, перелік учасників. chat_participants служить для реєстрації участі користувача в чаті й дозволяє розмежовувати їхні ролі: власник чи звичайний учасник. Всі обміни повідомленнями фіксуються у таблиці messages – тут вказується час, відправник, статус повідомлення, текст і зв'язок із відповідним чатом.

Таблиця courses описує освітні курси з усіма основними атрибутами: назва, опис, кредити, статус, зображення, дати проведення, зв'язок із освітньою програмою. У свою чергу, course_modules дозволяє структурувати курс на окремі логічні блоки, кожен із яких може містити кілька занять (таблиця lessons). Таким чином, легко організовується прогресивне вивчення матеріалу, контроль доступу та виконання проміжних завдань.

Взаємодія студентів і викладачів із курсами фіксується у таблицях student_courses та teacher_courses – тут відображається статус участі у курсі, прогрес, поточний модуль, спеціальні позначки, сповіщення, а також те, хто саме з викладачів є творцем або куратором курсу.

lessons містить інформацію про кожне заняття в межах модуля: тип заняття (лекція, практика, лабораторна робота), контент, посилання на матеріали, тривалість, дату публікації, умови доступу тощо. Це дозволяє автоматично будувати розклад, організовувати видачу домашніх завдань і тестів.

У таблиці grades фіксуються всі оцінки студентів із детальним описом: виставлений бал, коментар викладача, зв'язки із завданням, курсом, подачею, дата, статус апеляції тощо. Це дає змогу не лише зберігати академічну історію, а й організовувати прозору систему контролю та звітності.

Безпека та керування доступом реалізовані через таблиці sessions (сесії користувача), totp_secrets (двохфакторна аутентифікація), а також службову alembic_version, яка відслідковує актуальний стан міграцій БД. Нарешті, audit_logs виступає ключовим інструментом для контролю, фіксації подій і моніторингу дій користувачів, що особливо важливо з точки зору безпеки та прозорості роботи системи.

Такий підхід до структурування даних забезпечує не лише повну відповідність потребам навчального закладу, а й легкість масштабування, інтеграції нових модулів, гнучке налаштування прав і можливість організувати сучасний освітній процес на якісно новому рівні.

Ключові особливості реалізації для всіх ідентифікаторів використовується тип UUID, що забезпечує унікальність записів навіть у випадку масштабування чи роботи в розподіленому середовищі. Це дозволяє уникнути дублювання даних і спрощує інтеграцію з іншими системами. Поліморфне наслідування (polymorphic_identity) у SQLAlchemy дало змогу створити єдину універсальну модель користувача (User), від якої спадкуються типи студентів, викладачів та адміністраторів. Завдяки цьому стало можливо централізовано керувати базовими властивостями всіх користувачів, а також розширювати модель новими специфічними полями для кожної ролі без дублювання коду. Зв'язки між таблицями реалізовано за допомогою потужних механізмів SQLAlchemy relationship із підтримкою каскадного видалення, що особливо важливо для чутливих даних - наприклад, для токенів автентифікації чи секретних ключів двофакторної

автентифікації (TOTP). Це забезпечує автоматичне очищення пов’язаних записів при видаленні основного об’єкта, мінімізуючи ризики залишення "сміттєвих" записів у базі.

Таким чином, поточна реалізація моделі бази даних є не лише гнучкою та масштабованою, а й високодоступною для подальшого розвитку системи. Це дозволяє ефективно керувати академічними ресурсами, забезпечує надійну аутентифікацію, авторизацію й захист даних користувачів, а також відкриває широкі можливості для подальшої кастомізації та адаптації системи під індивідуальні вимоги будь-якого освітнього закладу. Водночас структура таблиць продумана так, щоб якнайпростіше було реалізовувати новий функціонал, інтегруватися з існуючими сервісами університету та підтримувати високий рівень безпеки й цілісності даних.

Далі буде представлено опис таблиць, задіяних у роботі нашого додатку:

Таблиця 2.3

Опис моделі Users

Поле	Тип даних	Опис
id	UUID	Унікальний ідентифікатор користувача в системі. Генерується автоматично, використовується як головний ключ для зв’язку з іншими таблицями.
created	DateTime	Дата та час створення облікового запису. Дозволяє відстежувати, коли користувача було зареєстровано у системі.
modified	DateTime	Дата й час останнього оновлення профілю. Це поле автоматично змінюється при будь-яких змінах даних користувача.
is_active	Boolean	Показник того, чи обліковий запис є активним. Якщо встановлено значення false – користувач не може входити в систему та взаємодіяти з сервісом.
is_superuser	Boolean	Ознака того, що користувач має розширені права адміністратора (суперкористувача). Такі користувачі мають доступ до усіх функцій без обмежень.

Продовження таблиці 2.3

email	String	Електронна пошта користувача. Вона є унікальною для кожного запису
name	String	Ім’я користувача. Використовується для персоналізації інтерфейсу, створення профілів і в офіційному документообігу.
surname	String	Прізвище користувача. Необхідне для ідентифікації та формування звітів, сертифікатів чи документів.
middle_name	String	По батькові користувача (якщо є). Додає додаткову ідентифікацію, часто потрібне у формальних документах.
mobile_phone	String	Номер мобільного телефону. Використовується для зв’язку, а також може застосовуватися для двофакторної автентифікації або відновлення доступу.
address	String	Фізична адреса користувача. Може бути потрібною для адміністративних цілей, звітності чи організації доставки матеріалів.
birth_date	DateTime	Дата народження користувача. Дозволяє перевіряти вік, формувати аналітику чи обмежувати певні функції для неповнолітніх.
photo_url	String	Посилання на фотографію профілю. Додає зручності у візуальній ідентифікації користувача в системі.
hashed_password	Text	Хеш паролю користувача, збережений у безпечному вигляді. Справжній пароль не зберігається; це критично для безпеки даних.
totp_enabled	Boolean	Чи активовано для користувача двофакторну автентифікацію через одноразові паролі (TOTP). Якщо так, то для входу потрібне додаткове підтвердження.
user_type	String	Тип користувача: може бути "student", "teacher" або "admin". Визначає роль і доступні функції у системі.
role_id	UUID	Посилання на запис із таблиці ролей. Дозволяє розширено керувати правами й доступом користувача до різних модулів системи.

Центральна таблиця для всіх користувачів системи. Тут зберігаються основні персональні та контактні дані, інформація про тип користувача (студент, викладач, адміністратор), роль, статус активності, облікові дані, історію автентифікацій та безпекові налаштування (двофакторна автентифікація).

Таблиця 2.4

Опис моделі Admins

Поле	Тип даних	Опис
user_id	UUID	Унікальний ідентифікатор користувача, який виступає адміністратором системи. Це головний ключ таблиці та водночас зовнішній ключ до таблиці users. Завдяки цьому адміністратор має всі атрибути звичайного користувача, а також додаткові права та функціонал.
admin_id	String	Унікальний ідентифікатор адміністратора у межах системи, який використовується для внутрішнього обліку, інтеграції з іншими модулями або сервісами, а також при формуванні звітів чи логування адміністративних дій.
faculty_id	UUID	Посилання на факультет (зовнішній ключ до таблиці faculties), до якого належить адміністратор. Дозволяє визначити зону відповідальності та розподілити адміністративні права.
department_id	UUID	Посилання на кафедру (зовнішній ключ до таблиці departments), яку адміністратор може обслуговувати або за яку відповідає. Дає змогу гнучко керувати повноваженнями в межах окремих структурних підрозділів.
access_level	Integer	Числовий показник, який визначає рівень доступу адміністратора до функцій і даних системи. Від цього залежить, які саме дії адміністратор може виконувати.

Ця таблиця містить записи про адміністраторів системи, які мають розширені повноваження щодо управління платформою, налаштування політик доступу, моніторингу дій користувачів і керування структурними одиницями закладу. Адмін пов’язаний з певним користувачем, факультетом або кафедрою і може мати різний рівень доступу.

Таблиця 2.5

Опис моделі Teachers

Поле	Тип даних	Опис
user_id	UUID	Унікальний ідентифікатор користувача, який є викладачем. Це поле виконує роль первинного ключа та одночасно є зовнішнім ключем до таблиці users
teacher_id	String	Унікальний ідентифікатор викладача в межах освітньої системи, який використовується для швидкого пошуку
department_id	UUID	Посилання на кафедру (зовнішній ключ до таблиці departments), до якої закріплений викладач. Це дозволяє організувати структуру підрозділів навчального закладу
academic_title	String	Офіційний науковий або академічний титул викладача, наприклад "доцент", "професор", "старший викладач" тощо.
academic_degree	String	Науковий ступінь викладача, такий як "кандидат наук", "доктор наук", "старший викладач" або відсутність ступеня.
position	String	Посада викладача у закладі – наприклад, "завідувач кафедри", "викладач", "асистент".

Тут зберігається інформація про викладачів, включаючи їх унікальні ідентифікатори, зв’язок із кафедрою, науковий ступінь, академічне звання та посаду. Таблиця дозволяє призначати викладачам курси, контролювати їх роль у навчальному процесі й вести історію їх діяльності.

Таблиця 2.6

Опис моделі Students

Поле	Тип даних	Опис
user_id	UUID	Унікальний ідентифікатор користувача, який є студентом. Це головний ключ таблиці та одночасно зовнішній ключ до основної таблиці users.

Продовження таблиці 2.6

student_id	String	Унікальний ідентифікатор студента, який використовується для внутрішнього обліку, реєстрації, формування електронних документів, журналів успішності, а також для швидкого пошуку чи синхронізації з іншими інформаційними системами навчального закладу.
faculty_id	UUID	Посилання на факультет, до якого належить студент (зовнішній ключ до таблиці faculties).
department_id	UUID	Посилання на кафедру (зовнішній ключ до таблиці departments), до якої закріплений студент.
educational_degree_id	UUID	Зовнішній ключ до таблиці освітніх ступенів (наприклад, бакалавр, магістр, доктор філософії).
education_form_id	UUID	Посилання на форму навчання – наприклад, денна, заочна чи дистанційна (зовнішній ключ до таблиці education_forms).
specialty_id	UUID	Посилання на спеціальність, яку опановує студент (зовнішній ключ до таблиці specialties).
education_program_id	UUID	Зовнішній ключ до освітньої програми, за якою навчається студент (education_programs).
group_id	UUID	Посилання на навчальну групу (зовнішній ключ до таблиці groups), до якої належить студент.

У цій таблиці містяться всі ключові дані про студентів: від персональних ідентифікаторів до їх належності до групи, спеціальності, факультету, освітньої програми, освітнього рівня та форми навчання. Вона дає змогу будувати академічні траєкторії, вести оцінювання й автоматично формувати доступ до матеріалів.

Завдяки продуманій індексації ключових полів, система дозволяє оперативно виконувати фільтрацію та агрегацію великих масивів інформації навіть при значному навантаженні. Такий підхід дає змогу швидко генерувати звіти, відстежувати динаміку навчального процесу й ефективно взаємодіяти з іншими підсистемами без затримок і втрати продуктивності.

Таблиця 2.7

Опис моделі Chats

Поле	Тип даних	Опис
id	UUID	Унікальний ідентифікатор чату. Генерується автоматично системою і виступає первинним ключем для цієї таблиці.
type	String	Тип чату, що визначає його призначення та особливості використання. Може бути "dialog" (приватний діалог між двома користувачами) або "group" (груповий чат, у якому бере участь більше двох осіб).
title	String	Назва чату, яка відображається учасникам у списках чатів. Для діалогів це поле може бути порожнім або містити ім’я співрозмовника, а для групових чатів – зрозумілу назву для всієї команди чи проєкту.

Таблиця для організації різних типів чатів – як приватних діалогів між двома користувачами, так і групових обговорень. Вона містить інформацію про тип чату, його назву, а також зв’язки з учасниками та повідомленнями.

Таблиця 2.8

Опис моделі Chat Participants

Поле	Тип даних	Опис
id	UUID	Унікальний ідентифікатор запису про участь користувача у чаті. Дозволяє однозначно визначити кожен зв’язок між користувачем і чатом, навіть якщо користувач може брати участь у багатьох чатах.
chat_id	UUID	Посилання на конкретний чат (зовнішній ключ до таблиці chats), в якому бере участь користувач. Дозволяє легко знаходити всіх учасників певного чату та будувати структуру обговорень.
user_id	UUID	Посилання на користувача (зовнішній ключ до таблиці users), який є учасником цього чату. Це забезпечує зв’язок між профілем користувача та всіма чатами, де він присутній.
role	String	Роль користувача у чаті, наприклад "owner" (власник, адміністратор чату) або "member" (звичайний учасник).

Зберігає відомості про те, хто з користувачів є учасником конкретного чату, з зазначенням їхньої ролі (наприклад, власник або просто учасник). Це дозволяє реалізувати індивідуальні права в межах одного чату.

Таблиця 2.9

Опис моделі Messages

Поле	Тип даних	Опис
id	UUID	Унікальний ідентифікатор кожного повідомлення в системі, який використовується для зв’язку з іншими сутностями та точного пошуку потрібного запису.
chat_id	UUID	Посилання на чат, у якому надіслано повідомлення. Дозволяє згрупувати всі повідомлення відповідно до їхньої розмови.
sender_id	UUID	Ідентифікатор користувача, який є автором повідомлення. Забезпечує зв’язок із таблицею users і дає змогу відстежувати автора кожного повідомлення.
text	String	Основний вміст повідомлення, тобто текст, який надсилається у чаті.
created_at	DateTime	Дата й час відправлення повідомлення. Дозволяє впорядковувати повідомлення в хронологічному порядку та формувати історію переписки.
is_read	Boolean	Позначає, чи було це повідомлення прочитане отримувачем, що дозволяє реалізувати індикатори прочитання.
status	String	Статус доставки повідомлення: може бути "sent" (надіслано), "delivered" (доставлено), "read" (прочитано). Дозволяє відслідковувати, чи дійшло повідомлення до адресата.

У цій таблиці зберігаються всі повідомлення, що надсилаються у чатах. Вказується текст, автор, дата та час відправлення, статус прочитання й належність до конкретного чату. Завдяки цьому модуль чату підтримує миттєву синхронізацію та відображення нових повідомлень для всіх учасників, а також зберігає історію спілкування для подальшого перегляду й аналітики активності користувачів.

Таблиця 2.10

Опис моделі Courses

Поле	Тип даних	Опис
id	UUID	Унікальний ідентифікатор курсу, який використовується як основний ключ для зв’язку з іншими таблицями (модулями, студентами, викладачами).
name	String	Офіційна назва курсу, яка відображається у списках і профілях користувачів.
description	Text	Детальний опис змісту курсу, його мети та основних тем.
credits	Integer	Кількість академічних кредитів ECTS, яку отримує студент за проходження цього курсу.
is_active	Boolean	Вказує, чи курс зараз активний, чи вже архівований або не доступний для запису.
cover_image_url	String	Посилання на зображення-обкладинку курсу для візуалізації в інтерфейсі.
start_date	DateTime	Дата початку навчання за цим курсом
end_date	DateTime	Дата завершення курсу, після якої він може перейти до архіву.
tags	Text	Список тегів у форматі JSON для категоризації й швидкого пошуку курсів за темами чи напрямами.
is_free_entered	Boolean	Вказує, чи можна вступати на курс вільно, без додаткового запрошення або підтвердження.
created_at	DateTime	Дата та час створення курсу – фіксується автоматично при додаванні.
updated_at	DateTime	Дата останнього оновлення курсу, корисно для відстеження актуальності інформації.
light_remove	Boolean	Позначає, чи курс помічений для "легкого" видалення (наприклад, прихований, але не видалений остаточно з бази).
education_program_id	UUID	Посилання на освітню програму, до якої належить курс

Тут описані всі навчальні курси, їх характеристики, кредити, описи, зв’язок з освітньою програмою, дата початку й завершення, статус активності та інші параметри, необхідні для організації навчального процесу.

Таблиця 2.11

Опис моделі Student Courses

Поле	Тип даних	Опис
id	UUID	Унікальний ідентифікатор запису про зв’язок студента з конкретним курсом.
enrolled_at	DateTime	Дата та час, коли студент записався на курс.
completed_at	DateTime	Дата завершення курсу студентом; може бути порожнім, якщо курс ще не завершено.
status	String	Статус участі студента у курсі: “enrolled” – навчається, “completed” – завершив, “dropped” – відмовився.
current_module_id	UUID	Посилання на модуль, який студент проходить у даний момент (зовнішній ключ до course_modules).
progress	Integer	Відсоток виконання програми курсу студентом; допомагає відстежувати успішність навчання.
last_accessed	DateTime	Час останнього відвідування курсу студентом, корисно для моніторингу активності.
favorite	Boolean	Позначає, чи доданий курс у вибране студентом для швидкого доступу.
notifications_enabled	Boolean	Вказує, чи ввімкнені сповіщення про події на курсі для цього студента.
student_id	UUID	Посилання на студента (зовнішній ключ до students.user_id), якому належить цей запис.
course_id	UUID	Посилання на курс (зовнішній ключ до courses.id), до якого стосується запис.

Таблиця служить для фіксації участі студентів у курсах: коли вони записалися, як просувається їх навчання, поточний прогрес, статуси, вибрані курси, налаштування сповіщень тощо.

Таблиця 2.12

Опис моделі Teacher Courses

Поле	Тип даних	Опис
id	UUID	Унікальний ідентифікатор запису, що пов’язує викладача з певним курсом.
role	String	Роль викладача на курсі (наприклад, основний викладач, асистент чи куратор).
is_active	Boolean	Вказує, чи бере викладач активну участь у проведенні курсу на даний момент.
notes	Text	Додаткові примітки щодо ролі або особливостей роботи викладача на цьому курсі.
is_creator	Boolean	Чи є цей викладач ініціатором/творцем курсу.
teacher_id	UUID	Посилання на викладача (зовнішній ключ до teachers.user_id).
course_id	UUID	Посилання на курс (зовнішній ключ до courses.id).

Аналогічно до попередньої, ця таблиця фіксує, які курси викладає певний викладач, його роль на курсі, активність, особливі примітки чи статус творця курсу.

Таблиця 2.13

Опис моделі Course Modules

Поле	Тип даних	Опис
id	UUID	Унікальний ідентифікатор кожного модуля в курсі.
name	String	Назва модуля, що відображається студентам і викладачам у структурі курсу.
description	Text	Опис змісту модуля, коротке резюме основних тем або завдань.
order	Integer	Порядковий номер модуля у курсі для формування правильної послідовності навчання.
is_required	Boolean	Вказує, чи є модуль обов’язковим для проходження студентом.
light_remove	Boolean	Позначає, чи модуль прихований для користувачів, але не видалений повністю з бази.
course_id	UUID	Посилання на курс (зовнішній ключ до courses.id), до якого належить модуль.

У цій таблиці зберігаються модулі, на які поділений кожен курс. Вона містить їх порядок, назви, описи, обов'язковість проходження, а також зв'язок із курсом.

Таблиця 2.14

Опис моделі Lessons

Поле	Тип даних	Опис
id	UUID	Унікальний ідентифікатор заняття (лекції, практики чи іншого елементу модуля).
title	String	Назва заняття, що відображається в курсі й у розкладі.
description	Text	Короткий опис змісту заняття.
content	Text	Основний навчальний матеріал у форматі HTML (текст, зображення, посилання).
order	Integer	Порядок заняття у межах модуля для формування послідовності проходження.
lesson_type	String	Тип заняття: наприклад, «lecture», «practice», «lab» тощо.
video_url	String	Посилання на відеоматеріали до заняття (за наявності).
presentation_url	String	Посилання на презентацію, яку можна переглянути чи завантажити.
additional_materials	Text	Додаткові матеріали у форматі JSON (посилання на статті, файли тощо).
duration_minutes	Integer	Орієнтовна тривалість заняття у хвилинах.
is_published	Boolean	Вказує, чи заняття опубліковано та доступне студентам.
publish_date	DateTime	Дата й час публікації заняття.
unlock_conditions	Text	Умови відкриття заняття (наприклад, проходження попереднього), у форматі JSON.
thumbnail_url	String	Посилання на мініатюру або зображення до заняття.
download_materials_url	String	Посилання для завантаження архіву з матеріалами до заняття.
external_id	String	Зовнішній ідентифікатор для інтеграції із сторонніми платформами.

Продовження таблиці 2.14

completion_type	String	Тип завершення заняття: наприклад, «view», «assignment», «test».
light_remove	Boolean	Позначає, чи заняття приховано (тимчасово не відображається), але не видалено остаточно.
module_id	UUID	Посилання на модуль (зовнішній ключ до course_modules.id), до якого належить заняття.

Тут міститься інформація про кожне заняття всередині модулів: назва, тип заняття (лекція, практика, лабораторна), контент, матеріали, тривалість, статус публікації та інші додаткові параметри.

Таблиця 2.15

Опис моделі Grades

Поле	Тип даних	Опис
id	UUID	Унікальний ідентифікатор оцінки, що дозволяє однозначно визначити кожний запис.
score	Integer	Виставлений бал за виконане завдання чи підсумкову роботу.
feedback	Text	Коментар або відгук викладача щодо результату студента.
graded_at	DateTime	Дата й час, коли оцінку було виставлено або оновлено.
is_final	Boolean	Позначає, чи це підсумкова оцінка за весь курс, чи поточна/проміжна.
draft_score	Integer	Чорновий бал, який ще не підтверджено й може бути змінений.
is_approved	Boolean	Чи підтверджена ця оцінка як остаточна (наприклад, після перевірки).
rubric_scores	Text	Оцінювання за критеріями у форматі JSON (деталізує за різними аспектами).
late_submission	Boolean	Чи здано роботу із запізненням відносно дедлайну.
late_penalty	Integer	Розмір штрафу (у балах), якщо роботу здано пізніше встановленого терміну.
appeal_status	String	Статус апеляції: наприклад, «none», «pending», «approved», «rejected».

Продовження таблиці 2.15

appeal_comment	Text	Додаткові пояснення щодо апеляції або її результату.
assignment_id	UUID	Посилання на завдання (зовнішній ключ до assignments.id), за яке отримана оцінка.
student_id	UUID	Посилання на студента (зовнішній ключ до students.user_id), який отримав оцінку.
teacher_id	UUID	Посилання на викладача (зовнішній ключ до teachers.user_id), який виставив оцінку.
submission_id	UUID	Посилання на конкретне подання студентом роботи (submission), яке оцінюється.
course_id	UUID	Посилання на курс (зовнішній ключ до courses.id), до якого належить ця оцінка.

Зберігає всі оцінки, що виставляються студентам за різні види діяльності: роботи, тести, проекти чи підсумкові оцінки за курс. Також містить інформацію про апеляції, запізнення, коментарі та зв’язок із конкретним викладачем чи завданням.

Таблиця 2.16

Опис моделі Sessions

Поле	Тип даних	Опис
id	UUID	Унікальний ідентифікатор сесії для точного відстеження кожного входу користувача.
user_id	UUID	Посилання на користувача (зовнішній ключ до users.id), який відкрив цю сесію.
ip_address	String	IP-адреса пристрою, з якого відбулося підключення – використовується для безпеки та аудиту.
user_agent	String	Інформація про пристрій і браузер користувача (User-Agent) для детального журналювання і виявлення підозрілих неправомірних активностей.
refresh_token_hash	Text	Хешований refresh-токен, який використовується для безпечного поновлення доступу до системи.
created_at	DateTime	Час створення сесії – коли користувач увійшов у систему.

Продовження таблиці 2.16

last_active	DateTime	Дата й час останньої активності користувача у цій сесії, дозволяє визначати неактивні або "забуті" сесії.
expires_at	DateTime	Час закінчення дії сесії – після цієї дати сесія автоматично вважається неактивною, і користувач має повторно пройти автентифікацію.

Таблиця для зберігання активних сесій користувачів: містить інформацію про токени, пристрої, IP-адреси, тривалість і статуси сесії, що необхідно для безпеки та зручного керування входами.

Таблиця 2.17

Опис моделі Totp Secrets

Поле	Тип даних	Опис
id	UUID	Унікальний ідентифікатор секретного запису для TOTP (одноразових паролів), що дає змогу швидко знаходити й оновлювати потрібний ключ.
user_id	UUID	Посилання на користувача (унікальний зовнішній ключ до users.id), для якого активовано двофакторну автентифікацію.
secret	Text	Секретний ключ, який використовується для генерації тимчасових одноразових паролів (TOTP) у додатку типу Google Authenticator.
counter	Integer	Лічильник використання TOTP-кодів, застосовується для підвищення безпеки й захисту від повторного використання кодів.
created_at	DateTime	Дата й час створення цього секрету; дозволяє відстежувати історію підключення двофакторки користувачем.

Призначена для зберігання секретних ключів двофакторної автентифікації кожного користувача, які використовуються для захисту акаунтів і підвищення рівня безпеки при вході. Кожен запис містить унікальний секрет, лічильник та дату створення, що дозволяє гнучко керувати налаштуваннями TOTP і швидко відновлювати або оновлювати методи автентифікації у разі потреби. Такий підхід дозволяє мінімізувати ризики несанкціонованого доступу до персональних даних.

Таблиця 2.18

Опис моделі Audit Logs

Поле	Тип даних	Опис
id	UUID	Унікальний ідентифікатор кожного запису логу для точного відстеження подій у системі.
user_id	UUID	Посилання на користувача (зовнішній ключ до users.id), який виконав дію або зробив запит.
session_id	UUID	Посилання на сесію (зовнішній ключ до sessions.id), в межах якої відбулась дія.
method	String	HTTP-метод запиту, наприклад GET, POST, PUT, DELETE – дозволяє зрозуміти, яку операцію було виконано.
path	String	URL-шлях, до якого було здійснено запит (наприклад, /api/v1/courses/123).
status_code	Integer	Код відповіді сервера (наприклад, 200, 403, 404), який дозволяє швидко визначити, чи завершився запит успішно.
ip_address	String	IP-адреса пристрою користувача для контролю безпеки та виявлення підозрілої активності.
user_agent	String	Дані про браузер та операційну систему користувача (User-Agent) – допомагає у розслідуванні інцидентів.
duration	Float	Час у секундах, який знадобився серверу для обробки цього запиту.
timestamp	DateTime	Дата й час виконання дії, що дає змогу побудувати хронологію подій.
query_params	JSON	Параметри запиту у вигляді JSON-структури для точнішого аудиту, зокрема GET-запитів.
action	String	Короткий опис або тип дії, наприклад, "login", "update_profile", "delete_course" – для полегшення аналітики логів.
request_body	Text	Вміст тіла запиту, якщо він був (наприклад, дані з форми чи JSON-повідомлення).

Це журнал аудиту всієї системи: фіксує всі дії користувачів, інформацію про HTTP-запити, IP-адреси, статуси, параметри, тіло запитів, дати й час – все, що

дозволяє відслідковувати безпеку, аналізувати активність і знаходити потенційні проблеми.

Уся взаємодія серверної частини системи з базою даних відбувається через ORM (Object-Relational Mapping) – на практиці, у цьому проєкті використовується SQLAlchemy. Саме ORM дозволяє розробникам працювати з даними у вигляді об'єктів, а не сирих SQL-запитів. Кожна модель у коді є відображенням відповідної таблиці в базі: наприклад, клас Student відповідає таблиці students, а всі поля класу – це відповідні стовпці у цій таблиці.

