Міністерство освіти і науки України
Тернопільський національний технічний університет імені Івана Пулюя

Факультет комп'ютерно-інформаційних систем і програмної інженерії
(повна назва факультету)
Кафедра програмної інженерії
(повна назва кафедри)

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття освітнього ступеня

бакалавр
(назва освітнього ступеня)

на тему: Розробка програмного забезпечення веб-системи управління навчальним контентом з використанням мови програмування Python

Виконав(ла): студент(ка) IV курсу, групи СП-42
спеціальності 121 Інженерія програмного забезпечення
(шифр і назва спеціальності)

	(підпис)	Чигрин М.М (прізвище та ініціали)
Керівник	(підпис)	Михалик Д.М. (прізвище та ініціали)
Нормоконтроль	(підпис)	Стоянов Ю.М. (прізвище та ініціали)
Завідувач кафедри	(підпис)	Петрик М.Р. (прізвище та ініціали)
Рецензент	(підпис)	(прізвище та ініціали)

Тернопіль
2026

Міністерство освіти і науки України
Тернопільський національний технічний університет імені Івана Пулюя

Факультет комп'ютерно інформаційних систем і програмної інженерії
(повна назва факультету)

Кафедра програмної інженерії
(повна назва кафедри)

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Петрик М.Р.
(підпис) (прізвище та ініціали)
« » 2026 р.

З А В Д А Н Н Я
НА КВАЛІФІКАЦІЙНУ РОБОТУ

на здобуття освітнього ступеня бакалавр
(назва освітнього ступеня)

за спеціальністю 121 Інженерія програмного забезпечення
(шифр і назва спеціальності)

студенту Чигрин Максим Мирославович
(прізвище, ім'я, по батькові)

1. Тема роботи Розробка програмного забезпечення веб-системи управління навчальним контентом з використанням мови програмування Python

Керівник роботи к.т.н., доц. Михалик Д.М.
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

Затверджені наказом ректора від «*06*» *квітня* 2026 року № *4/9-170*

2. Термін подання студентом завершеної роботи

3. Вихідні дані до роботи Предметна область, технічне завдання, вимоги та специфікація, програмне рішення.

4. Зміст роботи (перелік питань, які потрібно розробити)
Вступ. 1 Аналіз вимог до програмної системи.
2 Проєктування та розробка програмної системи.
3 Тестування, впровадження та підтримка.
4 Безпека життєдіяльності, основи охорони праці.
Висновки. Список використаних джерел. Додатки.

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень, слайдів)
Ілюстративні зображення, інформативні зображення для доповнення тексту, діаграми, знімки екрану з проробленою роботою.
1. Тема роботи. 2. Актуальність, мета, задачі дослідження
3. Існуючі технології реалізації подібних систем.
4. Функціональні та нефункціональні вимоги .5. Загальна архітектура системи.
6. Варіанти використання. 7. Компоненти програми для налаштування параметрів системи.
8. Програмні засоби та технології. 9. Інтерфейси реалізації застосунку..
10. Тестування. 11. Висновки по роботі. 12. Слайди презентації.

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Безпека життєдіяльності та основи охорони праці			

7. Дата видачі завдання _____

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів роботи	Термін виконання етапів роботи	Примітка
1	Розробка технічного завдання	6.04 – 12.04	
2	Робота над першим розділом «Аналіз вимог до програмної системи»	13.04 – 26.04	
3	Робота над другим розділом «Проєктування та розробка програмної системи»	27.04 – 10.05	
4	Робота над третім розділом «Тестування, впровадження та підтримка»	11.05 – 24.05	
5	Робота над четвертим розділом «Безпека життєдіяльності, основи охорони праці»	25.05 – 31.05	
6	Оформлення пояснювальної записки та графічного матеріалу	1.06 – 7.06	
7	Перевірка на академічну доброчесність, перевірка керівником і консультантами	8.06 – 14.06	
8	Попередній захист кваліфікаційної роботи бакалавра	15.06 – 21.06	
9	Захист кваліфікаційної роботи бакалавра		

Студент _____ Чигрин М.М.
(підпис) (прізвище та ініціали)

Керівник роботи _____ Михалик Д.М.
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Розробка програмного забезпечення веб-системи управління навчальним контентом з використанням мови програмування Python // Кваліфікаційна робота освітнього рівня “Бакалавр” // Чигрин Максим Мирославович // ТНТУ ім. І. Пулюя, факультет комп’ютерно-інформаційних систем і програмної інженерії, кафедра програмної інженерії, група СП-42 // Тернопіль, 2026. Сторінок – 79, рисунків – 25, таблиць – 7, слайдів – 17, додатків – 3, посилань – 33, лістингів – 8.

Ключові слова: веб-система, управління навчальним контентом, Python, Django, Django Ninja, REST API, PostgreSQL, Redis, JWT, Docker, pytest, тестування.

Кваліфікаційна робота досліджує процеси аналізу, проєктування, розробку та тестування програмного рішення веб-системи управління навчальним контентом передовими технологіями та інструментами у сфері веб-розробки на основі мови програмування Python.

У першому розділі кваліфікаційної роботи досліджено предметну область, конкурентні рішення, проаналізовано вимоги до системи та засоби й інструменти розробки. Розділ дозволяє ознайомитись з темою та актуальністю проблематики цієї теми.

У другому розділі кваліфікаційної роботи описано процеси вибору архітектури рішення, проєктування та програмної реалізації. Розділ дозволяє ознайомитись з процесами безпосередньої розробки на основі даних із першого розділу.

У третьому розділі кваліфікаційної роботи описано тестування та верифікацію розробленого програмного рішення. Розділ дозволяє ознайомитись з інструментами та підходами до тестування і сформулювати кінцеву оцінку задовільності виконаної роботи.

ABSTRACT

Software development for a web-based learning content management system using the Python programming language // Bachelor's Qualification Work // Chyhryn Maksym Myroslavovych // Ternopil Ivan Puluj National Technical University, Faculty of Computer Information Systems and Software Engineering, Department of Software Engineering, group SP-42 // Ternopil, 2026. Pages – 79, figures – 25, tables – 7, slides – 17, appendices – 3, references – 33, listings – 8.

Keywords: web system, learning content management, Python, Django, Django Ninja, REST API, PostgreSQL, Redis, JWT, Docker, pytest, testing.

The qualification work explores the processes of analysis, design, development and testing of a software solution for a web-based learning content management system using modern technologies and tools in the field of web development based on the Python programming language.

The first section examines the subject area, competitive solutions, analyses the system requirements and the means and tools of development. The section provides an introduction to the topic and the relevance of its problems.

The second section describes the processes of choosing the system architecture, designing and the software implementation. The section provides an overview of the actual development processes based on the data from the first section.

The third section describes the testing and verification of the developed software solution. The section presents the tools and approaches to testing and allows formulating the final assessment of the completed work.

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

БД – база даних.

БЖД – безпека життєдіяльності.

ВДТ – відеотермінал (візуальний дисплейний термінал).

ДСТУ – державний стандарт України.

ПЗ – програмне забезпечення.

API – Application Programming Interface – програмний інтерфейс застосунку.

CI/CD – Continuous Integration / Continuous Delivery – безперервна інтеграція та доставка.

Docker – платформа контейнеризації застосунків.

HTTP – HyperText Transfer Protocol – протокол передачі гіпертексту.

JSON – JavaScript Object Notation – формат обміну даними.

JWT – JSON Web Token – веб-токен автентифікації у форматі JSON.

LMS – Learning Management System – система управління навчанням.

ORM – Object-Relational Mapping – об’єктно-реляційне відображення.

REST – Representational State Transfer – архітектурний стиль веб-сервісів.

S3 – Simple Storage Service – об’єктне хмарне сховище AWS.

URL – Uniform Resource Locator – уніфікований локатор ресурсу.

ЗМІСТ

ВСТУП

Стрімкий розвиток цифрових технологій кардинально змінив підходи до організації освітнього процесу та індивідуального розвитку. Веб-системи управління навчальним контентом перетворилися з простих репозиторіїв навчальних матеріалів на повноцінні платформи, що поєднують доставку контенту, оцінювання, спілкування з куратором та збір зворотного зв'язку. Особливої актуальності такі рішення набули в умовах дистанційного та змішаного навчання, коли учасники освітнього процесу взаємодіють виключно через цифрові канали.

Поряд із загальновідомими комерційними та відкритими LMS-платформами (Moodle, Canvas LMS, Google Classroom, Blackboard Learn) існує значний попит на спеціалізовані рішення, які розробляються під конкретні освітні програми, цільову аудиторію або методологію навчання. Такі системи дозволяють реалізувати специфічні бізнес-правила, наприклад послідовне відкриття модулів за розкладом групи, гнучке оцінювання прогресу слухача за переглянутими уроками та пройденими квізами, інтеграцію з зовнішніми чат-каналами для оперативної комунікації, а також підсистему вхідного та підсумкового опитування слухачів (зокрема для оцінки самооцінки рівня знань та мотивації слухача) до початку та після завершення курсу для оцінки ефективності освітнього впливу. Розробка власного програмного забезпечення також забезпечує повний контроль над персональними даними та архітектурою системи.

Актуальність теми. Розробка програмного забезпечення власної веб-системи управління навчальним контентом зумовлена: зростанням попиту на гнучкі цифрові освітні платформи; необхідністю підтримки різноманітних форматів навчальних матеріалів і автоматизованого оцінювання; потребою у групово-орієнтованому проходженні курсів з керованим розкладом відкриття уроків; доцільністю проведення вхідного та підсумкового опитування слухачів зі шкалюванням відповідей до й після проходження курсу для об'єктивної оцінки його освітнього ефекту; необхідністю інтеграції з зовнішніми сервісами (емейл-

розсилка, об’єктне хмарне сховище, моніторинг помилок) при збереженні єдиного програмного інтерфейсу для клієнтських застосунків.

Мета роботи. Розробити програмне забезпечення веб-системи управління навчальним контентом з використанням мови програмування Python, яке забезпечить керування навчальними модулями, уроками, домашніми завданнями, квізами та користувачами, а також підсистему вхідного та підсумкового опитування слухачів (зокрема для оцінки рівня знань) у веб-середовищі.

Для досягнення поставленої мети необхідно вирішити такі завдання:

1. проаналізувати предметну область управління навчальним контентом та виконати огляд існуючих рішень;

2. визначити функціональні та нефункціональні вимоги до системи;

3. спроєктувати архітектуру програмного рішення та схему бази даних;

4. розробити серверну частину системи на основі мови програмування Python, фреймворку Django та бібліотеки Django Ninja, реалізувати REST API із JWT-автентифікацією;

5. реалізувати підсистему вхідного та підсумкового опитування слухачів з налаштовуваним переліком питань зі шкалою балів та двома фіксованими спробами – перед першим уроком і після завершення курсу – для оцінки динаміки самооцінки рівня знань та мотивації слухача слухача;

6. провести модульне та інтеграційне тестування системи за фреймворком pytest, забезпечити покриття коду не менше 80 %;

7. підготувати інфраструктуру розгортання на основі Docker та конвеєр безперервної інтеграції GitHub Actions.

Об’єкт дослідження. Процес управління навчальним контентом у сучасних веб-системах.

Предмет дослідження. Методи та засоби розробки програмного забезпечення веб-системи управління навчальним контентом на основі мови Python та фреймворку Django.

Методи дослідження. У роботі використано методи аналізу та синтезу для дослідження предметної області, об’єктно-орієнтованого проєктування при

розробці архітектури системи, методи REST-архітектури при проєктуванні API, реляційного моделювання при розробці схеми бази даних, методи модульного та інтеграційного тестування для верифікації коду, а також методи статичного аналізу за допомогою лінтера ruff.

Наукова новизна одержаних результатів. Вперше запропоновано серверну архітектуру веб-системи управління навчальним контентом, яка одночасно інтегрує групово-орієнтоване проходження курсу з керованим календарним розкладом відкриття модулів, вбудоване логування пристроїв слухачів для подальшої аналітики аудиторії, вбудовану підсистему вхідного та підсумкового опитування з налаштовуваним переліком питань зі шкалою балів для вимірювання динаміки рівня знань слухача в межах освітнього циклу, а також REST API з автогенерацією `OpenAPI`-специфікації для уніфікованої взаємодії з веб- і мобільними клієнтами в межах єдиного бекенду.

Практичне значення одержаних результатів. Розроблене програмне забезпечення є самостійним серверним рішенням з повноцінним REST API, що може використовуватися як основа для будь-якої спеціалізованої освітньої платформи. Модульна архітектура на основі Django-додатків полегшує подальше розширення функціоналу, контейнеризація через Docker спрощує розгортання, а наявність автоматизованого тестування та CI/CD гарантує контроль якості при внесенні змін.

Апробація результатів. Окремі результати роботи представлено на IX Міжнародній студентській науково-технічній конференції “Природничі та гуманітарні науки. Актуальні питання” (Тернопіль, 2026 р.), секція “Інформаційні технології”, у тезах “Архітектурні підходи до побудови веб-системи управління навчальним контентом”.

Структура роботи. Кваліфікаційна робота складається зі вступу, чотирьох розділів, висновків, списку використаних джерел та додатків. Бібліографічний опис джерел оформлений згідно з ДСТУ 8302:2015. Загальний обсяг пояснювальної записки становить 79 сторінок основного тексту, включаючи 25 рисунків, 7 таблиць, 33 бібліографічних джерела, 8 лістингів та 3 додатки.

Використання інструментів штучного інтелекту. Під час підготовки кваліфікаційної роботи інструменти генеративного штучного інтелекту використовувалися як допоміжний засіб для технічних завдань: Claude (Anthropic, модель Opus 4.8) – для стилістичного редагування та перевірки граматики тексту пояснювальної записки, формулювання окремих фрагментів тексту на основі наданих автором матеріалів і вихідного коду проєкту, генерації сценаріїв модульних та інтеграційних тестів (з подальшою перевіркою, запуском і аналізом результатів автором) і звірки оформлення роботи з методичними вказівками; Perplexity – для попереднього пошуку бібліографічних джерел, які надалі перевірено за першоджерелами. Постановку задачі, проєктні та архітектурні рішення, формулювання наукової новизни, висновки і практичні рекомендації виконано автором самостійно. Усі використані джерела перевірено за першоджерелами. Автор несе повну відповідальність за зміст роботи.

1 АНАЛІЗ ВИМОГ ДО ПРОГРАМНОЇ СИСТЕМИ

У цьому розділі досліджено предметну область управління навчальним контентом, виконано порівняльний аналіз провідних рішень на ринку, сформульовано функціональні та нефункціональні вимоги до розроблюваної системи й обґрунтовано вибір технологічного стеку, який буде використано в подальших розділах при проєктуванні та реалізації програмного забезпечення.

1.1 Аналіз предметної області

Актуальність створення спеціалізованих систем управління навчальним контентом, орієнтованих на конкретну освітню програму, обґрунтовано, зокрема, у публікації автора [1]. Системи управління навчальним контентом (Learning Content Management System, LCMS, у ширшому контексті – Learning Management System, LMS) – це програмні платформи, призначені для адміністрування, документування, відстеження прогресу та доставки навчальних матеріалів учасникам освітнього процесу [2, 3, 4]. Сучасні LMS виходять за межі простих електронних репозиторіїв та надають комплекс інструментів: створення курсів і уроків, керування користувачами та групами, оцінювання, формування зворотного зв’язку, аналітику, інтеграцію з зовнішніми сервісами комунікації та аналізу даних.

LMS-системи традиційно належать до категорії корпоративних інформаційних систем. Відповідно до рекомендацій SWEBOK [5], для таких систем критичними є чітке моделювання доменних об’єктів, масштабованість та інтеграційність. Для веб-орієнтованих рішень типовим є архітектурний стиль REST (Representational State Transfer), сформульований Р. Філдінгом [6]. Цей підхід задає принципи безстанової взаємодії клієнта і сервера через єдиний інтерфейс.

Типова веб-система управління навчальним контентом охоплює такі компоненти: сховище навчального контенту з чотирирівневою ієрархічною структурою (курс, модуль, урок, матеріал); підсистему оцінювання, що включає квізи з різними типами запитань та домашні завдання; підсистему управління

користувачами з декількома рівнями доступу та реєстрацією за єдиним ідентифікатором (як правило, email або номер телефону); підсистему сповіщень електронною поштою; інфраструктурні компоненти для зберігання медіа-файлів і логування подій.

Предметна область характеризується такими особливостями. По-перше, контент має чітку ієрархічну структуру (курс складається з модулів, модулі – з уроків, уроки можуть містити текст, відео, аудіо, квіз або домашнє завдання), що природно лягає в реляційну модель бази даних із зовнішніми ключами та табелями зв’язку. По-друге, послідовність проходження матеріалу часто керована: уроки можуть відкриватися або згідно з графіком групи (наприклад, по одному модулю на тиждень), або після виконання попередніх кроків. По-третє, навколо змісту з’являється шар оцінювання – нараховуються бали за переглянуті уроки, пройдені квізи та зараховані домашні завдання, на основі яких будується індивідуальний прогрес слухача. По-четверте, для організації групової роботи характерним є додатковий шар комунікації – чат-канали для груп, надсилання запрошень за демографічним зрізом, а також логування пристроїв і браузерів, з яких слухачі звертаються до системи, для аналізу аудиторії на різних платформах.

Серед ключових факторів успіху сучасної навчальної платформи можна виділити: зручність використання та інтерфейс, оптимізований під мобільні пристрої; надійність і доступність сервісу не менше 99,5 %; гнучкість налаштування під методологію конкретного освітнього продукту; інтеграція з зовнішніми каналами комунікації; підтримка інтернаціоналізації; коректна робота з персональними даними згідно з нормативними вимогами.

У контексті цієї роботи предметна область обмежується розробкою серверної частини (back-end) системи. Серверна частина надає клієнтським застосункам (веб-фронтенду, мобільному додатку, адміністративному інтерфейсу) уніфікований REST API. Інтерфейс кінцевого користувача (front-end) реалізується окремо й виходить за межі цієї роботи. Основні варіанти використання системи з боку слухача – реєстрація та активація, перегляд каталогу модулів, проходження уроків,

здача домашніх завдань і квізів, перегляд власного прогресу та балів, оновлення профілю й отримання чат-запрошень – наведено на рисунку 1.1.

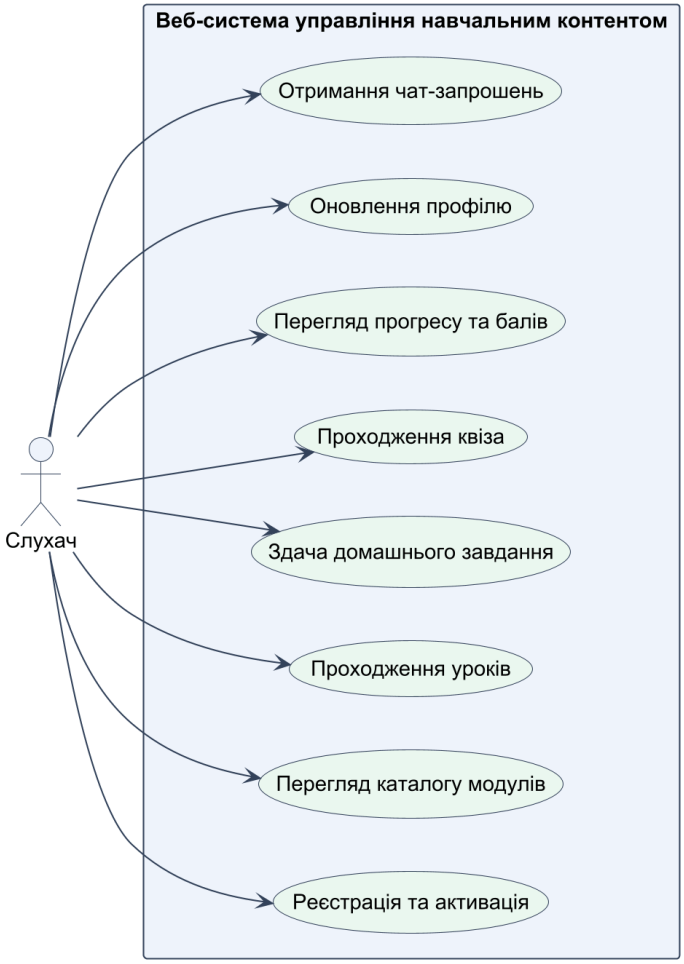


Рисунок 1.1 – Варіанти використання системи з боку слухача

Варіанти використання з боку куратора (адміністратора) охоплюють адміністрування навчального контенту та супровід освітнього процесу: управління модулями й уроками, перевірку домашніх завдань, створення квізів та опитувань, управління користувачами та групами слухачів, перегляд статистики, розсилання email-запрошень і контроль доступу до контенту. Їх наведено на рисунку 1.2.