Завдяки ORM вдається досягти чистого розділення між логікою додатку й механізмами зберігання даних, що спрощує не лише розробку, але й тестування, масштабування, рефакторинг і міграції структури БД. Дані у вигляді об'єктів можна передавати між різними модулями, обробляти їх, фільтрувати, агрегувати – і все це, не вдаючись до ручного написання складних SQL-запитів.

Велика перевага ORM у можливості легко будувати складні запити з використанням механізмів фільтрації, агрегації, приєднання пов'язаних об'єктів (joinedload, selectinload тощо), що значно підвищує ефективність при отриманні даних для складних інтерфейсів: наприклад, вивести сторінку студента разом з усіма його курсами, оцінками та повідомленнями можна всього кількома запитами. Оскільки SQLAlchemy підтримує асинхронний режим роботи, це дозволяє не блокувати сервер навіть при великій кількості паралельних запитів, що є ключовим для сучасних освітніх платформ із великим навантаженням.

Окремо варто зупинитися на стратегіях роботи з транзакціями. Кожен критичний запис чи оновлення в системі (реєстрація студента, виставлення оцінки, додавання повідомлення в чат) відбувається в межах транзакції – це гарантує, що або всі зміни будуть збережені, або в разі помилки система повернеться до попереднього стану без втрати цілісності. Також важливо, що ORM дозволяє дуже гнучко керувати міграціями схеми БД – у цьому проєкті використовується Alembic, що дає змогу поступово змінювати структуру бази даних, не порушуючи роботу сервісу.

Ще одним аспектом є кешування та оптимізація доступу. Для пришвидшення часто виконуваних запитів (наприклад, отримання інформації про курси чи профілі

користувачів) у системі застосовується Redis – це дає змогу зменшити навантаження на основну БД, прискорити рендеринг сторінок і знизити час очікування для користувача. Для тривалих операцій або тих, що не вимагають негайної реакції користувача (наприклад, масова розсилка чи аналітика) використовується черга завдань (Celery), де завдання виконуються асинхронно, а результати зберігаються у БД.

Завдяки правильній організації доступу до БД, використанню асинхронних механізмів і продуманій структурі моделей, система забезпечує не лише швидкий і надійний доступ до будь-якої інформації, а й гарантує цілісність, захист і масштабованість даних навіть при значному рості кількості користувачів та ускладненні функціоналу. Всі ці аспекти дозволяють системі бути сучасною, надійною та такою, що легко розвивається у відповідь на нові виклики в освітній сфері.

На цій діаграмі класів (рис. 2.4.1) відображена моя структура основних моделей інформаційної системи управління академічними ресурсами студентів. Центральне місце займає клас User, від якого через спадкування утворені ролі Student, Teacher та Admin. Реалізовано зв’язки з іншими сутностями: сесіями (Session), журналом аудиту (AuditLog), токенами скидання пароля (PasswordResetToken), ключами двофакторної автентифікації (TOTPSecret), чатами та повідомленнями. Кожна роль користувача пов’язана з відповідними навчальними, адміністративними або організаційними моделями, такими як факультети, кафедри, групи, освітні програми.

Також є діаграми присвячено освітньому процесу. Тут відображено зв’язки між курсами (Course), модулями курсу (CourseModule), уроками (Lesson), реєстрації на курс (StudentCourse), участі викладачів (TeacherCourse). Уся система побудована з урахуванням можливості деталізованого відстеження взаємодії користувачів із навчальними матеріалами, оцінюванням, процесом авторизації, комунікацією (чат), поданням документів і контролем доступу. Це дозволяє системі бути не лише структурно цілісною, але й максимально гнучкою для впровадження в різних навчальних закладах.

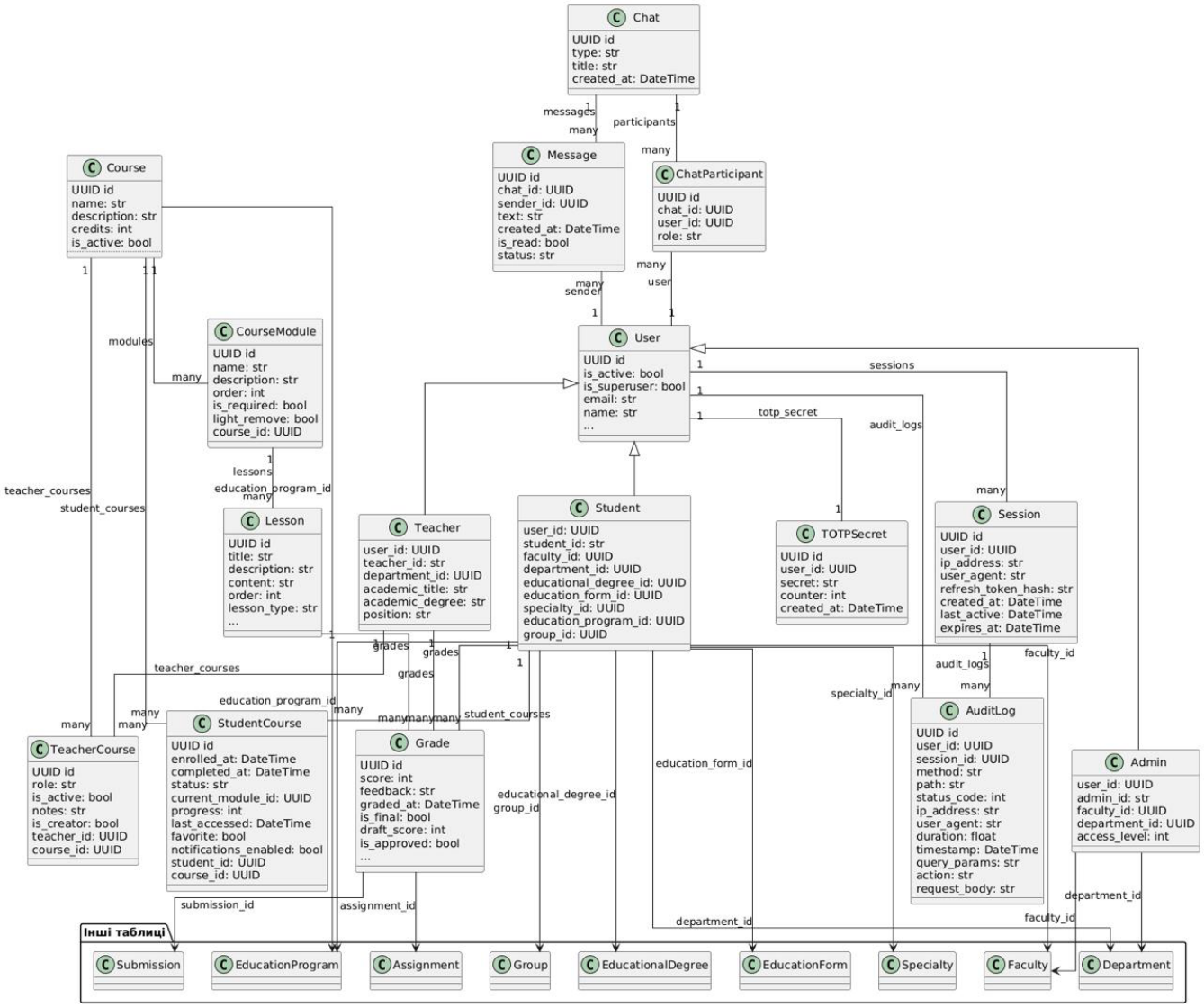


Рис. 2.4.1 – Діаграма всіх класів проекта

2.4. Проектування та розробка алгоритмів роботи системи.

На сучасному етапі розвитку програмного забезпечення під алгоритмами розуміють не лише формальні послідовності команд чи циклів, а й широкий спектр організаційних і структурних сценаріїв: як відбувається обробка подій, у якій послідовності виконується взаємодія користувача із системою, що саме відбувається у відповідь на певну дію, яким чином забезпечується захист даних, контроль доступу, аудит, а також виконуються критично важливі перевірки.

Розробка освітньої платформи передбачає наявність великої кількості логічних ланцюжків, які визначають роботу таких підсистем, як автентифікація й авторизація користувачів, створення й керування курсами, робота з чатами,

система оцінювання, облік активних сесій, двофакторна автентифікація та формування детального журналу подій. Кожна з цих підсистем має власну структуру роботи, набір перевірок, механізми фіксації результатів і реагування на різні ситуації – від стандартних сценаріїв, як-от вхід у систему, до більш складних, як аудит змін чи інтеграція нових модулів.

Деталізація таких алгоритмів необхідна не лише для розуміння внутрішньої логіки додатку, а й для забезпечення прозорості, масштабованості та підтримуваності всієї системи. Саме тому для кожного із ключових функціональних блоків наведено послідовність операцій, які відбуваються при виконанні відповідних дій, із акцентом на контроль доступу, безпеку, цілісність і збереження даних. Для кращої візуалізації й наочності для кожного важливого елемента наведено відповідну UML-діаграму або схему, що дозволяє швидко зрозуміти загальну логіку й порядок виконання дій навіть тим, хто не брав участі у безпосередній розробці системи.

Процес автентифікації та авторизації є критичним для будь-якої інформаційної системи, де необхідно контролювати доступ до ресурсів і захищати особисті дані користувачів. У розробленій системі цей алгоритм охоплює декілька логічних етапів, які забезпечують не лише перевірку особистості користувача, але й визначення його ролі та рівня доступу.

Коли користувач ініціює спробу входу до системи, його запит обробляється на рівні бекенду, де перш за все перевіряється наявність облікового запису за вказаною електронною поштою. Якщо запис знайдено, система звіряє пароль, порівнюючи його хеш із тим, що зберігається в базі даних. При успішній перевірці активності облікового запису — якщо обліковий запис ще не активований, користувач перенаправляється до процедури активації через email чи спеціальний ідентифікатор.

Для користувачів, у яких увімкнена двофакторна автентифікація (TOTP), після базової перевірки системою повертається запит на введення додаткового одноразового коду, що генерується окремим додатком. Лише після успішного проходження цього етапу користувач отримує доступ до системи.

Якщо всі перевірки пройдено, створюється нова сесія користувача з прив'язкою до його пристрою, IP-адреси та інформації про браузер (user-agent). Генеруються токени доступу — access_token і refresh_token, які повертаються у відповідь, а також встановлюються як cookies, що захищає від спроб несанкціонованого доступу. У токенах закодована інформація про роль користувача (студент, викладач, адміністратор), що дозволяє контролювати рівень доступу до функцій системи. Подальші запити користувача перевіряються через цей токен — у разі зміни сесії, втрати дійсності або підозрілої активності відбувається автоматичне завершення сесії й блокування доступу (рис. 2.5).

Завдяки цьому алгоритму система забезпечує не лише високий рівень безпеки, але й гнучкість – можлива інтеграція додаткових ролей, розмежування прав доступу та підключення додаткових механізмів захисту (наприклад, геолокаційних обмежень чи аналізу аномальної поведінки).

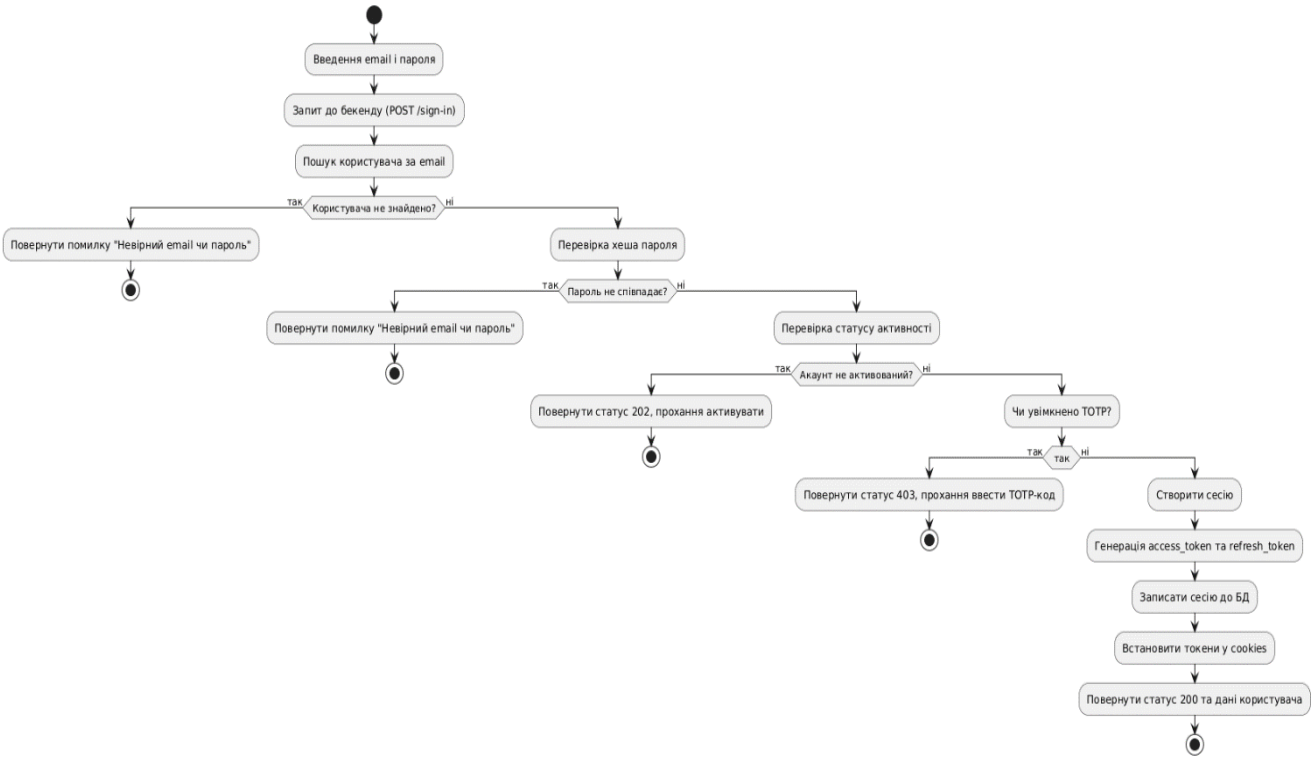


Рис. 2.5 - Процес входу в систему

Реєстрація нового користувача – це багатоступеневий процес, який передбачає перевірку унікальності електронної пошти, створення нового облікового запису та відправку підтвердження на email. Спочатку користувач надсилає дані для реєстрації (ім'я, email, пароль, тип користувача – студент, викладач чи

адміністратор). Сервер одразу перевіряє, чи існує такий email у системі: якщо вже є відповідний обліковий запис, процес зупиняється з повідомленням про помилку.

У разі, якщо email унікальний, система формує новий об’єкт користувача потрібного підтипу (наприклад, Student або Teacher), ініціалізує унікальний ідентифікатор відповідної категорії, та зберігає всі дані в базу. Запис одразу містить потрібні зв’язки з факультетами, кафедрами чи навчальними програмами – це гарантує цілісність даних з самого початку (рис. 2.6).

Після успішного створення профілю, сервер надсилає електронного листа з підтвердженням реєстрації й унікальним ідентифікатором (активаційним кодом чи ID). Користувач, отримавши цей лист, може перейти до активації облікового запису. Завершення цього процесу відкриває повний доступ до функцій системи.

Усі перевірки та логіка розподілені між контролерами FastAPI, асинхронним CRUD-класом та спеціалізованими функціями для генерації ідентифікаторів і завантаження пов’язаних сутностей через selectinload.

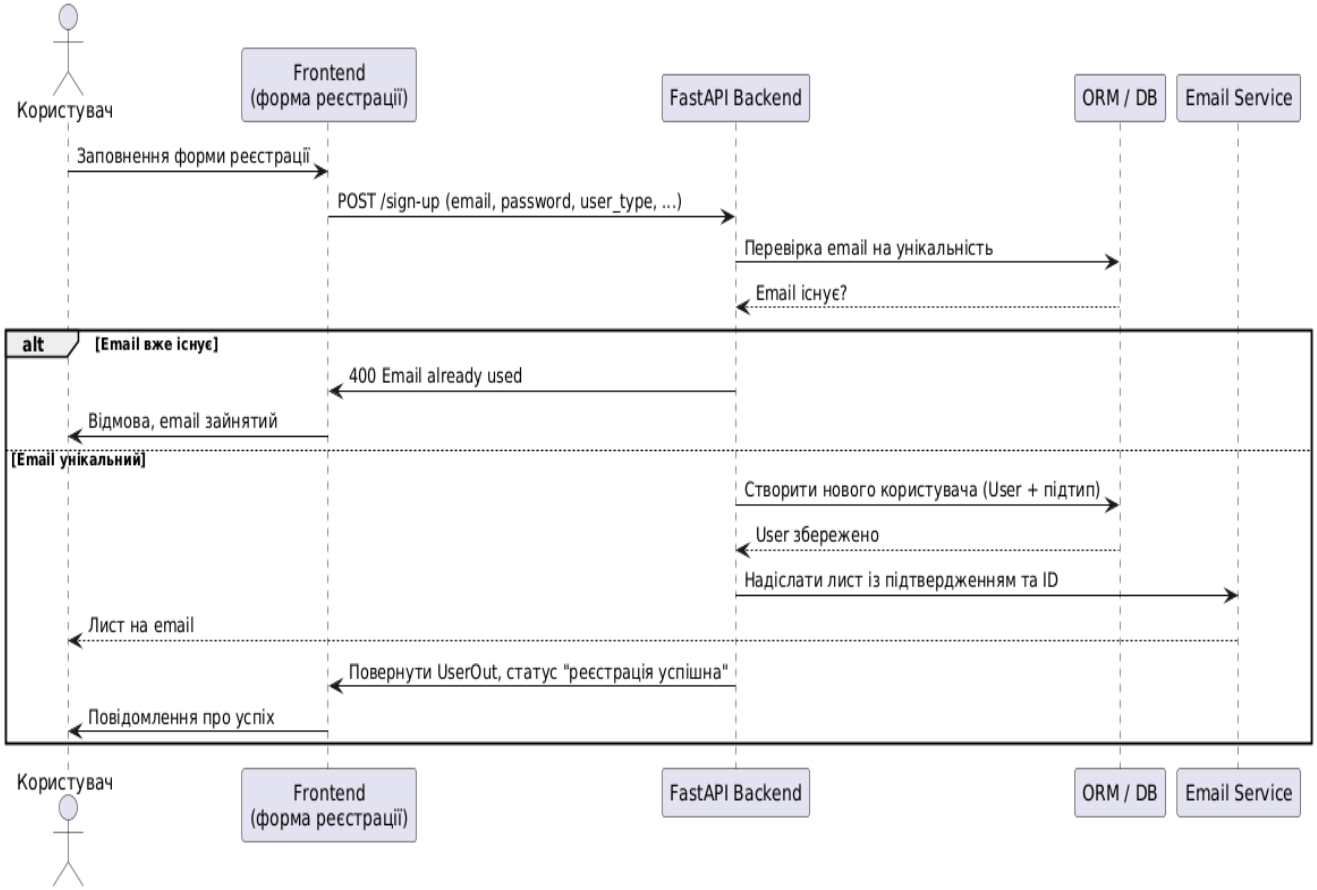


Рис. 2.6 - Sequence Diagram реєстрація користувача

Створення й редагування курсу – це процес, доступний лише адміністраторам і викладачам, причому викладач повинен бути приєднаний до курсу або бути його засновником. Користувач (admin чи teacher) надсилає дані курсу на сервер. Сервер спершу перевіряє права доступу: студент або сторонній користувач отримає відмову.

Далі йде перевірка унікальності назви курсу. Якщо курс із таким ім’ям уже існує, запит завершується з помилкою. Якщо ні – створюється новий об’єкт курсу з усіма введеними полями (назва, опис, кредити, прив’язка до освітньої програми, періоди проведення, тощо), після чого об’єкт записується до бази даних. Якщо ініціатор – викладач, його одразу додають до курсу як засновника (is_creator = True) (рис. 2.7).

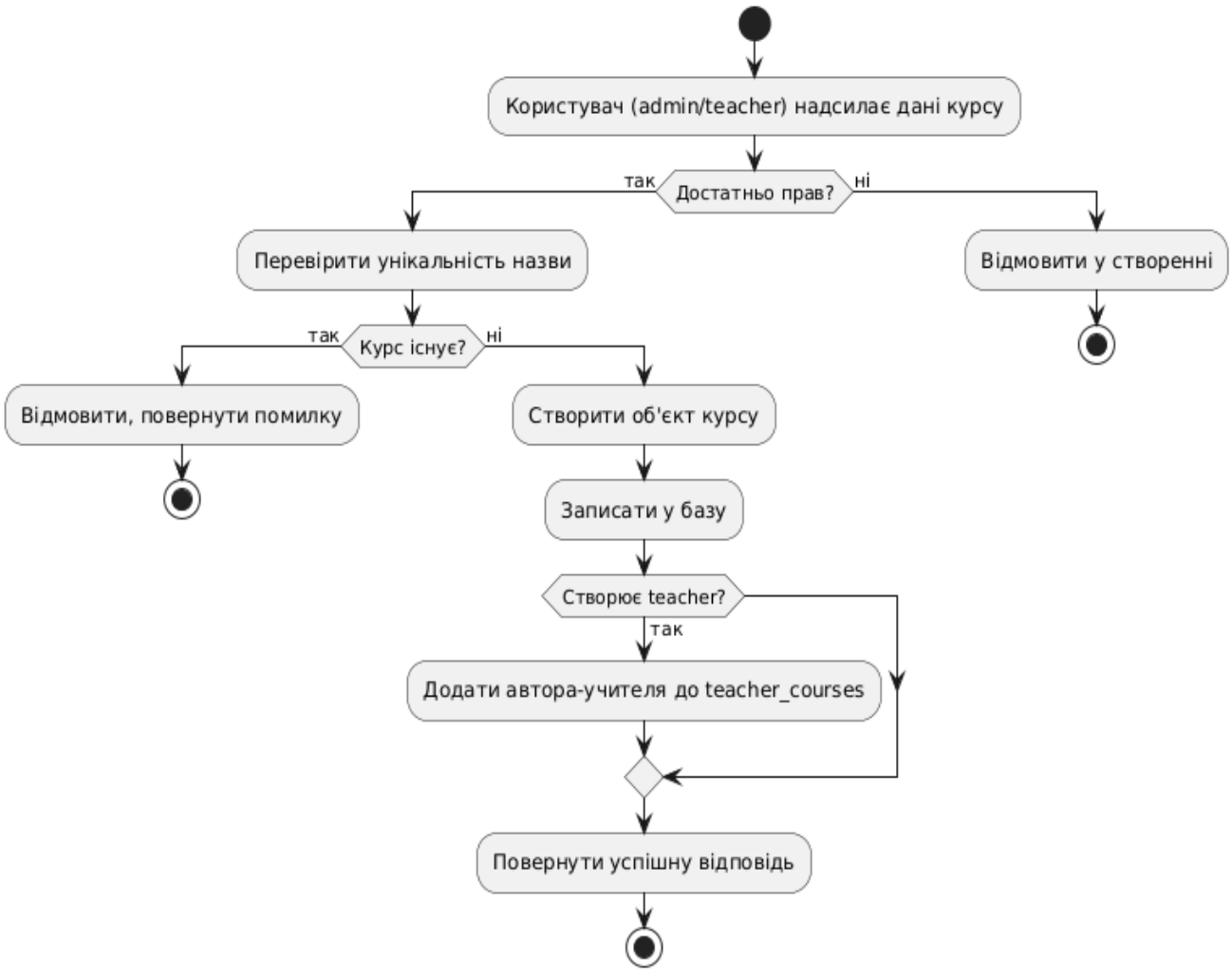


Рис. 2.7 - Activity Diagram створення курсу

Редагування курсу теж має перевірку доступу: редагувати може або адміністратор, або викладач, який вже приєднаний до цього курсу. Усі зміни проходять валідацію та зберігаються тільки після проходження перевірок.

Видалення (або «легке видалення») курсу можливе лише для адміністратора чи для засновника-учителя. При цьому курс фактично не видаляється з бази, а позначається як неактивний, що дозволяє зберегти історію та пов’язані дані.

Додавання/видалення викладача чи студента до курсу, а також робота з модулями курсу відбувається через аналогічну перевірку прав і маніпуляцію зв’язками у відповідних таблицях (teacher_courses, student_courses, course_modules) (рис. 2.8).

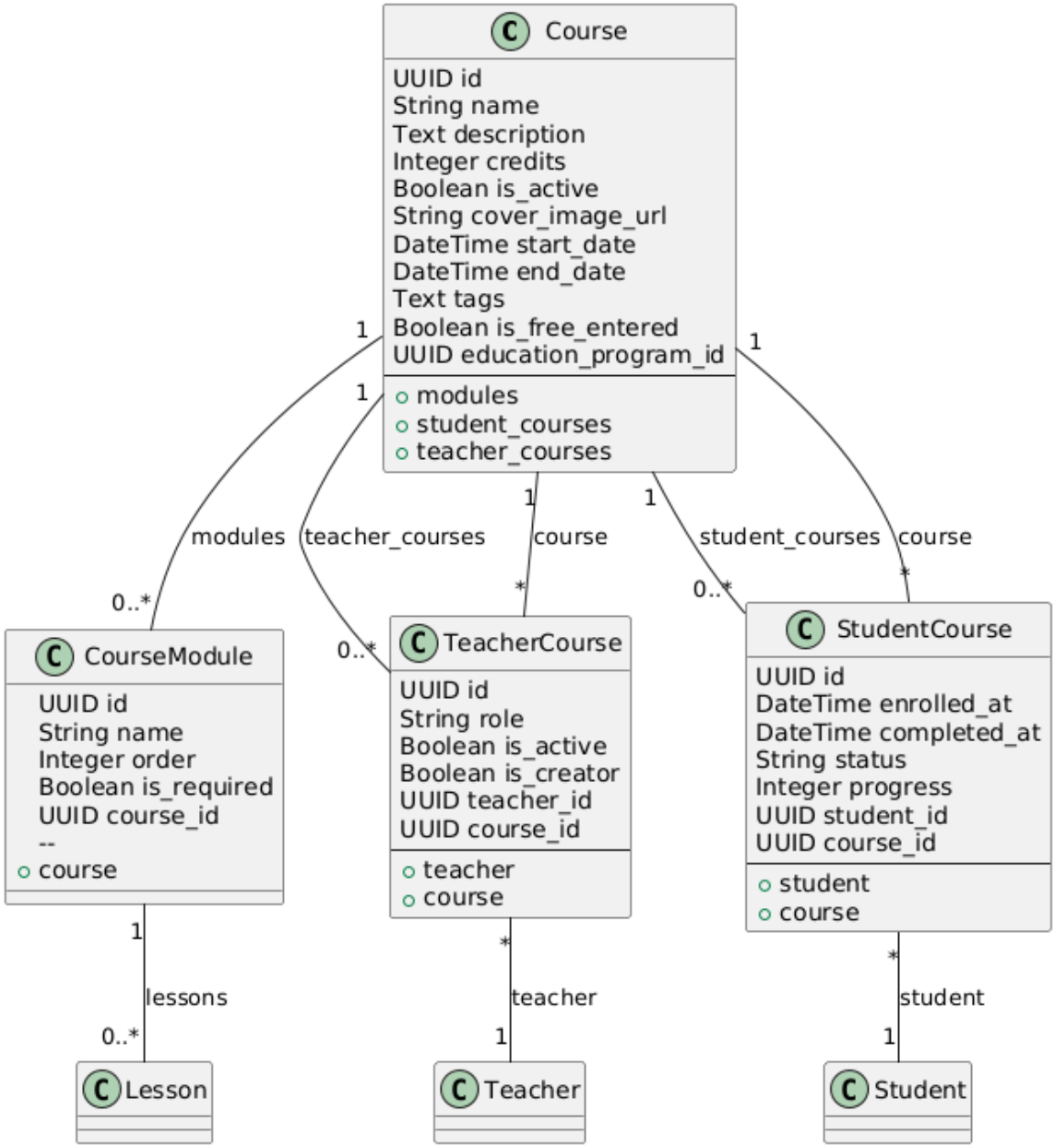


Рис. 2.8 - Class Diagram структура курсу та зв’язків

Система чатів реалізована через окремі сутності: чати, учасники чату та повідомлення. Діалог або груповий чат створюється тоді, коли користувач ініціює розмову з іншим користувачем або додає декількох учасників до нового чату. Для діалогу спочатку перевіряється, чи вже існує чат із заданими учасниками – якщо так, повертається існуючий; якщо ні, створюється новий запис Chat і для кожного учасника – відповідний запис у ChatParticipant (рис. 2.9).

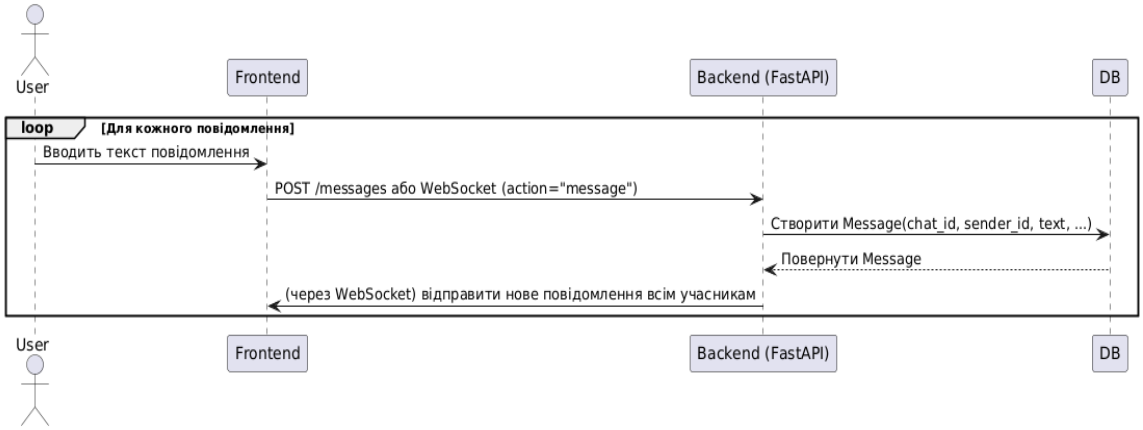


Рис. 2.9 - Sequence Diagram відправка повідомлення

Додавання повідомлення (через REST та WebSocket) – це створення нового об'єкта Message з прив’язкою до чату і відправника. Одразу після запису повідомлення система через WebSocket транслює це повідомлення всім підключеним учасникам цього чату (рис. 2.10).

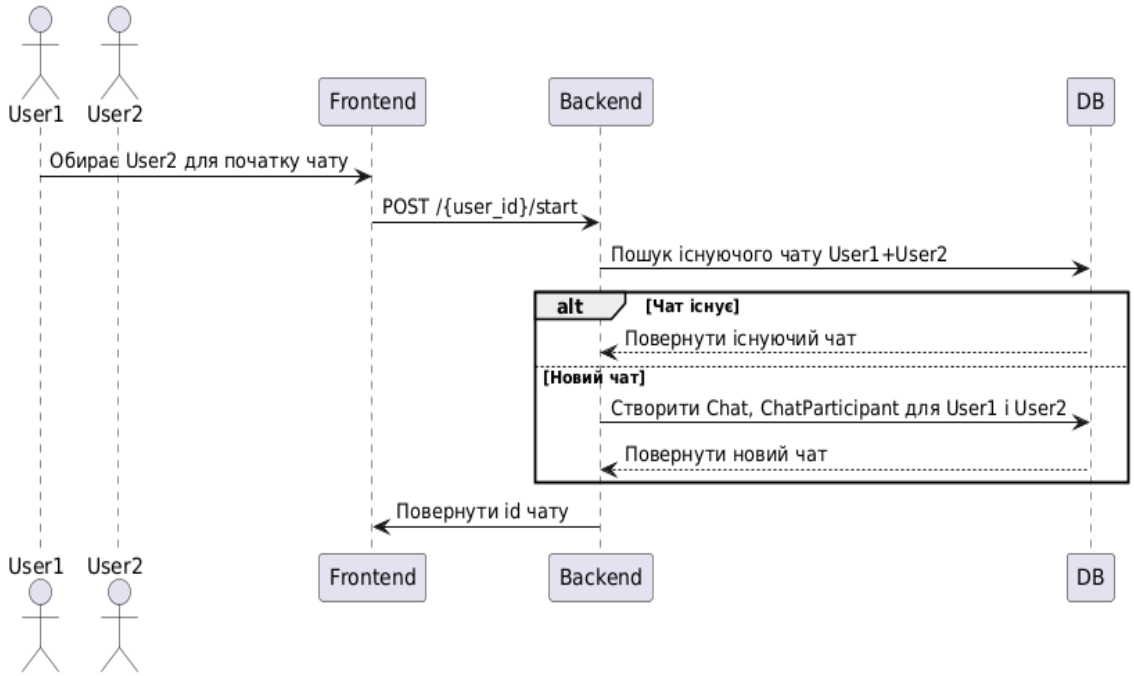


Рис. 2.10 - Sequence Diagram створення діалогу

Видалення чату доступне лише його учасникам: запит на видалення перевіряє, чи поточний користувач є серед учасників, і тільки після цього видаляє чат та всі пов’язані записи (рис. 2.11).

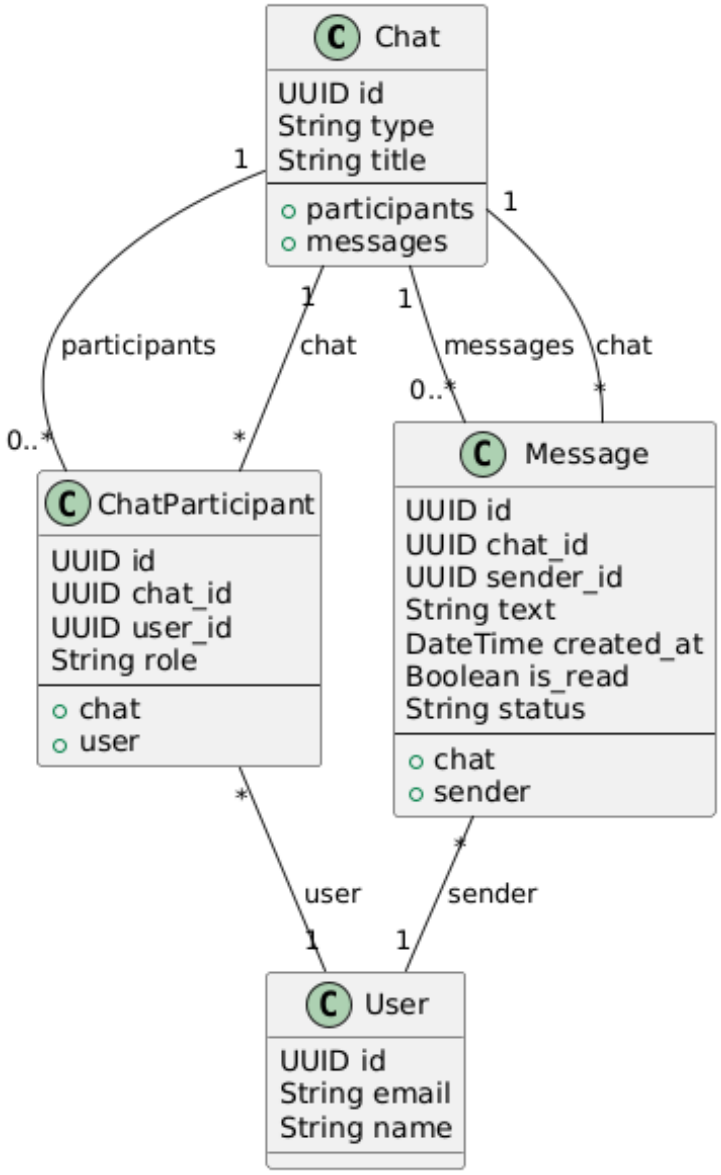


Рис. 2.11 - Class Diagram структура курсу та зв’язків

Система оцінювання реалізує класичний сценарій навчального журналу з можливістю як поетапного, так і підсумкового оцінювання студента. Основними діючими особами є викладач, студент і система. Викладач має право виставляти оцінки як за конкретні завдання (assignment), так і підсумкові оцінки за курс. При додаванні оцінки обов’язково вказуються студент, курс/завдання, бал, коментарі й параметри (наприклад, чи здав студент із запізненням, чи є апеляція) (рис. 2.13).

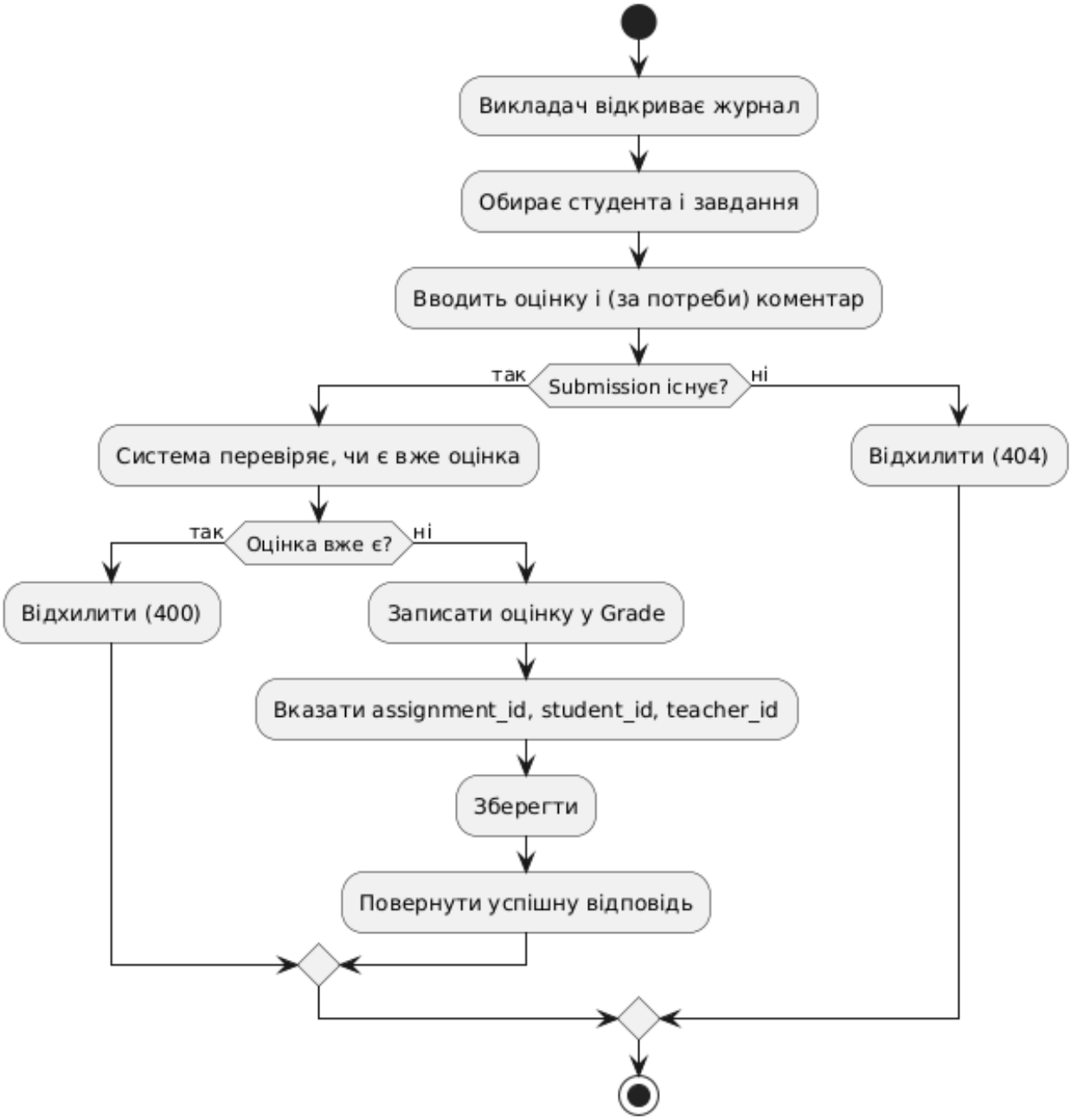


Рис. 2.13 - Activity Diagram виставлення оцінки за завдання

Оцінка створюється через відповідний endpoint тільки для студентів, які реально подали роботу (submission), а система перевіряє, чи не дублюється запис для того самого завдання й студента. Після створення або оновлення оцінки студент може переглядати її через особистий журнал або сторінку завдання, а також має право подати апеляцію, якщо не згоден із результатом.

Підтвердження оцінки студентом (approve/unapprove) реалізовано через окремі endpoints. Це дозволяє фіксувати остаточні результати і дає прозору історію змін для журнала (рис. 2.14).

Викладач бачить усі оцінки своїх студентів і може швидко отримати підсумковий бал за курс. Адміністратор має доступ до всієї історії оцінок у системі. Всі дії – створення, оновлення, видалення оцінок, подання апеляції – фіксуються у журналі аудиту для контролю прозорості та безпеки.

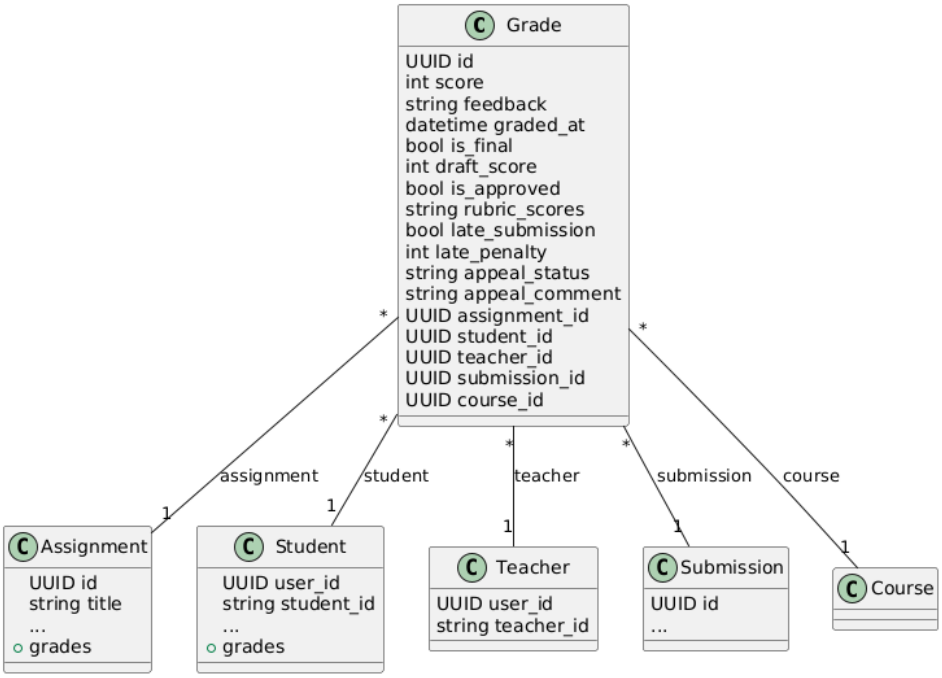


Рис. 2.14 - Class Diagram структура оцінювання

Двофакторна автентифікація (TOTP) реалізована для підвищення рівня безпеки при вході до системи. Коли у користувача активовано двофакторний захист, під час авторизації (після введення правильного email і пароля) система перевіряє прапорець totp_enabled у профілі користувача. Якщо TOTP активний, на наступному етапі користувач повинен ввести тимчасовий код, згенерований мобільним додатком (наприклад, Google Authenticator).

Усі секретні ключі для генерації кодів зберігаються в окремій таблиці totp_secrets (прив’язаній до користувача, поле user_id). Валідація відбувається на бекенді – отриманий від користувача код звіряється з кодом, який генерується по збереженому секрету для поточного часу (рис. 2.15).

Поки користувач не підтвердить правильний TOTP-код, доступ до основної системи не надається, а сесія авторизації не завершується. Після успішної перевірки TOTP сесія стає активною, а користувач отримує токени для роботи з API.

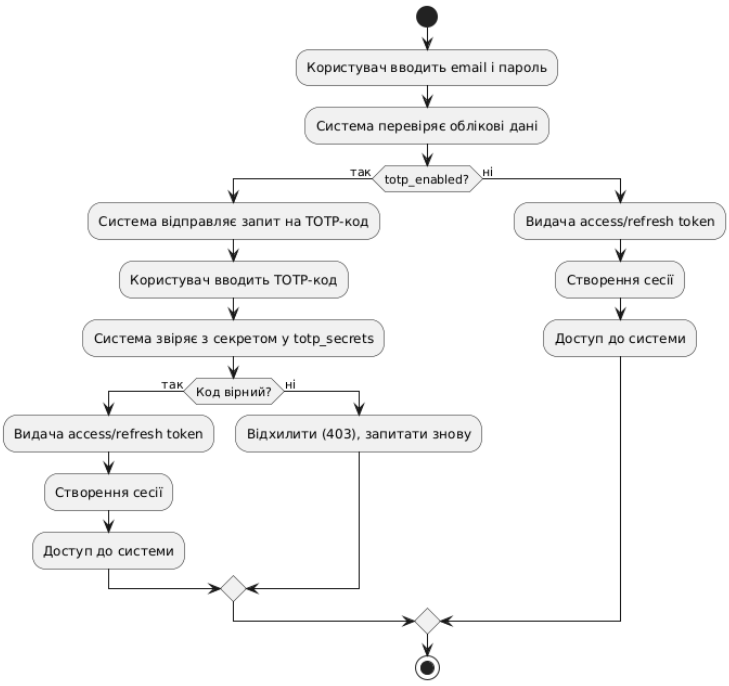


Рис. 2.15 - Activity Diagram TOTP-перевірка при вході

Усі суттєві дії користувача в системі (автентифікація, зміна даних, доступ до чутливої інформації, спроби несанкціонованого доступу) автоматично фіксуються в таблиці audit_logs**.** Для кожного HTTP-запиту, що змінює або отримує дані, створюється запис із прив’язкою до користувача (user_id), поточної сесії (session_id), параметрами запиту, HTTP-методом, шляхом, статус-кодом відповіді, IP-адресою, user-agent’ом та іншими атрибутами (рис. 2.16).

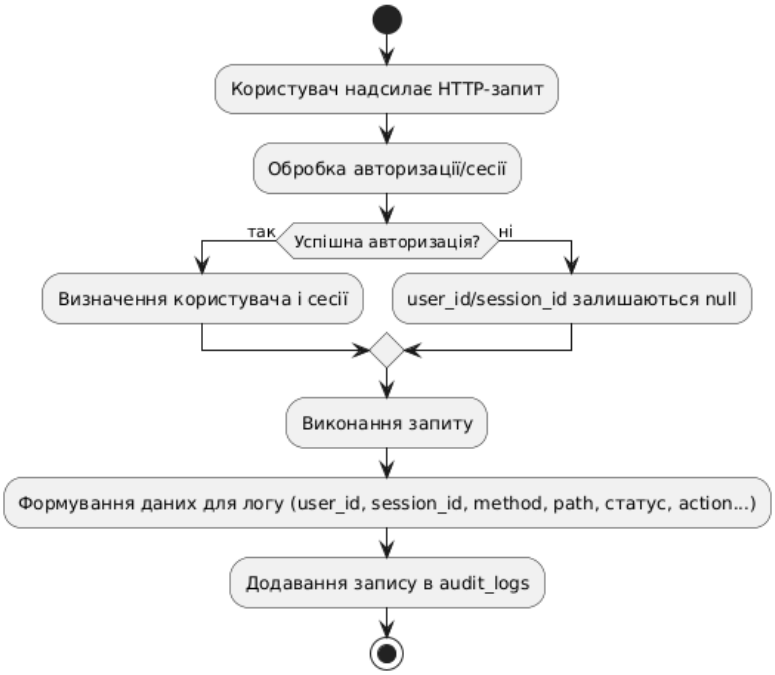


Рис. 2.16 - Activity Diagram створення логів аудиту під час запиту

Кожен вхід у систему супроводжується створенням сесії (таблиця sessions). Сесія містить інформацію про користувача, refresh-токен (у хешованому вигляді), дату створення, час останньої активності й час завершення дії сесії. При кожній дії користувача оновлюється поле last_active. Видалення сесії (логаут, закінчення терміну, вихід із усіх пристроїв) автоматично розриває зв'язок із журналом аудиту – всі логи залишаються, але нові події не додаються до цієї сесії (рис. 2.17).

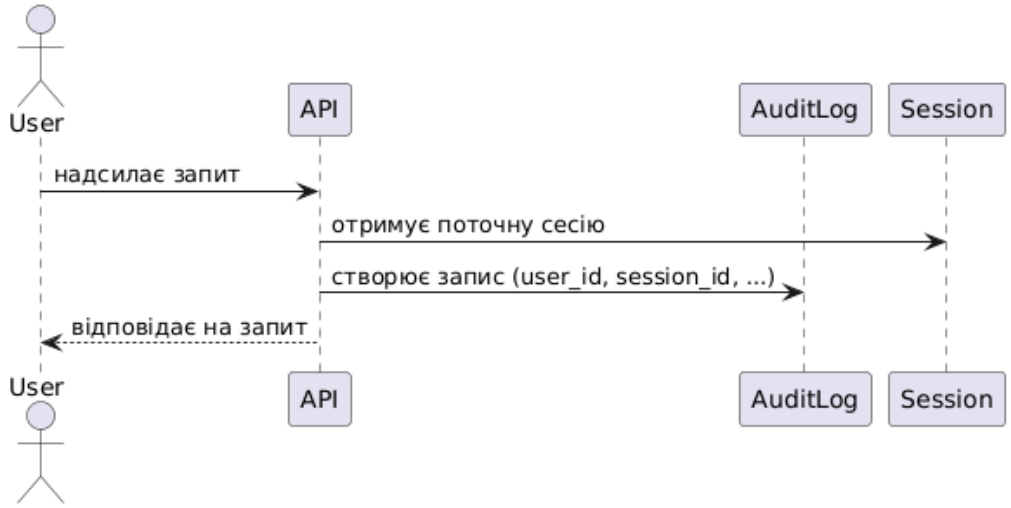


Рис. 2.17 - Sequence Diagram як формується лог

Записи журналу аудиту використовуються не тільки для контролю безпеки, а й для аналітики: підрахунок популярних endpoint, моніторинг помилок, пошук спроб підбору паролів чи спроб доступу до заборонених ресурсів (рис. 2.18).

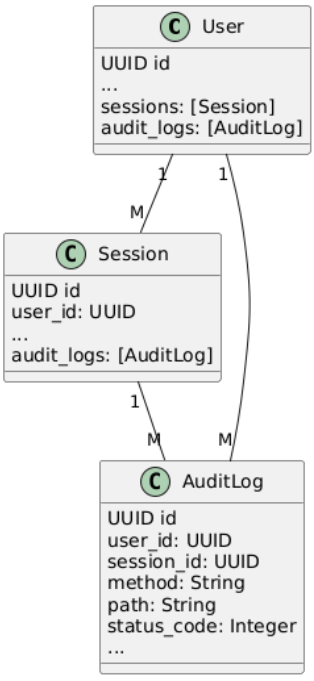


Рис. 2.18 - Class Diagram схематично

Завершуючи можна підкреслити, що грамотне опрацювання логіки й сценаріїв взаємодії – це основа надійної, гнучкої та захищеної інформаційної платформи. Для кожного ключового блоку – автентифікації, реєстрації, навчальних курсів, чатів, оцінювання, двофакторної верифікації, ведення журналу сесій та аудиту – було визначено чіткі алгоритми роботи, рольову структуру доступу, а також оптимальні способи взаємодії між компонентами системи.

Використання сучасних підходів, таких як ORM, централізоване логування подій, розподіл ролей і асинхронна обробка запитів, забезпечує не лише функціональність, а й масштабованість, зручність для користувачів і відповідність сучасним вимогам безпеки. Діаграми UML і описані послідовності операцій дозволяють легко інтегрувати нові функції або змінювати вже реалізовані блоки без втрати цілісності проєкту. В результаті, система залишається зрозумілою для розробників, прозорою для адміністраторів та зручною для кінцевих користувачів.

Висновки до другого розділу

У цьому розділі було всебічно проаналізовано та деталізовано ключові аспекти проєктування інформаційної системи управління академічними ресурсами студентів. На етапі визначення вимог велика увага приділялася не лише формальним функціональним можливостям, а й практичним питанням масштабованості, продуктивності, безпеки та інтеграції із зовнішніми сервісами. Завдяки цьому вдалося сформувати концептуальну основу, що орієнтує всю подальшу розробку на реальні потреби кінцевих користувачів: студентів, викладачів, адміністраторів та представників адміністрації.

Важливою частиною роботи стало моделювання ролей і сценаріїв взаємодії користувачів із системою. Це дало змогу не лише структуровано представити логіку розподілу прав доступу, а й закласти основи для ефективної організації бізнес-процесів і захисту даних. Візуалізація архітектури за допомогою UML-діаграм і схем послідовності сприяла кращому розумінню внутрішніх процесів як для розробників, так і для майбутніх користувачів системи [54].

Окремий акцент зроблено на об’єктно-орієнтованому підході до проєктування ядра системи: були визначені основні сутності, їх атрибути та зв’язки, що

лягли в основу структурної моделі. Заздалегідь продумана ієрархія класів, чіткий розподіл відповідальності й інкапсуляція дозволили не лише підвищити гнучкість коду, а й спростити підтримку та масштабування проєкту [50].

Особлива увага приділялася побудові структури бази даних, обґрунтованому вибору реляційної моделі на основі PostgreSQL та застосуванню ORM SQLAlchemy[53]. Такий підхід забезпечив цілісність і консистентність даних, а також полегшив подальшу інтеграцію з іншими сервісами через стандартні протоколи й API [51].