Рисунок 1.2 – Варіанти використання системи з боку куратора/адміністратора

Наведені діаграми варіантів використання визначають функціональні межі системи з боку двох укрупнених акторів та слугують основою для деталізованого формулювання вимог у наступних підрозділах.

1.2 Огляд існуючих рішень

Ринок систем управління навчальним контентом є зрілим і представлений як комерційними, так і відкритими (open-source) рішеннями. Для обґрунтування технологічних рішень та визначення вимог до розроблюваної системи виконано порівняльний аналіз чотирьох представників ринку, результати якого узагальнено в таблиці 1.1.

Moodle – найпоширеніша відкрита LMS-платформа у світі, написана мовою PHP. Підтримує розгалужену систему плагінів, що дозволяє адаптувати її під різні освітні потреби. Має велику спільноту розробників, безкоштовно поширюється під ліцензією GPL. Серед недоліків зазначають перевантажений інтерфейс, складність

налаштування для непідготовлених адміністраторів та орієнтацію переважно на класичний університетський сценарій з курсами, лекціями та оцінюванням за стобальною шкалою [7].

Canvas LMS – комерційна хмарна платформа компанії Instructure. Має сучасний інтерфейс, добре оптимізована під мобільні пристрої, надає публічний REST API. Інтегрується з низкою зовнішніх сервісів (відеоконференції, антиплагіат, бібліографічні менеджери). Основні недоліки – висока вартість для освітніх установ та обмежені можливості кастомізації шаблонів інтерфейсу [8].

Google Classroom – безкоштовний сервіс компанії Google, тісно інтегрований з Google Workspace (Drive, Meet, Calendar). Завдяки простоті використання поширений у школах та невеликих курсах. До недоліків належать: обмежений функціонал у частині автоматизованого тестування, відсутність гнучкої системи ролей, складність побудови курсу з контрольованим послідовним відкриттям уроків [9].

Blackboard Learn – комерційна LMS-платформа для вищої освіти. Має багатий функціонал та потужну аналітику. До недоліків відносять складність впровадження, високу вартість та застарілий інтерфейс [10].

Таблиця 1.1 – Порівняння існуючих систем управління навчанням

Критерій	**Moodle**	**Canvas LMS**	**Google Classroom**	**Власна система**
Тип ліцензії	GPL (open source)	Комерційна, SaaS	Безкоштовний для шкіл	Розробка під замовлення
Мова реалізації	PHP	Ruby on Rails	Закрита, Google	Python / Django
Публічний REST API	Частковий	Так	Так, але обмежений	Так, Django Ninja
Групова сегментація	Так (cohorts)	Так (sections)	Базова	`UserGroup` з розкладом відкриття модулів
Логування пристроїв слухачів	Через плагіни	Так, частково	Немає	Вбудовано (`UserDeviceLog` middleware)
Контроль над БД	Повний (self-hosted)	Немає	Немає	Повний
Інтеграція з SendGrid	Через плагіни	Так	Через Workspace	Вбудовано
Контейнеризація	Так	SaaS	SaaS	Docker / Compose

Узагальнюючи результати порівняння, можна дійти таких висновків. Готові LMS-платформи добре розв’язують типові задачі університетського навчання, проте не завжди дозволяють реалізувати специфічні бізнес-правила (графік відкриття модулів для конкретної групи, гнучке нарахування балів за переглянуті уроки і пройдені квізи, інтеграція з зовнішніми чат-каналами за демографічними сегментами, логування пристроїв слухачів для подальшої аналітики). Самостійна розробка програмного забезпечення на сучасному технологічному стеку забезпечує гнучкість при подальшому розвитку продукту, повний контроль над персональними даними та інтеграцію з обраними зовнішніми сервісами без посередників.

Таким чином, обґрунтованим є рішення про розробку власного програмного забезпечення серверної частини веб-системи управління навчальним контентом, яке поєднує перевіреність архітектурних підходів open-source-рішень із сучасним стеком Python/Django, контейнеризацією та хмарними сервісами зберігання.

1.3 Постановка задачі та функціональні вимоги

Кваліфікаційну роботу виконано відповідно до методичних вказівок випускової кафедри [11], які регламентують її структуру, обсяг та оформлення. На основі аналізу предметної області та виявлених обмежень існуючих рішень сформульовано перелік функціональних вимог до розроблюваної системи. Функціональні вимоги описують поведінку системи з погляду користувача та визначають перелік підтримуваних операцій. Для кожної з вимог далі у розділах 2 та 3 наводитимуться відповідні реалізації та тестові сценарії.

Основні функціональні вимоги до системи:

1. Управління користувачами та автентифікація. Система повинна підтримувати реєстрацію нових користувачів за email, обов’язковим унікальним номером телефону, з можливістю заповнити демографічний профіль (стать, вікова група, країна, місто, наявність дітей, сімейний статус, інтереси). Підтверджується email через токен активації. Автентифікація реалізована за стандартом JWT (RFC 7519) з парою токенів access/refresh.

2. Управління навчальними модулями. Система повинна підтримувати створення, редагування та видалення модулів і уроків з боку адміністраторів. Кожен модуль містить впорядкований список уроків кількох типів (text, video, audio, quiz, homework). Контент уроків вводиться через WYSIWYG-редактор з підтримкою HTML.

3. Групово-орієнтоване проходження курсу. Система повинна підтримувати сутність "група" (`UserGroup`) з заданими параметрами реєстраційного періоду, дати старту курсу та інтервалу відкриття модулів у днях. Слухачі автоматично приєднуються до активної групи при реєстрації.

4. Підсистема домашніх завдань. Викладач / адміністратор створює завдання прив'язане до уроку, а слухач здає завдання у форматі тексту або файлу (pdf/docx/txt). Підтримується автоматичне затвердження здачі або ручне рев'ю кураторами з коментарем.

5. Підсистема квізів. Адміністратор створює квіз з питаннями різних типів (одиничний вибір, множинний вибір, відкритий текст). Система автоматично підраховує бал слухача, фіксує спроби проходження, повертає індивідуалізовані коментарі за обрані відповіді.

6. Підсистема індивідуального прогресу. Система повинна підраховувати поточний бал слухача (`User`.score) на основі переглянутих уроків, проходжених квізів та зарахованих домашніх завдань. Окремий API-метод повертає поточну позицію слухача (current_module, current_lesson) для відображення у клієнтському застосунку.

7. Підсистема вхідного та підсумкового опитування. Система повинна надавати слухачу налаштовуване адміністратором опитування зі шкалою балів (за замовчуванням 0–5): одну спробу до початку навчання і одну після завершення курсу, з обмеженням однієї спроби кожного типу. Зіставлення результатів дозволяє оцінити динаміку самооцінки рівня знань та мотивації слухача слухача.

8. Інтеграція з чат-каналами. Адміністратор формує перелік запрошень у зовнішні чат-канали (Telegram) із сегментацією за віковою групою або загальним

призначенням; слухач отримує запрошення на email або через окремий API-ендпоінт.

9. Сповіщення електронною поштою. Система повинна надсилати email-листи для активації акаунта, скидання пароля та запрошень у чат-канали через сервіс SendGrid.

10. Адміністративний інтерфейс. Система повинна надавати розширену адмін-панель на основі бібліотеки django-unfold для оперативного управління контентом, переглядом статистики та модерацією здач.

Окремою функціональною вимогою є логування сесій пристроїв користувачів (`UserDeviceLog`), яке реалізовується через спеціальне Django middleware, що зчитує заголовок `User`-Agent та фіксує метадані входу. Така можливість підвищує спостережуваність системи та допомагає аналізувати поведінку аудиторії на різних платформах.

Усі перелічені функціональні вимоги мають бути доступні через єдиний REST API, документований за стандартом `OpenAPI` 3.0. Окремих кінцевих точок для веб-клієнта та мобільного клієнта не передбачено – клієнтські застосунки взаємодіють з одним і тим самим API.

1.4 Нефункціональні вимоги до системи

Нефункціональні вимоги задають якісні характеристики системи, що безпосередньо не пов’язані з конкретним функціоналом, але впливають на надійність, безпеку, продуктивність та зручність експлуатації [5]. Для розроблюваної системи визначено такі групи нефункціональних вимог.

– Продуктивність. 95-й персентиль часу відповіді API не повинен перевищувати 300 мс на навантаженні 100 одночасних користувачів за умови, що запити не передбачають генерації важких звітів. Для часто запитуваних даних застосовується кешування у Redis [12].

– Масштабованість. Система повинна підтримувати горизонтальне масштабування рівня Gunicorn-воркерів та запуск декількох екземплярів за зворотним проксі Nginx без зміни прикладного коду.

– Надійність. Покриття коду тестами не нижче 80 %, забезпечене конфігурацією pytest --cov-fail-under=80. CI-конвеєр відмовляє при падінні цього порогу, що унеможливлює потрапляння неперевіреного коду до головної гілки.

– Безпека. Паролі зберігаються у вигляді PBKDF2-хешу засобами Django; автентифікація – за JWT з короткоживучим access-токеном (15 хв) та довгоживучим refresh-токеном (7 діб). Підтримується чорний список refresh-токенів (django-ninja-jwt blacklist). Усі ендпоінти, що змінюють стан, доступні лише за HTTPS на рівні зворотного проксі.

– Підтримка інтернаціоналізації. Рядки, що повертаються користувачу, обгорнуті у функції `gettext_lazy()`; файли перекладу зберігаються у `locale/uk` та `locale/en`. У базі даних рядкові поля підтримують Unicode (PostgreSQL з кодуванням UTF-8).

– Зручність розгортання. Розгортання здійснюється у Docker-контейнерах. Конфігурація різних середовищ (development, production) задається через `.env`-файли, які не комітяться у репозиторій.

– Спостережуваність. Усі помилки та неочікувані виключення відправляються у систему моніторингу Sentry. Структуроване логування реалізоване через бібліотеку loguru з виведенням у файл `app.log` і автоматичним передаванням критичних подій до Sentry.

– Якість коду. Лінтинг та автоматичне форматування коду здійснюються інструментом ruff згідно з конфігурацією у `pyproject.toml`. Конвенції фіксації змін у репозиторії відповідають Conventional Commits, що контролюється бібліотекою commitizen.

Окрім перелічених вимог, при розробці системи дотримуються вимог до якості коду: дотримання стилістичних правил мови Python (PEP 8) та принципів чистої архітектури за Р. Мартіном [13], тобто чіткого відокремлення доменних об’єктів від інфраструктурних залежностей.

1.5 Технології та інструменти розробки

Вибір технологічного стеку є рішенням, від якого залежить швидкість розробки, можливості подальшого супроводу та продуктивність системи. На основі вимог, сформульованих у попередніх підрозділах, та з урахуванням сучасних практик веб-розробки [14] обрано стек, наведений у таблиці 1.2. Основу стека становить мова програмування Python 3.13 [15] завдяки її зрілій екосистемі бібліотек, розвинутій підтримці асинхронного програмування та широкому набору фреймворків для веб-розробки.

Таблиця 1.2 – Обраний технологічний стек системи

Категорія	**Інструмент**	**Призначення у проєкті**
Мова програмування	Python 3.13	Основна мова реалізації серверної частини
Веб-фреймворк	Django 5.2	ORM, адміністративна панель, інтернаціоналізація, сигнали
REST-фреймворк	Django Ninja 1.4	Маршрутизація, валідація `Pydantic`-схемами, авто-генерація `OpenAPI`
Автентифікація	django-ninja-jwt 5.3	JWT з підтримкою чорного списку refresh-токенів
СУБД	PostgreSQL 17	Реляційне сховище даних, `JSONB`-поля, повнотекстовий пошук
Кеш / черга	Redis 7	Кешування часто запитуваних даних, сесійне сховище
Об’єктне сховище	AWS S3 (django-storages + boto3)	Зберігання медіа-файлів (відео, аудіо, аватари)
Email-сервіс	SendGrid (django-sendgrid-v5)	Транзакційні листи, шаблони, підтримка масових розсилок
Адмін-панель	django-unfold 0.74	Кастомний інтерфейс адміністратора
WYSIWYG-редактор	django-tinymce	Редагування HTML-вмісту уроків
Контейнеризація	Docker, Docker Compose	Уніфікація середовища розробки і продакшну
Зворотний проксі	Nginx	TLS-термінація, статика, балансування Gunicorn-воркерів
WSGI-сервер	Gunicorn	Запуск Python-застосунку у продакшні
Тестування	pytest, pytest-django, pytest-cov	Модульні та інтеграційні тести, аналіз покриття
Логування	loguru	Структуроване логування з ротацією у `app.log`
Моніторинг	Sentry SDK	Збір помилок і трейсів у реальному часі
Якість коду	ruff, commitizen	Лінтинг, форматування, конвенції повідомлень коміту

Продовження таблиці 1.2

1	2	3
CI/CD	GitHub Actions	Автоматичні тести і збірка Docker-образів

Висновки до першого розділу

У першому розділі досліджено предметну область систем управління навчальним контентом і визначено її ключові особливості: ієрархічну структуру контенту, кероване відкриття матеріалів, шар оцінювання прогресу слухача та комунікаційні функції.

Проаналізовано наявні рішення (Moodle, Canvas, Google Classroom, Blackboard), сформульовано функціональні та нефункціональні вимоги до системи й обґрунтовано вибір технологічного стека (Python 3.13, Django, Django Ninja, PostgreSQL, Redis, Docker). Отримані результати є основою для проєктування архітектури системи у наступному розділі.

2 ПРОЄКТУВАННЯ ТА РОЗРОБКА ПРОГРАМНОЇ СИСТЕМИ

На основі результатів аналізу, виконаного у розділі 1, цей розділ присвячено вибору процесу розробки, архітектурним рішенням, проєктуванню сутностей бази даних, розробці REST API та реалізації бізнес-логіки системи. Окремий підрозділ описує інтерфейс адміністрування та інтеграцію з зовнішніми сервісами, які забезпечують повноцінну роботу системи у продуктивному середовищі.

2.1 Вибір процесу розробки

Перш ніж переходити до проєктування архітектури, обґрунтовано вибір процесу (моделі життєвого циклу) розробки програмного забезпечення. Модель життєвого циклу визначає порядок та характер виконання основних етапів – аналізу вимог, проєктування, реалізації, тестування та супроводу [5]. Від обраного процесу залежать гнучкість внесення змін, швидкість отримання працездатних версій системи та керованість якістю впродовж усієї розробки.

Для цієї роботи обрано ітеративно-інкрементну модель життєвого циклу на засадах гнучкої методології (Agile). На відміну від каскадної (водоспадної) моделі, у якій етапи виконуються послідовно й одноразово, ітеративний підхід передбачає поступове нарощування функціональності: на кожній ітерації проходяться всі етапи життєвого циклу для окремого набору вимог, а обсяг наступної ітерації планується з урахуванням уже отриманого результату. Такий підхід відповідає характеру предметної області, у якій вимоги до навчальної платформи уточнюються в міру розвитку продукту.

Розробку структуровано за функціональними модулями (Django-додатками), кожен з яких реалізовувався окремою ітерацією: керування користувачами та автентифікація, навчальні модулі й уроки, домашні завдання, квізи, підсистема опитування та адміністрування. Інструментальною основою ітеративного процесу слугують система контролю версій Git, конвеєр безперервної інтеграції та доставки (CI/CD) на GitHub Actions, автоматизоване модульне й інтеграційне тестування з

контролем покриття коду, а також статичний аналіз коду та уніфіковані повідомлення комітів (Conventional Commits). Завдяки цьому кожна зміна автоматично перевіряється, а працездатний стан головної гілки репозиторію підтримується протягом усього циклу розробки.

Обраний ітеративний процес визначає й логіку викладу цього розділу: детальне проєктування архітектури, схеми бази даних, REST API та бізнес-логіки, виконане в його межах, описано в наступних підрозділах.

2.2 Архітектура програмної системи

Архітектура програмного рішення побудована за принципом багаторівневої (layered) архітектури з чітким розділенням відповідальностей. Такий підхід є рекомендованим у класичних роботах з архітектури корпоративних застосунків [16, 13, 17] і дозволяє ізольовано тестувати кожен шар, замінювати інфраструктурні залежності та полегшує супровід коду.

Логічна архітектура системи містить шість рівнів. Верхнім рівнем є клієнтський рівень (веб-браузер або мобільний додаток), який взаємодіє з системою виняткова через HTTPS-запити до REST API. Запити приймає зворотний проксі Nginx, який термінує TLS, віддає статику й передає динамічні запити у Gunicorn-воркери з Django-застосунком. У межах Django-процесу запит проходить через рівень API (Django Ninja), що виконує маршрутизацію та валідацію вхідних даних за `Pydantic`-схемами; далі викликається рівень бізнес-логіки (сервіси, сигнали, кастомні менеджери), який оперує доменними об’єктами та делегує читання/запис до рівня доступу до даних (Django ORM). Нижній рівень – інфраструктурний – об’єднує PostgreSQL, Redis, S3-сумісне сховище, SendGrid та Sentry. Загальна схема архітектури показана на рисунку 2.1.

Рисунок 2.1 – Багаторівнева архітектура веб-системи

Модульна структура реалізована у вигляді шести Django-додатків, що відповідають предметним підобластям системи: users, modules, homeworks, quizzes, mental_health та admin_custom. Додаток `mental_health` реалізує підсистему вхідного та підсумкового опитування слухачів з налаштовуваним адміністратором переліком питань і шкалою відповідей; її детальний опис наведено у підрозділах 2.3 та 2.5. Кожен додаток має стандартну структуру файлів: models.py – доменні моделі; api.py – маршрути Django Ninja; schemas.py – вхідні та вихідні `Pydantic`-схеми; services.py – бізнес-логіка; signals.py – реакція на події сигналів Django; managers.py – кастомні QuerySet-менеджери; exceptions.py – доменні виключення; `tests.py` – модульні та інтеграційні тести.

Центральним компонентом є файл `apps/routers`.py, у якому усі додатки реєструють свої роутери через список `API_ROUTERS`. У такий спосіб реалізовано принцип компонентної архітектури: для додавання нового функціонального модуля

достатньо створити окремий Django-додаток та додати один рядок у цей файл. Структура додатків показана на рисунку 2.2.

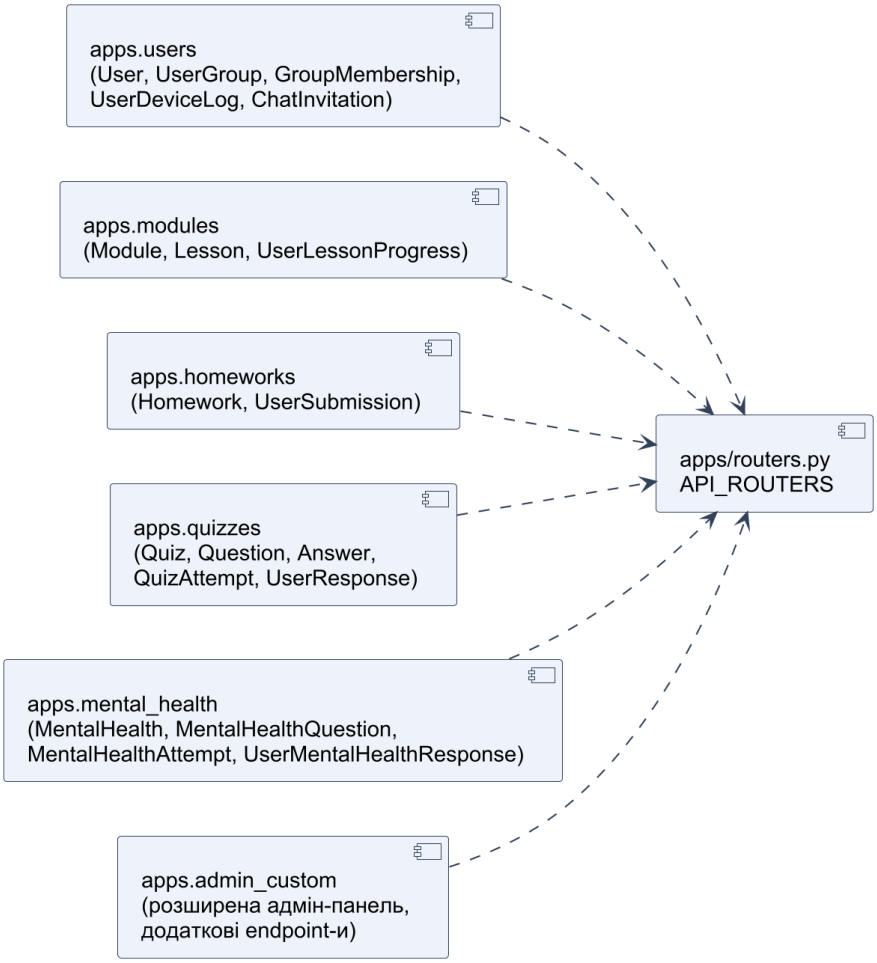


Рисунок 2.2 – Структура Django-додатків та центрального роутера

Усі ендпоінти HTTP-API доступні за префіксом /api/v1/. Інтерфейс адміністрування підключено за стандартним префіксом `/admin/`, статичні файли віддаються Nginx з директорії `staticfiles/`, а медіа-файли користувачів зберігаються в AWS S3 і доступні через адресу окремого S3-бакета. У середовищі розробки використовується Docker Compose для одночасного запуску усіх сервісів (Django + PostgreSQL + Redis + Nginx) одним командним рядком. Загальна схема розгортання компонентів у контейнеризованому середовищі показана на рисунку 2.3.