На додаток до цього, у розділі розкрито питання алгоритмізації ключових процесів системи: від автентифікації й контролю доступу до роботи з навчальними модулями, чатами, оцінюванням і веденням журналів дій. Описані послідовності операцій і логічні сценарії роботи забезпечують надійність, безпеку й прозорість усіх функціональних блоків.

У результаті, на основі глибокого аналізу вимог, сценаріїв використання, моделювання БД і розробки алгоритмів, закладено міцне підґрунтя для ефективної, масштабованої та безпечної системи управління академічними ресурсами, яка відповідає сучасним викликам цифрової освіти. Комплексний підхід дозволяє створити дійсно зручне, надійне й адаптивне середовище для всіх учасників освітнього процесу – студентів, викладачів, адміністраторів та управлінців [52].

РОЗДІЛ 3. ІНТЕРФЕЙС ТА АЛГОРИТМ РОБОТИ ТА ВИКОРИСТАННЯ СИСТЕМИ УПРАВЛІННЯ АКАДЕМІЧНИМИ РЕСУРСАМИ

3.1. Порядок налаштування параметрів системи управління

Розроблена система управління академічними ресурсами студентів орієнтована на інтеграцію саме в середовище конкретного закладу вищої освіти. При проєктуванні було враховано, що кожен університет має свою унікальну організаційну структуру, набір навчальних програм, політики доступу й власні бізнес-процеси. Саме тому система має гнучкі механізми налаштування, які дозволяють адаптувати платформу під потреби і масштаби конкретного навчального закладу.

Для роботи платформи передбачається встановлення на виділеному сервері університету або у хмарному середовищі з надійним резервуванням даних. Доступ до системи відбувається через сучасний вебінтерфейс, що розміщується на унікальному доменному імені університету (наприклад, stud-res.university.edu), що гарантує захищений вхід та інтеграцію з внутрішньою інфраструктурою закладу.

Початкове налаштування системи стартує із запуску адміністративного модуля – це спеціальний інтерфейс, доступний після першого розгортання, де створюється головний акаунт адміністратора. Саме адміністратор має найширші права доступу й може виконувати всі дії по конфігурації: додавання факультетів, кафедр, навчальних програм, створення структури груп та спеціальностей, а також заповнення всіх необхідних довідників. Для цього передбачені зручні візуальні майстри та імпорт даних із файлів CSV або навіть інтеграція із зовнішніми електронними реєстрами (наприклад, для автоматичного підключення реальних списків студентів чи викладачів).

У панелі керування адміністратор може одразу налаштовувати базові параметри – вибрати логотип університету, змінити кольорову гаму, прописати контактні дані, налаштувати політики автентифікації (наприклад, чи обов’язкова двофакторна автентифікація, політики щодо складності паролів, тощо), а також обрати

оптимальний спосіб реєстрації для користувачів (реєстрація за посиланням, через запрошення чи ручне додавання).

Дуже важливим етапом є налаштування організаційної та нормативної інформації. Система передбачає зручну ієрархічну структуру для довідників факультетів, кафедр, спеціальностей, груп, освітніх програм, за якими надалі здійснюється автоматизоване формування курсів, групування студентів і викладачів, організація доступу до навчального контенту тощо. Така модель дозволяє з легкістю управляти великими обсягами інформації, уникати дублювання даних та забезпечити цілісність і актуальність довідників.

Окрему увагу приділено гнучкому розподілу ролей та прав доступу. Адміністратор може призначати додаткових адміністраторів, делегувати повноваження для керування окремими факультетами, кафедрами чи навчальними програмами, а також створювати та активувати облікові записи для викладачів і студентів. Додаткові налаштування дозволяють автоматично розсилати повідомлення про активацію акаунтів, зміну ролей чи інші важливі події.

Після базового налаштування адміністратор може поступово вводити в роботу інші модулі системи: додавати навчальні курси, призначати викладачів і студентів на відповідні програми, організовувати розклад занять, структурувати навчальні матеріали, а також використовувати аналітичні інструменти для моніторингу ефективності освітнього процесу.

Всі подальші адміністративні та організаційні дії виконуються через інтуїтивно зрозумілий вебінтерфейс. Система підтримує адаптивний дизайн, завдяки чому керування доступне як із настільних комп’ютерів, так і з мобільних пристроїв. Кожна дія супроводжується підказками й допоміжними інструкціями, що значно спрощує роботу навіть для недосвідчених користувачів.

Таким чином, університет отримує у своє розпорядження інструмент, який дозволяє не лише організувати всю освітню інфраструктуру в єдиному цифровому середовищі, а й швидко адаптувати систему до нових потреб, змін у структурі чи вимогах законодавства, не залучаючи розробників до кожної дрібної зміни. Система є максимально масштабованою та може бути розгорнута як у невеликих

коледжах, так і у великих університетах із тисячами студентів і сотнями освітніх програм (рис. 3.1).

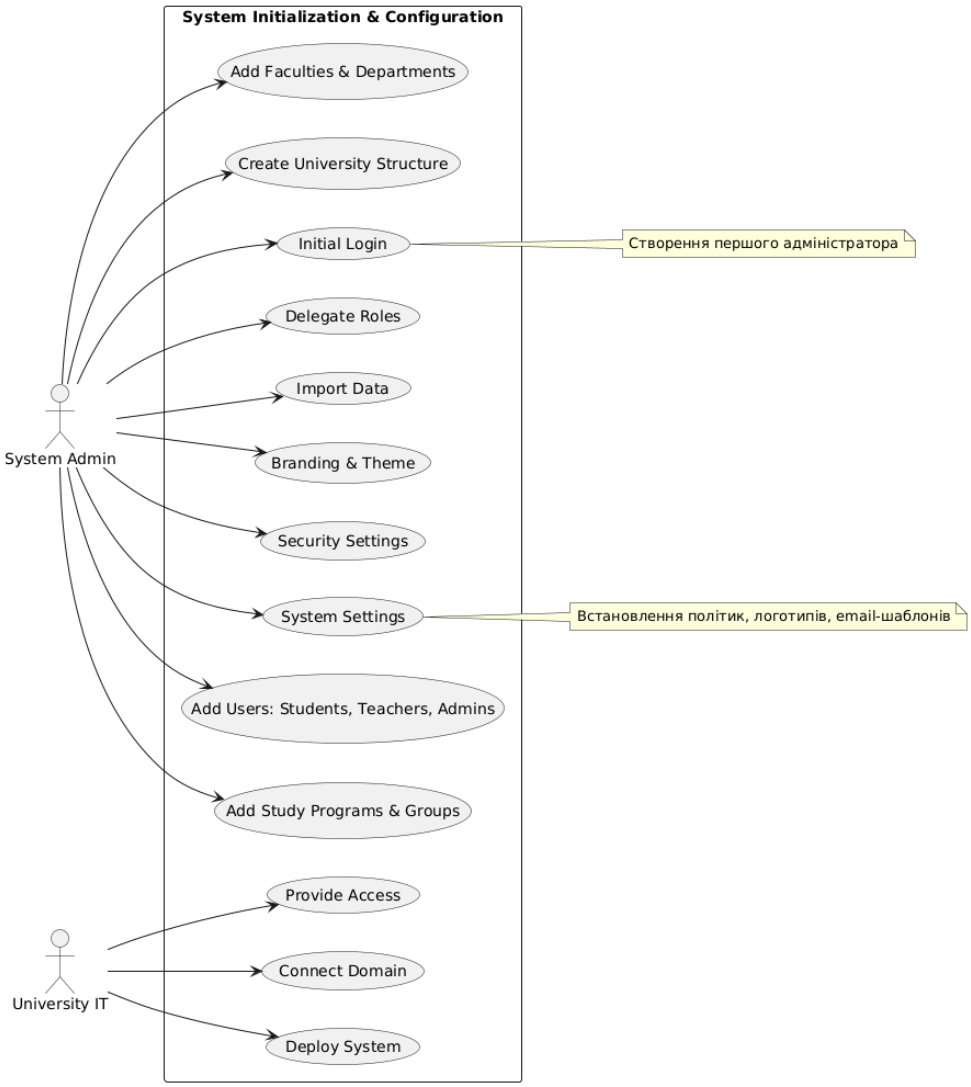


Рис. 3.1 - Порядок налаштування параметрів системи управління

3.2. Структура та алгоритм використання інтерфейсу системи

Тепер опишемо ключові сценарії взаємодії користувачів із інтерфейсом системи управління академічними ресурсами. Для кращого розуміння логіки роботи наведено послідовність дій користувачів, а також відповідні ілюстрації основних екранів і функціональних можливостей. Описані операції охоплюють процеси автентифікації, навігації, взаємодії з навчальними матеріалами, роботу з курсами та модулем чату, а також дії щодо адміністрування й оцінювання. Такий підхід дозволяє прозоро відобразити алгоритм використання системи та продемонструвати її гнучкість, функціональність і зручність для різних ролей.

На даному екрані зображено, як студент отримує повідомлення із запрошенням до системи. Повідомлення містить унікальне посилання для швидкої реєстрації або активації облікового запису. Завдяки цьому процес приєднання до системи спрощується для нових користувачів. Дана операція дозволяє адміністратору або викладачу швидко додавати студентів до навчального процесу. Це значно скорочує час на інтеграцію нових учасників (рис. 3.2).

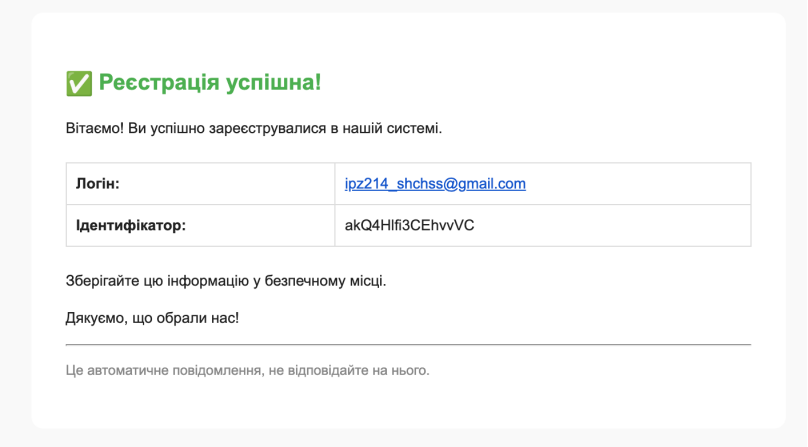


Рис. 3.2 – Приклад отриманого повідолмення запрошеним студентом

Після переходу за посиланням студент бачить форму авторизації, спеціально налаштовану для запрошених користувачів. Тут необхідно ввести електронну пошту та код, наданий в запрошенні. Система перевіряє введені дані та проводить автентифікацію. Така форма мінімізує складність входу для нових студентів і водночас гарантує безпеку даних. Додатково, інтерфейс орієнтований на швидкий доступ до навчальних ресурсів одразу після авторизації (рис. 3.3).

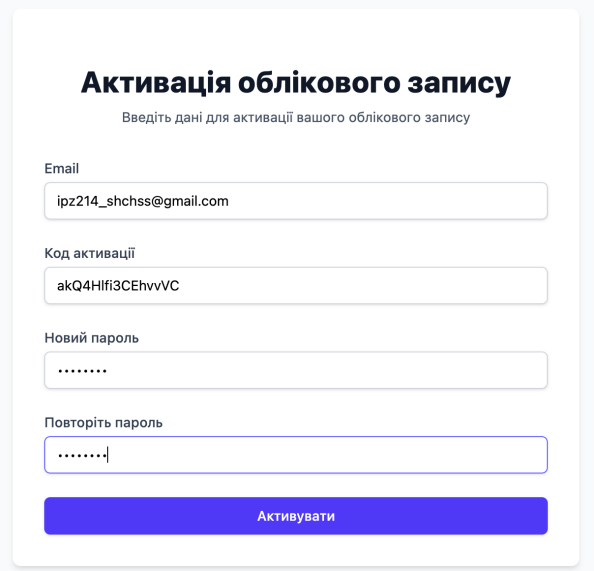


Рис. 3.3 – Форма авторизації запрошеного студента

Сторінка авторизації забезпечує доступ до системи для всіх типів користувачів: студентів, викладачів та адміністраторів. Верифікація облікових даних відбувається із застосуванням сучасних стандартів безпеки. Сторінка є універсальною для всіх ролей, що спрощує підтримку та розширення системи. Інтерфейс містить підказки для зручності користувачів (рис. 3.4).

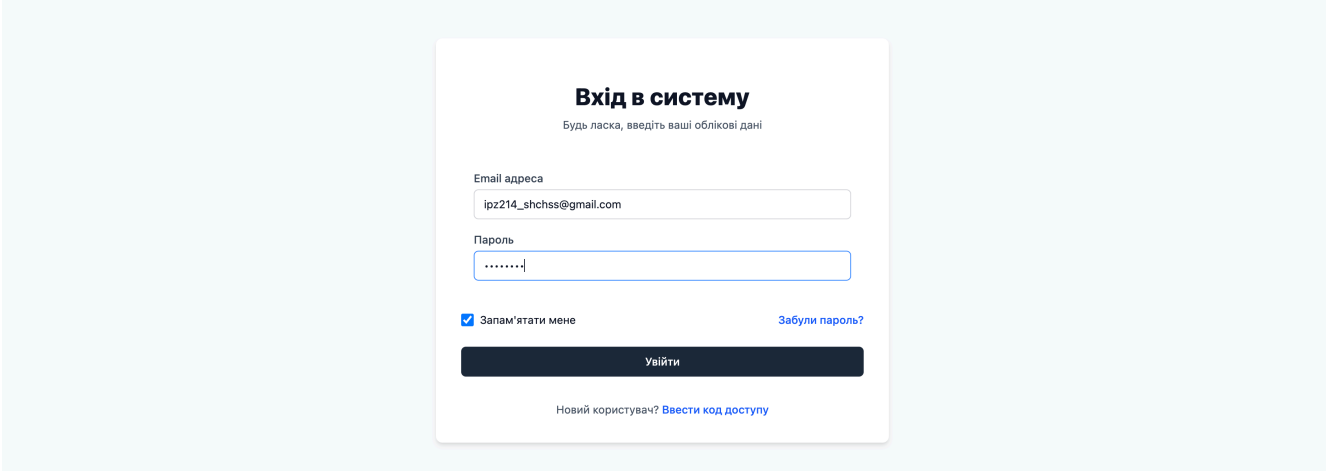


Рис. 3.4 – Сторінка авторизації

Задля безпеки в систему додана можливість активувати додатковий рівень захисту через OTP-додаток або SMS. Користувач має просканувати QR-код або ввести секретний ключ у додатку-генераторі. Після активації двофакторна автентифікація захищає обліковий запис навіть у випадку компрометації паролю. Це важливий крок для підвищення безпеки освітнього середовища (рис. 3.5).

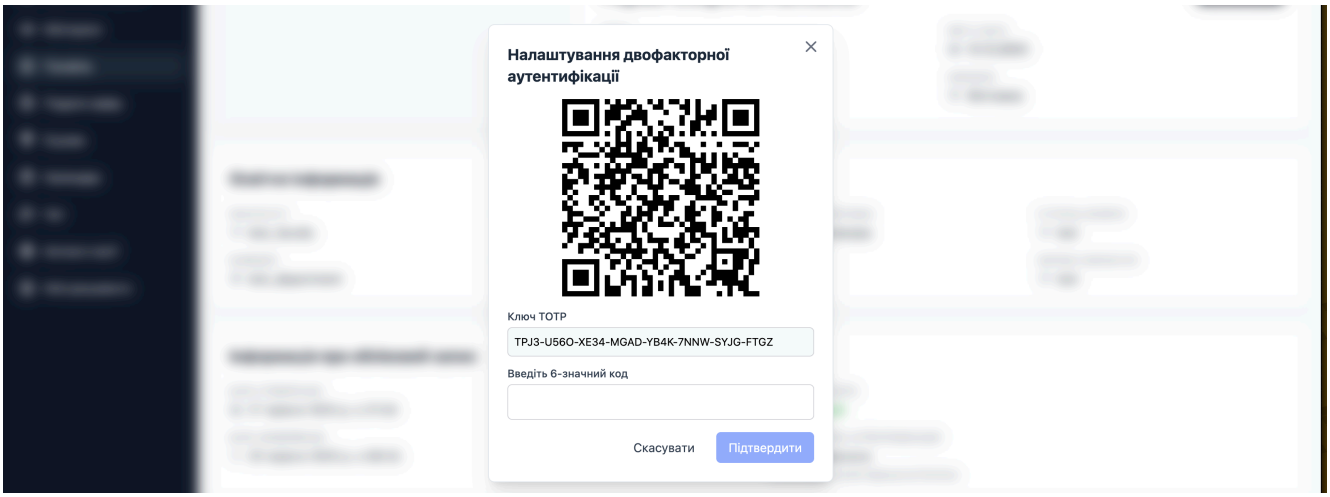


Рис. 3.5 – Сторінка активації двофакторної авторизації

На цьому екрані користувач вводить одноразовий код із додатку для підтвердження входу. Система автоматично перевіряє валідність коду та дозволяє або

блокує доступ до облікового запису. У разі помилки можна повторити спробу або запросити новий код. Цей крок гарантує, що доступ до облікового запису матиме тільки справжній власник пристрою. Оформлення сторінки інтуїтивно зрозуміле й зручне для різних пристроїв. Також тут можна не натискати кнопку «Підтвердити», якщо код вірний – форма сама сабмітнется (рис. 3.6).

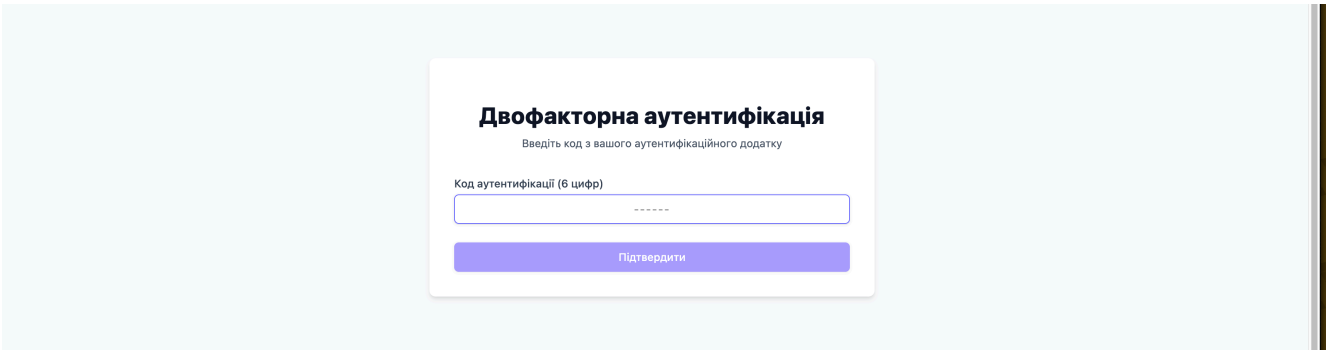


Рис. 3.6 – Сторінка введення отп коду

Головна сторінка адміністратора містить оглядову інформацію про стан системи та швидкий доступ до основних адміністративних функцій. Тут можна переглядати та керувати користувачами, курсами, ролями, а також бачити статистику активності. Інтерфейс забезпечує швидкий пошук і фільтрацію даних. Для безпеки адміністративні дії логуються та мають багаторівневу систему доступу. Це підвищує контроль і прозорість у системі управління (рис. 3.7).

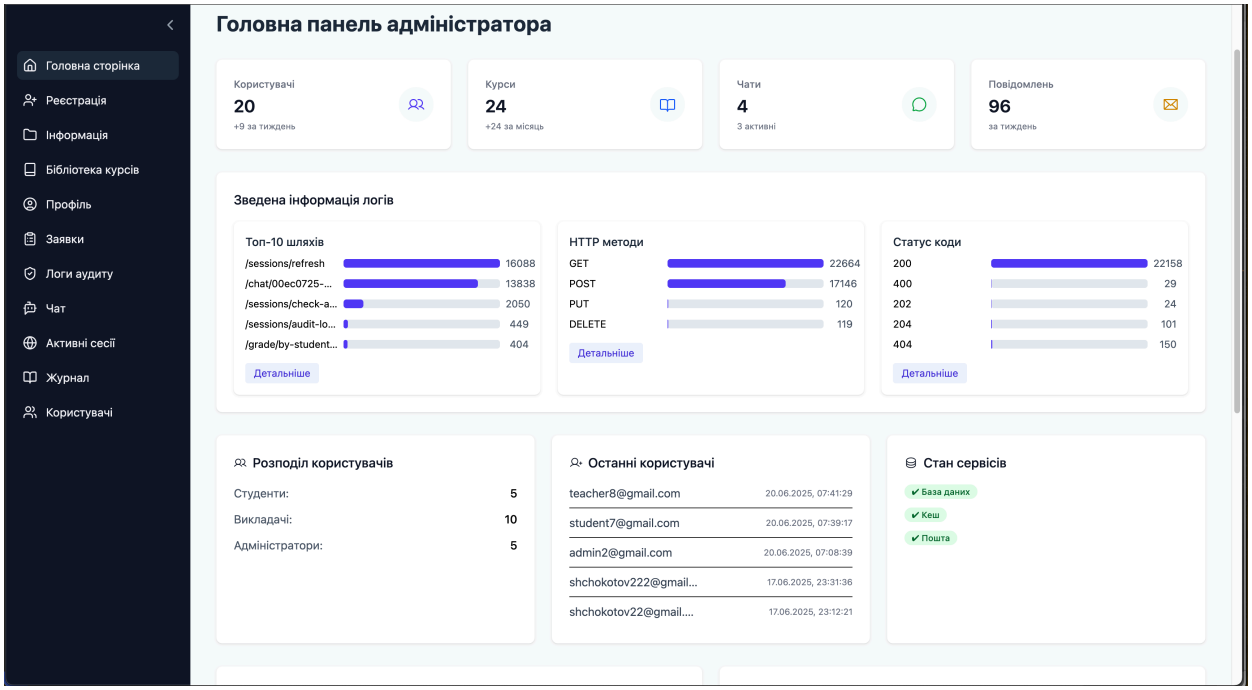


Рис. 3.7 – Головна сторінка адміністратора

На головній сторінці студент бачить перелік доступних курсів, стан проходження, важливі повідомлення та нагадування про дедлайни. Також відображається загальний прогрес, нещодавні завдання та сповіщення від викладачів. Завдяки структурованому вигляду студент швидко орієнтується у своїх поточних навчальних активностях. Присутні елементи швидкого доступу до навчальних матеріалів і персональних налаштувань. Інтерфейс підтримує адаптивність для різних розмірів екранів (рис. 3.8).

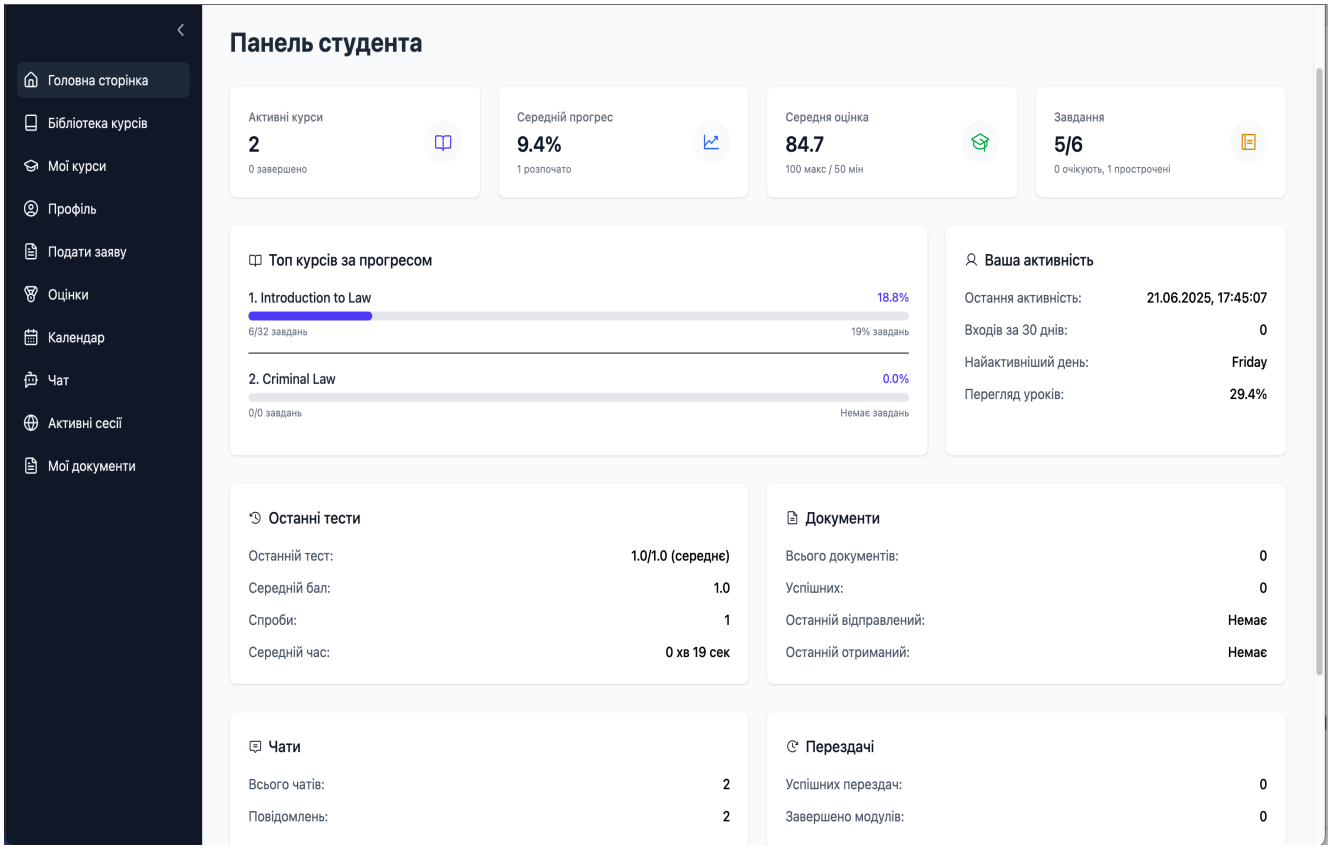


Рис. 3.8 – Головна сторінка студента

Викладач на головній сторінці отримує доступ до своїх курсів, журналу оцінювання, повідомлень і нових заявок студентів. Інтерфейс дозволяє швидко переходити до редагування матеріалів курсу, перегляду статистики відвідувань і проведення оцінювання. Всі ключові елементи згруповані для максимального комфорту роботи викладача. Додатково надається інформація про поточні групи та активність студентів. Такий підхід сприяє ефективній організації навчального процесу (рис. 3.9).