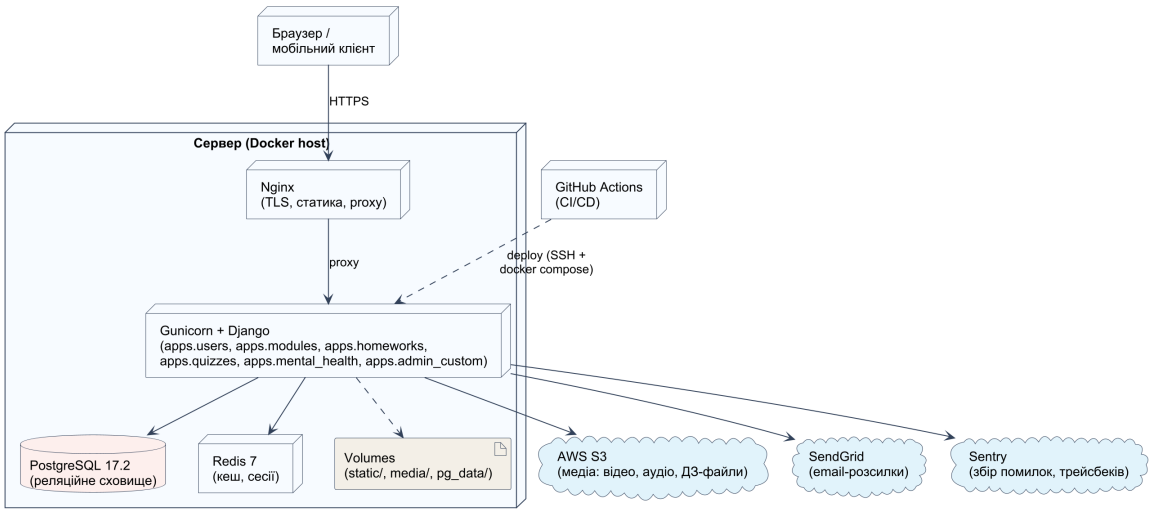


Рисунок 2.3 – Діаграма розгортання системи у контейнеризованому середовищі

Окремо слід відзначити кастомне Django middleware `DeviceLogMiddleware` (файл `config/middleware.py`), яке зчитує заголовок `User`-Agent з бібліотеки user-agents та фіксує мета-дані пристрою користувача (тип браузера, операційна система, мобільний/десктоп) у моделі `UserDeviceLog`. Це middleware вмикається у налаштуваннях Django раніше за middleware автентифікації та не блокує запит у разі помилки парсингу. Код цього middleware наведено у Додатку А (лістинг А.6).

Окрім самого принципу багаторівневої архітектури, під час реалізації системи свідомо застосовано низку патернів проєктування. Породжувальні та поведінкові патерни використано за класичною класифікацією «банди чотирьох» [18], а патерни рівня застосунку – за каталогом архітектури корпоративних застосунків [16, 13]. Нижче наведено ті патерни, що мають безпосереднє відображення у вихідному коді системи.

– фабричний метод (Factory Method, породжувальний) – кастомний менеджер `UserManager` з методами `create_user()` та `create_superuser()` інкапсулює створення користувача (нормалізація email, хешування пароля, встановлення службових прапорів is_staff/is_superuser), приховуючи деталі ініціалізації об’єкта від решти коду;

– одинак (Singleton, породжувальний) – абстрактна модель `SingletonModel` (`mixins/singleton.py`) з кешованим методом доступу `get_solo()` гарантує існування лише одного екземпляра; її застосовано до моделі `MentalHealth`, тож у

системі підтримується рівно одна анкета вхідного та підсумкового опитування зі спільним для всіх слухачів переліком питань;

– спостерігач (Observer, поведінковий) – реалізований штатним механізмом сигналів Django (`post_save`, `post_delete`): обробник `assign_user_to_group` реагує на створення моделі `User`, `handle_user_score_update` – на зміну `UserLessonProgress` і перераховує агрегатне поле `User`.score, а `update_attempt_score` – на зміну `UserMentalHealthResponse`; у такий спосіб об'єкти-обробники реагують на події доменних моделей без жорсткого зв'язування з ними;

– шар сервісів (Service Layer) – бізнес-логіку винесено у класи та функції файлів services.py (`UserProgressService`, `LessonNavigationService`, а також функції підрахунку балу у `apps/quizzes/services.py`), завдяки чому ендпоінти API лишаються «тонкими» – вони лише перевіряють авторизацію, валідують запит та делегують роботу сервісу;

– репозиторій через кастомні менеджери Django – `ActiveManager` та `ActiveLessonManager` приховують деталі фільтрації QuerySet (is_active=True), унеможливлюючи «протікання» неактивних об'єктів у клієнтську відповідь;

– об'єкт передавання даних (DTO) – `Pydantic`-схеми у файлах schemas.py відокремлюють вхідні та вихідні контракти REST API від внутрішніх моделей бази даних, запобігаючи витоку службових полів у клієнтську відповідь;

– фасад (Facade) – файл `apps/routers`.py служить єдиною точкою реєстрації роутерів усіх Django-додатків, що дає змогу додавати нові функціональні модулі без зміни наявної логіки.

Такий набір патернів полегшує тестування (кожен шар можна перевіряти ізольовано через моки сусідніх шарів) та подальший розвиток системи з дотриманням принципів чистої архітектури [13].

Описана багаторівнева архітектура у поєднанні з модульною структурою додатків та контейнеризованим середовищем розгортання забезпечує чітке розділення відповідальностей між компонентами та дозволяє у наступних

підрозділах деталізувати схему бази даних, REST API і бізнес-логіку незалежно один від одного.

2.3 Проєктування сутностей системи та бази даних

Схема бази даних спроєктована відповідно до вимог третьої нормальної форми (3NF). Як СУБД обрано PostgreSQL 17 [19] завдяки підтримці типу `JSONB`, повнотекстового пошуку, масивних полів (`ArrayField`) та чіткої сумісності з Django ORM. Загальний вигляд ER-діаграми системи наведено на рисунках 2.4 і 2.5 (відповідно домени користувачів і навчального контенту та домени оцінювання й самооцінювання знань), а доменну модель у вигляді UML-діаграми класів – на рисунку 2.6.

Користувацька модель apps.users.models.`User` розширює стандартну Django-модель `AbstractUser` і використовує email як первинний ідентифікатор. Окрім стандартних полів, модель містить унікальний номер телефону (`PhoneNumberField`), поточну кількість набраних балів (score), демографічні поля – стать (`GenderChoice`), вікову групу (`AgeGroupChoice`), країну, місто, наявність дітей (`ChildrenChoice`), сімейний статус (`FamilyStatusChoice`), а також `ArrayField` інтересів (`InterestTypeChoice`). Така схема дозволяє виконувати сегментацію аудиторії та формувати персоналізовані запрошення у чат-канали. Фрагмент моделі користувача наведено у Додатку А (лістинг А.1).

Групи користувачів моделюються через сутність `UserGroup` (apps.users.models.`UserGroup`). Кожна група має реєстраційне вікно (registration_started_at, registration_finished_at), дату старту курсу (course_started_at) та інтервал відкриття модулів у днях (opening_interval_days). Метод `clean()` забезпечує консистентність: не можна одночасно деактивувати останню активну групу, не можна зробити групу активною після закінчення реєстраційного періоду тощо. Зв’язок користувача з групою реалізовано через проміжну модель `GroupMembership`.

Сутність `Module` (apps.modules.models.`Module`) описує навчальний модуль з полями name, description (HTML), order (порядок), is_scored (чи рахуються бали), is_active. Підлеглі модулю об'єкти `Lesson` містять прив'язку до модуля (module_fk), тип контенту (`ContentType`: text, video, audio, quiz, homework), файлові поля video_url та audio_url з валідаторами розширень, HTML-опис та текстовий контент, бал score та порядок order. Прогрес проходження уроку слухачем фіксується у моделі `UserLessonProgress` з унікальним обмеженням (user_fk, lesson_fk), щоб одна особа не могла повторно отримати бали за один і той же урок.

Підсистема домашніх завдань складається з двох моделей: `Homework` (опис завдання, прив'язка до уроку через lesson_fk, прапор is_auto_approved) та `UserSubmission` (здача користувача з text_answer, file_answer, feedback, is_approved, created_at, date_review). Метод `clean()` для `UserSubmission` забороняє здачу, у якій порожні і текст, і файл. Метод `save()` при кожному оновленні полів feedback або is_approved автоматично виставляє дату рецензування. Можливі стани здачі та переходи між ними (чернетка, надсилання на рецензію, затвердження або відхилення куратором) показані на діаграмі станів на рисунку 2.7.

Квізи реалізовані через моделі `Quiz`, `Question`, `Answer`, `QuizAttempt` та `QuizAttemptAnswer`. Питання має тип `QuestionTypes` (text, single, multiple) і прив'язане до квіза через quiz_fk. Для кожного питання визначаються коректна та некоректна шаблонні відповіді (`QuestionResultMessageTemplate`), які повертаються користувачу як зворотний зв'язок після проходження. Загальний максимальний бал квіза обчислюється як сума балів за всі питання: 2 бали за текстове, 1 бал за вибіркове. Послідовності взаємодії при проходженні квіза та здачі домашнього завдання детально розглянуто у підрозділі 2.5.

Окремою службовою сутністю є `UserDeviceLog`, яка зберігається у Django-додатку apps.users та містить мета-дані пристрою користувача (user_agent, browser, os, is_mobile, ip, created_at). Записи створюються автоматично кастомним middleware `DeviceLogMiddleware` при кожному автентифікованому запиті, що дозволяє у подальшому аналізувати, з яких пристроїв і браузерів слухачі

звертаються до системи. Така схема забезпечує спостережуваність без додаткових інструментів сторонніх постачальників.

Підсистему вхідного та підсумкового опитування реалізують моделі `MentalHealth`, `MentalHealthQuestion`, `MentalHealthAttempt` та `UserMentalHealthResponse` (Django-додаток `mental_health`). `MentalHealth` – єдина для системи анкета-сінглтон з упорядкованим переліком питань `MentalHealthQuestion`, кожне з власною числовою шкалою (за замовчуванням 0–5). `MentalHealthAttempt` фіксує спробу проходження з ознакою етапу (до початку чи після завершення курсу), а унікальне обмеження не дозволяє слухачу пройти опитування одного типу двічі; `UserMentalHealthResponse` зберігає відповіді на окремі питання. На відміну від квізів, що виставляють об'єктивний бал за правильність відповідей, ця підсистема фіксує суб'єктивну самооцінку слухача за заданою шкалою, а зіставлення підсумкового та вхідного результатів дає змогу оцінити динаміку самооцінки рівня знань та мотивації слухача – залежно від налаштованих адміністратором питань.

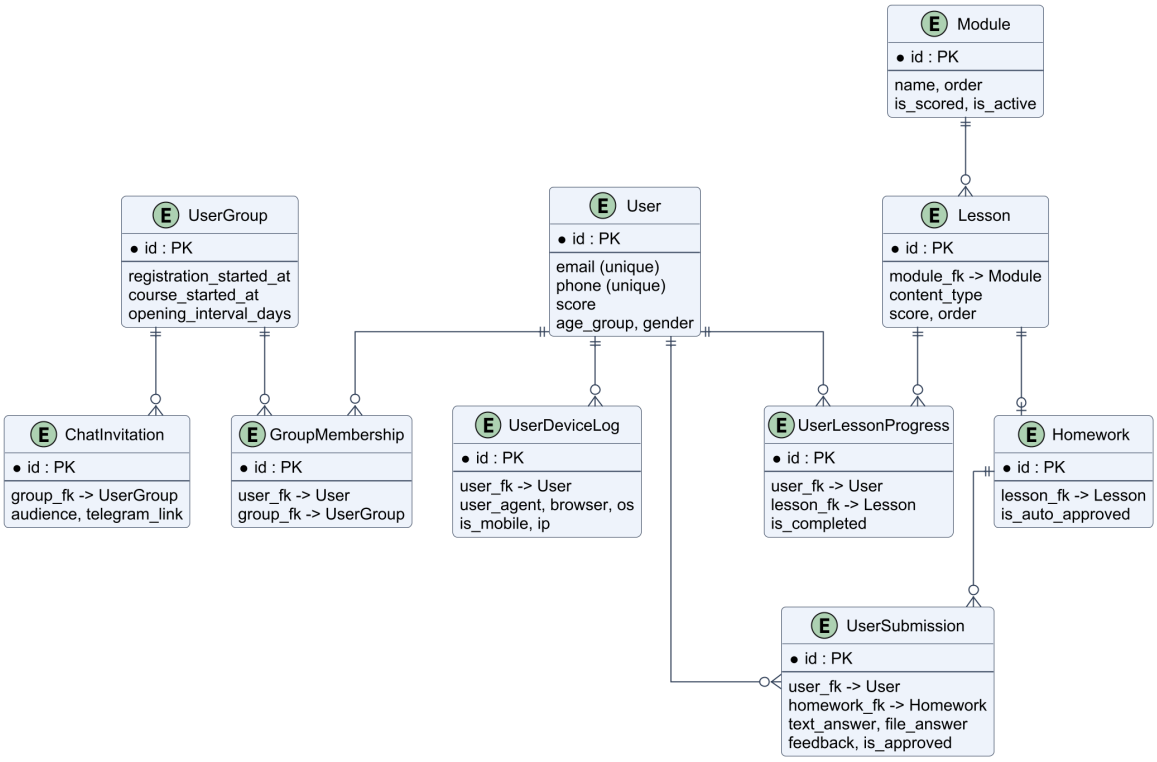


Рисунок 2.4 – ER-діаграма бази даних: користувачі та навчальний контент

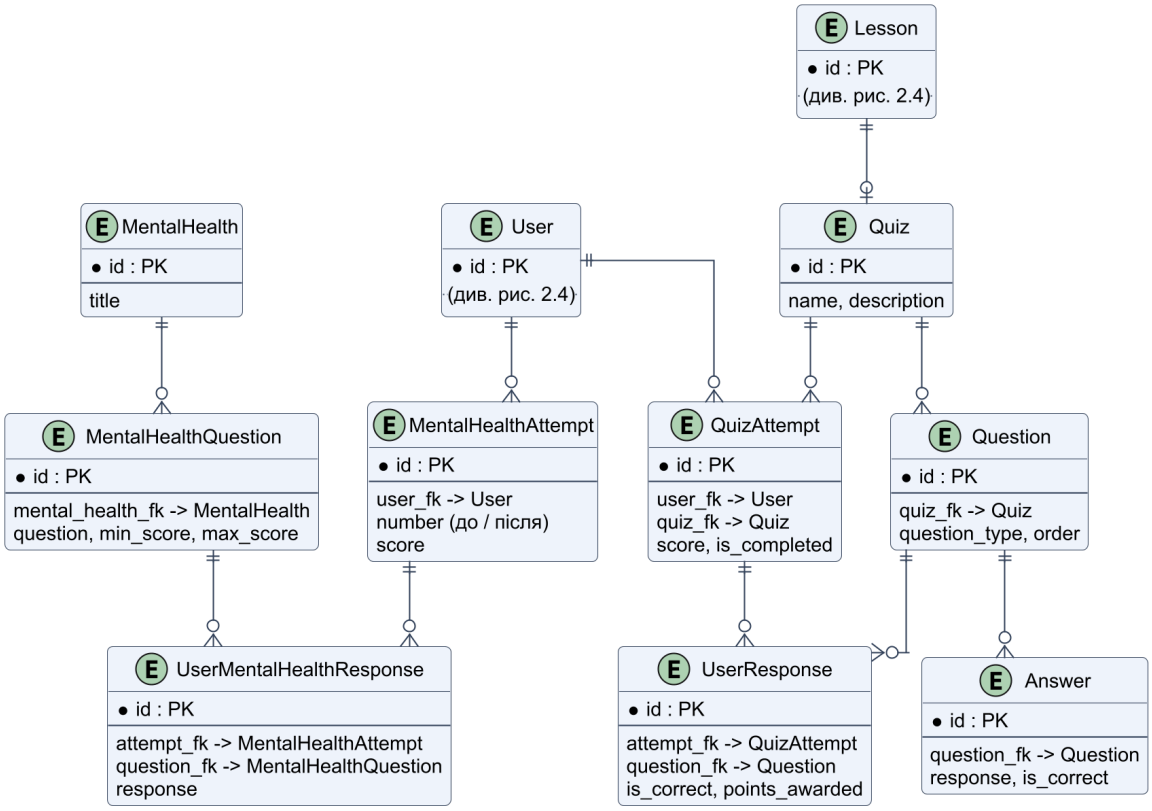


Рисунок 2.5 – ER-діаграма бази даних: оцінювання та самооцінювання знань

На рис. 2.5 доменну модель подано у вигляді UML-діаграми класів, згрупованої за Django-додатками: users (користувачі, групи та членство у групах), modules (модулі, уроки, прогрес проходження), homeworks (завдання та здачі), quizzes (квізи, питання, спроби) і mental_health (вхідне та підсумкове опитування). Зв’язки композиції відображають життєвий цикл підлеглих об’єктів (наприклад, видалення модуля каскадно видаляє його уроки), а асоціації Lesson–Homework та Lesson–Quiz з кратністю 0..1 показують, що урок може містити щонайбільше одне завдання або один квіз залежно від типу контенту. Діаграма доповнює ER-модель поведінковим аспектом: для ключових класів наведено основні методи бізнес-логіки (`clean()`, `save()`, `max_score()`, `finish()`, `get_module_unlock_date()`).

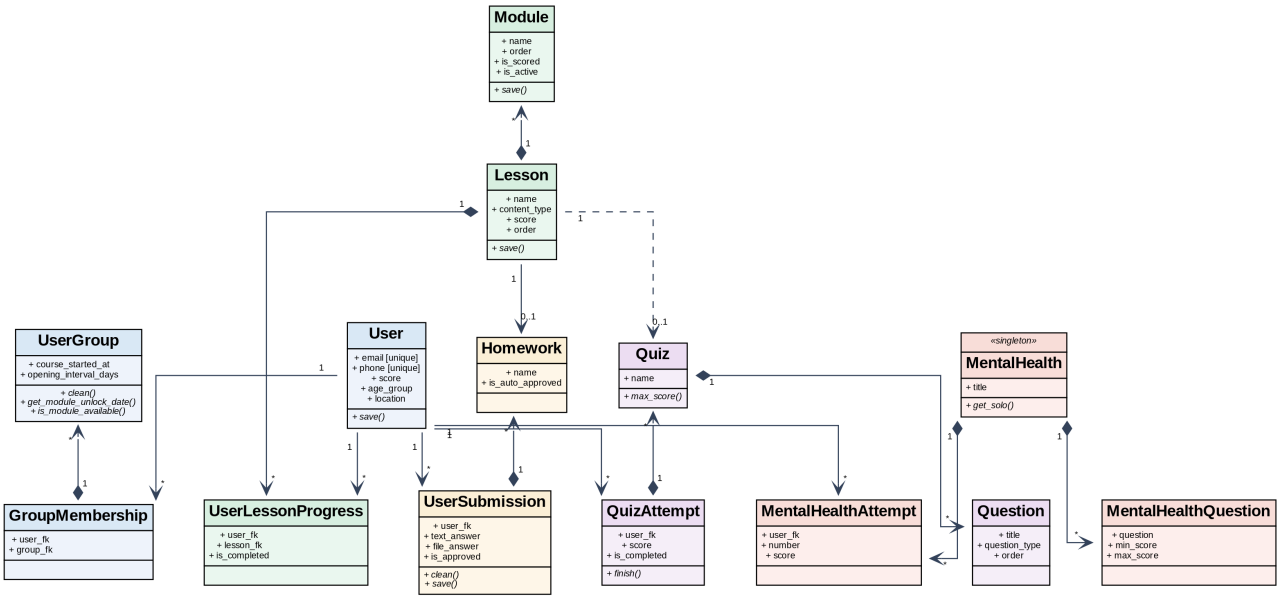


Рисунок 2.6 – UML-діаграма класів доменної моделі системи

Узагальнюючи, основні сутності бази даних та їх призначення наведено у таблиці 2.1.