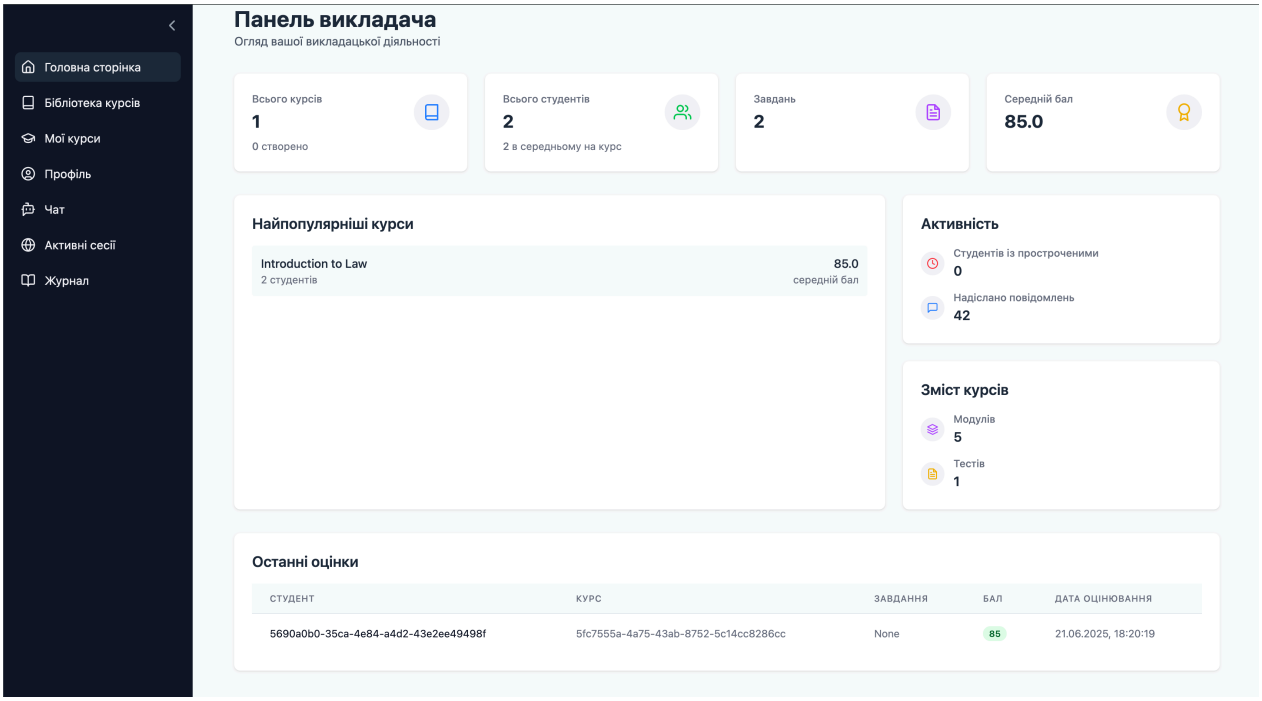


Рис. 3.9 – Головна сторінка викладача

Дана форма використовується для створення нових облікових записів адміністраторів. Необхідно заповнити основну інформацію, таку як ПІБ, email, роль та інші атрибути, визначені політикою безпеки. Після заповнення дані проходять валідацію, а новий акаунт додається до системи. Форма інтегрує підказки для запобігання помилок. Всі нові адміністратори отримують базовий доступ, який можна налаштувати пізніше (рис. 3.10).

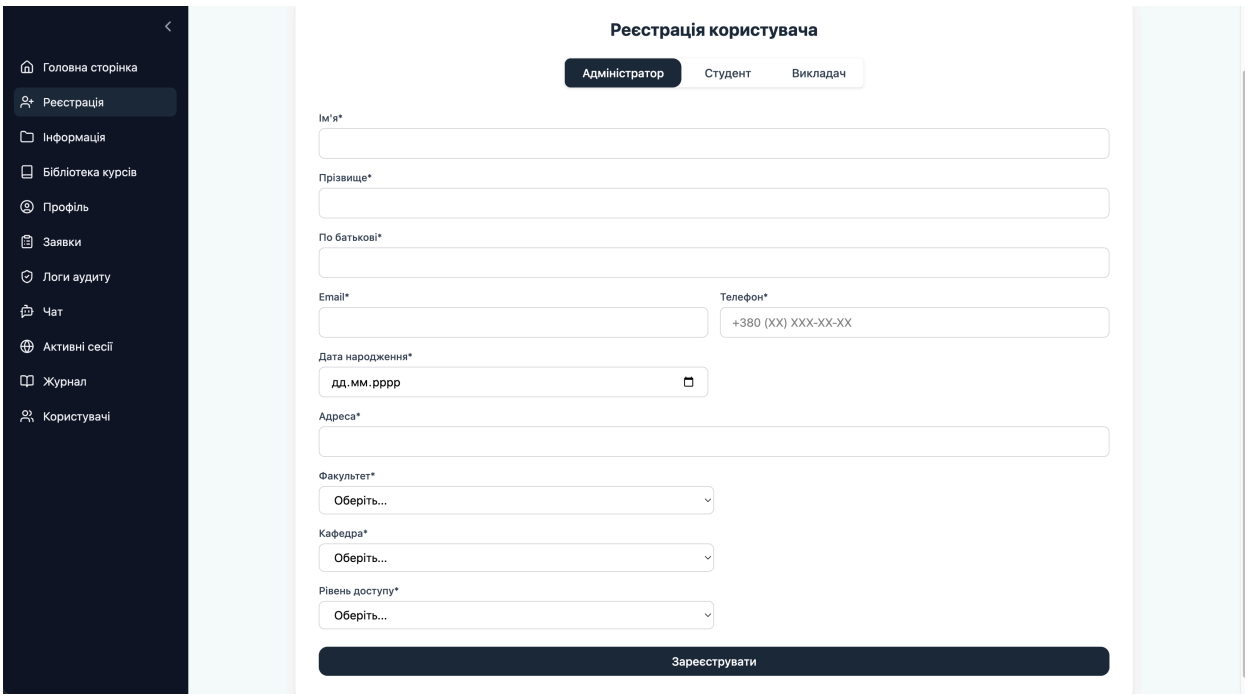


Рис. 3.10 – Форма реєстрації користувача(адміністратор)

Для реєстрації студента використовується окрема форма, яка враховує специфічні поля, такі як група, спеціальність, факультет, освітня програма та інші важливі характеристики. Кожне поле перевіряється на унікальність і валідність - наприклад, система не дозволяє зареєструвати студента з вже існуючою електронною поштою або студентським ідентифікатором. Після успішного введення і підтвердження даних студент автоматично додається до навчальної групи, отримує доступ до відповідних навчальних матеріалів, курсів і може одразу користуватися всіма можливостями системи.

Додатково, такий підхід до реєстрації дозволяє уникнути помилок і дублювання студентських облікових записів, а також забезпечує прозору структуру для подальшої аналітики й управління. Усі ці дії виконуються в єдиному інтерфейсі, що значно підвищує якість та швидкість організаційних процесів (рис. 3.11).

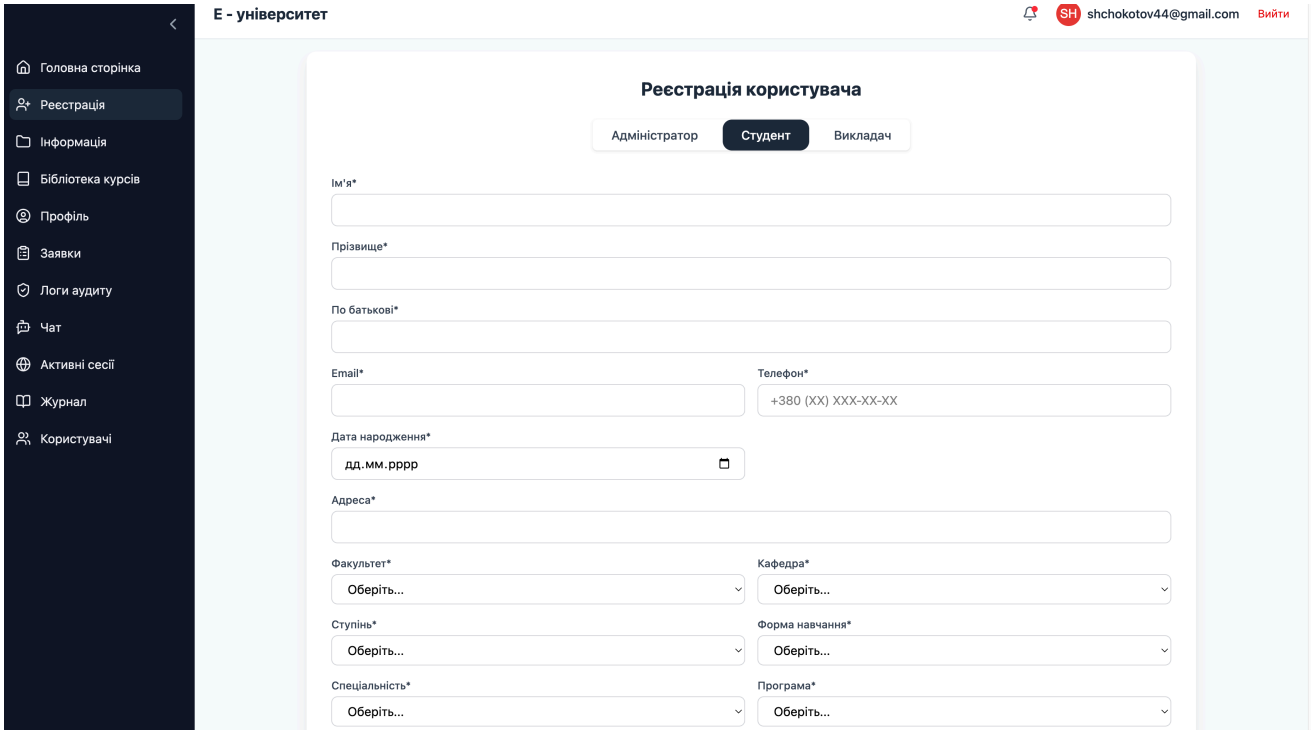


Рис. 3.11 – Форма реєстрації користувача(студент)

На цій сторінці студент може переглянути всі курси, доступні для запису або проходження. Вказується назва, короткий опис, кількість кредитів та статус проходження. Зручна система фільтрів допомагає швидко знайти потрібний курс. Натискання на курс відкриває його детальний опис та матеріали (рис. 3.12).

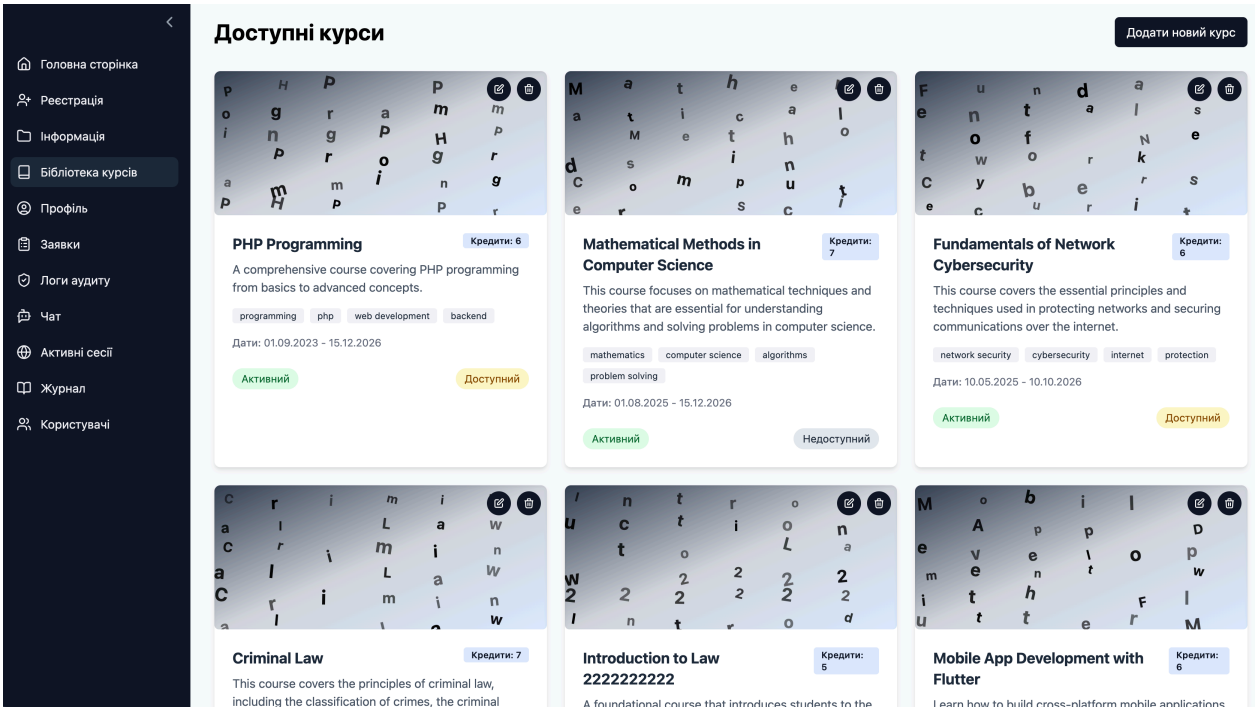


Рис. 3.12 – Сторінка з доступними курсами

Ця сторінка призначена для викладачів та адміністраторів, які можуть створювати нові курси. Необхідно вказати основні параметри: назву, опис, кількість кредитів, тривалість, та інші. Всі поля проходять перевірку на коректність. Після збереження курс стає доступним для призначення викладачів та запису студентів. Інтерфейс оптимізовано для швидкого введення даних та багатократного використання (рис. 3.13).

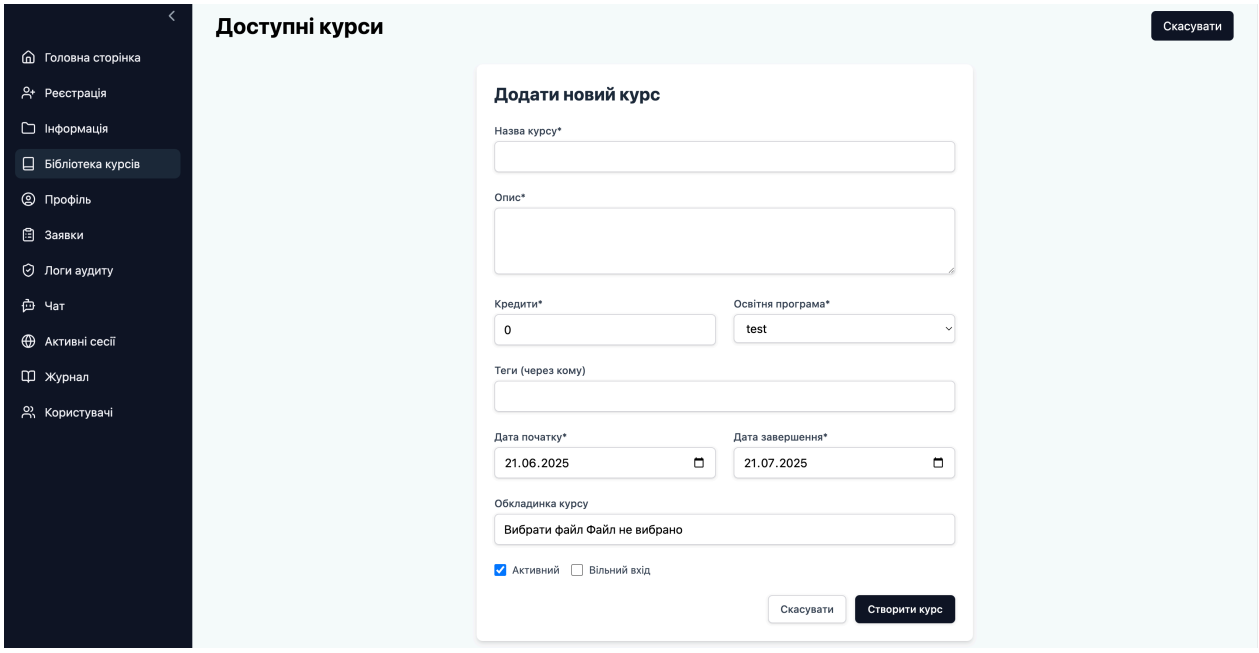


Рис. 3.13 – Сторінка з створенням курсу

На цій сторінці відображаються всі курси, у яких користувач бере участь – як студент, так і викладач. Інтерфейс дозволяє швидко перемикатися між ролями та переглядати актуальний статус кожного курсу. Можна переходити до детального перегляду матеріалів чи до списку учасників курсу. Завдяки цьому користувач завжди бачить повну картину свого навчального процесу. (рис. 3.14)

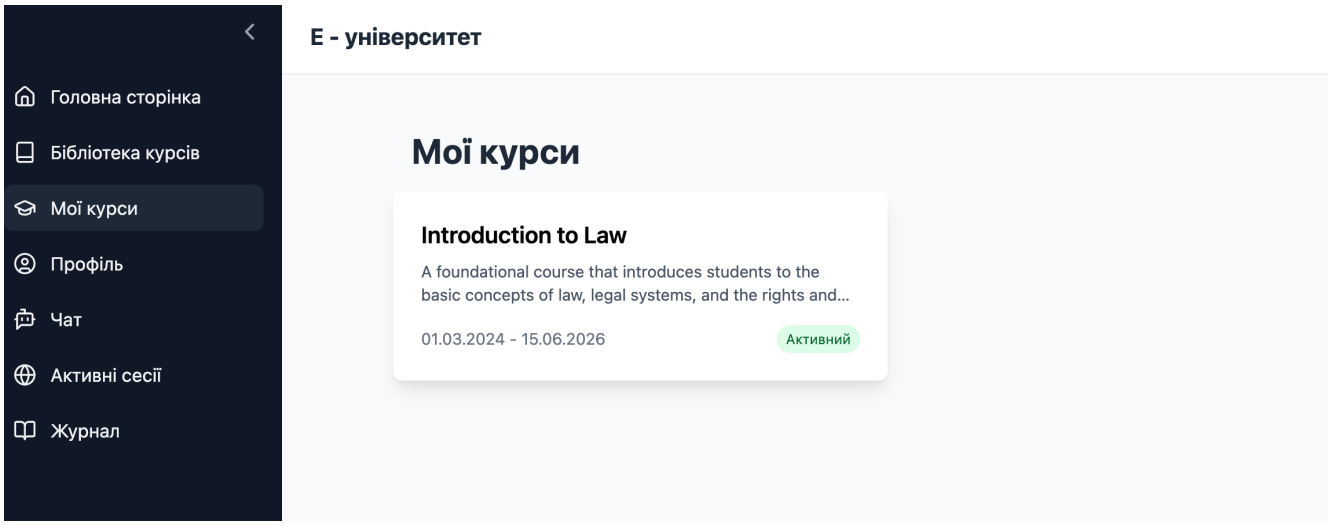


Рис. 3.14 – Сторінка з власними курсами викладача чи студента

Форма редагування дозволяє змінювати основну інформацію про курс, додавати або видаляти матеріали, призначати викладачів та студентів. Всі зміни зберігаються автоматично. Це дозволяє адаптувати курс під поточні потреби навчальної програми (рис. 3.15).

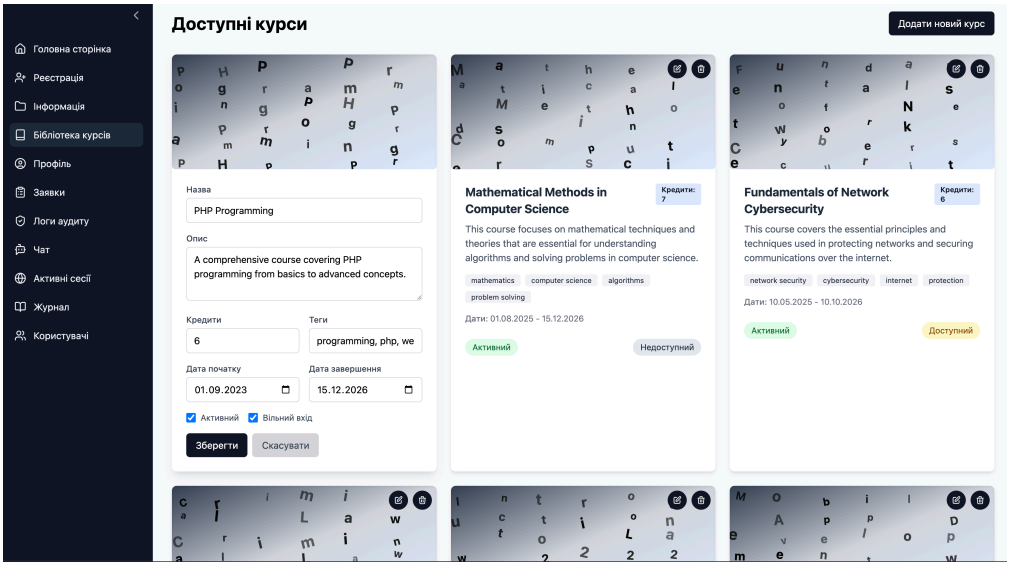


Рис. 3.15 – Редагування курсу

На сторінці курсу відображається структура модулів, які входять до навчального плану. Кожен модуль містить короткий опис, список занять, тести та завдання.

Студент може переходити до кожного модуля для ознайомлення з матеріалами та виконання завдань. Модулі розміщено у хронологічному або логічному порядку для зручності навігації. Всі зміни доступні в реальному часі (рис. 3.16).

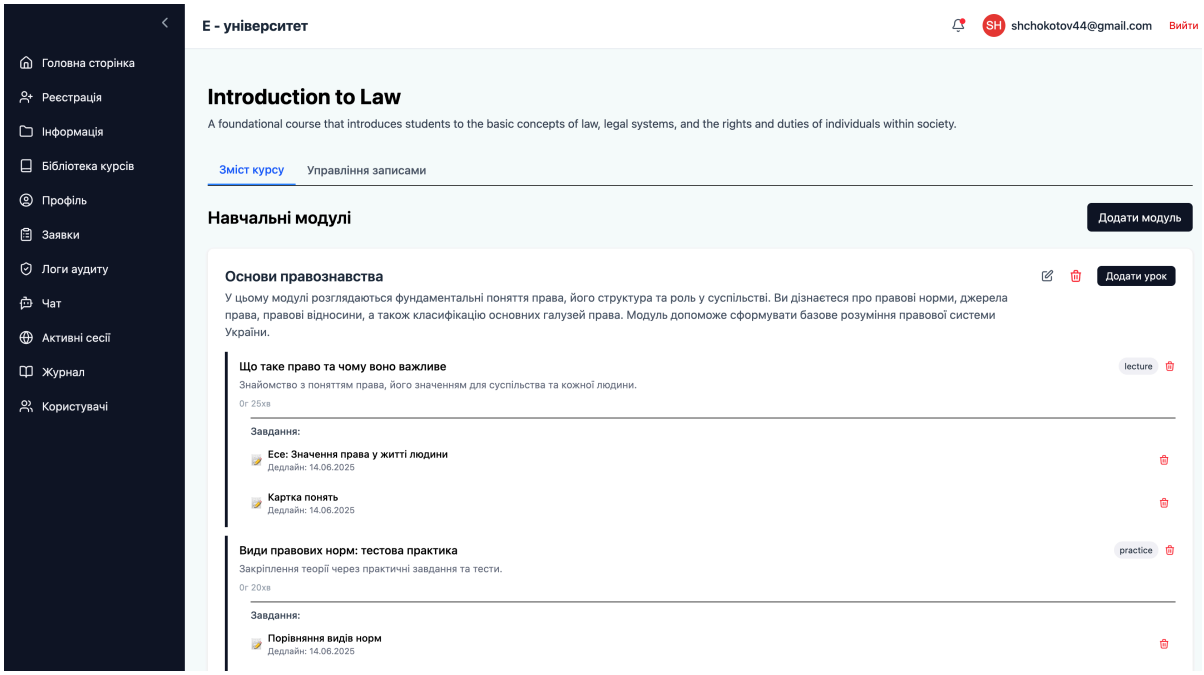


Рис. 3.16 – Сторінка курсу, відображення модулів

Ця сторінка дозволяє створювати нові модулі в курсі. Викладач чи адміністратор вводить назву модуля, опис та визначає порядок слідування. Можна одразу додати початкові заняття або тести для модуля. Після збереження модуль стає видимим для студентів. Це значно полегшує поетапне наповнення курсу контентом (рис. 3.17).

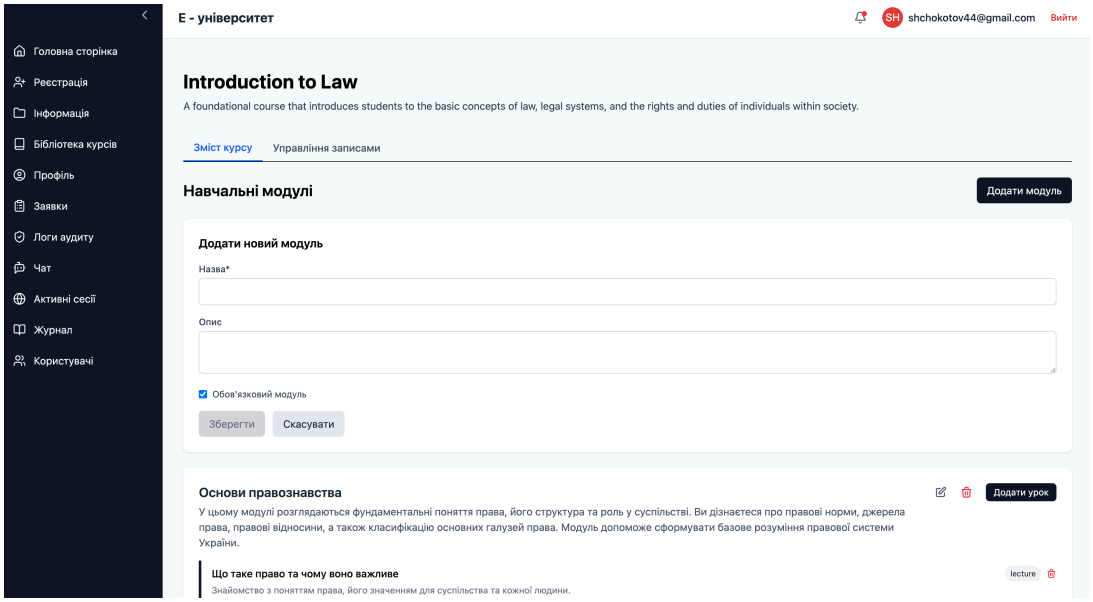


Рис. 3.17 – Створення модуля

Редагування модуля дозволяє змінити його опис, склад, додати або прибрати заняття, а також встановити умови доступу. Оновлення зберігаються одразу після підтвердження користувачем. Інтерфейс орієнтований на швидке виконання типових дій та має підказки для спрощення роботи. Завдяки цьому викладачі можуть швидко адаптувати програму під нові вимоги (рис. 3.18).