Таблиця 2.1 – Основні сутності бази даних та їх призначення

Сутність	**Призначення**	**Ключові поля**
`User`	Користувач системи (слухач/адміністратор)	email, phone, score, age_group, gender, interests
`UserGroup`	Група слухачів з керованим розкладом	registration_started_at, course_started_at, opening_interval_days
`GroupMembership`	Зв’язок користувача з групою	user_fk, group_fk
`Module` / `Lesson`	Навчальний модуль і його уроки	module_fk, content_type, score, order
`UserLessonProgress`	Прогрес проходження уроку	user_fk, lesson_fk, is_completed
`Homework` / `UserSubmission`	Завдання та здача слухача	lesson_fk, is_auto_approved, text_answer, file_answer
`Quiz` / `Question` / `QuizAttempt`	Квіз, питання, спроба проходження	quiz_fk, question_type, score
`ChatInvitation`	Запрошення у Telegram-канал	audience, telegram_link
`UserDeviceLog`	Лог сесії пристрою користувача	user_fk, user_agent, ip, created_at
`MentalHealth` / `MentalHealthQuestion` / `MentalHealthAttempt`	Вхідне та підсумкове опитування слухача (самооцінка)	title, number (до/після), score, response

Окремої уваги заслуговує життєвий цикл здачі домашнього завдання. Кожна здача послідовно проходить через визначені стани, а переходи між ними керуються

прапорами is_auto_approved та is_approved і наявністю відгуку куратора. Відповідну діаграму станів моделі `UserSubmission` наведено на рисунку 2.7.

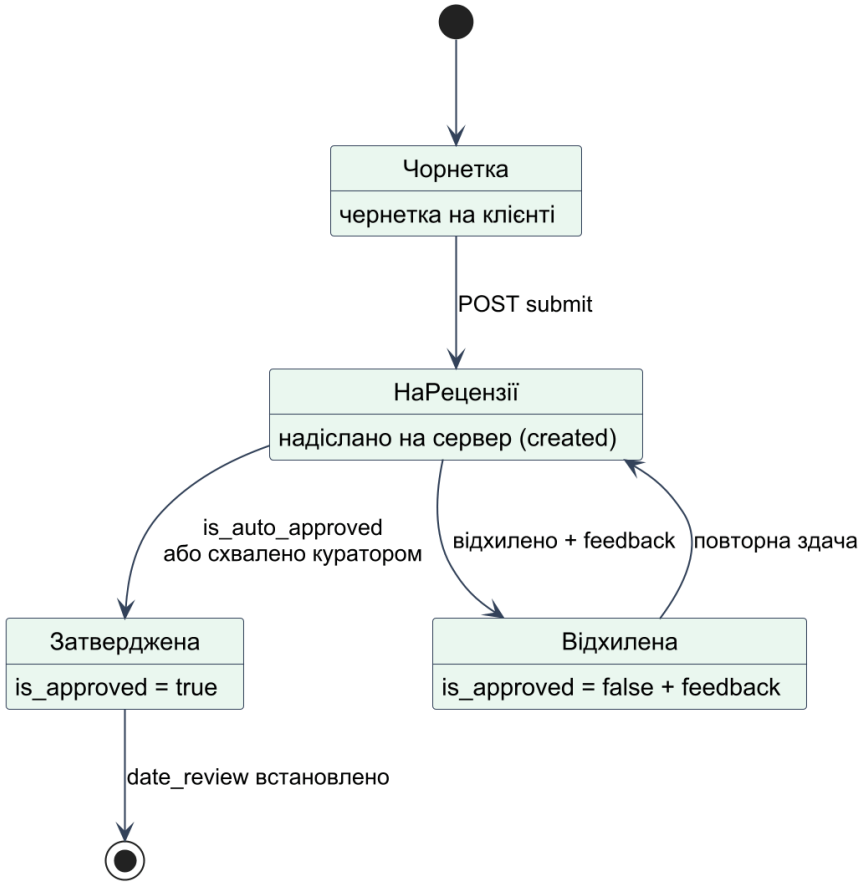


Рисунок 2.7 – Діаграма станів моделі UserSubmission (домашнє завдання)

Діаграма станів моделі `UserSubmission` демонструє повний життєвий цикл здачі домашнього завдання – від чорнового запису до затвердженого або відхиленого результату – і слугує основою для побудови тестових сценаріїв переходів станів у Розділі 3.

2.4 Розробка REST API

Інтерфейс взаємодії клієнтських застосунків із серверною частиною реалізовано як REST API згідно з принципами, описаними у роботах [20, 6]. API будується на базі бібліотеки Django Ninja [21], яка використовує `Pydantic`-схеми для опису вхідних та вихідних даних і автоматично генерує специфікацію `OpenAPI`

3.0. У такий спосіб клієнтська команда (фронтенд, мобільна) має завжди актуальний контракт, доступний через стандартний інтерфейс Swagger UI за адресою `/api/docs/`.

Маршрутизація запитів організована у вигляді центрального реєстру роутерів у файлі `apps/routers`.py. Кожен Django-додаток реєструє власний `NinjaRouter` з префіксом, що відповідає предметній області. Перелік основних маршрутів REST API наведено у таблиці 2.2.

Автентифікація реалізована через бібліотеку django-ninja-jwt, що підтримує стандарт JWT (RFC 7519) [22]. При вході (POST `/users/login`) клієнт отримує пару токенів: access-токен з обмеженим часом дії 15 хвилин та refresh-токен з часом дії 7 діб. Усі захищені ендпоінти декорують класом `JWTAuth`, який автоматично перевіряє валідність access-токена у заголовку Authorization: Bearer <token>. Для виходу з системи refresh-токен потрапляє у чорний список (ninja_jwt.token_blacklist). Послідовність взаємодії при автентифікації наведено на рисунку 2.8.

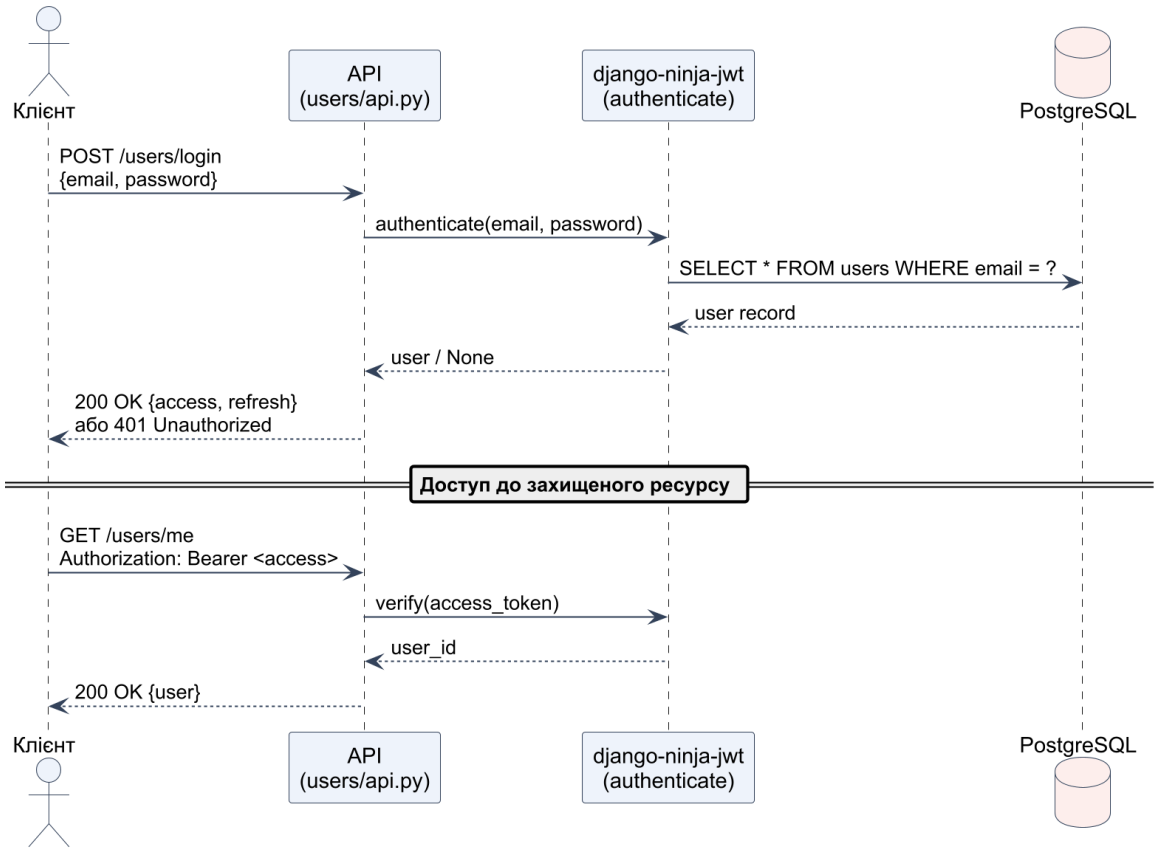


Рисунок 2.8 – Послідовність взаємодії при автентифікації за JWT

Валідація вхідних даних реалізована через `Pydantic`-схеми у файлах schemas.py кожного додатку (наприклад, `UserRegisterSchema`, `ActivateUserSchema`, `UserLoginSchema`). Django Ninja автоматично перевіряє структуру JSON-запиту, типізацію полів, мінімальні та максимальні значення, а у разі невідповідності повертає клієнту HTTP 422 із зрозумілою помилкою у JSON-форматі. Вихідні дані описуються окремими схемами (`UserResponseSchema`, `UserProgressSchema`), що дозволяє приховувати службові поля та повертати лише потрібні для клієнта атрибути. Інтерфейс автоматичної документації Swagger UI показано на рисунку 2.9.

Обробка помилок централізована через окремі виключення у файлах apps/<app>/exceptions.py та глобальний хендлер виключень Django Ninja. Кожне виключення має фіксований HTTP-код та структуроване тіло відповіді, наприклад: `EmailAlreadyExistsError` повертає 409 Conflict, `UserInvalidCredentialError` – 401 Unauthorized, `InvalidActivationTokenError` – 400 Bad Request. Усі неперевірені виключення фіксуються у Sentry разом із трейсбеком та параметрами запиту.

Обмеження частоти запитів (throttling) застосоване до публічних ендпоінтів автентифікації за допомогою класу `AnonRateThrottle`, що дозволяє не більше визначеної кількості запитів від одного IP-адреси за хвилину. Це знижує ризик атак типу brute-force на пару email/password.

Лістинг А.3 у Додатку А наводить приклад ендпоінту реєстрації користувача з валідацією, обробкою помилок та виклику сервісу.

Таблиця 2.2 – Перелік основних маршрутів REST API

Маршрут	**Метод**	**Призначення**
`/api/v1/users/register`	POST	Реєстрація нового користувача, надсилання листа активації
`/api/v1/users/activate`	POST	Активація акаунта за токеном з листа
`/api/v1/users/login`	POST	Отримання пари JWT-токенів
`/api/v1/users/token/refresh`	POST	Оновлення access-токена за refresh

Продовження таблиці 2.2

1	2	3
/api/v1/users/me	GET, PATCH	Перегляд та оновлення власного профілю
/api/v1/users/forgot-password	POST	Запит на скидання пароля
/api/v1/users/progress	GET	Поточний прогрес слухача
/api/v1/modules/	GET	Список модулів, доступних для групи слухача
/api/v1/modules/{id}/lessons	GET	Уроки модуля з прогресом
/api/v1/modules/lessons/{id}/complete	POST	Позначити урок як завершений
/api/v1/quizzes/{id}/attempts	POST	Розпочати спробу квіза, надіслати відповіді
/api/v1/homeworks/{id}/submit	POST	Здача домашнього завдання
/api/v1/users/chats	GET	Чат-запрошення для поточної аудиторії
/api/v1/admin/...	*	Внутрішні адмін-ендпоінти

Рисунок 2.9 – Інтерфейс автоматичної документації API (Swagger UI)

Автоматично згенерована документація OpenAPI слугує формальним контрактом між сервером і клієнтськими застосунками, що дозволяє командам

розробки фронтенду та мобільного додатку працювати паралельно та незалежно один від одного.

2.5 Реалізація бізнес-логіки

Бізнес-логіка системи відокремлена від маршрутів API та доменних моделей у вигляді сервісних класів у файлах apps/<app>/services.py. Такий підхід відповідає принципу одиничної відповідальності та полегшує тестування: ендпоінти API залишаються тонкими (вони тільки перевіряють авторизацію, валідують запит та викликають сервіс), а бізнес-правила інкапсульовано у функціях/класах, які приймають доменні об’єкти та повертають результат.

Найбільш значущим прикладом сервісу є `UserProgressService` (`apps/users/services.py`), що відповідає за підрахунок балу слухача. Сервіс містить методи `get_total_possible_score()`, `get_user_current_score()`, `get_user_current_position()` та `recalculate_user_score()`. Метод `get_user_current_position()` повертає словник з поточним модулем та уроком, на якому перебуває слухач, обчислюючи його як перший за порядком урок, який ще не позначений як завершений у моделі `UserLessonProgress`. Метод `recalculate_user_score()` агрегує бали уроків та квізів через ORM-запит з функціями Sum/Max/Case/When і записує загальний бал у поле `User`.score. Послідовність взаємодії між клієнтом, API-шаром і сервісом при перерахунку балу показана на рисунку 2.10. Повний лістинг цього сервісу наведено у Додатку А (лістинг А.2).

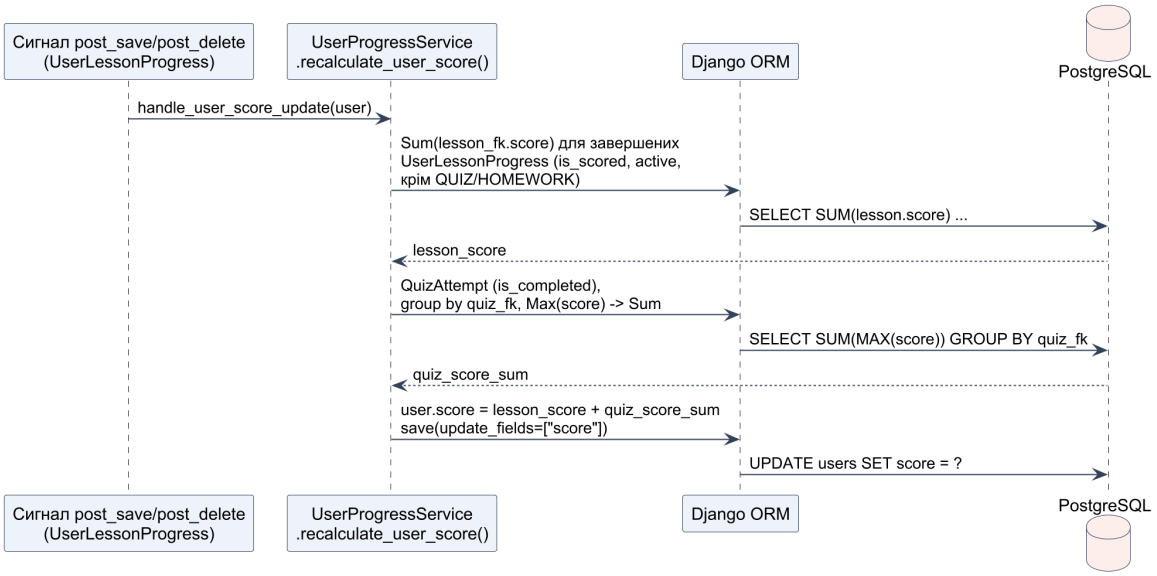


Рисунок 2.10 – Послідовність перерахунку балу слухача (UserProgressService)

Доступність навчального модуля для слухача визначається на рівні моделі `UserGroup`: метод `get_module_unlock_date()` обчислює дату відкриття модуля заданого порядкового номера як course_started_at + (module_order − 1) × opening_interval_days, а метод `is_module_available()` повертає, чи вже настала ця дата. Якщо модуль ще закритий, відповідний ендпоінт повертає помилку через доменне виключення `ModuleClosedError` (`apps/users/utils.py`). Таке рішення дозволяє адміністратору один раз створити курс і потім запускати його для нових груп без копіювання контенту. Навігацію між уроками (попередній та наступний урок з урахуванням уже завершених) інкапсульовано в окремому сервісі `LessonNavigationService` (`apps/modules/services.py`).

Сервіс домашніх завдань реалізує логіку автоматичного або ручного затвердження здач. У моделі `UserSubmission` перевизначено метод `save()`: якщо в моделі `Homework` встановлено прапор is_auto_approved=True, то здача автоматично отримує is_approved=True; при будь-якій зміні поля feedback або is_approved виставляється дата рецензування date_review = timezone.`now()`. У такий спосіб збереження дати рецензування не покладається на код адміністративного інтерфейсу або API, а гарантується на рівні моделі. Послідовність взаємодії при здачі домашнього завдання показано на рисунку 2.11.

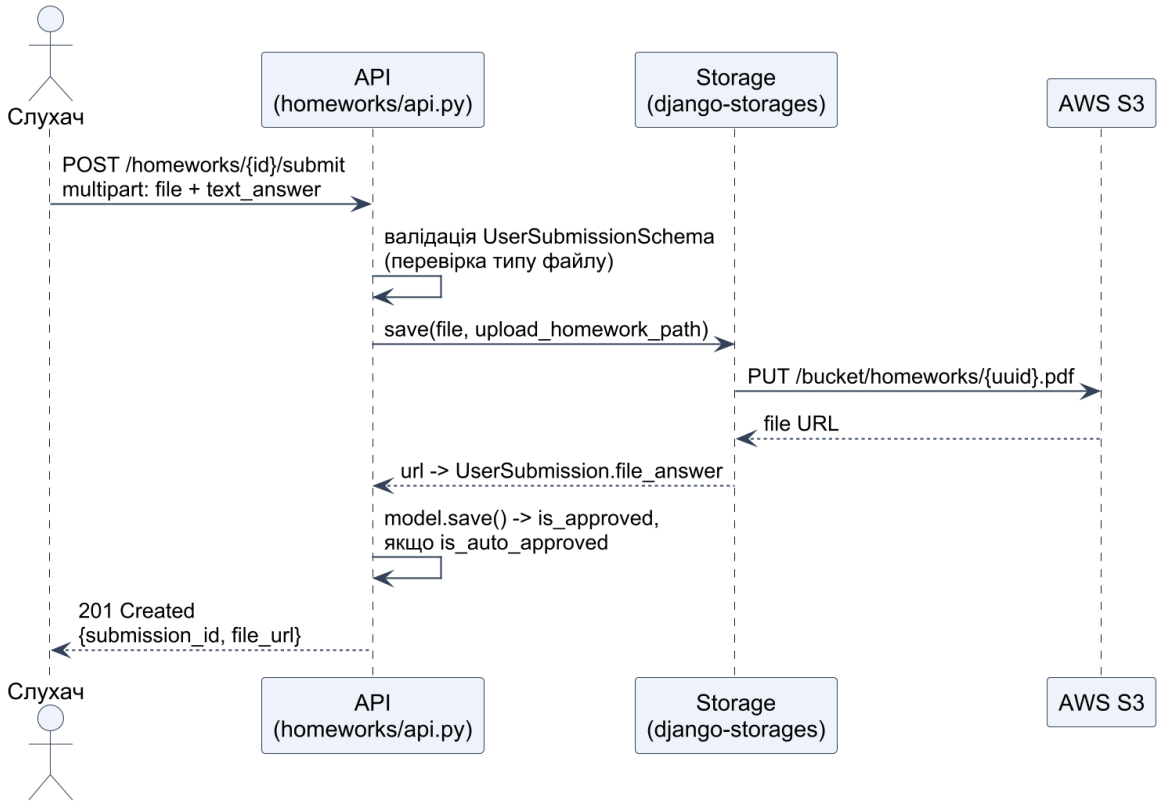


Рисунок 2.11 – Послідовність взаємодії при здачі домашнього завдання

Сигнали Django (signals.py) використовуються для реакції на події у системі. Сигнал post_save моделі `User` викликає функцію `assign_user_to_group()`, яка автоматично прикріплює новостворених користувачів до активної групи `UserGroup` з найближчою датою старту реєстрації. Сигнали post_save та post_delete моделі `UserLessonProgress` викликають обробник `handle_user_score_update()`, який перераховує бал слухача через `UserProgressService.recalculate_user_score()`; оскільки завершення уроку-квіза фіксується саме у моделі `UserLessonProgress`, перерахунок балу після проходження квіза відбувається автоматично. Додаткові обробники сигналів моделей `Module` та `Lesson` упорядковують об’єкти після видалення та перераховують бали при зміні їх активності. У такий спосіб обчислення балу зосереджено в одному місці, а будь-яка зміна стану прогресу автоматично оновлює агреговане поле `User`.score. Реалізацію сигналу автоматичного прикріплення слухача до групи наведено у Додатку А (лістинг А.4). Послідовність взаємодії при проходженні квіза слухачем, що завершується таким перерахунком балу, показано на рисунку 2.12.