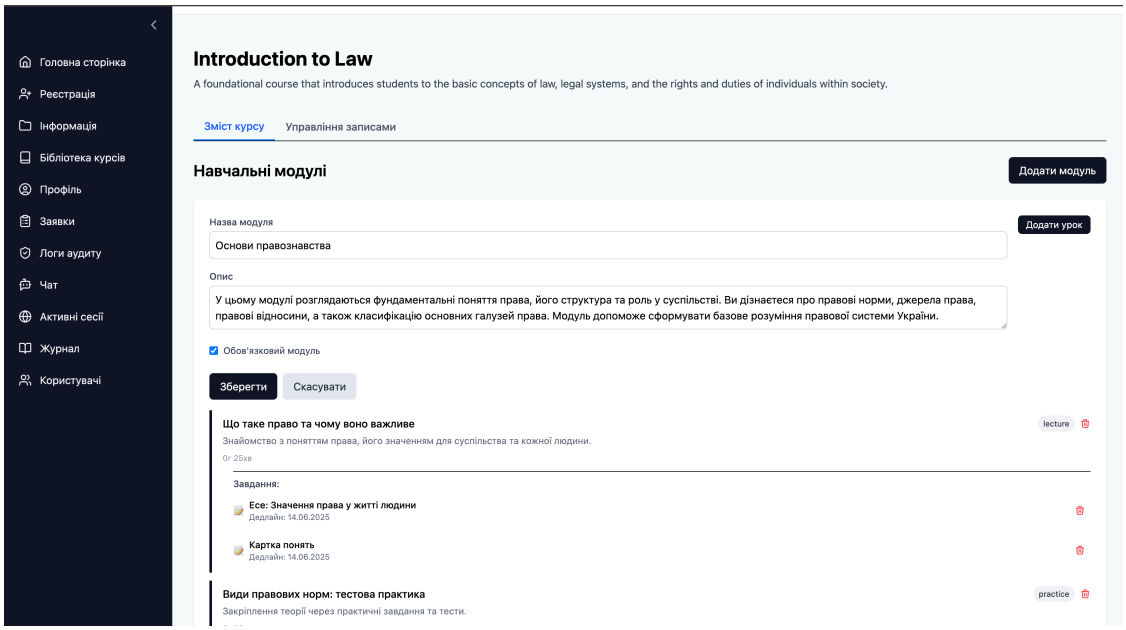


Рис. 3.18 – Редагування модуля

Сторінка чату містить перелік усіх доступних діалогів і групових розмов. Можна здійснювати пошук за іменем користувача чи темою чату. Всі діалоги відсортовані за датою останньої активності. Додано можливість створювати нові чати безпосередньо з цієї сторінки. Це полегшує спілкування та координацію між студентами і викладачами (рис. 3.19).

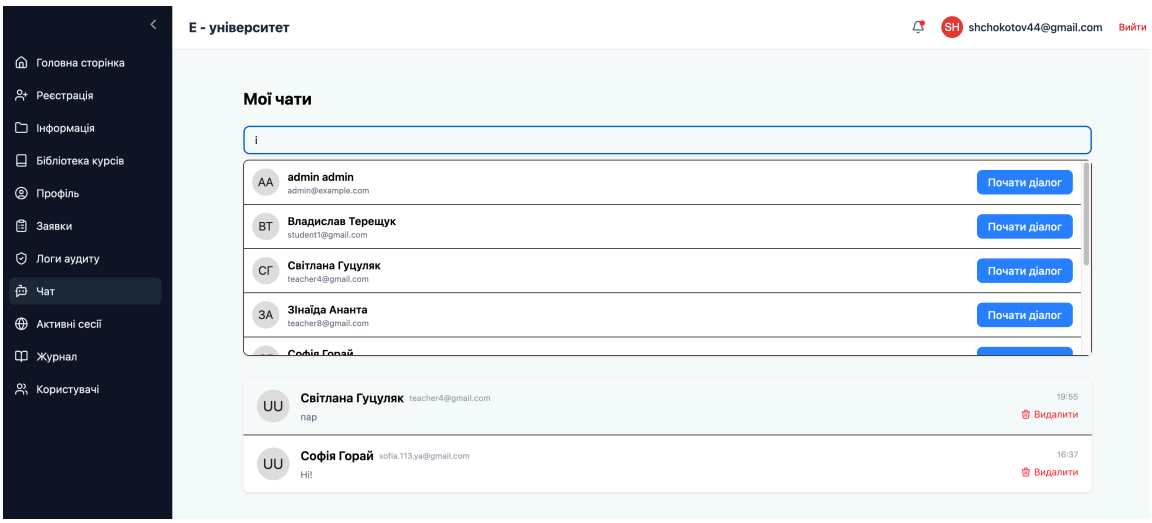


Рис. 3.19 – Сторінка чатів та пошуку

На сторінці чату відображаються повідомлення між учасниками розмови. Кожне повідомлення містить інформацію про час відправлення, автора та статус перегляду. Інтерфейс оптимізований для комфортного спілкування в навчальному середовищі (рис. 3.20).

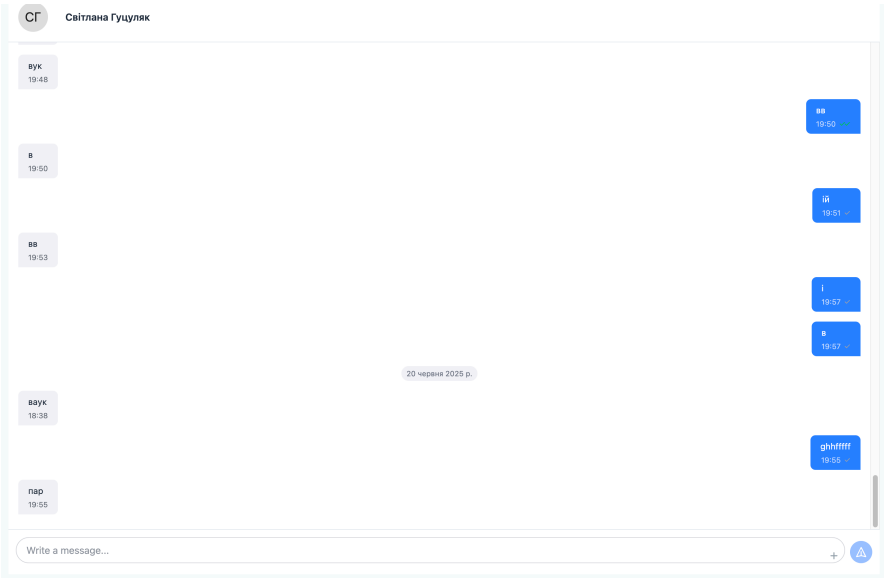


Рис. 3.20 – Сторінка чату спілкування

Цей екран ілюструє механізм реального часу: система показує, що співрозмовник зараз пише повідомлення. Використовується WebSocket для миттєвої синхронізації статусу “набирає”. Це підвищує інтерактивність і забезпечує природній досвід спілкування (рис. 3.21).

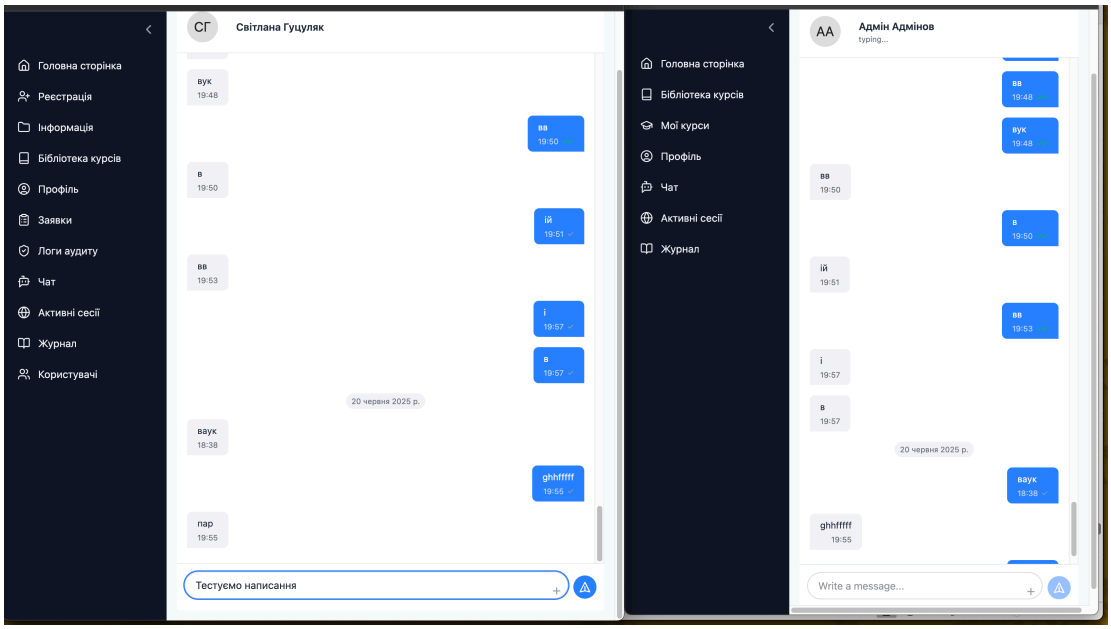


Рис. 3.21 – Приклад роботи веб сокетів на сповіщенні про написання

Тут демонструється обмін повідомленнями в чаті у режимі реального часу. Кожен користувач одразу бачить нові повідомлення без оновлення сторінки. Вебсокети забезпечують миттєву доставку інформації та підтвердження про прочитання. Це важливо для організації командної роботи та обговорення навчальних питань (рис. 3.22).

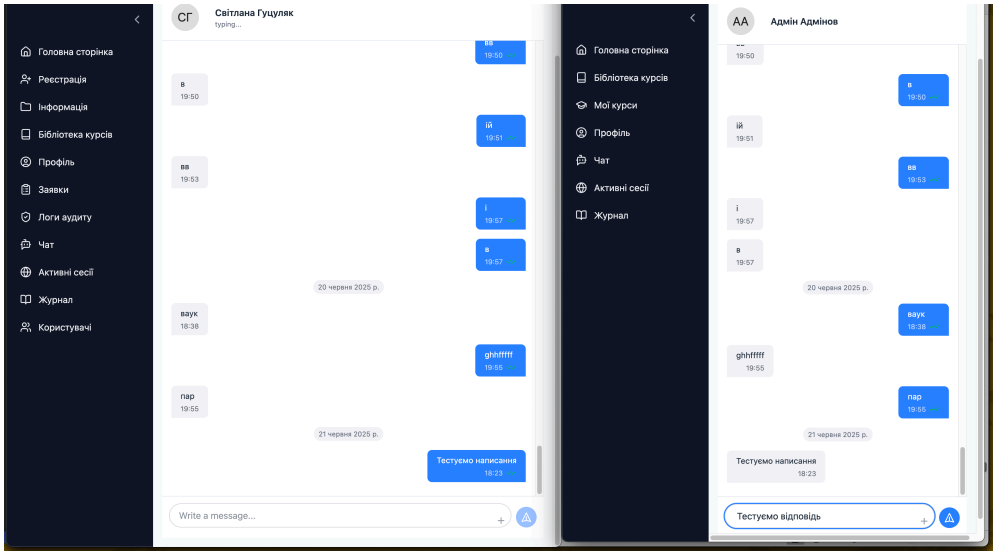


Рис. 3.22 – Приклад роботи веб сокетів на спілкування

На сторінці журналу аудиту відображаються всі ключові дії користувачів у системі. Є можливість фільтрувати логи за типом операції, часом або користувачем. Це дозволяє адміністраторам контролювати активність і швидко знаходити потенційні порушення. Додано функцію експорту логів у різних форматах. Система автоматично зберігає всю історію для подальшого аналізу (рис. 3.23).

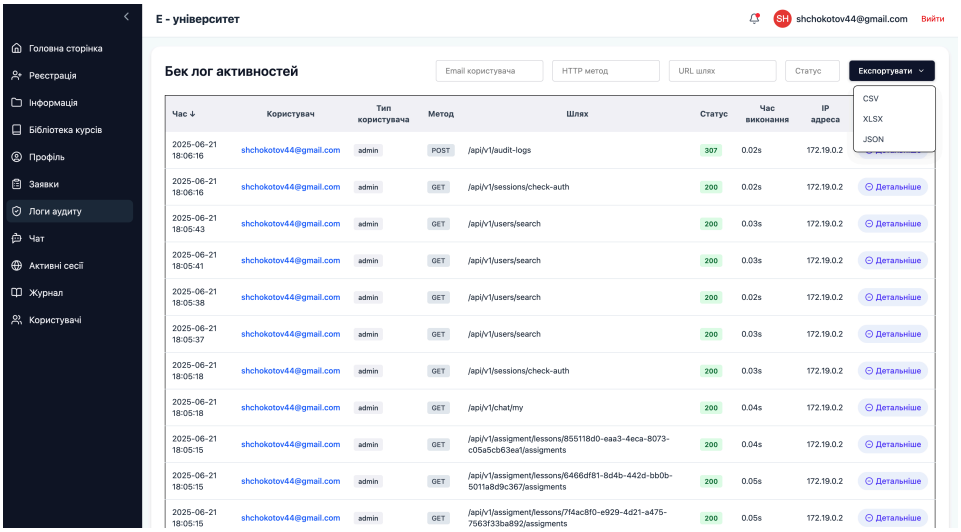


Рис. 3.23 – Сторінка з логами аудиту

Ця сторінка дозволяє переглянути розширену інформацію про конкретну дію з журналу аудиту. Відображаються всі параметри запиту, відповіді та додаткові технічні дані. Такий підхід допомагає оперативно розслідувати інциденти або відновлювати послідовність дій. Інтерфейс адаптовано для роботи з великими обсягами даних. Також реалізовано захист доступу до чутливої інформації (рис. 3.24).

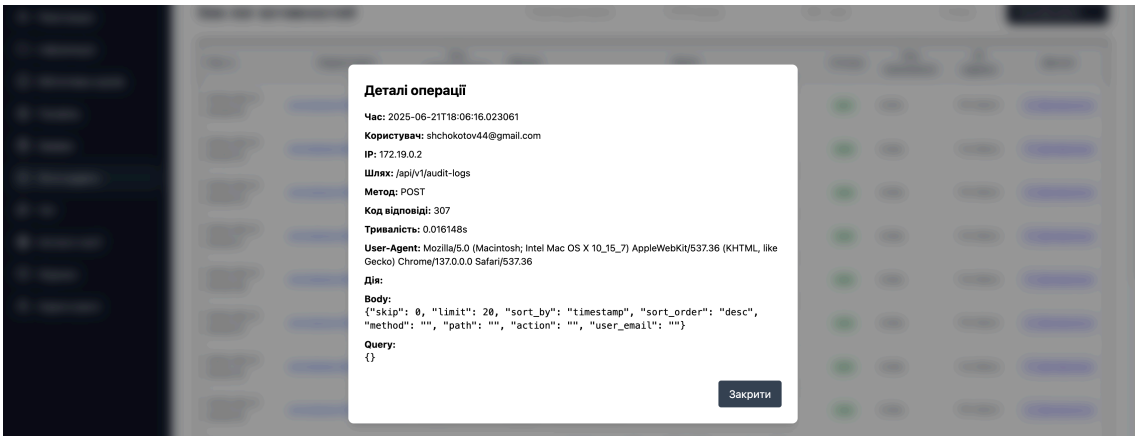


Рис. 3.24 – Сторінка з детальною інформацією операції

На цьому екрані видно журнал дій із підтримкою пагінації для зручної навігації. Користувачі можуть переходити між сторінками та швидко знаходити потрібні записи. Передбачена оптимізація відображення великих обсягів логів. Пошук і фільтрація залишаються доступними на всіх сторінках. Це спрощує аналіз подій навіть при великій кількості користувачів (рис. 3.25).

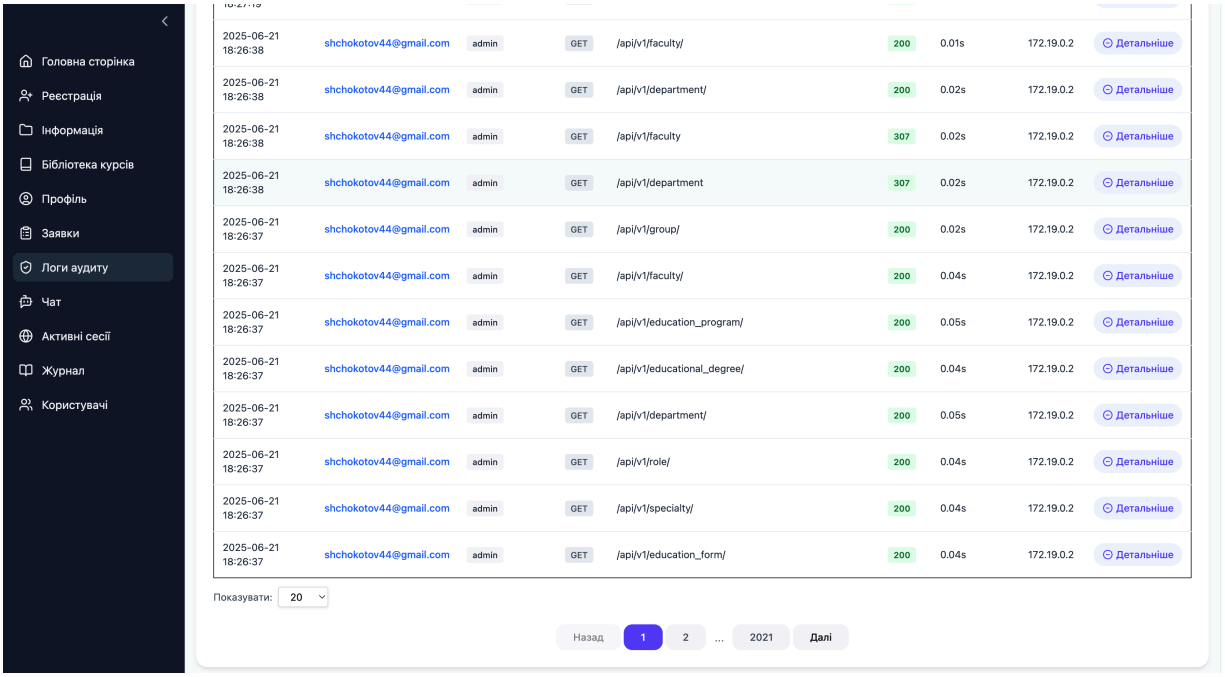


Рис. 3.25 – Сторінка з логами аудиту, пагінація

Тут відображаються всі поточні активні сесії користувача в системі. Кожна сесія містить інформацію про пристрій, IP-адресу та час останньої активності. Користувач може завершити одну або кілька сесій вручну. Це підвищує рівень безпеки облікового запису. Інтерфейс зрозумілий навіть для некваліфікованих користувачів (рис. 3.26).

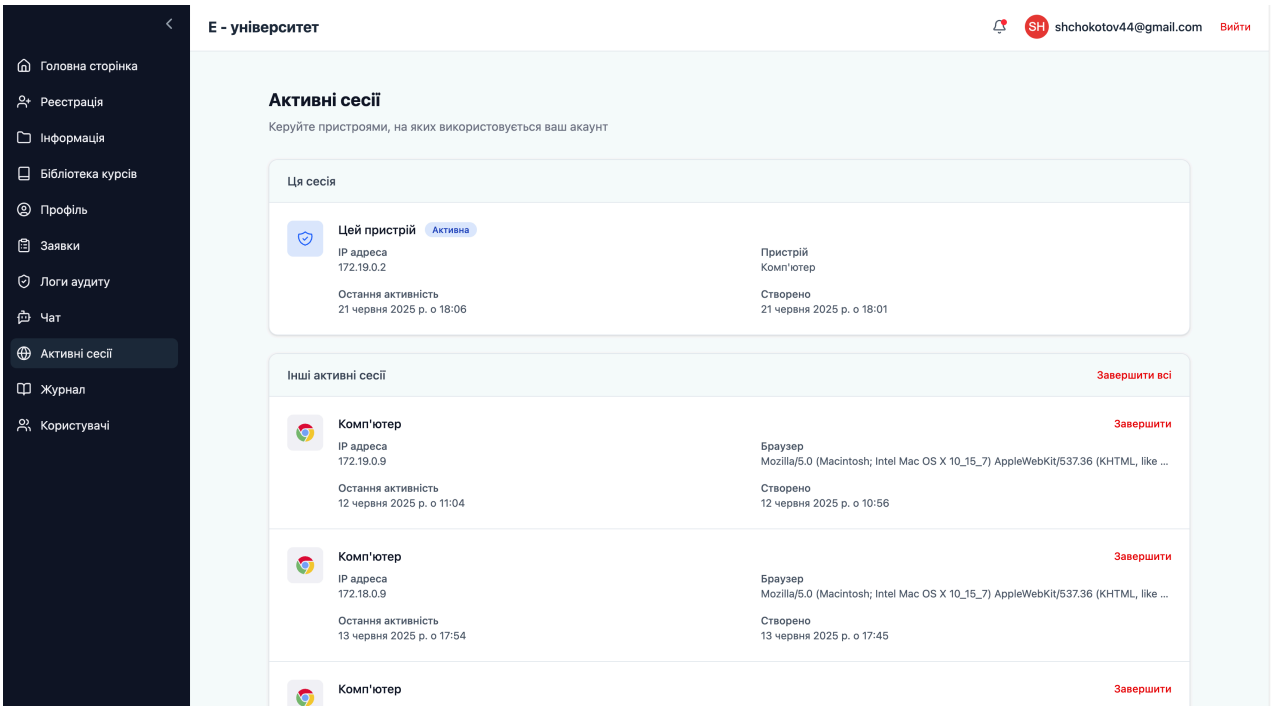


Рис. 3.26 – Сторінка з активними сесіями

На сторінці завершення сесії користувач підтверджує свій намір вийти з вибраної сесії. Система додатково запитує підтвердження для запобігання випадковим діям. Після завершення сесія зникає зі списку активних. Це дозволяє краще контролювати доступ до акаунту в різних пристроях. Функція корисна для забезпечення приватності та безпеки (рис. 3.27).

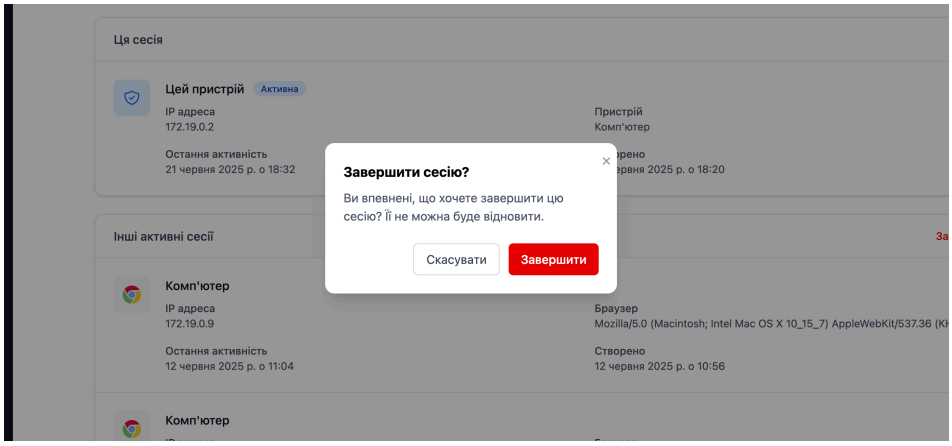


Рис. 3.27 – Завершення активної сесії

Далі у викладача є доступний журнал, в якому викладач може вести успішність студентів, сортованих за групою, попередньо обравши за курсом (рис. 3.28).

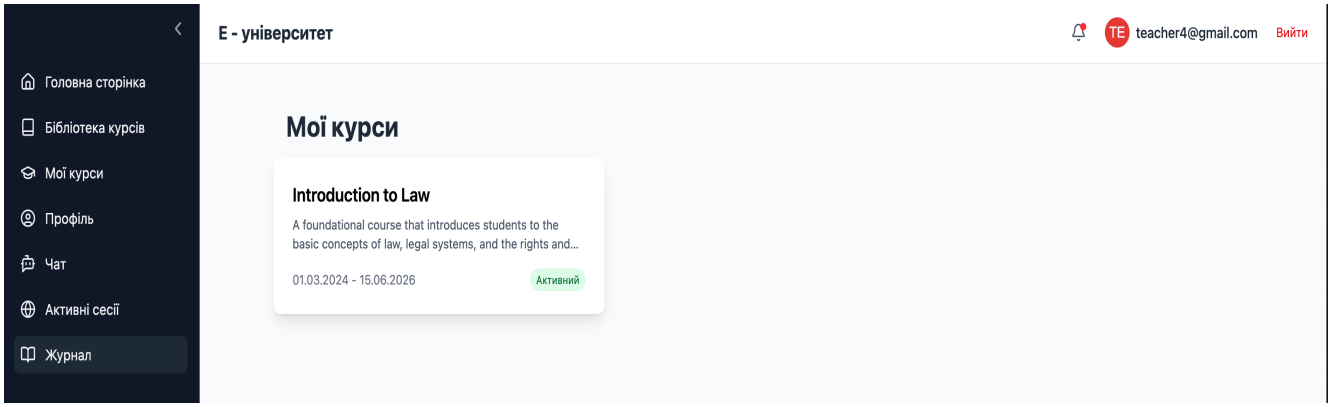


Рис. 3.28 – Вибір курсу в журналі викладача

Після вибору курсу у викладача є можливість обирати навчальну групу, в якій він бажає виставити оцінки (рис. 3.29).

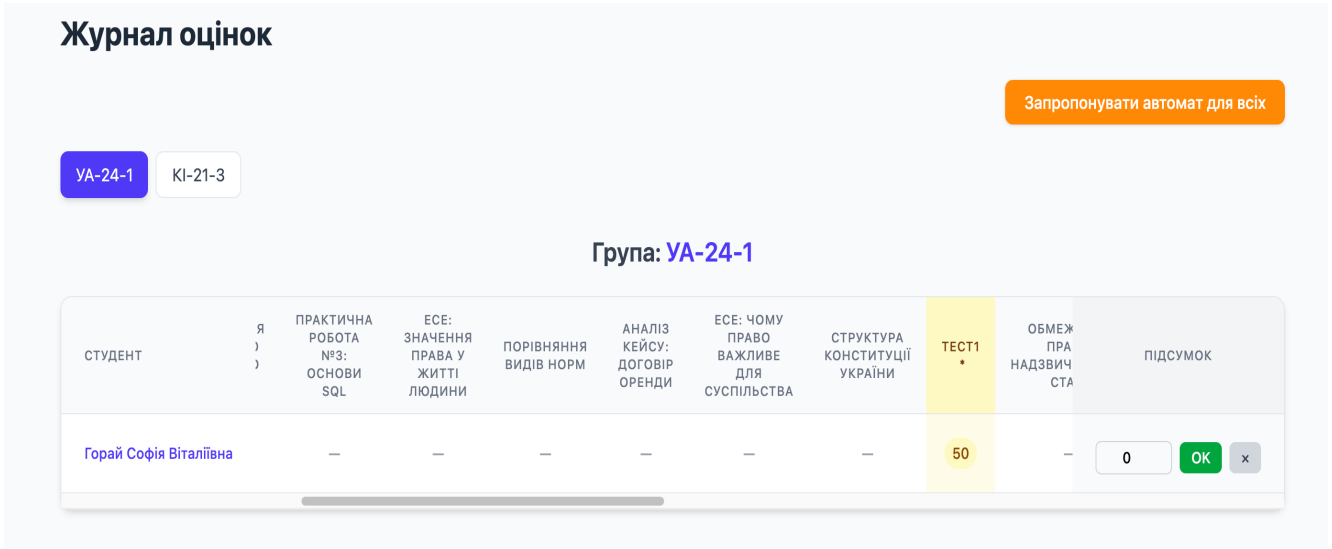


Рис. 3.29 – Сторінка журналу викладача

Журнал викладача містить функціонал по пропозиції виставлення підсумкового балу з урахуванням алгоритму вагових коефіцієнтів і обовязковості завдань. Також журнал місить інформацію про результати студентів, виконані завдання та загальний прогрес у курсах. Дозволяє переглядати історію оцінювання та відслідковувати активність кожного студента. Зручний фільтр і сортування полегшують пошук потрібних даних. Така функція є ключовою для контролю та аналізу навчального процесу (рис. 3.30).