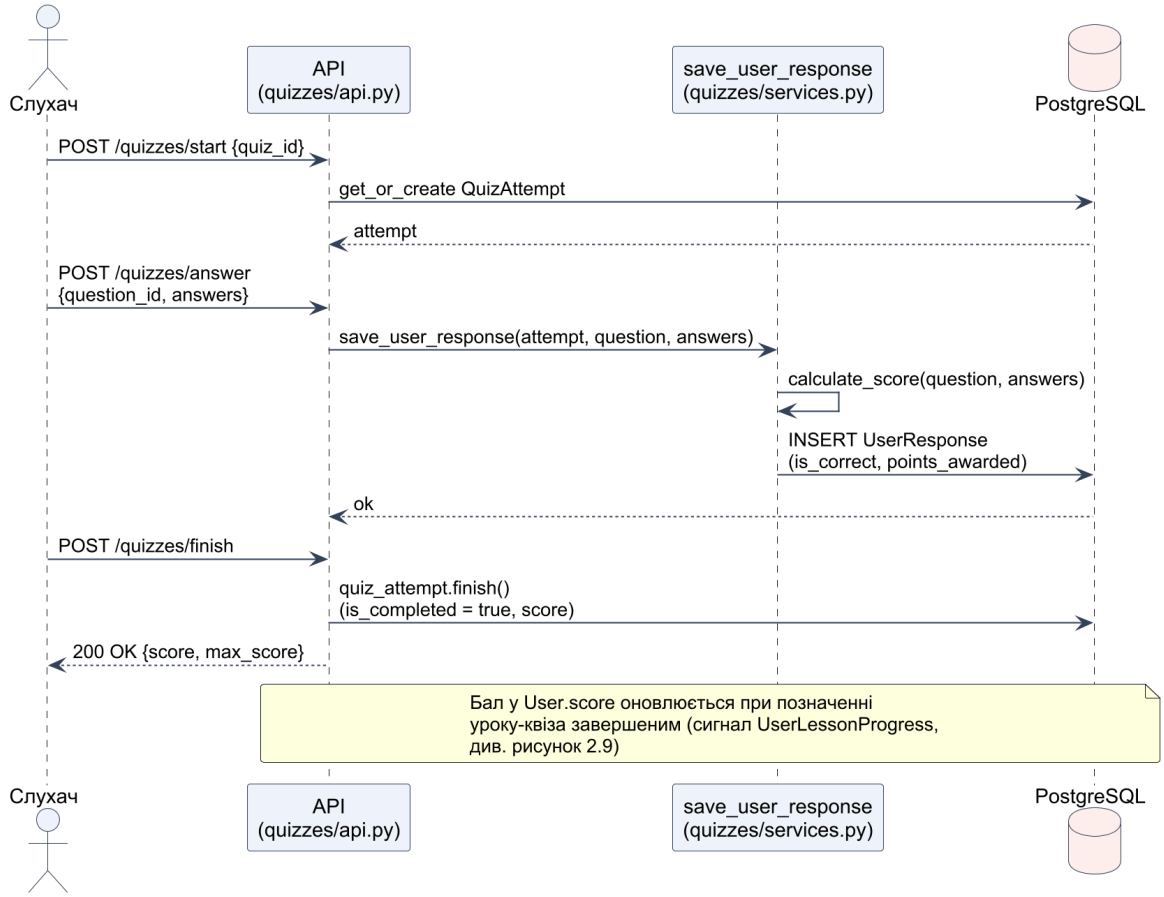


Рисунок 2.12 – Послідовність взаємодії при проходженні квіза

Логіку підсистеми вхідного та підсумкового опитування зосереджено на рівні API-ендпоінтів і сигналів, без окремого сервісного класу. Під час надсилання підсумкової спроби сервер перевіряє наявність вхідної спроби та факт завершення останнього за порядком уроку курсу – інакше повертається відповідь 403 з доменним виключенням (`NoPreviousAnswerException` або `NotCompletedEducationException`). Сигнал `post_save/post_delete` моделі `UserMentalHealthResponse` автоматично перераховує сумарний бал спроби як суму відповідей, тож результат завжди узгоджений незалежно від інтерфейсу, через який внесено зміни.

Кастомні менеджери моделей (`apps/modules/managers.py`) розширюють стандартний QuerySet-API Django. Так, `ActiveManager` повертає лише модулі з is_active=True, а `ActiveLessonManager` – лише уроки з is_active=True. Це дозволяє писати у вьюхах простий запит `Module`.active.`all()` замість `Module`.objects.filter(is_active=True), що, у свою чергу, зменшує ризик “протікання”

неактивних об’єктів у клієнтську відповідь. Узагальнений процес проходження уроку слухачем – від запиту модуля до фіксації прогресу – показано у вигляді діаграми діяльності на рисунку 2.13.

Інтернаціоналізація реалізована стандартним механізмом Django: усі рядки, які можуть бути показані користувачеві, обгорнуті у функції `gettext_lazy()`; переклади зберігаються у форматі .po у директоріях `locale/uk/` та `locale/en/`, компіляція виконується командою django-admin compilemessages. У середовищі розробки додано короткий Makefile-таргет make locale, який автоматизує оновлення .po-файлів.

Завантаження медіа-файлів реалізовано через django-storages та boto3 на S3-сумісне сховище. Файлові поля video_url, audio_url моделі `Lesson` та file_answer моделі `UserSubmission` безпосередньо завантажують файл у бакет, а у базі даних зберігається лише посилання. Розмір та розширення файлу валідуються на рівні FileExtensionValidator з визначеними допустимими списками: відео – mp4, webm, mov; аудіо – mp3, m4a, wav, ogg; документи – pdf, docx, txt. Імена файлів формуються через окрему функцію upload_homework_path, яка додає UUID-частину для уникнення колізій імен.

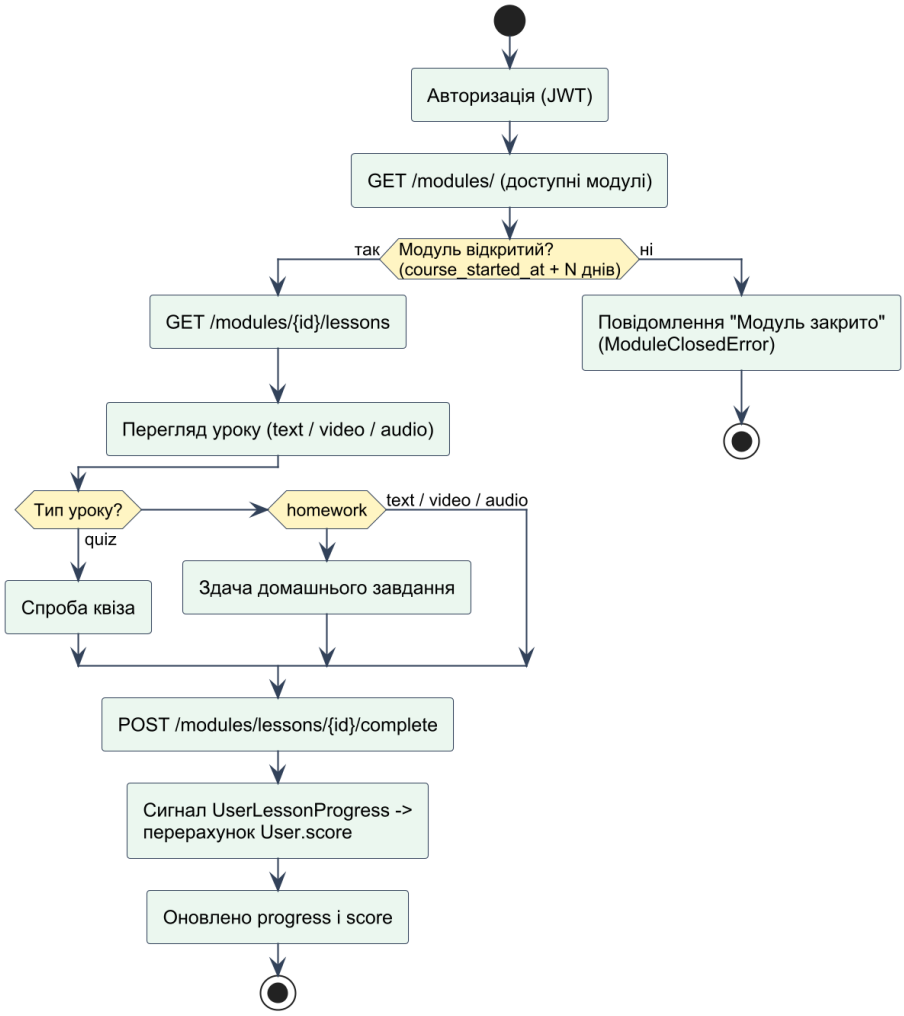


Рисунок 2.13 – Діаграма діяльності процесу проходження уроку слухачем

Наведена діаграма діяльності завершує опис основної бізнес-логіки системи: вона показує реальну послідовність дій слухача та фонові процеси на боці сервера, що в подальшому стає основою для проєктування інтеграційних тестів у Розділі 3.

2.6 Інтерфейс адміністрування та інтеграція з зовнішніми сервісами

Інтерфейс адміністрування побудовано на основі стандартної admin-панелі Django з підключеною темою django-unfold, яка надає сучасний інтерфейс із підтримкою бічної панелі, темної теми, фільтрів та збереження користувацьких налаштувань. Список навчальних модулів в адмін-панелі наведено на рисунку 2.14.

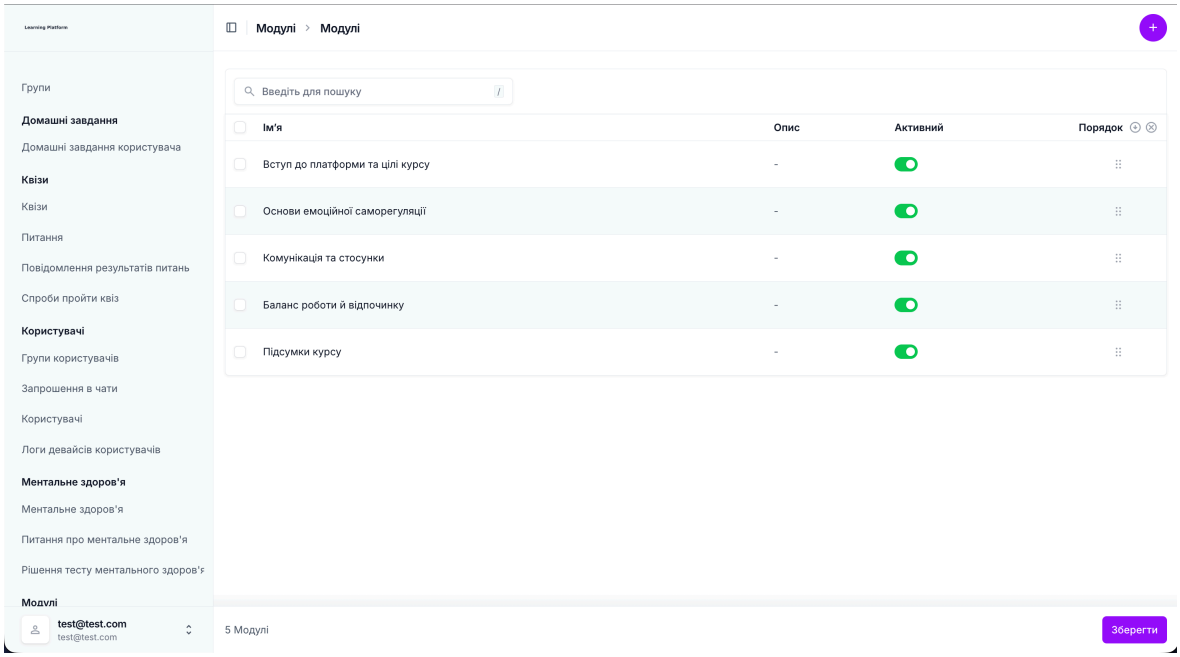


Рисунок 2.14 – Адмін-панель django-unfold: список навчальних модулів

Для редагування HTML-вмісту уроків інтегровано редактор TinyMCE через бібліотеку django-tinymce, що дозволяє адміністраторам форматувати текст, вставляти зображення, посилання та інші елементи без знання HTML. Форму редагування уроку з цим редактором показано на рисунку 2.15.

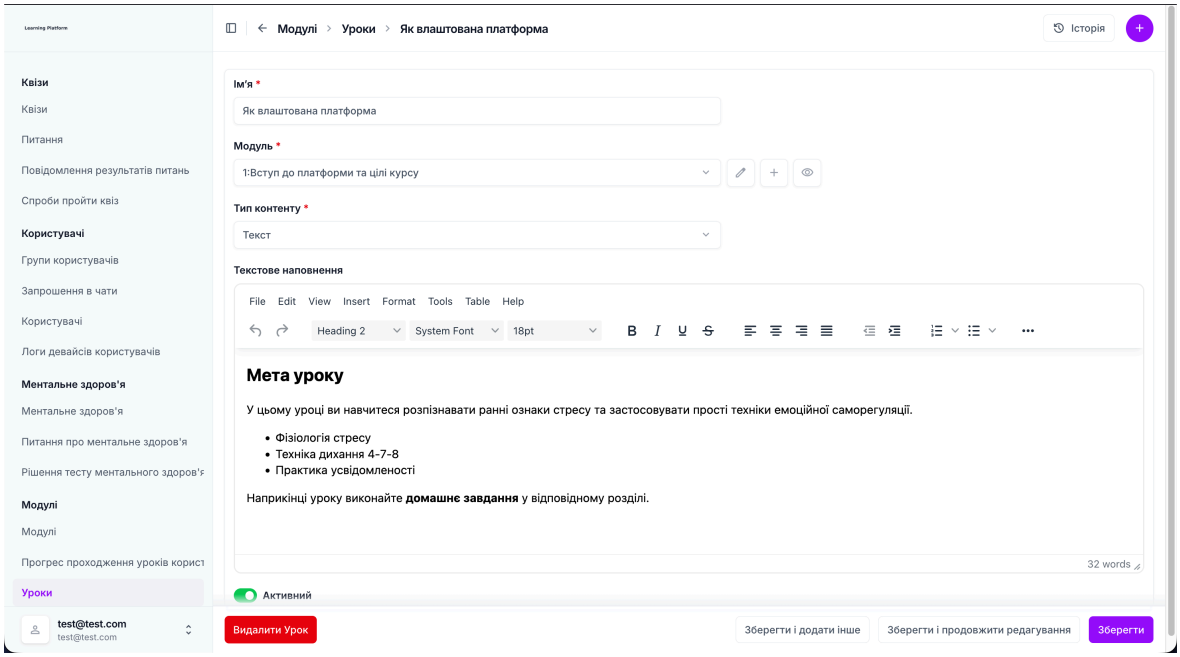


Рисунок 2.15 – Форма редагування уроку з WYSIWYG-редактором TinyMCE

Керування користувачами та групами реалізовано через стандартні Django-ендпоінти адмінки з додатковими фільтрами за віковою групою, статусом та

активністю (рисунок 2.16). Окремий додаток apps.admin_custom містить додаткові ендпоінти, що використовуються адмін-панеллю для оперативного отримання статистики (загальна кількість слухачів, активні групи, відсоток виконаних завдань) та керування дозволами доступу. Усі адмін-маршрути доступні лише користувачам з is_staff=True.

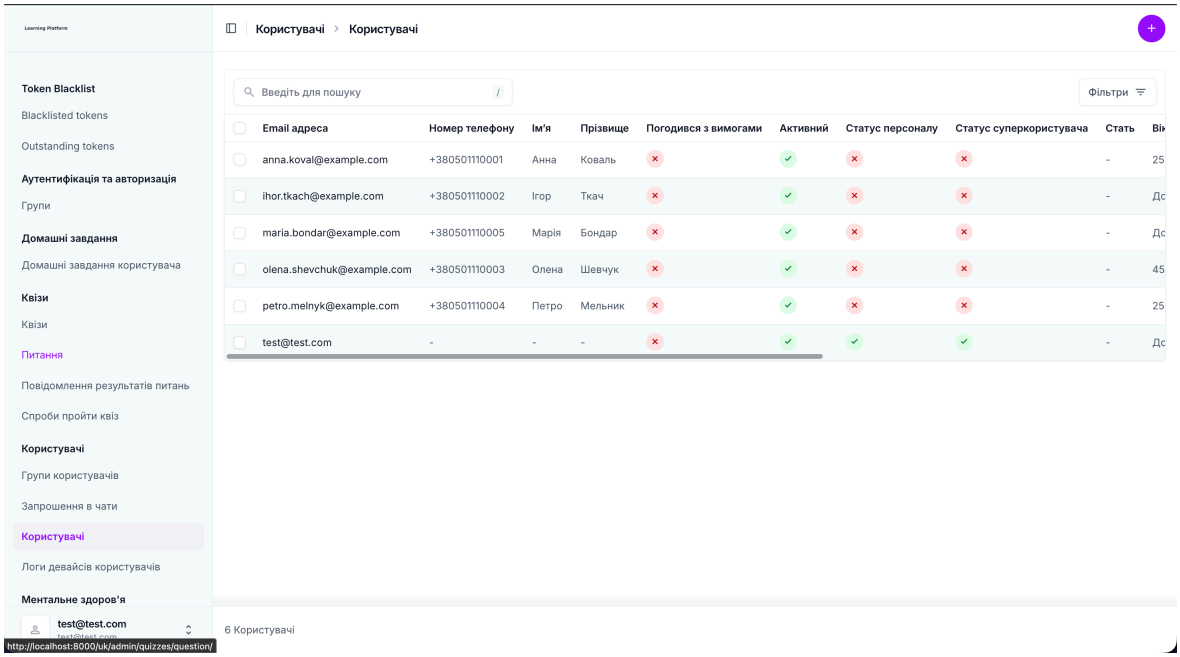


Рисунок 2.16 – Адмін-панель: керування користувачами та групами

Інтеграція з зовнішніми сервісами здійснюється через окремі модулі-помічники. Для email-розсилок використовується django-sendgrid-v5, який підставляється як стандартний `EMAIL_BACKEND` Django. У файлі `apps/users/utils.py` реалізовано функції `send_activation_email()`, `send_reset_password_email()`, `send_chat_invitation_email()`, що формують HTML-листи з шаблонів та надсилають їх через SendGrid. У разі помилки відправки виключення фіксується у Sentry, але автентифікація користувача не блокується.

Для моніторингу помилок підключено Sentry SDK з інтеграціями DjangoIntegration та LoguruIntegration. У файлі `config/logging.py` налаштовано loguru з ротацією файлів `app.log` та автоматичною передачею логів рівня ERROR і вище до Sentry. У середовищі розробки Sentry-DSN залишається порожнім, тож трейси не надсилаються; у продакшні – задається через змінну середовища

`SENTRY_DSN`. Окремий `config/middleware.py` додає до кожного запиту унікальний ідентифікатор `RequestIDMiddleware`, який потрапляє у логи та у Sentry-event і дозволяє швидко знайти усі логи, пов'язані з конкретним запитом.

Конфігурація різних середовищ задається через файл `.env` (зразок – у `.env.template`, який зберігається у репозиторії) та підвантажується через python-dotenv. Усі секретні значення (`SECRET_KEY`, `SENTRY_DSN`, AWS-ключі, SendGrid API-ключ) ніколи не комітяться у репозиторій. Налаштування PostgreSQL, Redis, `ALLOWED_HOSTS` та `CSRF_TRUSTED_ORIGINS` читаються з `.env` та переключаються між дев- та прод-режимами лише значенням змінної `DEBUG`.

Висновки до другого розділу

У другому розділі спроєктовано та реалізовано серверну частину системи: побудовано багаторівневу архітектуру з розмежуванням відповідальностей, спроєктовано схему бази даних з десяти основних сутностей, реалізовано REST API на основі Django Ninja з JWT-автентифікацією, виокремлено бізнес-логіку у сервісні класи та сигнали й підключено інтеграції з зовнішніми сервісами PostgreSQL, Redis, S3, SendGrid і Sentry. Дотримання шарової архітектури дозволяє у наступному розділі ізольовано тестувати кожен компонент системи.

Обрані архітектурні та інфраструктурні рішення узгоджуються із сучасними підходами до побудови надійних і захищених хмарних інформаційних систем [23] та високопродуктивних інтелектуальних інформаційних технологій [24], що підтверджує обґрунтованість прийнятих проєктних рішень.