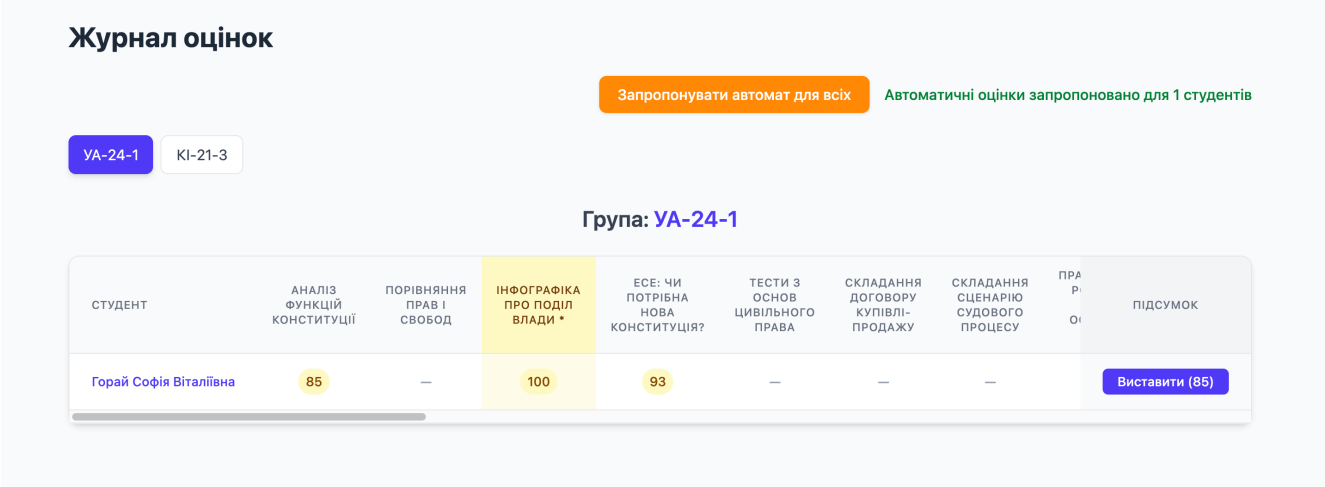


Рис. 3.30 – Приклад виконання виставлення оцінки

На цьому екрані показано, як викладач оцінює завдання студента. Відразу після введення балу й коментаря система оновлює результат у журналі. Операція підтверджується для виключення випадкових змін. Після виставлення оцінки студент отримує відповідне сповіщення. Це спрощує зворотній зв'язок і стимулює своєчасне виконання завдань. (рис. 3.31)

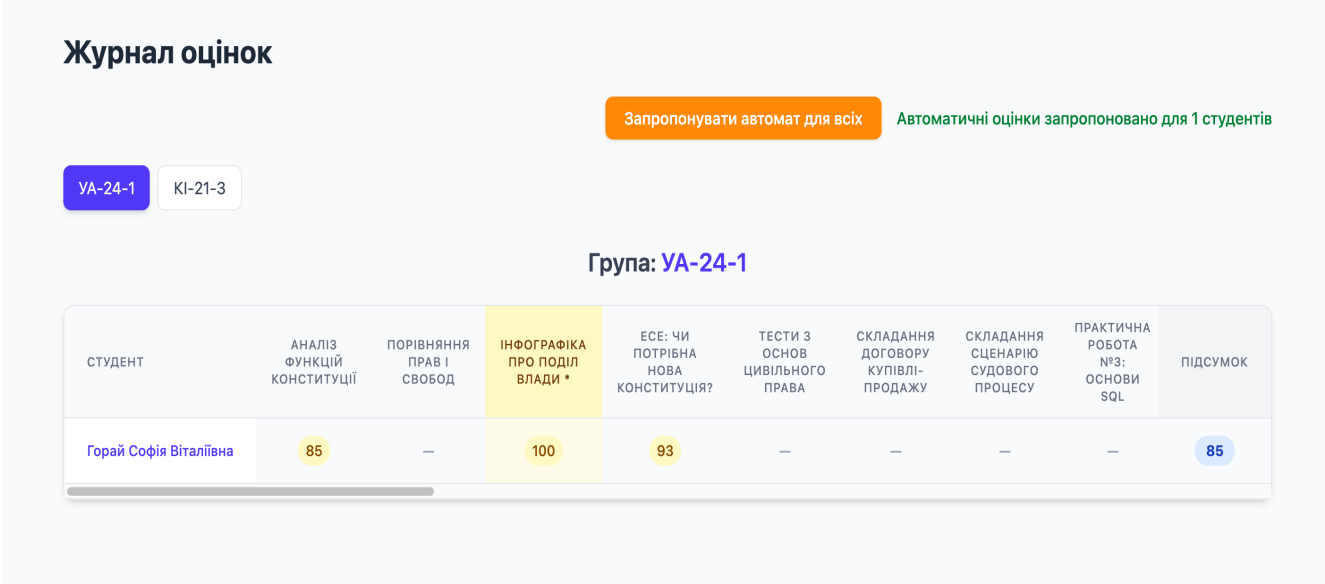


Рис. 3.31 – Результат виставлення оцінки

Журнал дозволяє швидко перейти до профілю будь-якого студента чи викладача. Це спрощує взаємодію між учасниками освітнього процесу. Таке рішення підвищує прозорість і персоналізує навчальний процес. Інтерфейс інтегрує додаткові можливості адміністрування користувачів (рис. 3.32).

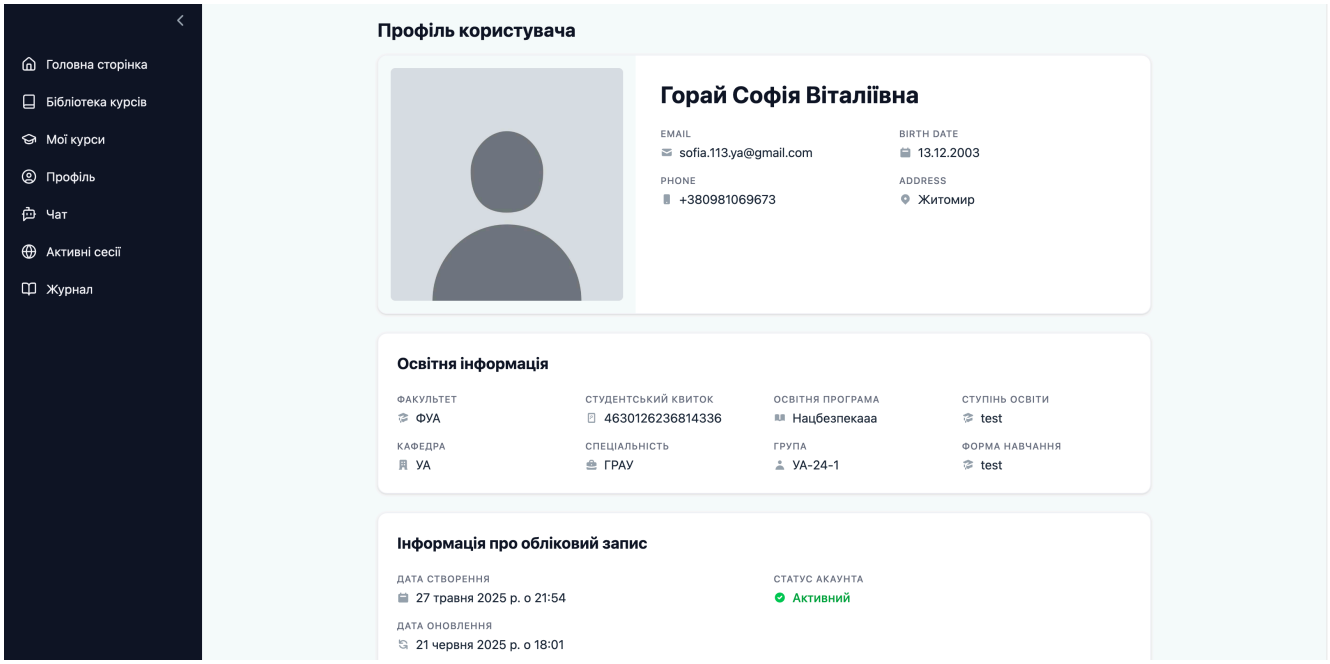


Рис. 3.32 – Можливість переходу до користувача з журналу

Сторінка користувачів містить список усіх активних облікових записів у системі з можливістю пошуку та фільтрації. Доступні функції редагування, видалення, зміни ролей і перегляду історії дій. Вся інформація відображається у зрозумілому та структурованому вигляді. Додано захист від випадкового видалення важливих акаунтів. Адміністратор може керувати доступами та налаштуваннями для кожного користувача (рис. 3.33).

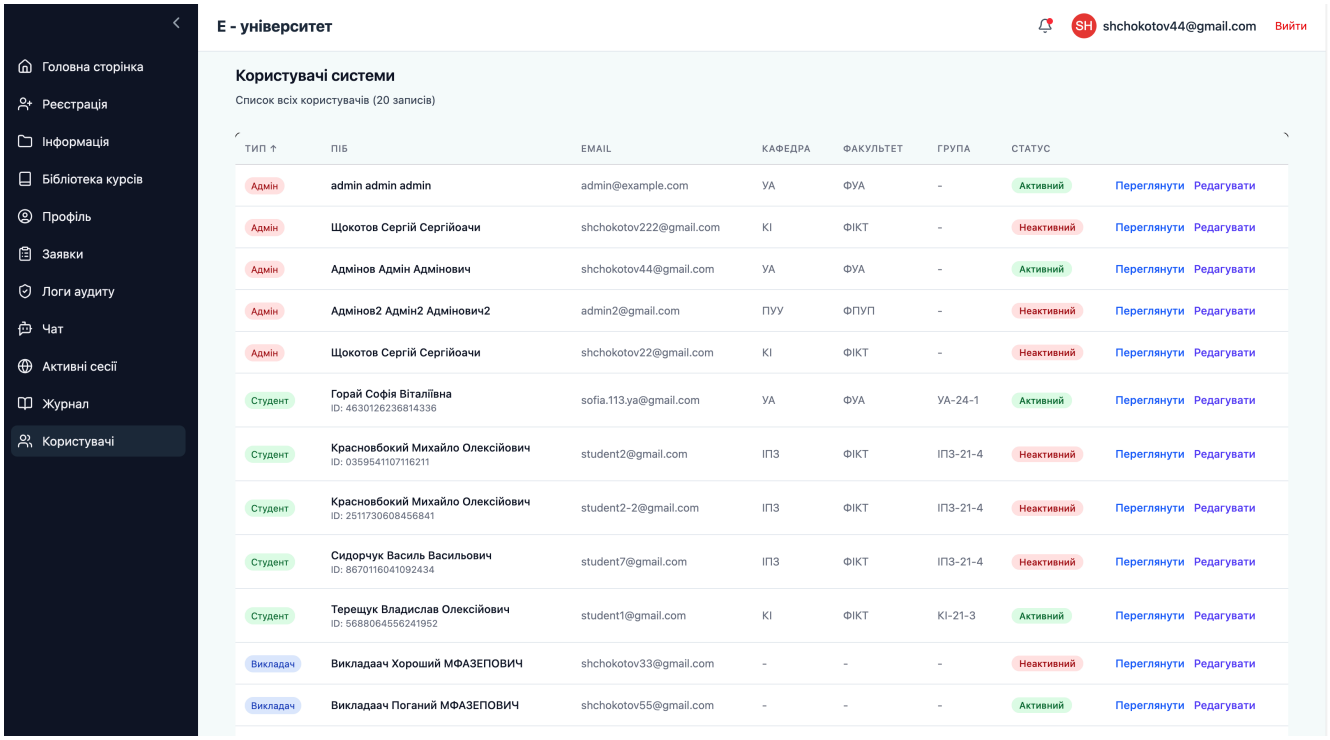


Рис. 3.33 – Сторінка користувачів

Цей екран дозволяє редагувати дані адміністратора – ПІБ, email, роль, права доступу тощо. Зміни зберігаються одразу після підтвердження, а про всі дії ведеться аудит. Форма оптимізована для швидкого внесення змін. Це підвищує ефективність адміністрування та підтримки системи (рис. 3.34).

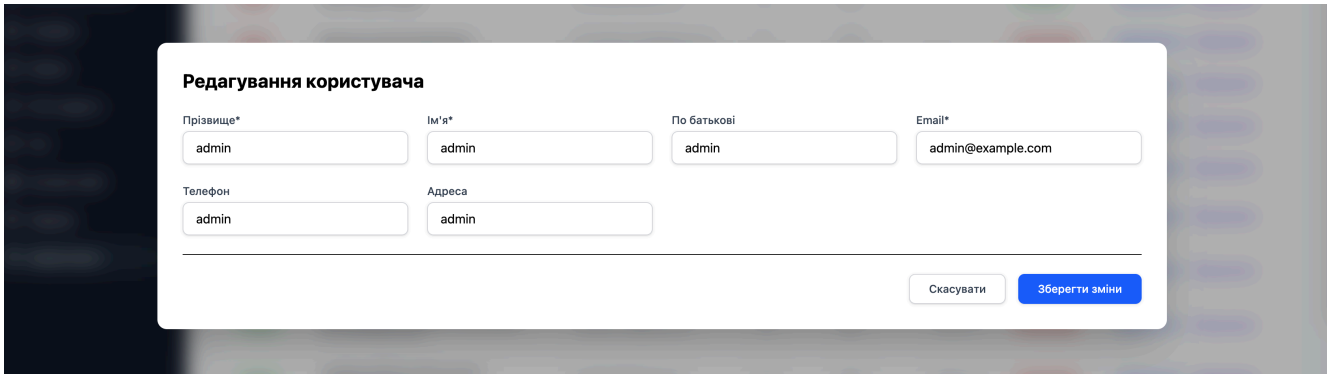


Рис. 3.34 – Можливість редагування адміністратора

На цьому екрані адміністратор або викладач може оновити дані студента: групу, спеціальність, email, контактну інформацію тощо. Система контролює унікальність і коректність змінюваних полів. Всі редагування фіксуються для прозорості та безпеки. Інтерфейс орієнтований на простоту та зручність роботи навіть при великій кількості студентів. Це полегшує підтримку актуальних даних у навчальному процесі (рис. 3.35).

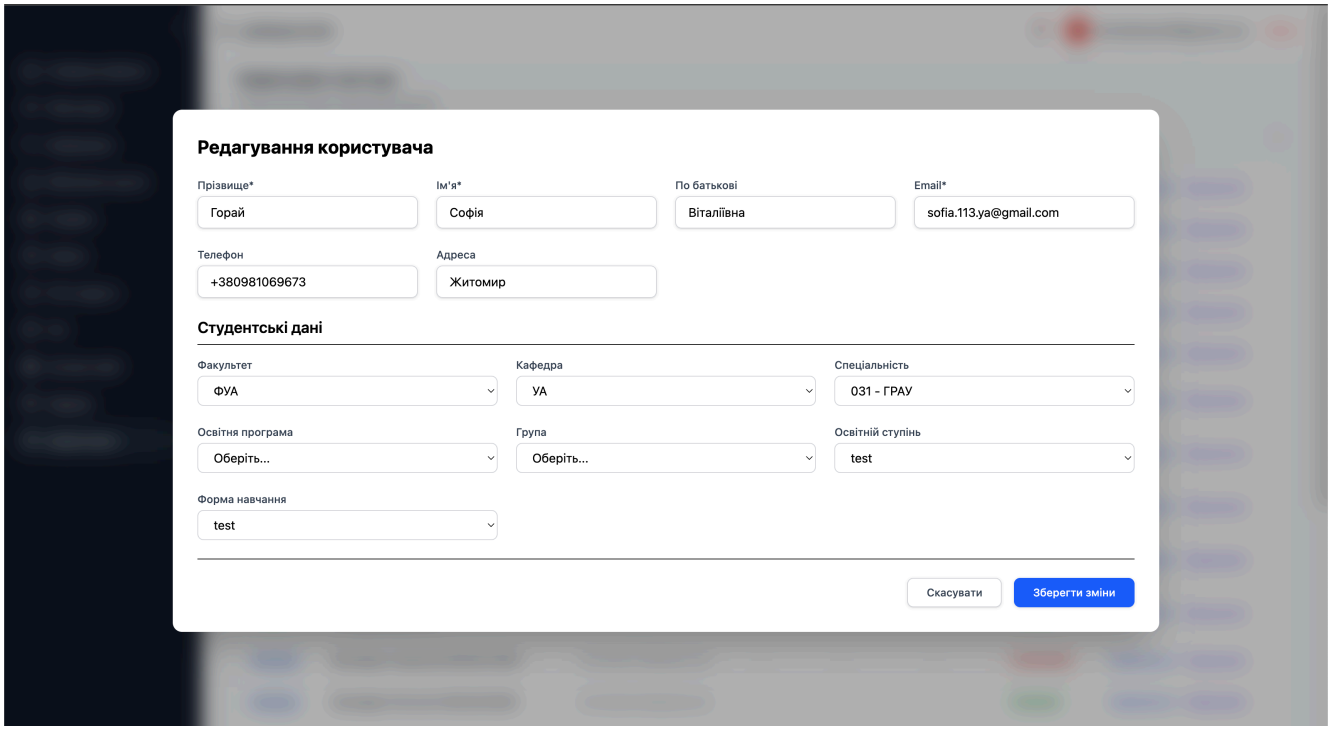


Рис. 3.35 – Можливість редагування студента

Запропонований інтерфейс системи забезпечує зручність роботи для всіх ролей, надаючи прозорий доступ до ключових функцій і матеріалів. Детальний опис та ілюстрації дозволяють чітко зрозуміти алгоритм взаємодії користувачів із системою, а також підтверджують її сучасність та функціональність.

3.3. Тестування роботи системи управління академічними ресурсами

Якість та надійність програмної системи визначається не лише її функціоналом, а й здатністю стабільно працювати у різних сценаріях використання. Для інформаційної системи управління академічними ресурсами особливе місце займає етап тестування, адже саме він дозволяє виявити та усунути можливі помилки до впровадження платформи у навчальному процесі.

У межах проєкту було застосовано комплексний підхід до тестування, що поєднує ручне, автоматизоване та інтеграційне тестування. Для перевірки роботи всіх API-ендпоінтів використовувався Swagger UI – вбудований інтерфейс інтерактивної документації, що автоматично генерується FastAPI. За допомогою Swagger усі функціональні можливості (реєстрація, авторизація, керування курсами, виставлення оцінок, робота з чатами тощо) були протестовані у різних ролях: студент, викладач, адміністратор (рис. 3.36).

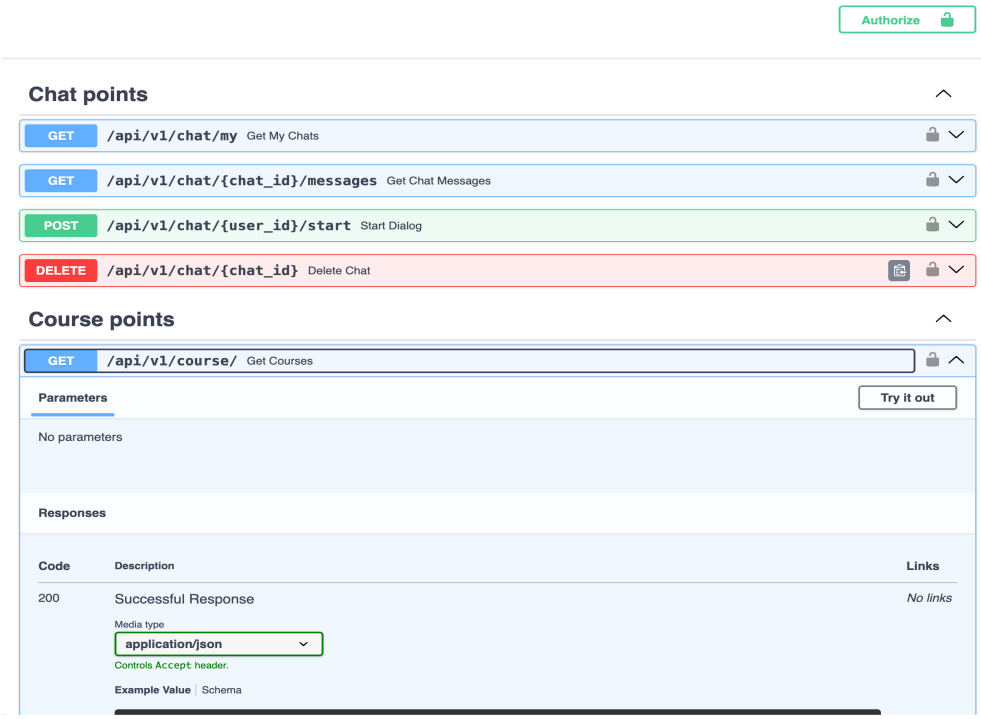


Рис. 3.36 – Демонстрація Swagger UI

Окрему увагу було приділено тестуванню безпеки: перевірялася робота з токенами, обмеження доступу до критичних API для різних ролей, стійкість до типових атак (SQL Injection, підбір паролів, спроби обходу аутентифікації). Для цього використовувалися як спеціалізовані тести, так і ручна перевірка сценаріїв.

У процесі тестування також застосовувалися такі підходи:

- Тестування “чорного ящика” (Black-box testing) – фокус на зовнішніх результатах без знання коду;
- Тестування “білого ящика” (White-box testing) – аналіз логіки та гілок виконання на рівні коду;
- Smoke Testing – базова перевірка працездатності основних функцій системи після внесення змін;
- Regression Testing – контроль коректності старих функцій після додавання нових можливостей.

Додатково для симуляції масових сценаріїв була проведена навантажувальна перевірка з допомогою інструментів Locust і парсинга на Python, що дозволило оцінити масштабованість і відгук системи під час пікових навантажень.

Завдяки використанню сучасних підходів та інструментів тестування, система продемонструвала стабільність, коректність виконання ключових бізнес-процесів і готовність до інтеграції у реальне навчальне середовище.

Висновки до третього розділу

У третьому розділі було детально розглянуто практичні аспекти впровадження, налаштування, використання та перевірки функціонування інформаційної системи управління академічними ресурсами студентів. На основі створеної архітектури було описано весь цикл роботи з системою: від первинного налаштування параметрів та організації доступу до адміністрування, до безпосередньої експлуатації користувачами різних ролей.

Особливу увагу приділено універсальності платформи, можливості розгортання у конкретному освітньому закладі, гнучкому налаштуванню структури факультетів, навчальних програм, ролей і політик безпеки. Важливим елементом є інтуїтивний і багатофункціональний вебінтерфейс, що забезпечує ефективну

взаємодію для адміністраторів, викладачів і студентів. Описано основні сценарії використання, логіку навігації та управління освітніми ресурсами.

Розкрито підходи до тестування працездатності системи, зокрема використання автоматизованих та інтерактивних інструментів перевірки (Swagger, юніт- та інтеграційні тести, ручне тестування ключових функцій). Це дозволило впевнитися у стабільності, безпеці й готовності системи до реального використання в умовах освітньої установи.

Таким чином, розроблена система не лише задовольняє сучасні вимоги до освітніх платформ, а й демонструє високу адаптивність, масштабованість та зручність для різних категорій користувачів. Практична реалізація та проведене тестування підтверджують готовність платформи до інтеграції у навчальні процеси й подальшого розвитку.

ВИСНОВКИ

У результаті виконання дипломної роботи було здійснено повний цикл розробки інформаційної системи управління академічними ресурсами студентів – від аналізу предметної області та формування вимог до впровадження, тестування й апробації в умовах сучасного навчального закладу.

На першому етапі було проведено глибокий аналіз існуючих аналогів, визначено основні завдання автоматизації освітнього процесу й особливості, характерні для університетського середовища. Це дозволило сформувати цілісне бачення майбутньої системи, окреслити пріоритети та визначити сучасну архітектуру на основі кращих світових практик. Аргументовано обрано технологічний стек, що забезпечує масштабованість, безпеку та швидку інтеграцію з іншими цифровими сервісами.

Окремо варто відмітити розподіл відповідальностей між двома розробниками та детальний огляд усіх потрібних ресурсів для досягнення максимальної синергії між розробниками. Описано принципи й цінності спільної роботи, організацію доступу до спільних ресурсів, контроль версій, вирішення конфліктів та досвід використання системи Jira для ефективного планування, розподілу задач і моніторингу прогресу. Це дозволило забезпечити високу якість коду, прозорість кожного етапу розробки та оперативне реагування на зміни вимог.

Другий розділ був присвячений проектуванню архітектури системи, специфікації функціональних і нефункціональних вимог, моделюванню бази даних, об'єктно-орієнтованій структурі та реалізації ключових алгоритмів. Особливу увагу приділено ролям користувачів, контролю доступу, логіці роботи з курсами, навчальними модулями, завданнями та механізмам оцінювання. Створена структура БД забезпечує надійне збереження, цілісність та швидкий доступ до всіх даних. Реалізовані алгоритми охоплюють як базові, так і складні сценарії взаємодії користувачів із системою, що дозволяє максимально автоматизувати освітні процеси та контроль успішності.

В кінці було деталізовано практичні аспекти впровадження системи: особливості початкового налаштування, алгоритм роботи вебінтерфейсу, доступність і

зручність для різних категорій користувачів. Проведено комплексне тестування з використанням автоматизованих інструментів, що підтвердило працездатність, надійність та безпеку створеного програмного комплексу.

У підсумку, розроблена система повністю відповідає сучасним вимогам до цифрових освітніх платформ: вона підтримує багаторівневу авторизацію, двофакторну автентифікацію, гнучкий розподіл ролей, журнал оцінювання, електронний документообіг, модулі інтеграції, сучасний вебінтерфейс і гнучке адміністрування. Комплексність, масштабованість і адаптивність системи дозволяють ефективно впроваджувати її у закладах вищої освіти незалежно від їх розміру й структури.