3 ТЕСТУВАННЯ, ВПРОВАДЖЕННЯ ТА ПІДТРИМКА

Тестування є невід'ємною складовою процесу розробки програмного забезпечення, що гарантує відповідність системи функціональним вимогам та запобігає внесенню регресій при подальших змінах коду. У цьому розділі описано стратегію тестування, визначено види та план тестування, описано розробку тестових сценаріїв, проаналізовано результати їх виконання, розглянуто верифікацію програмної системи та процедуру розгортання й підтримки.

3.1 Стратегія тестування

Стратегія тестування системи опирається на класичні принципи піраміди тестування, описані К. Беком [25] та підтримані рекомендаціями SWEBOK [5]. Згідно з цим підходом, основну частку покриття становлять швидкі модульні тести, поверх них розташовується менша кількість інтеграційних тестів, а на верхівці – невелика частина ручних або "димових" сценаріїв, які перевіряють критичні шляхи системи у цілому. Узагальнена структура піраміди тестування для проєкту наведена на рисунку 3.1.

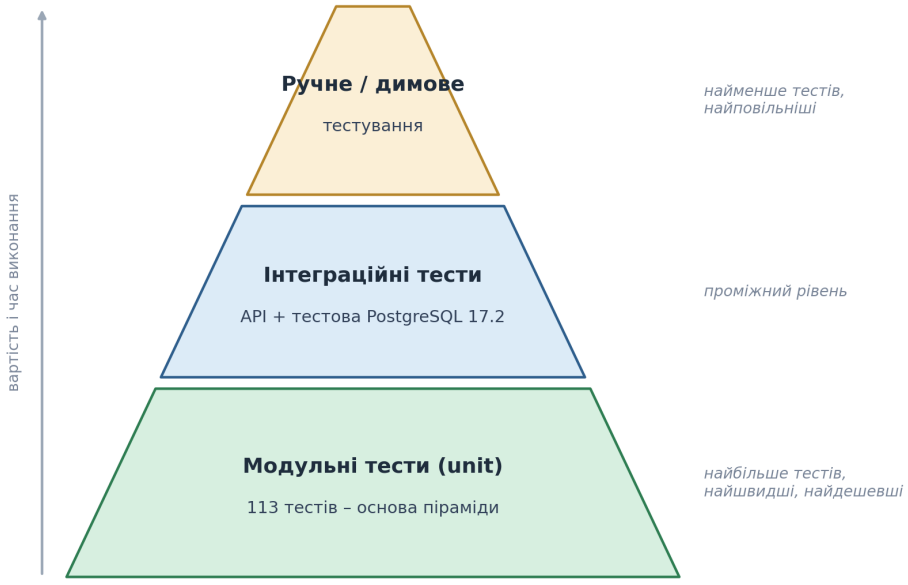


Рисунок 3.1 – Піраміда тестування проєкту

Основні принципи прийнятої стратегії тестування:

– мінімальне покриття коду тестами – не менше 80 %, що зафіксовано у конфігурації `pyproject.toml` ключем --cov-fail-under=80. У разі зниження покриття CI-конвеєр повертає помилку, а PR не може бути злитий у головну гілку;

– ізоляція тестів – кожен тест повинен бути незалежним від стану, що залишився від попереднього тесту; для цього використовуються транзакції pytest-django, які автоматично відкочуються після кожного тесту;

– підготовка тестових даних – спільні тестові об'єкти створюються у методі `setUp()` (Django `TestCase`) або через pytest-фікстури, що зменшує дублювання коду й робить тести читабельними;

– використання окремої тестової БД PostgreSQL з повторним використанням (--reuse-db), що значно прискорює запуск тестів локально;

– автоматичний запуск тестів у CI/CD конвеєрі на кожному push у репозиторій.

Інструменти тестування. Основним фреймворком обрано pytest 8.4 [26] разом з розширенням pytest-django 4.11, що додає підтримку Django-специфічних компонентів – тестового HTTP-клієнта, фікстури django_db, інтеграцію з settings. Покриття коду вимірюється плагіном pytest-cov, який формує консольний звіт з відсотком покриття та переліком непокритих рядків. Конфігурація тестів повністю описана у файлі `pyproject.toml` (Додаток А, лістинг А.7) у секції [tool.pytest.ini_options] з рядком addopts = "--ds=config.settings --reuse-db --cov=apps --cov-report=term-missing --cov-fail-under=80".

3.2 Види та план тестування

Відповідно до прийнятої стратегії у проєкті застосовуються такі види тестування. Модульне тестування перевіряє ізольовані одиниці коду: методи моделей `clean()` і `save()`, властивості, кастомні менеджери та сервісні класи. Інтеграційне тестування перевіряє взаємодію компонентів: API-ендпоінти викликаються через тестовий HTTP-клієнт pytest-django із реальною тестовою

базою даних, що дозволяє перевірити повний шлях запиту – від маршрутизації та автентифікації до ORM-запитів і сигналів. Контрактне тестування забезпечується `Pydantic`-схемами: невалідні запити мають відхилятися з кодом 422, а структура відповідей – відповідати специфікації `OpenAPI`. Нарешті, ручне димове тестування покриває наскрізні користувацькі сценарії, які складно автоматизувати у межах серверної частини.

Оскільки предметом роботи є серверна частина системи, функціональне тестування зосереджено на REST API. Окремо перевіряються аспекти безпеки: коректність кодів 401/403 для неавтентифікованих і неавторизованих запитів, перевірка прав доступу до чужих об'єктів (наприклад, здач інших слухачів), обмеження частоти запитів до публічних ендпоінтів автентифікації.

План тестування передбачає такі етапи та середовища виконання. Локально розробник запускає тестовий набір перед кожним комітом командою pytest; повторне використання тестової БД (--reuse-db) скорочує час прогону. На сервері безперервної інтеграції повний набір виконується автоматично на кожен push або pull-request у середовищі з сервісним контейнером PostgreSQL 17.2. Критеріями успішного проходження є: усі тести завершуються успішно, покриття коду не нижче 80 % (інакше прогін завершується помилкою через --cov-fail-under=80), статичний аналізатор не повертає зауважень.

Узагальнений план тестування за рівнями системи наведено у таблиці 3.1.

Таблиця 3.1 – План тестування за рівнями системи

Рівень	**Об'єкт перевірки**	**Інструменти**	**Критерій проходження**
Модульний	Методи моделей, сервіси, менеджери	pytest, setUp / @pytest.fixture	Усі тести успішні
Інтеграційний	API-ендпоінти, сигнали, middleware	Тестовий HTTP-клієнт, тестова БД	Коди відповідей та дані збігаються з очікуваними
Контрактний	Схеми запитів і відповідей API	`Pydantic`, `OpenAPI` 3.0	422 на невалідні дані; відповіді за схемою
Статичний аналіз	Стиль та якість коду	ruff, ruff-format, pre-commit	Відсутність зауважень
Димовий (ручний)	Наскрізні сценарії користувача	Браузер, httpie, Swagger UI	Сценарії виконуються без помилок

Такий план відповідає прийнятій ітеративній моделі розробки: кожна ітерація завершується повним прогоном автоматизованих рівнів, а димове тестування виконується перед розгортанням на цільове середовище.

3.3 Розробка тестових сценаріїв

Тестові сценарії проєктувалися на основі артефактів Розділу 2. Діаграма станів моделі `UserSubmission` (рис. 2.6) задає мінімальний набір переходів, які слід покрити тестами: створення чорнетки, надсилання на рецензію, затвердження і відхилення куратором, автоматичне затвердження при is_auto_approved=True. Діаграма діяльності проходження уроку (рис. 2.12) визначає кроки наскрізного сценарію – від запиту списку модулів до фіксації прогресу та перерахунку балу. Розроблені сценарії співвіднесено з функціональними вимогами підрозділу 1.3, що забезпечує перевірюваність кожної вимоги.

Тести організовані у відповідних файлах `tests.py` кожного Django-додатку. Перевіряються три рівні системи: моделі (зокрема, методи `clean()`, `save()` та властивості); сервіси (бізнес-логіка, виокремлена у services.py); ендпоінти API (через HTTP-клієнт pytest-django, який надсилає запити до Ninja-роутерів та перевіряє відповіді). Поглиблений приклад тестового класу для перевірки API наведено у Додатку А (лістинг А.5).

Тестову інфраструктуру побудовано на двох взаємодоповнювальних підходах. Для додатків users, modules, homeworks, quizzes та admin_custom застосовано стандартний Django `TestCase`: спільні тестові об'єкти (користувачі, групи, модулі, уроки) готуються у методі `setUp()` кожного тестового класу, який виконується перед кожним тестом і гарантує ізольований стан бази даних. Підсистему mental_health протестовано у стилі pytest із застосуванням фікстур (`@pytest.fixture`), що повертають готові доменні об'єкти – анкету, користувача та спробу проходження – для повторного використання в кількох тестах.

Граничні, позитивні та негативні сценарії перевіряються окремими тестовими методами у межах відповідних тестових класів: для функціональних

вимог передбачено як перевірку коректної поведінки, так і реакцію на невалідні дані, порушення обмежень цілісності та відсутність прав доступу (повернення кодів 401/403/422).

Взаємодію із зовнішніми сервісами у тестах ізольовано, щоб результати не залежали від мережі. Для цього використовується стандартна бібліотека `unittest.mock`: менеджер контексту `patch` підміняє функцію надсилання листа (наприклад, `send_activation_email`), а тест перевіряє сам факт і параметри її виклику без реального звернення до SendGrid.

Фрагмент тестового коду одного з найскладніших модулів – `UserProgressService` – наведено на рисунку 3.2.

Рисунок 3.2 – Фрагмент тестів сервісу UserProgressService

Приклад pytest-фікстур, що готують спільні тестові дані для повторного використання, наведено на рисунку 3.3.

apps > mental_health > tests.py > ...

```python
@pytest.fixture
def survey():
    cache.clear()
    return MentalHealth.objects.create(
        title="Самооцінювання рівня знань",
        additional_content="<p>Оцініть свій рівень за шкалою 0–5</p>",
    )


@pytest.fixture
def user():
    u = User.objects.create_user(
        email="learner@example.com",
        password="Str0ng!Pass",
        first_name="Olha",
        last_name="Petrenko",
        phone="+380501112233",
    )
    u.has_approved_requirements = True
    u.save(update_fields=["has_approved_requirements"])
    return u


@pytest.fixture
def auth(user):
    token = str(RefreshToken.for_user(user).access_token)
    return {"HTTP_AUTHORIZATION": f"Bearer {token}"}


def _post(client, url, payload, headers):
    return client.post(url, data=json.dumps(payload),
                       content_type="application/json", **headers)
```

Рисунок 3.3 – Приклад pytest-фікстур для підготовки тестових даних

Тести модуля users охоплюють: реєстрацію нового користувача з валідними та невалідними даними; перевірку унікальності email та телефону; активацію акаунта за токеном (включно з невалідним і простроченим токеном); вхід з невірними обліковими даними (повернення 401); вхід для неактивного користувача (повернення помилки `UserNotActiveError`); оновлення профілю; зміну пароля; відновлення пароля через email; перевірку прав доступу при PATCH-запитах.

Тести модуля modules перевіряють: створення та редагування навчальних модулів адміністратором; коректну роботу властивості is_active та менеджера `Module`.active; правильність обчислення дати відкриття модуля методом `UserGroup.get_module_unlock_date()` для груп з різними значеннями opening_interval_days; коректність послідовності уроків у відповіді API; перевірку прав доступу (звичайний слухач не може створити модуль).

Тести модуля homeworks включають: створення завдань адміністратором; перевірку, що клієнт може здати завдання тільки в режимі або тексту, або файлу, але не порожньої здачі (валідується методом `clean()`); коректне виставлення date_review при оновленні поля feedback; автоматичне затвердження здачі при is_auto_approved=True; неможливість здати завдання після дедлайну (якщо це передбачено бізнес-правилом); неможливість одного слухача побачити здачу іншого.

Тести модуля quizzes перевіряють: створення квіза з питаннями різних типів; обчислення max_score квіза як суми балів за всі питання; підрахунок індивідуального балу слухача після проходження квіза; повернення коректних шаблонних повідомлень `QuestionResultMessageTemplate`; коректну роботу обмеження повторної спроби.

Тести модуля mental_health перевіряють: автоматичне впорядкування питань анкети та їх прив'язку до сінглтон-моделі; обмеження однієї спроби кожного типу (до початку та після завершення курсу) на слухача; автоматичний перерахунок балу спроби сигналом при збереженні та видаленні відповідей; коректну роботу ендпоінтів отримання анкети й надсилання відповідей; блокування підсумкової спроби без вхідної або без завершення курсу (відповідь 403).

Окрему групу тестів становлять інтеграційні сценарії, що перевіряють взаємодію кількох модулів: наприклад, проходження уроку спричиняє автоматичне перерахування балу слухача через сервіс `UserProgressService`; здача домашнього завдання спричиняє виставлення дати рев'ю при оновленні поля feedback; перевірка коректної роботи middleware `DeviceLogMiddleware` (фіксація запису у `UserDeviceLog` при автентифікованому запиті).

Обсяг тестового набору та покриття за модулями наведено у таблиці 3.2.

Таблиця 3.2 – Обсяг тестового набору та покриття за модулями

Модуль	**Файлів коду**	**Тестів**	**Покриття коду, %**
apps.users	11	52	85
apps.modules	11	20	82
apps.homeworks	9	11	89
apps.quizzes	9	12	89

Продовження таблиці 3.2

1	2	3	4
apps.admin_custom	4	2	67
apps.mental_health	8	16	97
Усього	52	113	86

Чисельні значення у стовпці “Покриття коду” отримано з консольного звіту pytest --cov, який виводиться в кінці кожного запуску. Конкретні рядки, які не покриті тестами, переважно належать до обробників граничних випадків інтеграції з зовнішніми сервісами (помилки мережі при відправці email через SendGrid, помилки звернення до Sentry) – їх перевірка вимагає окремих заглушок (mocks) і запланована як подальший розвиток.

3.4 Аналіз результатів тестування

За результатами запуску повного тестового набору всі 113 тестів виконуються успішно. Загальне покриття коду модулів проєкту становить 85,5 %, що перевищує встановлений у вимогах поріг 80 %. Зведений звіт pytest --cov генерується у форматі term-missing, що додатково виводить перелік рядків коду, не покритих тестами (рисунок 3.4); ці рядки використовуються як орієнтир для написання нових тестів у наступних ітераціях.

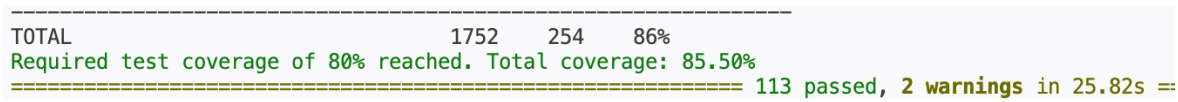
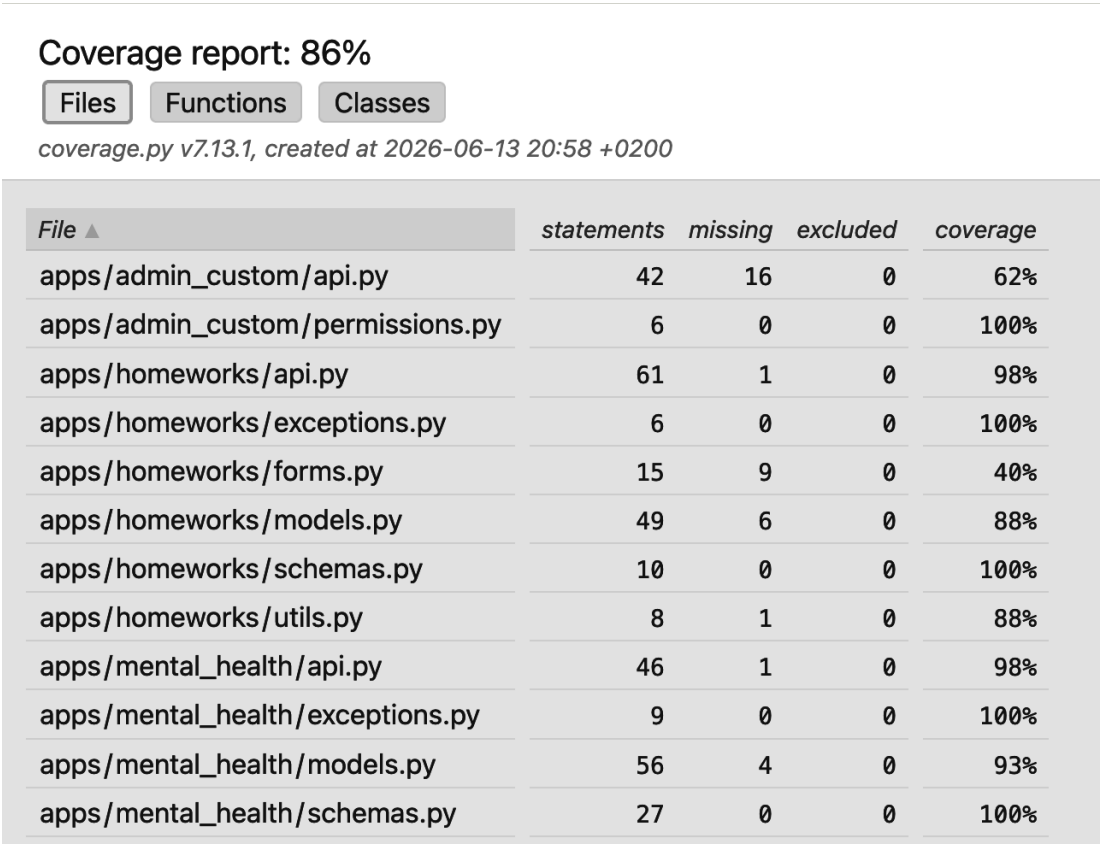


Рисунок 3.4 – Вивід звіту pytest --cov у терміналі

Окрім текстового звіту, плагін pytest-cov може формувати інтерактивний HTML-звіт (`htmlcov/index.html`), де для кожного файлу проєкту візуально підсвічуються рядки, які виконано тестами (зелений) і непокриті (червоний). Приклад такого звіту наведено на рисунку 3.5.



Coverage report: 86%

Files | Functions | Classes

coverage.py v7.13.1, created at 2026-06-13 20:58 +0200

File ▲	statements	missing	excluded	coverage
apps/admin_custom/api.py	42	16	0	62%
apps/admin_custom/permissions.py	6	0	0	100%
apps/homeworks/api.py	61	1	0	98%
apps/homeworks/exceptions.py	6	0	0	100%
apps/homeworks/forms.py	15	9	0	40%
apps/homeworks/models.py	49	6	0	88%
apps/homeworks/schemas.py	10	0	0	100%
apps/homeworks/utils.py	8	1	0	88%
apps/mental_health/api.py	46	1	0	98%
apps/mental_health/exceptions.py	9	0	0	100%
apps/mental_health/models.py	56	4	0	93%
apps/mental_health/schemas.py	27	0	0	100%

Рисунок 3.5 – HTML-звіт покриття коду (htmlcov/index.html)

У ході розробки тестовий набір зафіксував низку дефектів ще до потрапляння коду у головну гілку. Помилки можна умовно поділити на кілька категорій за рівнем коду, у якому вони виявлені. Категоризація дефектів та орієнтовна кількість зафіксованих кейсів наведена у таблиці 3.3.

Таблиця 3.3 – Категоризація дефектів, виявлених тестами

Категорія дефектів	**Рівень коду**	**Орієнт. кількість**	**Приклад**
Валідація моделей	models.py / `clean()`	12	Порожня здача без тексту і файлу проходила `save()`
Бізнес-логіка	services.py	9	Невірний розрахунок балу при повторній спробі квіза
API: HTTP-коди	api.py / Ninja	7	Замість 401 повертався 500 при невалідному JWT
API: контракт	schemas.py	5	У відповіді бракувало поля current_module
Сигнали / прав доступу	signals.py / permissions	4	`GroupMembership` не створювалося для нового користувача
Інтеграційні сценарії	tests інтеграційні	3	Дата відкриття модуля рахувалась без часової зони
Усього	–	40	–

Аналіз звіту покриття показав, що найвище покриття досягнуто у модулях з чітко вираженою бізнес-логікою (users, modules), що пов’язано з достатньою кількістю edge-кейсів у відповідних тестах. Модуль admin_custom має мінімальне покриття 80 %, що пояснюється меншою кількістю гілок умовної логіки і, відповідно, меншою потребою у спеціальних граничних тестах.