Результати роботи підтверджують доцільність впровадження розробленої системи для оптимізації освітнього процесу, підвищення прозорості оцінювання та взаємодії між студентами, викладачами й адміністрацією, а також створення єдиного інформаційного середовища для підтримки сучасної освіти.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Statistics on the use of digital technologies in Ukraine. URL: https://www.researchgate.net/figure/Statistics-on-the-use-of-digital-technologies-in-Ukraine_fig1_354823564 (дата звернення: 01.05.2025).
2. UNESCO’s global education coalition expanding online learning opportunities for Ukrainian learners and teachers. URL: https://www.unesco.org/en/articles/unescos-global-education-coalition-expanding-online-learning-opportunities-ukrainian-learners-and (дата звернення: 01.05.2025).
3. Pros and Cons of Moodle as an LMS platform. URL: https://courseorbit.com/blog/pros-and-cons-of-moodle-as-an-lms-platform/ (дата звернення: 01.05.2025).
4. International Review of Research in Open and Distributed Learning. URL: https://www.irrodl.org/index.php/irrodl/article/view/5769/5677 (дата звернення: 01.05.2025).
5. Jomswsge. URL: https://jomswsge.com (дата звернення: 02.05.2025).
6. Academy of Mine. URL: https://academyofmine.com (дата звернення: 02.05.2025).
7. Ser.net.ua. URL: https://ser.net.ua (дата звернення: 02.05.2025).
8. Іваненко В. Методологія вибору LMS в університеті // Журнал освітніх технологій. 2023.
9. Foreman S. D. The LMS Guidebook: Learning Management Systems Demystified. Routledge, 2017.
10. Офіційна документація Moodle. URL: https://moodle.org (дата звернення: 12.05.2025).
11. Google for Education. URL: https://classroom.google.com (дата звернення: 12.05.2025).
12. Instructure. Canvas LMS Overview. URL: https://instructure.com (дата звернення: 12.05.2025).
13. IT-Enterprise. Smart-School LMS Overview. 2024.
14. Міністерство освіти і науки України. Платформа «Єдина школа». URL: https://mon.gov.ua (дата звернення: 12.05.2025).

15. HatchWorks AI. URL: https://hatchworks.ai (дата звернення: 12.05.2025).
16. AtlassianIBM - United States. URL: https://atlassian.com (дата звернення: 12.05.2025).
17. GeeksforGeeks. URL: https://geeksforgeeks.org (дата звернення: 14.05.2025).
18. Josuttis N. SOA in Practice: The Art of Distributed System Design. O'Reilly Media, 2007.
19. Newman S. Building Microservices: Designing Fine-Grained Systems. O'Reilly Media, 2015.
20. Atlassian. URL: https://atlassian.com (дата звернення: 14.05.2025).
21. Connect, protect, and build everywhere. URL: https://cloud.ibm.com (дата звернення: 14.05.2025).
22. Roberts M. Serverless Architectures on AWS: With examples using AWS Lambda. Manning Publications, 2018.
23. Coulouris G., Dollimore J., Kindberg T., Blair G. Distributed Systems: Concepts and Design (5th Edition). Addison-Wesley, 2011.
24. Stevens W. R. UNIX Network Programming, Volume 1: The Sockets Networking API. Addison-Wesley, 2003.
25. Orfali R., Harkey D., Edwards J. Client/Server Survival Guide. Wiley, 2003.
26. Coulouris G., Dollimore J., Kindberg T., Blair G. Distributed Systems: Concepts and Design (5th Edition). Addison-Wesley, 2011.
27. Josuttis N. SOA in Practice: The Art of Distributed System Design. O'Reilly Media, 2007.
28. Fowler M. Patterns of Enterprise Application Architecture. Addison-Wesley, 2002. URL: https://martinfowler.com/eaaCatalog/ (дата звернення: 19.05.2025).
29. Official Microsoft Docs. Separation of concerns in .NET applications. URL: https://learn.microsoft.com/en-us/dotnet/architecture/modern-web-apps-azure/architectural-principles#separation-of-concerns (дата звернення: 19.05.2025).
30. Fielding R. Architectural Styles and the Design of Network-based Software Architectures. URL:

https://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm (дата звернення: 19.05.2025).
31. Mozilla Developer Network. Single Page Applications (SPA). URL: https://developer.mozilla.org/en-US/docs/Glossary/SPA (дата звернення: 19.05.2025).
32. Official Redux Documentation. Redux Fundamentals. URL: https://redux.js.org/tutorials/fundamentals/part-1-overview (дата звернення: 19.05.2025).
33. Fielding R. Architectural Styles and the Design of Network-based Software Architectures (REST). URL: https://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm (дата звернення: 19.05.2025).
34. Fielding R. Architectural Styles and the Design of Network-based Software Architectures (REST). URL: https://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm (дата звернення: 20.05.2025).
35. JWT.IO. Introduction to JSON Web Tokens. URL: https://jwt.io/introduction/ (дата звернення: 20.05.2025).
36. FastAPI Docs. WebSockets. URL: https://fastapi.tiangolo.com/advanced/websockets/ (дата звернення: 20.05.2025).
37. Bass L., Clements P., Kazman R. Software Architecture in Practice (4th Edition). Addison-Wesley, 2021.
38. Martin R. C. Clean Architecture: A Craftsman's Guide to Software Structure and Design. Prentice Hall, 2017.
39. Van Rossum G., Drake F.L. The Python Language Reference Manual. Python Software Foundation, 2024. URL: https://docs.python.org/3/reference/ (дата звернення: 20.05.2025).
40. FastAPI. Official Documentation. URL: https://fastapi.tiangolo.com (дата звернення: 21.05.2025).

41. Sebastián Ramírez. FastAPI: The modern, fast (high-performance), web framework for building APIs with Python 3.7+. URL: https://fastapi.tiangolo.com (дата звернення: 22.05.2025).
42. PostgreSQL Global Development Group. PostgreSQL Documentation. URL: https://www.postgresql.org/docs/ (дата звернення: 22.05.2025).
43. FastAPI Documentation. Dependencies - Dependency Injection. URL: https://fastapi.tiangolo.com/tutorial/dependencies/ (дата звернення: 22.05.2025).
44. Martin R.C. Clean Architecture: A Craftsman's Guide to Software Structure and Design. Prentice Hall, 2017.
45. Docker Docs. Get Started with Docker. URL: https://docs.docker.com/get-started/ (дата звернення: 24.05.2025).
46. Kubernetes Documentation. Production-Grade Container Orchestration. URL: https://kubernetes.io/docs/concepts/overview/what-is-kubernetes/ (дата звернення: 24.05.2025).
47. GitHub Docs. About continuous integration. URL: https://docs.github.com/en/actions/automating-builds-and-tests/about-continuous-integration (дата звернення: 24.05.2025).
48. OWASP Cheat Sheet Series. Password Storage Cheat Sheet. URL: https://cheatsheetseries.owasp.org/cheatsheets/Password_Storage_Cheat_Sheet.html (дата звернення: 25.05.2025).
49. Celery Documentation. Distributed Task Queue. URL: https://docs.celeryq.dev/en/stable/index.html (дата звернення: 25.05.2025).
50. Michael Bayer. Essential SQLAlchemy: Mapping Python to Databases, 2nd Edition. O'Reilly Media, 2016.
51. Офіційна документація SQLAlchemy ORM. URL: https://docs.sqlalchemy.org (дата звернення: 25.05.2025).
52. Офіційна документація Alembic. URL: https://alembic.sqlalchemy.org (дата звернення: 25.05.2025).
53. Офіційна документація Redis. URL: https://redis.io (дата звернення: 25.05.2025).

54. Офіційна документація Celery. URL: https://docs.celeryq.dev (дата звернення: 25.05.2025).
55. Брукс Ф. Міфічний людино-місяць: есе про розробку програмних систем. К.: Літопис, 2021.
56. Офіційна документація Git. URL: https://git-scm.com/doc (дата звернення: 25.05.2025).
57. Atlassian Git Tutorials – Resolving merge conflicts. URL: https://www.atlassian.com/git/tutorials/using-branches/merge-conflicts (дата звернення: 01.06.2025).
58. Chacon S., Straub B. Pro Git. Apress, 2014.
59. Офіційний сайт Jira Software. URL: https://www.atlassian.com/software/jira (дата звернення: 01.06.2025).
60. Atlassian Agile Coach. URL: https://www.atlassian.com/agile (дата звернення: 02.06.2025).

ДОДАТКИ

Додаток А

Middleware по роботі з аудитом та оновлення останнього стану користувача

```
import json
from starlette.middleware.base import BaseHTTPMiddleware
from starlette.requests import Request
from starlette.responses import Response
from datetime import datetime
from app.models.audit_log.audit_log import AuditLog
from sqlalchemy.ext.asyncio import AsyncSession
from app import crud
from app.api import deps
from app.utilities.data_middleware import restore_request_body


SENSITIVE_FIELDS = {"password", "new_password", "confirm_password", "token"}


def sanitize_data(data: dict) -> dict:
    return {key: ("***" if key in SENSITIVE_FIELDS else value) for key, value in data.items()}


class AuditMiddleware(BaseHTTPMiddleware):
    async def dispatch(self, request: Request, call_next):
        db: AsyncSession = await deps.get_db().__anext__()

        skip_paths = ["/static", "/favicon.ico", "/check_user", "/audit-logs"]
        if any(request.url.path.startswith(p) for p in skip_paths):
            return await call_next(request)

        start_time = datetime.utcnow()

        body_bytes = await restore_request_body(request)

        try:
            body_data = json.loads(body_bytes.decode())
            safe_body = sanitize_data(body_data)
        except Exception:
            safe_body = None

        response: Response = await call_next(request)
        duration = (datetime.utcnow() - start_time).total_seconds()

        user = None
        session = None
        user_session = await deps.get_user_from_request(request, db)
        if user_session:
            user, session = user_session

        audit_log = AuditLog(
            user_id=user.id if user else None,
            session_id=session.id if session else None,
            method=request.method,
            path=request.url.path,
            status_code=response.status_code,
            ip_address=request.client.host,
```

```
            user_agent=request.headers.get("user-agent"),
            duration=duration,
            timestamp=start_time,
            query_params=dict(request.query_params),
            action=getattr(request.state, "audit_action", None),
            request_body=json.dumps(safe_body) if safe_body else None,
        )

        try:
            db.add(audit_log)
            await db.commit()
        except Exception as e:
            print(f"Audit log error: {e}")

        return response


from starlette.middleware.base import BaseHTTPMiddleware
from starlette.requests import Request
from sqlalchemy.ext.asyncio import AsyncSession
from app import crud
from app.api import deps
from fastapi import Request

class UpdateLastActiveMiddleware(BaseHTTPMiddleware):
    async def dispatch(self, request: Request, call_next):
        if request.url.path.startswith(("/docs", "/openapi.json", "/static")):
            return await call_next(request)
        try:
            db: AsyncSession = await deps.get_db().__anext__()
            user_session = await deps.get_user_from_request(request, db)
            if user_session:
                user, session = user_session
            if user:
                ip = request.client.host
                ua = request.headers.get("user-agent")

                await crud.session.update_last_active_for_user(user.id, ip, ua, db)
        except Exception as e:
            print(f"Error updating last active: {e}")

        response = await call_next(request)
        return response
```

Додаток Б

Сінгелтон клас ініціалізації бази даних та веб додатку

```
import logging
from typing import AsyncGenerator
from sqlalchemy.ext.asyncio import AsyncEngine, create_async_engine, AsyncSession
from sqlalchemy.orm import sessionmaker
from app.core.config import settings
from sqlalchemy.exc import SQLAlchemyError
from loguru import logger
from sqlalchemy import text

logging.basicConfig()
logging.getLogger('sqlalchemy.engine').setLevel(logging.INFO)

class _PostgresClientSingleton:
    _engine: AsyncEngine | None = None
    _session_factory: sessionmaker | None = None

    def __new__(cls):
        if not hasattr(cls, "instance"):
            cls.instance = super(_PostgresClientSingleton, cls).__new__(cls)
            cls._initialize()
        return cls.instance

    @classmethod
    def _initialize(cls):
        logger.info("Initializing database connection...")
        try:
            if cls._engine is None:
                logger.info(f"Creating async engine...{settings.SQLALCHEMY_DATABASE_URI}")
                cls._engine = create_async_engine(
                    settings.SQLALCHEMY_DATABASE_URI,
                    pool_pre_ping=True,
                )
                logger.info(f"Engine created: {cls._engine}")
            if cls._session_factory is None:
                logger.info("Creating session factory...")
                cls._session_factory = sessionmaker(
```

```
                bind=cls._engine,
                class_=AsyncSession,
                expire_on_commit=False,
            )
            logger.info(f"Session factory created: {cls._session_factory}")
        except SQLAlchemyError as e:
            logger.exception("Failed to initialize database connection")

    @property
    def engine(self) -> AsyncEngine:
        if self._engine is None:
            raise RuntimeError("Engine is not initialized.")
        return self._engine

    @property
    def session_factory(self) -> sessionmaker:
        if self._session_factory is None:
            raise RuntimeError("Session factory is not initialized.")
        return self._session_factory



async def get_db_session() -> AsyncGenerator[AsyncSession, None]:
    session = _PostgresClientSingleton().session_factory()
    try:
        yield session
    finally:
        await session.close()



async def ping():
    try:
        async with _PostgresClientSingleton().engine.connect() as connection:
            await connection.execute(text("SELECT 1"))
    except Exception as e:
        raise ConnectionError("Unable to connect to the database.") from e
```

```
__all__ = ["get_db_session", "ping"]


from fastapi import FastAPI
from starlette.middleware.cors import CORSMiddleware
from contextlib import asynccontextmanager

from app.api.api_v1.api import api_router
from app.core.config import settings
from app.middleware.update_last_active import UpdateLastActiveMiddleware
from app.middleware.audit_middleware import AuditMiddleware


@asynccontextmanager
async def app_init(app: FastAPI):
    app.include_router(api_router, prefix=settings.API_V1_STR)
    yield


app = FastAPI(
    title=settings.PROJECT_NAME,
    openapi_url=f"{settings.API_V1_STR}/openapi.json",
    lifespan=app_init,
    debug=settings.DEBUG
)

app.add_middleware(UpdateLastActiveMiddleware)
app.add_middleware(AuditMiddleware)
# Set all CORS enabled origins
if settings.BACKEND_CORS_ORIGINS:
    app.add_middleware(
        CORSMiddleware,
        # Trailing slash causes CORS failures from these supported domains
        allow_origins=[str(origin).strip("/") for origin in settings.BACKEND_CORS_ORIGINS], # noqa
        allow_credentials=True,
        allow_methods=["*"],
        allow_headers=["*"],
    )
```

Додаток В

Описані моделі баз даних

```
class User(Base):
    __tablename__ = "users"

    id = Column(UUID(as_uuid=True), primary_key=True,
                default=uuid4, index=True)

    created = Column(DateTime, default=datetime_now_sec, nullable=False)
    modified = Column(DateTime, default=datetime_now_sec,
                      nullable=False, onupdate=datetime_now_sec)
    is_active = Column(Boolean, default=False, nullable=False)
    is_superuser = Column(Boolean, default=False, nullable=False)
    email = Column(String, unique=True, index=True, nullable=False)
    name = Column(String, default="", nullable=False)
    surname = Column(String, default="", nullable=False)
    middle_name = Column(String, default="", nullable=False)
    mobile_phone = Column(String, default="", nullable=False)
    address = Column(String, default="", nullable=False)
    birth_date = Column(DateTime, default=None, nullable=False)
    photo_url = Column(String, default="", nullable=True)
    hashed_password = Column(Text, nullable=True)
    totp_enabled = Column(Boolean, default=False)

    user_type = Column(String, nullable=False)  # "student", "teacher", "admin"

    role_id = Column(UUID(as_uuid=True), ForeignKey(
        "roles.id"), nullable=False)

    # courses_link = relationship(
    #     "UserCourse", back_populates="user", cascade="all, delete")
    role = relationship("Role", back_populates="users")
    totp_secret = relationship(
        "TOTPSecret", uselist=False, back_populates="user", cascade="all, delete-orphan")
    sessions = relationship(
        "Session", back_populates="user", cascade="all, delete-orphan")
    audit_logs = relationship("AuditLog", back_populates="user", cascade="all, delete-orphan")
```

```
    def __repr__(self):
        return f"<User(id={self.id}, email={self.email})>"

class Admin(User):
    __tablename__ = "admins"

    user_id = Column(UUID(as_uuid=True), ForeignKey(
        "users.id"), primary_key=True)
    admin_id = Column(String, unique=True, nullable=False, default="admin")

    faculty_id = Column(UUID(as_uuid=True), ForeignKey(
        "faculties.id", ondelete="CASCADE"), nullable=True)
    department_id = Column(UUID(as_uuid=True), ForeignKey(
        "departments.id", ondelete="CASCADE"), nullable=True)
    access_level = Column(Integer, default="standard", nullable=False)

    faculty = relationship("Faculty", backref="admins")
    department = relationship("Department", backref="admins")

    __mapper_args__ = {
        'polymorphic_identity': 'admin'
    }

    def __repr__(self):
        return f"<AdminInfo(id={self.id}, admin_id={self.user_id})>"

class Student(User):
    __tablename__ = "students"

    user_id = Column(UUID(as_uuid=True), ForeignKey(
        "users.id"), primary_key=True)
    student_id = Column(String, unique=True, nullable=False)
    faculty_id = Column(UUID(as_uuid=True), ForeignKey(
        "faculties.id", ondelete="CASCADE"), nullable=False)
    department_id = Column(UUID(as_uuid=True), ForeignKey(
        "departments.id", ondelete="CASCADE"), nullable=False)
    educational_degree_id = Column(
```

```
        UUID(as_uuid=True), ForeignKey("educational_degrees.id", ondelete="CASCADE"), nullable=False
    )
    education_form_id = Column(UUID(as_uuid=True), ForeignKey(
        "education_forms.id", ondelete="CASCADE"), nullable=False)
    specialty_id = Column(UUID(as_uuid=True), ForeignKey(
        "specialties.id", ondelete="CASCADE"), nullable=False)
    education_program_id = Column(
        UUID(as_uuid=True), ForeignKey("education_programs.id", ondelete="CASCADE"), nullable=False
    )
    group_id = Column(UUID(as_uuid=True), ForeignKey(
        "groups.id", ondelete="CASCADE"), nullable=False)

    faculty = relationship("Faculty")
    department = relationship("Department")
    educational_degree = relationship("EducationalDegree")
    education_form = relationship("EducationForm")
    specialty = relationship("Specialty")
    education_program = relationship("EducationProgram")
    group = relationship("Group")
    grades = relationship(
        "Grade", back_populates="student", foreign_keys="Grade.student_id"
    )
    student_courses = relationship(
        "StudentCourse", back_populates="student", cascade="all, delete-orphan")

    __mapper_args__ = {"polymorphic_identity": "student"}

    def __repr__(self):
        return f"<Student(id={self.id}, student_id={self.student_id})>"

class Teacher(User):
    __tablename__ = "teachers"

    user_id = Column(UUID(as_uuid=True), ForeignKey("users.id"), primary_key=True)
    teacher_id = Column(String, unique=True, nullable=False)
    department_id = Column(UUID(as_uuid=True), ForeignKey(
        "departments.id", ondelete="CASCADE"), nullable=False)
    academic_title = Column(String, nullable=True)
```

```
    academic_degree = Column(String, nullable=True)
    position = Column(String, nullable=True)

    department = relationship("Department", foreign_keys=[department_id])

    # grades = relationship("Grade", back_populates="teacher", foreign_keys="Grade.teacher_id")
    teacher_courses = relationship("TeacherCourse", back_populates="teacher", cascade="all, delete-orphan")



    __mapper_args__ = {
        'polymorphic_identity': 'teacher'
    }

    def __repr__(self):
        return f"<Teacher(id={self.id}, teacher_id={self.teacher_id})>"


class Course(Base):
    __tablename__ = "courses"

    id = Column(UUID(as_uuid=True), primary_key=True, default=uuid4)
    name = Column(String, nullable=False)
    description = Column(Text, nullable=True)
    credits = Column(Integer, nullable=False)  # Кредити ECTS
    is_active = Column(Boolean, default=True, nullable=False)
    cover_image_url = Column(String, nullable=True)
    start_date = Column(DateTime, nullable=True)  # Початок курсу
    end_date = Column(DateTime, nullable=True)  # Кінець курсу
    tags = Column(Text, nullable=True)  # JSON з тегами курсу
    is_free_entered = Column(Boolean, default=True, nullable=False)
    created_at = Column(DateTime, default=datetime_now_sec, nullable=False)
    updated_at = Column(DateTime, default=datetime_now_sec, onupdate=datetime_now_sec, nullable=False)
    light_remove = Column(Boolean, default=False, nullable=False)

    education_program_id = Column(UUID(as_uuid=True), ForeignKey("education_programs.id"),
nullable=False)

    education_program = relationship("EducationProgram", back_populates="courses")
```

```
    modules = relationship("CourseModule", back_populates="course", cascade="all, delete")
    student_courses = relationship("StudentCourse", back_populates="course", cascade="all, delete")
    teacher_courses = relationship("TeacherCourse", back_populates="course", cascade="all, delete")


class StudentCourse(Base):
    __tablename__ = "student_courses"

    id = Column(UUID(as_uuid=True), primary_key=True, default=uuid4)
    enrolled_at = Column(DateTime, default=datetime_now_sec, nullable=False)
    completed_at = Column(DateTime, nullable=True)
    # "enrolled", "completed", "dropped"
    status = Column(String, default="enrolled", nullable=False)
    current_module_id = Column(UUID(as_uuid=True), ForeignKey("course_modules.id"), nullable=True)
    # Відсоток завершення
    progress = Column(Integer, default=0, nullable=False)
    last_accessed = Column(DateTime, nullable=True)
    favorite = Column(Boolean, default=False, nullable=False)
    notifications_enabled = Column(Boolean, default=True, nullable=False)

    student_id = Column(UUID(as_uuid=True), ForeignKey("students.user_id"), nullable=False)
    course_id = Column(UUID(as_uuid=True), ForeignKey("courses.id"), nullable=False)

    student = relationship("Student", back_populates="student_courses")
    course = relationship("Course", back_populates="student_courses")


class TeacherCourse(Base):
    __tablename__ = "teacher_courses"

    id = Column(UUID(as_uuid=True), primary_key=True, default=uuid4)
    role = Column(String, nullable=True)
    # Чи активний викладач на курсі
    is_active = Column(Boolean, default=True, nullable=False)
    notes = Column(Text, nullable=True)  # Примітки про викладача на курсі
    is_creator = Column(Boolean, default=False, nullable=False)

    teacher_id = Column(UUID(as_uuid=True), ForeignKey("teachers.user_id"), nullable=False)
```

```
    course_id = Column(UUID(as_uuid=True), ForeignKey("courses.id"), nullable=False)

    teacher = relationship("Teacher", back_populates="teacher_courses")
    course = relationship("Course", back_populates="teacher_courses")


class CourseModule(Base):
    __tablename__ = "course_modules"

    id = Column(UUID(as_uuid=True), primary_key=True, default=uuid4)
    name = Column(String, nullable=False)
    description = Column(Text, nullable=True)
    order = Column(Integer, nullable=False)  # Порядок модулів у курсі
    is_required = Column(Boolean, default=True, nullable=False)  # Обов'язковий модуль
    light_remove = Column(Boolean, default=False, nullable=False)
    course_id = Column(UUID(as_uuid=True), ForeignKey("courses.id"), nullable=False)

    course = relationship("Course", back_populates="modules")
    lessons = relationship("Lesson", back_populates="module", cascade="all, delete")



class Chat(Base):
    __tablename__ = "chats"
    id = Column(UUID(as_uuid=True), primary_key=True, default=uuid4)
    type = Column(String, default="dialog")  # dialog, group
    title = Column(String, nullable=True)
    participants = relationship("ChatParticipant", back_populates="chat", cascade="all, delete")
    messages = relationship("Message", back_populates="chat", cascade="all, delete")
    created_at = Column(DateTime, default=datetime.utcnow, nullable=False)


class ChatParticipant(Base):
    __tablename__ = "chat_participants"
    id = Column(UUID(as_uuid=True), primary_key=True, default=uuid4)
    chat_id = Column(UUID(as_uuid=True), ForeignKey("chats.id"))
    user_id = Column(UUID(as_uuid=True), ForeignKey("users.id"))
    role = Column(String, default="member")  # owner, member
```

```
    chat = relationship("Chat", back_populates="participants")
    user = relationship("User")


class Message(Base):
    __tablename__ = "messages"
    id = Column(UUID(as_uuid=True), primary_key=True, default=uuid4)
    chat_id = Column(UUID(as_uuid=True), ForeignKey("chats.id"))
    sender_id = Column(UUID(as_uuid=True), ForeignKey("users.id"))
    text = Column(String)
    created_at = Column(DateTime, default=datetime.utcnow)
    is_read = Column(Boolean, default=False)
    status = Column(String, default="sent")  # sent, delivered, read
    chat = relationship("Chat", back_populates="messages")
    sender = relationship("User")



class Session(Base):
    __tablename__ = "sessions"

    id = Column(UUID(as_uuid=True), primary_key=True, default=uuid4, index=True)
    user_id = Column(UUID(as_uuid=True), ForeignKey("users.id", ondelete="CASCADE"), nullable=False)
    ip_address = Column(String, nullable=True)
    user_agent = Column(String, nullable=True)
    refresh_token_hash = Column(Text, nullable=False)
    created_at = Column(DateTime, default=datetime.utcnow)
    last_active = Column(DateTime, default=datetime.utcnow)
    expires_at = Column(DateTime, nullable=False)
    user = relationship("User", back_populates="sessions")
    audit_logs = relationship("AuditLog", back_populates="session")

    def is_expired(self) -> bool:
        return datetime.utcnow() > self.expires_at
class TOTPSecret(Base):
    __tablename__ = "totp_secrets"

    id = Column(UUID(as_uuid=True), primary_key=True, default=uuid4, index=True)
```

```
    user_id = Column(UUID(as_uuid=True), ForeignKey("users.id", ondelete="CASCADE"), nullable=False,
unique=True)
    secret = Column(Text, nullable=False)
    counter = Column(Integer, default=0, nullable=False)
    created_at = Column(DateTime, default=datetime_now_sec, nullable=False)

    user = relationship("User", back_populates="totp_secret")

User.totp_secret = relationship("TOTPSecret", uselist=False, back_populates="user", cascade="all, delete-
orphan")
class AuditLog(Base):
    __tablename__ = "audit_logs"
    id = Column(UUID(as_uuid=True), primary_key=True, default=uuid4, index=True)
    user_id = Column(UUID(as_uuid=True), ForeignKey("users.id"), nullable=True)
    session_id = Column(
        UUID(as_uuid=True),
        ForeignKey("sessions.id", ondelete="SET NULL"),
        nullable=True
    )

    method = Column(String, nullable=False)
    path = Column(String, nullable=False)
    status_code = Column(Integer, nullable=False)
    ip_address = Column(String, nullable=True)
    user_agent = Column(String, nullable=True)
    duration = Column(Float)
    timestamp = Column(DateTime, default=datetime.utcnow)
    query_params = Column(JSON, nullable=True)
    action = Column(String, nullable=True)
    request_body = Column(Text, nullable=True)

    user = relationship("User", back_populates="audit_logs")
    session = relationship(
        "Session", back_populates="audit_logs", passive_deletes=True)
```