Виконання повного тестового набору у середовищі GitHub Actions на стандартному ubuntu-latest runner з PostgreSQL 17 у Docker-сервісі займає у середньому 2 хв 40 с. Це прийнятний час для щоденної розробки і дозволяє виконувати локальний прогін тестів перед кожним коміт-натисканням.

Окремо проведено ручне “димове” тестування системи у браузері та через утиліту httpie. Перевірено сценарії: реєстрація нового користувача, активація, вхід, перегляд списку доступних модулів, проходження першого уроку, здача домашнього завдання, проходження квіза та перегляд персонального прогресу. Усі ці сценарії виконано успішно у локальному Docker-середовищі та на стейджинг-сервері.

Інтерактивний HTML-звіт покриття використовується розробниками для оперативного виявлення непокритих гілок коду в окремих модулях і становить інструментальну основу для подальшого розширення тестового набору у наступних ітераціях розробки.

3.5 Верифікація програмної системи

Поряд із тестуванням (валідацією поведінки системи) у проєкті систематично застосовується верифікація – перевірка відповідності робочих продуктів встановленим специфікаціям і правилам, згідно з трактуванням SWEBOK [5]. Верифікація реалізована як поєднання автоматизованих перевірок на кількох рівнях: стилю та якості коду, повідомлень комітів, контрактів API і схеми бази даних.

Статична верифікація коду виконується аналізатором ruff, конфігурація якого зберігається у `pyproject.toml`: обмеження довжини рядка 120 символів,

автоматичне впорядкування імпортів (правило I), виключення каталогів міграцій з перевірки. Доповнює його форматер ruff-format, який гарантує єдиний стиль форматування. Перед кожним комітом локально запускаються pre-commit-хуки: видалення прикінцевих пробілів, перевірка коректності YAML-, JSON- та XML-файлів, заборона додавання великих файлів, а також повторний прогін ruff. Окремий хук commitizen верифікує формат повідомлень комітів за конвенцією Conventional Commits, на основі якої автоматично ведеться файл змін CHANGELOG.md та семантична версія проєкту.

Верифікація відповідності вимогам забезпечується трасуванням: кожній функціональній вимозі підрозділу 1.3 відповідають тестові сценарії підрозділу 3.3 або кроки димового тестування. Контракт REST API верифікується автоматично на двох рівнях: `Pydantic`-схеми відхиляють структурно невалідні запити з кодом 422 ще до виконання бізнес-логіки, а згенерована специфікація `OpenAPI` 3.0 слугує еталонним контрактом для клієнтських команд. Узгодженість схеми бази даних з доменними моделями верифікують Django-міграції: кожна зміна моделей фіксується у версійованому файлі міграції, а розбіжність між моделями та схемою виявляється при запуску системи.

Формальним критерієм верифікації якості тестового набору є поріг покриття коду: ключ --cov-fail-under=80 у конфігурації pytest перетворює зниження покриття нижче 80 % на помилку збірки. У поєднанні з обов'язковим проходженням статичного аналізу та повного тестового набору в CI це гарантує, що у головну гілку репозиторію потрапляє лише код, який відповідає встановленим специфікаціям і правилам якості.

3.6 Розгортання та підтримка системи

Розгортання системи реалізовано на основі Docker та Docker Compose. У репозиторії проєкту є два конфігураційні файли: `docker-compose.dev.yaml` – для локальної розробки з гарячим перезавантаженням коду та докер-контейнером PostgreSQL/Redis; `docker-compose.yaml` – для продакшну, де ці сервіси можуть

бути замінені керованими хмарними варіантами. Скрипт `entrypoint.sh` виконує запуск Django-міграцій, збір статичних файлів та запуск Gunicorn з кількістю воркерів, що дорівнює кількості CPU.

Конвеєр безперервної інтеграції побудовано на GitHub Actions (файл `.github/workflows/ci.yml`). Загальний потік етапів зображений на рисунку 3.6. На кожному push або pull-request у гілки main і develop виконуються послідовні етапи: (1) встановлення залежностей через uv за зафіксованим файлом `uv.lock`; (2) статичний аналіз коду ruff; (3) повний тестовий набір pytest з контролем покриття, що виконується проти сервісного контейнера PostgreSQL 17.2. Окремий workflow (`deploy.yml`) при push у відповідну гілку розгортає систему через SSH: на цільовому сервері оновлюється код з репозиторію та перезбираються контейнери командою docker compose up --build. Завдяки такій схемі неуспіх будь-якого етапу негайно фіксується і не дозволяє неперевіреному коду потрапити у продуктивне середовище.

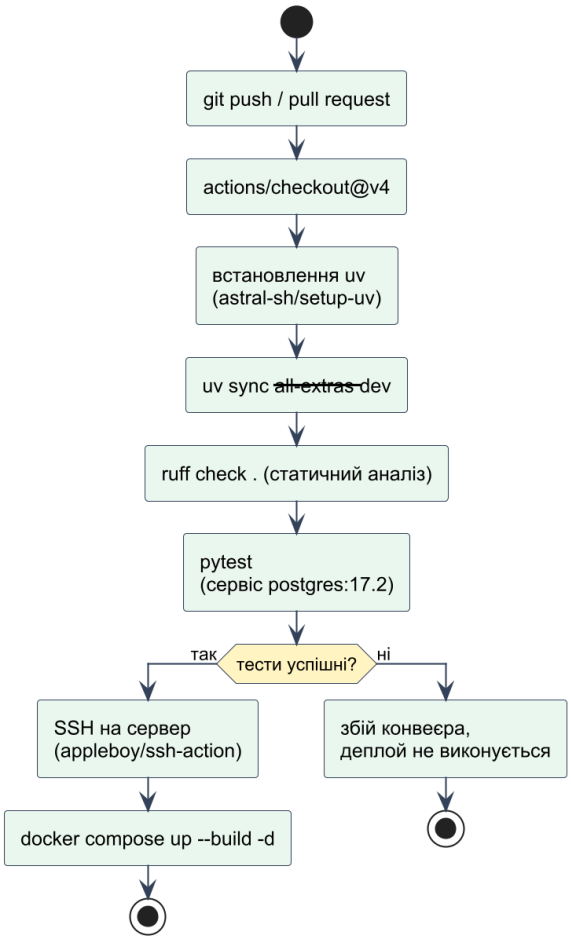


Рисунок 3.6 – Конвеєр безперервної інтеграції та доставки

Приклад успішного запуску конвеєра у веб-інтерфейсі GitHub Actions показано на рисунку 3.7. Фрагмент конфігурації конвеєра наведено у Додатку А (лістинг А.8).

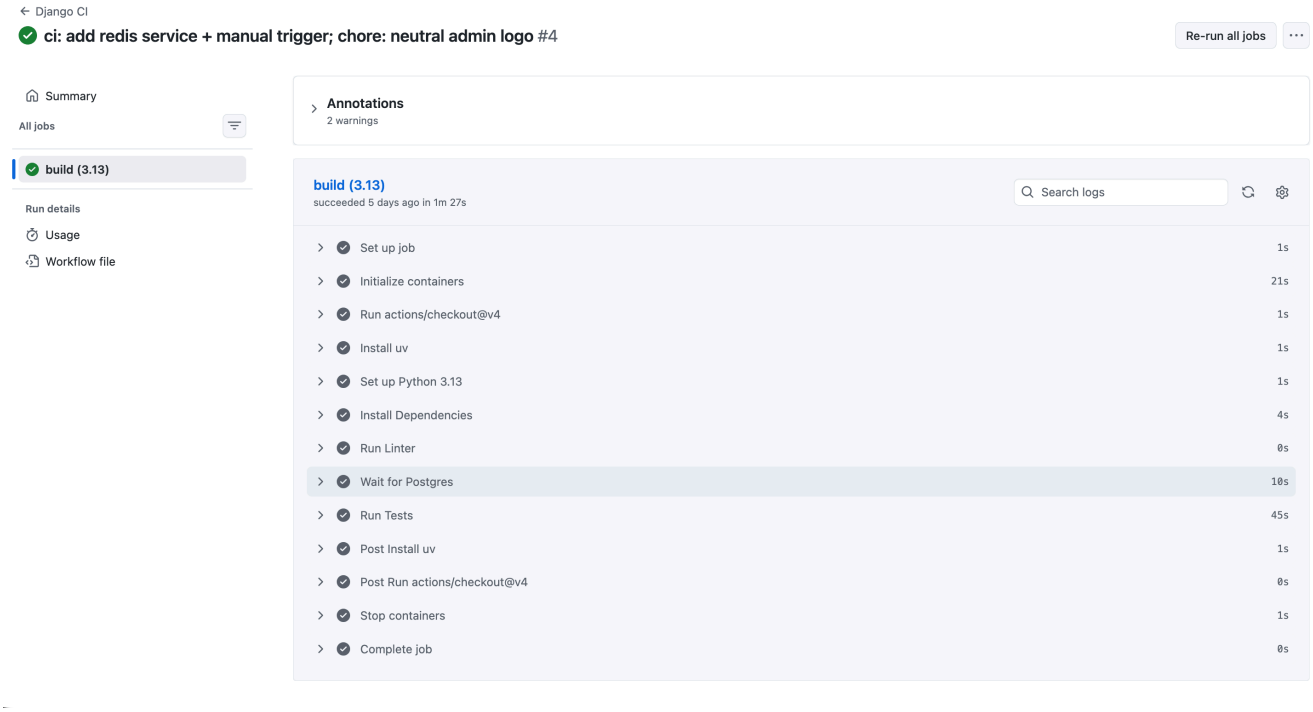


Рисунок 3.7 – Успішний запуск конвеєра у GitHub Actions

Підтримка системи у продакшні забезпечується через структуроване логування у файл `app.log` з ротацією за розміром (loguru) та автоматичну відправку критичних подій до Sentry. Усі логи помічені унікальним RequestID, що дозволяє швидко відновити контекст звернення при розслідуванні інцидентів. Для критичних адміністративних операцій (зміна групи, надсилання масової розсилки) у логах зберігається ім’я ініціатора та параметри запиту.

Висновки до третього розділу

У третьому розділі розроблено стратегію та засоби тестування системи: за принципом піраміди створено набір зі 113 модульних та інтеграційних тестів на pytest, який забезпечує покриття коду на рівні 85,5 %, що перевищує встановлений у вимогах поріг 80 %. Верифікацію якості (статичний аналіз ruff, pre-commit-хуки, контроль контрактів API та порогу покриття) поєднано з автоматизованим CI/CD-конвеєром на GitHub Actions, а розгортання у Docker-середовищі та структуроване логування з інтеграцією Sentry забезпечують контрольоване впровадження й подальшу підтримку системи.

4 БЕЗПЕКА ЖИТТЄДІЯЛЬНОСТІ, ОСНОВИ ОХОРОНИ ПРАЦІ

Відповідно до індивідуального завдання та методичних вказівок до розділу «Безпека життєдіяльності, основи охорони праці» у цій частині роботи розглянуто два питання, безпосередньо пов’язані з тематикою кваліфікаційної роботи – розробкою та експлуатацією веб-системи управління навчальним контентом. Перше стосується природних загроз і характеру їх впливу на об’єкти економіки, до яких належить і підприємство, що розробляє та надає доступ до програмного продукту. Друге – гігієнічних вимог до організації та обладнання робочих місць, оснащених відеотерміналами (ВДТ), оскільки і розробники, і користувачі системи значну частину часу працюють за персональним комп’ютером.

Безпека життєдіяльності та охорона праці становлять єдину систему правових, організаційно-технічних, санітарно-гігієнічних і профілактичних заходів, спрямованих на збереження життя, здоров’я та працездатності людини у процесі її діяльності [27]. Урахування цих вимог уже на етапі проєктування програмної системи та організації робочого процесу дозволяє знизити ризики як для персоналу, так і для безперервності функціонування самого IT-сервісу.

4.1 Природні загрози та характер їх проявів і дій на об’єкти економіки

Природні загрози (небезпеки) – це явища та процеси природного походження, які становлять загрозу життю і здоров’ю людей, завдають матеріальних збитків та порушують нормальне функціонування об’єктів господарювання. За походженням природні загрози поділяють на геологічні (землетруси, зсуви, обвали), метеорологічні (урагани, шквали, смерчі, грози, ожеледь), гідрологічні (повені, паводки, підтоплення), пожежні (лісові та торф’яні пожежі) і біологічні (епідемії, епізоотії) [27].

Характер прояву природних загроз визначається їх раптовістю, значною руйнівною силою та просторовим охопленням. Найтяжчі з них набувають масштабу надзвичайних ситуацій природного характеру, класифікацію та порядок реагування

на які встановлює Кодекс цивільного захисту України [28]. Окремі явища, зокрема сильні грози, шквали та ожеледь, безпосередньо впливають на енергетичну й телекомунікаційну інфраструктуру, від якої критично залежать сучасні інформаційні системи.

Дія природних загроз на об’єкти економіки проявляється у руйнуванні будівель і споруд, пошкодженні ліній електропостачання та зв’язку, виході з ладу інженерних мереж, зупинці виробничих і бізнес-процесів, а також у прямих і непрямих фінансових втратах [29]. Для підприємств, діяльність яких ґрунтується на безперервній обробці та зберіганні даних, особливо небезпечними є тривалі перебої електроживлення й порушення каналів зв’язку, що призводять до недоступності сервісів і ризику втрати інформації.

Розроблювана веб-система управління навчальним контентом як об’єкт економіки є програмним сервісом, працездатність якого забезпечується серверною інфраструктурою (центром обробки даних або хмарною платформою). Тому до типових наслідків природних загроз для такого об’єкта належать: відмова серверного обладнання через перебої живлення, втрата мережевого з’єднання, фізичне пошкодження апаратної частини центру обробки даних і, як наслідок, простій сервісу та потенційна втрата даних користувачів [29].

Для зниження зазначених ризиків застосовують комплекс організаційно-технічних заходів: резервне електроживлення (джерела безперебійного живлення та дизель-генератори), територіально рознесене резервне копіювання даних, розміщення сервісів у надійних центрах обробки даних або хмарних платформах із гарантованим рівнем доступності, а також розроблення плану відновлення після надзвичайних ситуацій, який визначає порядок дій персоналу при настанні аварійної події [29]. На рівні підприємства цивільний захист передбачає завчасний моніторинг загроз, систему оповіщення, навчання й тренування персоналу, що відповідає вимогам чинного законодавства [28].

4.2 Гігієнічні вимоги до організації та обладнання робочих місць з ВДТ

Робоче місце, обладнане відеотерміналом (ВДТ) на базі електронно-обчислювальної машини, є основним для розробників і користувачів програмної системи; основні шкідливі фактори під час роботи з ним – зорове та нервово-емоційне напруження, статичне навантаження на опорно-руховий апарат і електромагнітне випромінювання [30]. Гігієнічні вимоги до організації та обладнання таких робочих місць регламентують Державні санітарні правила і норми роботи з візуальними дисплейними терміналами [31] та Вимоги щодо безпеки і захисту здоров'я працівників під час роботи з екранними пристроями [32].

Вимоги до приміщень. Площа на одне робоче місце з ВДТ повинна становити не менше 6 м², а об'єм – не менше 20 м³. Приміщення з ВДТ обладнують природним і штучним освітленням; вікна доцільно орієнтувати на північ або північний схід та оснащувати регульованими сонцезахисними пристроями (жалюзі, шторами) для уникнення відблисків на екрані [31].

Освітлення. Робочі місця з ВДТ потребують комбінованого освітлення; рівень освітленості поверхні робочого стола в зоні розміщення документів має бути в межах від 300 до 500 лк. Слід уникати засліплювальної дії джерел світла та значного контрасту між яскравістю екрана й навколишніх поверхонь, забезпечуючи рівномірність освітлення робочої зони [31].

Мікроклімат і шум. Для робочих місць з ВДТ установлюють оптимальні параметри мікроклімату: температуру повітря в межах від 22 до 24 °С, відносну вологість від 40 до 60 % та швидкість руху повітря не більше 0,1 м/с. Рівень шуму на робочому місці не повинен перевищувати 50 дБА, оскільки підвищений шум знижує концентрацію уваги та продуктивність розумової праці [31].

Обладнання робочого місця та ергономіка. Екран монітора розміщують на відстані від 60 до 70 см від очей користувача так, щоб верхній край екрана був на рівні очей або трохи нижче. Робочий стіл має забезпечувати раціональне розміщення обладнання, а робочий стілець – бути підйомно-поворотним з регульованою висотою сидіння та кутом нахилу спинки; за потреби

використовують підставку для ніг і пюпітр для документів [30]. Раціональна організація робочого місця знижує статичне навантаження на хребет, м’язи шиї та рук [33].

Режим праці й відпочинку. Для запобігання перевтомі під час роботи з ВДТ установлюють регламентовані перерви – орієнтовно від 10 до 15 хвилин через кожну годину роботи, які доцільно використовувати для гімнастики для очей і нескладних фізичних вправ [32]. Працівники, зайняті роботою з екранними пристроями понад половину робочого часу, проходять попередні та періодичні медичні огляди [33].

Висновки до четвертого розділу

Таким чином, у розділі розглянуто два питання безпеки життєдіяльності та охорони праці, що відповідають тематиці роботи. Проаналізовано природні загрози й характер їх впливу на об’єкти економіки, зокрема на ІТ-інфраструктуру, що забезпечує роботу веб-системи, та визначено організаційно-технічні заходи для підвищення стійкості сервісу. Сформульовано гігієнічні вимоги до організації та обладнання робочих місць з відеотерміналами, дотримання яких зберігає здоров’я й працездатність розробників і користувачів системи та сприяє підвищенню продуктивності праці.

ВИСНОВКИ

У ході виконання кваліфікаційної роботи розроблено програмне забезпечення веб-системи управління навчальним контентом з використанням мови програмування Python та фреймворку Django. На основі проведеного дослідження можна зробити такі висновки.

1. Проведено аналіз предметної області управління навчальним контентом, виконано порівняльний аналіз чотирьох провідних рішень ринку (Moodle, Canvas LMS, Google Classroom, Blackboard Learn). Сформульовано перелік функціональних і нефункціональних вимог до системи з урахуванням специфіки групово-орієнтованого проходження курсу з керованим розкладом відкриття навчальних модулів.

2. Обґрунтовано вибір технологічного стеку: Python 3.13, Django 5.2, Django Ninja, PostgreSQL 17, Redis, django-ninja-jwt, AWS S3, SendGrid, Docker, Gunicorn, Nginx, pytest, GitHub Actions, Sentry.

3. Спроєктовано багаторівневу архітектуру системи з чітким розділенням рівнів API, бізнес-логіки, доступу до даних та інфраструктури. Реалізовано шість Django-додатків: users, modules, homeworks, quizzes, mental_health та admin_custom, які реєструються у центральному роутері `apps/routers`.py.

4. Розроблено схему бази даних з основних сутностей системи, нормалізовану до 3NF; реалізовано доменні моделі з валідацією на рівні методів `clean()` та `save()`, кастомні менеджери `ActiveManager` та `ActiveLessonManager`, проміжні моделі `GroupMembership` та модель `UserDeviceLog` для логування пристроїв слухачів.

5. Реалізовано REST API з 15+ маршрутами на основі бібліотеки Django Ninja з автогенерацією `OpenAPI` 3.0, валідацією `Pydantic`-схемами, JWT-автентифікацією та централізованою обробкою помилок.

6. Винесено бізнес-логіку у сервісні класи: `UserProgressService` обчислює бал слухача та його поточну позицію в курсі, `LessonNavigationService` забезпечує навігацію між уроками, а доступність модулів визначається за

календарем групи `UserGroup`; окремі функції-помічники надсилають email-листи через SendGrid.

7. Реалізовано підсистему вхідного та підсумкового опитування слухачів (моделі `MentalHealth`, `MentalHealthQuestion`, `MentalHealthAttempt`, `UserMentalHealthResponse`), яка вимірює динаміку самооцінки рівня знань та мотивації слухача до та після проходження курсу й покрита окремим набором модульних та інтеграційних тестів.

8. Розроблено тестовий набір зі 113 модульних та інтеграційних тестів на pytest з покриттям коду 85,5 %, що перевищує встановлений у вимогах поріг 80 %. Автоматизовано верифікацію коду через ruff і pre-commit-хуки та перевірку конвенцій комітів через commitizen.

9. Налаштовано конвеєр безперервної інтеграції на GitHub Actions (статичний аналіз ruff і повний тестовий набір проти PostgreSQL 17.2) та окремий workflow автоматичного розгортання через SSH з перезбиранням Docker-контейнерів на цільовому сервері.

Розроблена система відрізняється від конкурентних рішень наявністю групово-орієнтованого режиму проходження курсу з керованим розкладом відкриття модулів, сегментованих чат-запрошень за демографічними групами, прозорого механізму нарахування балів за переглянуті уроки та пройдені квізи, вбудованого вхідного та підсумкового опитування слухачів для вимірювання динаміки самооцінки рівня знань, а також повністю автоматизованим CI/CD-конвеєром.

Практична значущість роботи полягає в тому, що розроблене програмне забезпечення є самостійним серверним рішенням і може використовуватися як основа для різних спеціалізованих освітніх продуктів. Модульна структура коду полегшує подальший розвиток системи. Контейнеризація через Docker та чітко описаний CI/CD-конвеєр гарантують відтворюваність розгортання у будь-якому хмарному провайдері.

Перспективи подальшого розвитку проєкту охоплюють: впровадження рекомендаційних алгоритмів для персоналізації навчального маршруту;

розширення підсистеми аналітики з агрегацією поведінкових метрик; додавання асинхронних задач через брокер Celery або RQ; впровадження двофакторної автентифікації; розширення інтеграції з месенджерами не лише на рівні запрошень, а й активного зворотного зв’язку через ботів.

Структуру, обсяг та оформлення пояснювальної записки витримано відповідно до методичних вказівок кафедри програмної інженерії з виконання кваліфікаційної роботи бакалавра.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Чигрин М. М. Архітектурні підходи до побудови веб-системи управління навчальним контентом // Природничі та гуманітарні науки. Актуальні питання: матеріали IX Міжнародної студентської науково-технічної конференції. – Тернопіль: ТНТУ ім. І. Пулюя, 2026. – С. (у друці).

2. Garrison D. R. E-Learning in the 21st Century: A Community of Inquiry Framework for Research and Practice. 3rd ed. New York: Routledge, 2017. 220 p.

3. Anderson T., Dron J. Three Generations of Distance Education Pedagogy. International Review of Research in Open and Distributed Learning. 2011. Vol. 12, No 3. P. 80–97.

4. Биков В. Ю., Кухаренко В. М., Сиротенко Н. Г., Рибалко О. В., Богачков Ю. М. Технологія розробки дистанційного курсу: навч. посіб. – Київ: Міленіум, 2008. – 324 с.

5. IEEE Computer Society. SWEBOK v4.0: Guide to the Software Engineering Body of Knowledge. Los Alamitos: IEEE, 2024. 413 p.

6. Fielding R. T. Architectural Styles and the Design of Network-based Software Architectures: PhD dissertation. University of California, Irvine, 2000. 162 p.

7. Moodle Documentation. URL: https://docs.moodle.org (дата звернення: 01.03.2026).

8. Canvas LMS Platform Overview. URL: https://www.instructure.com/canvas (дата звернення: 01.03.2026).

9. Google Classroom Help Center. URL: https://support.google.com/edu/classroom (дата звернення: 02.03.2026).

10. Blackboard Learn Documentation. URL: https://help.blackboard.com (дата звернення: 02.03.2026).

11. Михалик Д. М., Цуприк Г. Б., Бревус В. М. Методичні вказівки до виконання кваліфікаційної роботи бакалавра для здобувачів спеціальності 121 – Інженерія програмного забезпечення (всіх форм навчання). – Тернопіль: ТНТУ ім. І. Пулюя, 2024. – 45 с.

12. Redis Documentation. URL: https://redis.io/docs/ (дата звернення: 08.03.2026).

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

14. Django Software Foundation. Django 5.2 Documentation. URL: https://docs.djangoproject.com/en/5.2/ (дата звернення: 05.03.2026).

15. Python Software Foundation. Python 3.13 Documentation. URL: https://docs.python.org/3.13/ (дата звернення: 05.03.2026).

16. Fowler M. Patterns of Enterprise Application Architecture. Boston: Addison-Wesley, 2002. 560 p.

17. Семчишин В., Михалик Д. М. Data-driven decision-making methods and hierarchical analysis in cloud-based medical service management systems // ITTAP-2025: 5th International Workshop on Information Technologies: Theoretical and Applied Problems. – Тернопіль: ТНТУ, 2025. – С. 1–9.

18. Швець О. Занурення в патерни проектування / пер. з англ.: В. Гальцев, О. Швець. Refactoring.Guru, 2022. 396 с. URL: https://refactoring.guru/uk/design-patterns/book (дата звернення: 11.06.2026).

19. PostgreSQL Global Development Group. PostgreSQL 17 Documentation. URL: https://www.postgresql.org/docs/17/ (дата звернення: 06.03.2026).

20. Richardson L., Ruby S. RESTful Web Services. Sebastopol: O’Reilly Media, 2007. 454 p.

21. Django Ninja Documentation. URL: https://django-ninja.dev/ (дата звернення: 20.04.2026).

22. Jones M., Bradley J., Sakimura N. JSON Web Token (JWT). RFC 7519. Internet Engineering Task Force, 2015. 30 p.

23. Семчишин В. М., Михалик Д. М. Модель комплексної інформаційної безпеки хмарної системи управління медичними послугами // Прикладні питання математичного моделювання. – 2025. – Т. 8, № 2. – С. 253–264.

24. Petryk M., Mykhalyk D. High-performance intellectual information technologies for the study of filtration systems in different-sized nanoporous particles

media // Scientific Journal of TNTU. – Тернопіль: ТНТУ, 2022. – Vol. 108, № 4. – P. 16–26.

25. Beck K. Test Driven Development: By Example. Boston: Addison-Wesley, 2002. 240 p.

26. Pytest Documentation. URL: https://docs.pytest.org/ (дата звернення: 08.03.2026).

27. Атаманчук П. С. Безпека життєдіяльності: навчальний посібник. – Київ: Центр учбової літератури, 2020. – 276 с.

28. Кодекс цивільного захисту України: Закон України від 02.10.2012 р. № 5403-VI (зі змінами) // Відомості Верховної Ради України. – 2013. – № 34–35. – Ст. 458.

29. Желібо Є. П., Зацарний В. В. Безпека життєдіяльності: підручник. – Київ: Каравела, 2023. – 344 с.

30. Жидецький В. Ц. Охорона праці користувачів комп’ютерів: підручник. – Львів: Афіша, 2020. – 176 с.

31. ДСанПіН 3.3.2.007-98. Державні санітарні правила і норми роботи з візуальними дисплейними терміналами електронно-обчислювальних машин. – Київ, 1998. – 22 с.

32. НПАОП 0.00-7.15-18. Вимоги щодо безпеки та захисту здоров’я працівників під час роботи з екранними пристроями: затв. наказом Міністерства соціальної політики України від 14.02.2018 р. № 207. – Київ, 2018. – 16 с.

33. Мелех Л. В. Безпека життєдіяльності та охорона праці: навчальний посібник. – Львів: Львівський державний університет внутрішніх справ, 2022. – 219 с.

ДОДАТКИ

ДОДАТОК А

Лістинги коду основних компонентів системи

Лістинг А.1 – Модель користувача apps/users/models.py (фрагмент)

```
class User(AbstractUser):
    """Користувач системи, email – унікальний ідентифікатор."""""
    first_name = models.CharField(_("first name"), max_length=150)
    last_name  = models.CharField(_("last name"),  max_length=150)
    email      = models.EmailField(_("email address"), unique=True)
    phone      = PhoneNumberField(_("phone number"), unique=True)
    score      = models.PositiveIntegerField(_("score"), default=0)

    gender = models.CharField(
        _("gender"), max_length=30, choices=GenderChoice.choices,
        blank=True, null=True,
    )
    age_group = models.CharField(
        _("age group"), max_length=20,
        choices=AgeGroupChoice.choices,
default=AgeGroupChoice.UNDER_24,
    )
    children = models.CharField(
        _("children"), max_length=30,
        choices=ChildrenChoice.choices, blank=True, null=True,
    )
    family_status = models.CharField(
        _("family status"), max_length=40,
        choices=FamilyStatusChoice.choices, blank=True, null=True,
    )
    interests = ArrayField(
        models.CharField(max_length=50,
choices=InterestTypeChoice.choices),
        blank=True, default=list, verbose_name=_("interests"),
    )

    USERNAME_FIELD = "email"
    REQUIRED_FIELDS = []
    objects = UserManager()

    class Meta:
        verbose_name = _("User")
        verbose_name_plural = _("Users")
        ordering = ["-date_joined"]

    def save(self, *args, **kwargs):
        if not self.username:
            self.username = self.email
        if self.password and not self.password.startswith("pbkdf2"):
            self.set_password(self.password)
        super().save(*args, **kwargs)
```

Лістинг А.2 – Сервіс прогресу слухача apps/users/services.py

```
class UserProgressService:
    @staticmethod
    def get_total_possible_score() -> float:
        lessons_score = Lesson.active.aggregate(
            total=Sum("score"))["total"] or 0.0
        quiz_ids = Lesson.active.filter(
            content_type=ContentType.QUIZ).values_list("quiz_fk",
flat=True)
        qs = Question.objects.filter(quiz_fk__in=quiz_ids)
        text_q  =
qs.filter(question_type=QuestionTypes.TEXT).count()
        other_q =
qs.exclude(question_type=QuestionTypes.TEXT).count()
        quizzes_total = text_q * 2.0 + other_q * 1.0
        return float(lessons_score) + quizzes_total

    @staticmethod
    def get_user_current_position(user) -> dict:
        completed_ids = set(
            UserLessonProgress.objects
            .filter(user_fk=user, is_completed=True)
            .values_list("lesson_fk_id", flat=True)
        )
        next_lesson = (
            Lesson.active.select_related("module_fk")
            .exclude(id__in=completed_ids)
            .order_by("module_fk__order", "order", "id")
            .first()
        )
        if next_lesson is None:
            return {"current_module": None, "current_lesson": None}
        return {
            "current_module": {
                "id": next_lesson.module_fk.id,
                "name": next_lesson.module_fk.name,
                "order": next_lesson.module_fk.order,
            },
            "current_lesson": {
                "id": next_lesson.id,
                "name": next_lesson.name,
                "order": next_lesson.order,
            },
        }

    @staticmethod
    def recalculate_user_score(user) -> None:
        lesson_score = (
            UserLessonProgress.objects
            .filter(user_fk=user, is_completed=True,
```

```
                    lesson_fk__module_fk__is_scored=True,
                    lesson_fk__is_active=True,
                    lesson_fk__module_fk__is_active=True)
            .exclude(lesson_fk__content_type__in=[
                ContentType.QUIZ, ContentType.HOMEWORK])
            .aggregate(s=Sum("lesson_fk__score"))["s"] or 0.0
        )
        attempts = QuizAttempt.objects.filter(
            user_fk=user, is_completed=True,
            quiz_fk__lessons__is_active=True,
        ).distinct()
        quiz_score = (
            attempts.values("quiz_fk")
            .annotate(mx=Max("score"))
            .aggregate(t=Sum("mx"))["t"] or 0.0
        )
        user.score = float(lesson_score) + float(quiz_score)
        user.save(update_fields=["score"])
```

Лістинг А.3 – Ендпоінт реєстрації apps/users/api.py (фрагмент)

```
@router.post(
    "/register",
    response={
        HTTP_201_CREATED:        UserResponseSchema,
        HTTP_409_CONFLICT:       ErrorSchema,
        HTTP_422_UNPROCESSABLE_ENTITY: ErrorSchema,
    },
    auth=None,
    throttle=AnonRateThrottle("5/m"),
)
def register(request, payload: UserRegisterSchema):
    try:
        user = get_user_model().objects.create_user(
            email=payload.email,
            password=payload.password,
            first_name=payload.first_name,
            last_name=payload.last_name,
            phone=payload.phone,
        )
    except EmailAlreadyExistsError as exc:
        return HTTP_409_CONFLICT, {"detail": str(exc)}
    except PhoneAlreadyExistsError as exc:
        return HTTP_409_CONFLICT, {"detail": str(exc)}

    send_activation_email(user)
    return HTTP_201_CREATED, UserResponseSchema.from_orm(user)
```

Лістинг А.4 – Сигнал автоматичного прикріплення до групи apps/users/signals.py

```
@receiver(post_save, sender=User)
def assign_user_to_group(sender, instance, created, **kwargs):
```

```
    if not created:
        return
    active_group = (
        UserGroup.objects
        .filter(is_active=True)
        .order_by("registration_started_at")
        .first()
    )
    if not active_group:
        return
    GroupMembership.objects.create(
        user=instance, group=active_group)
```

Лістинг А.5 – Приклад модульних тестів apps/users/tests.py (фрагмент)

```
@pytest.mark.django_db
class UserRegistrationTests:
    def test_register_creates_user(self, api_client):
        response = api_client.post("/api/v1/users/register", json={
            "email": "new@example.com",
            "password": "Str0ng!Pass",
            "first_name": "Olga",
            "last_name":  "Petrenko",
            "phone": "+380501112233",
        })
        assert response.status_code == 201
        user = User.objects.get(email="new@example.com")
        assert user.username == "new@example.com"
        assert user.is_active is False

    def test_register_duplicate_email(self, api_client, user):
        response = api_client.post("/api/v1/users/register", json={
            "email": user.email,
            "password": "Str0ng!Pass",
            "first_name": "X", "last_name": "Y",
            "phone": "+380501112244",
        })
        assert response.status_code == 409

    def test_login_returns_jwt_pair(self, api_client, active_user):
        response = api_client.post("/api/v1/users/login", json={
            "email": active_user.email,
            "password": "Str0ng!Pass",
        })
        assert response.status_code == 200
        body = response.json()
        assert "access"  in body
        assert "refresh" in body
```

Лістинг А.6 – Кастомне middleware логування пристроїв config/middleware.py (фрагмент)

```
class DeviceLogMiddleware:
    def __init__(self, get_response):
        self.get_response = get_response

    def __call__(self, request):
        response = self.get_response(request)
        try:
            ua_string = request.META.get("HTTP_USER_AGENT", "")
            user_agent = parse(ua_string)
            user = getattr(request, "user", None)
            if user and user.is_authenticated:
                UserDeviceLog.objects.create(
                    user_fk=user,
                    user_agent=ua_string[:512],
                    browser=user_agent.browser.family[:64],
                    os=user_agent.os.family[:64],
                    is_mobile=user_agent.is_mobile,
                    ip=self._client_ip(request),
                )
        except Exception as exc:
            logger.warning("DeviceLog skipped: {}", exc)
        return response

    @staticmethod
    def _client_ip(request):
        x_forwarded = request.META.get("HTTP_X_FORWARDED_FOR")
        if x_forwarded:
            return x_forwarded.split(",")[0].strip()
        return request.META.get("REMOTE_ADDR", "")
```

Лістинг А.7 – Конфігурація pytest у pyproject.toml

```
[tool.pytest.ini_options]
minversion = "6.0"
addopts = "--ds=config.settings --reuse-db --cov=apps " \
          "--cov-report=term-missing --cov-fail-under=80"
python_files     = ["tests.py", "test_*.py", "*_tests.py"]
python_classes   = ["*Tests"]
python_functions = ["test_*"]

[tool.coverage.run]
omit = [
    "*/migrations/*", "*/tests/*", "*/tests.py",
    "*/admin.py", "*/apps.py", "*/__init__.py",
    "manage.py", "config/*",
]

[tool.coverage.report]
exclude_lines = [
    "pragma: no cover",
    "def __repr__",
    "raise NotImplementedError",
```

```
    "if __name__ == .__main__.:",
]
```

Лістинг А.8 – Конвеєр GitHub Actions .github/workflows/ci.yml (фрагмент)

```
name: CI
on: [push, pull_request]
jobs:
  test:
    runs-on: ubuntu-latest
    services:
      postgres:
        image: postgres:16
        env:
          POSTGRES_USER: ci
          POSTGRES_PASSWORD: ci
          POSTGRES_DB: ci_db
        ports: ["5432:5432"]
      redis:
        image: redis:7
        ports: ["6379:6379"]
    steps:
      - uses: actions/checkout@v4
      - uses: actions/setup-python@v5
        with: { python-version: "3.13" }
      - run: pip install uv && uv sync --frozen
      - run: uv run ruff check .
      - run: uv run pytest
```

ДОДАТОК Б

Тези доповіді на IX Міжнародній студентській науково-технічній конференції «Природничі та гуманітарні науки. Актуальні питання»

УДК 004.42

Чигрин М.М. – ст. гр. СП-42
Тернопільський національний технічний університет імені Івана Пулюя

АРХІТЕКТУРНІ ПІДХОДИ ДО ПОБУДОВИ ВЕБ-СИСТЕМИ УПРАВЛІННЯ НАВЧАЛЬНИМ КОНТЕНТОМ

Науковий керівник: кандидат технічних наук., доцент Михалик Д.М.

Chyhryn M.
Ternopil Ivan Puluj National Technical University

ARCHITECTURAL APPROACHES TO BUILDING A WEB-BASED LEARNING CONTENT MANAGEMENT SYSTEM

Supervisor: PhD, Associate Professor Mykhalyk D.M.

Ключові слова: веб-система, навчальний контент, REST API, модульна архітектура
Keywords: web system, learning content, REST API, modular architecture

Сучасний розвиток дистанційної освіти супроводжується зростанням вимог до гнучкості, масштабованості та надійності платформ управління навчальним контентом. Більшість існуючих LMS-рішень (Moodle, Canvas) є монолітними та складними в адаптації до специфічних потреб освітніх установ, що зумовлює актуальність розробки власних архітектурно обґрунтованих систем.

Метою роботи є проектування та реалізація веб-системи управління навчальним контентом з чіткою модульною структурою та розмежуванням ролей користувачів: студент, викладач та адміністратор.

В основу архітектури системи покладено принцип розділення відповідальностей між компонентами – підхід, що забезпечує незалежність бізнес-логіки від інфраструктурних деталей та спрощує підтримку і розширення системи [1]. Систему розділено на незалежні модулі: навчальний контент, домашні завдання, тестування, управління користувачами та психологічна підтримка студентів. Надійність та масштабованість хмарних інформаційних систем досягається шляхом чіткого розмежування архітектурних рівнів і рольового доступу до даних [2].

Для реалізації програмного інтерфейсу застосовано фреймворк Django Ninja, який забезпечує автоматичну генерацію OpenAPI-документації та типізацію запитів і відповідей на основі анотацій типів [3]. Автентифікацію реалізовано через JWT-токени, що дозволяє

будувати stateless-взаємодію між клієнтом та сервером відповідно до принципів REST-архітектури [4]. Для підвищення продуктивності застосовано кешування на основі Redis, а зберігання медіафайлів організовано у хмарному S3-сховищі.

Діаграма розгортання (див. рис. 1) ілюструє інфраструктурну складову системи, яка базується на контейнеризації за допомогою Docker. Nginx виконує роль реверс-проксі та балансувальника навантаження, Django з Gunicorn обробляють запити REST API, PostgreSQL забезпечує персистентне зберігання даних, а Redis використовується для кешування часто запитуваних ресурсів.

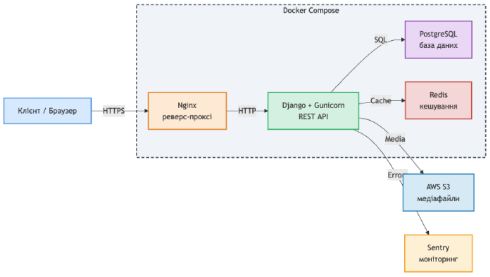


Рисунок 1 – Діаграма розгортання веб-системи управління навчальним контентом

Для забезпечення якості програмного коду налаштовано автоматизоване тестування з покриттям понад 80% за допомогою pytest, а моніторинг помилок у продакшн-середовищі здійснюється через Sentry. Застосування модульної архітектури у поєднанні з сучасним API-фреймворком, хмарною інфраструктурою та автоматизованим тестуванням дозволяє отримати масштабовану, надійну та легко підтримувану систему управління навчальним контентом.

Подальші дослідження можуть бути спрямовані на впровадження рекомендаційних алгоритмів для персоналізації навчального процесу та розширення модуля психологічної підтримки студентів.

Література:

1. Martin R. C. Clean Architecture: A Craftsman's Guide to Software Structure and Design. Boston: Prentice Hall, 2017. 432 p.
2. Semchyshyn V., Mykhalyk D. Data-driven decision-making methods and hierarchical analysis in cloud-based medical service management systems. ITTAP-2025: 5th International Workshop on Information Technologies: Theoretical and Applied Problems. Ternopil, 2025. P. 1–9.
3. Django Ninja – Fast Django REST Framework. – Електронний ресурс. – Режим доступу: https://django-ninja.dev (дата звернення: 20.04.2026)
4. Fielding R. T. Architectural Styles and the Design of Network-Based Software Architectures: Doctoral dissertation. University of California, Irvine, 2000. 162 p.

ДОДАТОК В

Посилання на репозиторій з вихідним кодом проєкту

Вихідний код програмної системи розміщено у публічному репозиторії системи контролю версій Git на платформі GitHub за адресою: https://github.com/Sn1kls/learning-platform-backend