МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ІНЖЕНЕРНИЙ НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ
ім. Ю.М. Потебні
ЗАПОРІЗЬКОГО НАЦІОНАЛЬНОГО УНІВЕРСИТЕТУ

КАФЕДРА ЕЛЕКТРОНІКИ, ІНФОРМАЦІЙНИХ СИСТЕМ ТА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Кваліфікаційна робота

другий (магістерський)
(рівень вищої освіти)

на тему **ОСОБЛИВОСТІ СТВОРЕННЯ КРОС-ПЛАТФОРМНОЇ СИСТЕМИ УПРАВЛІННЯ НАВЧАННЯМ (LMS) НА БАЗІ NEXT.JS, NEST.JS ТА REACT NATIVE**

Виконав: студент 2 курсу, групи 8.1214-іпз-2
спеціальності 121 Інженерія програмного забезпечення
(код і назва спеціальності)
освітньої програми Інженерія програмного забезпечення
(код і назва освітньої програми)
Коваленко К.М.
(ініціали та прізвище)
Керівник доцент, к.ф.-м.н., доцент Скрипник І.А
(посада, вчене звання, науковий ступінь, підпис, ініціали та прізвище)
Рецензент Директор ТОВ «Дискус»
Р.О.Лютий
(посада, вчене звання, науковий ступінь, підпис, ініціали та прізвище)

Запоріжжя
2025

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ІНЖЕНЕРНИЙ НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ
ім. Ю.М. Потебні
ЗАПОРІЗЬКОГО НАЦІОНАЛЬНОГО УНІВЕРСИТЕТУ

Кафедра __електроніки, інформаційних систем та програмного__ забезпечення
Рівень вищої освіти _____другий (магістерський)_____
Спеціальність ___121_Інженерія програмного забезпечення_____
(код та назва)
Освітня програма ___Інженерія програмного забезпечення
(код та назва)

ЗАТВЕРДЖУЮ

Завідувач кафедри _____Т.В. Критська_
"_01.04.2025_"

З А В Д А Н Н Я
НА КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТОВІ

Коваленку Кирилу Максимовичу
(прізвище, ім'я, по батькові)

1.Тема роботи __Особливості створення крос-платформної системи управління навчанням (LMS) на базі Next.js, Nest.js та React Native.__

керівник роботи ___к.ф.-м.н., доцент Скрипник Ірина Анатоліївна_____
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)
затверджені наказом ЗНУ від _28.03.2025 р. № 361-е__
2. Строк подання студентом кваліфікаційної роботи_____01.12.2025 р.___
3. Вихідні дані магістерської роботи
- комплект нормативних документів ;
- технічне завдання до роботи.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)
- огляд та збір літератури стосовно теми кваліфікаційної роботи;
- огляд та аналіз існуючих систем управління навчанням та їх функціональних можливостей;
- дослідження особливостей створення крос-платформної LMS та вибору технологічного стеку (Next.js, NestJS, React Native, PostgreSQL);
- створення програмного продукту та його опис;
- перелік вимог для роботи програми;
- дослідження поставленої проблеми та розробка висновків та пропозицій.

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

6. Консультанти розділів магістерської роботи

Розділ	Прізвище, ініціали та посада Консультанта	Підпис, дата Завдання прийняв

7. Дата видачі завдання 01.04.2025

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів магістерської роботи	Строк виконання етапів магістерської роботи	Примітка
1	Аналіз предметної області	05.03.2025 – 15.03.2025	виконано
2	Дослідження архітектурних підходів (моноліт, мікросервіси, REST/GraphQL)	06.04.2025 – 18.04.2025	виконано
3	Проєктування архітектури LMS (БД, модулі, API, ролі користувачів)	11.05.2025 – 31.05.2025	виконано
4	Реалізація бекенд-частини на Nest.js (автентифікація, користувачі, курси)	01.06.2025 – 30.06.2025	виконано
5	Реалізація фронтенд-частини (React або Next.js) — інтерфейс для студентів та викладачів	01.07.2025 – 25.07.2025	виконано
6	Розробка мобільного додатка на React Native (навігація, курсова стрічка, прогрес)	26.07.2025 – 15.08.2025	виконано
7	Тестування системи	16.08.2025 – 10.09.2025	виконано
8	Порівняння характеристик роботи LMS та аналіз результатів	11.09.2025 – 25.09.2025	виконано
9	Оформлення пояснювальної записки	26.09.2025 – 30.11.2025	виконано

Студент _____ Коваленко К.М.
(підпис) (прізвище та ініціали)

Керівник роботи _____ Скрипник І.А.
(підпис) (прізвище та ініціали)

Нормоконтроль пройдено

Нормоконтролер _____ Скрипник І.А,__
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Сторінок: 119

Рисунків: 36

Таблиць: 5

Джерел: 35

Коваленко К. М. Особливості створення крос-платформної системи управління навчанням (LMS) на базі Next.js, Nest.js та React Native: кваліфікаційна робота магістра спеціальності 121 «Інженерія програмного забезпечення» / наук. керівник І. А. Скрипник. Запоріжжя : ЗНУ, 2025. 119 с.

Метою роботи є дослідження сучасних підходів до розробки крос-платформних систем управління навчанням *(LMS)*, а також створення програмної системи, яка забезпечує доступ до навчальних матеріалів, керування курсами, відстеження прогресу користувачів та взаємодію між студентами і викладачами на різних типах пристроїв — веб, *Android* та *iOS*.

У процесі дослідження було проаналізовано предметну область *LMS-систем*, розглянуто сучасні веб-технології та архітектурні підходи, що використовуються при побудові освітніх платформ. На основі проведеного аналізу розроблено крос-платформну систему, до складу якої входять веб-інтерфейс, реалізований засобами React, бекенд-частина на основі *Nest.js* з підтримкою *REST API* та мобільний додаток, створений з використанням *React Native*.

Розроблену систему протестовано на етапах функціонального, інтеграційного та навантажувального тестування. Результати демонструють ефективність використання єдиного стеку технологій *JavaScript/TypeScript* для побудови сучасної *LMS*, яка може бути масштабована та розширена відповідно до потреб користувачів.

Ключові слова**:** *система управління навчанням, LMS, React, Nest.js, React Native, веб-додаток, крос-платформна розробка, освітні технології*.

SUMMARY

Pages: 119

Figures: 36

Tables: 5

Sources: 35

Kovalenko K. M. Features of Developing a Cross-Platform Learning Management System (LMS) Using Next.js, NestJS, and React Native: master's thesis in specialty 121 "Software Engineering" / scientific supervisor I. A. Skrypnyk. Zaporizhzhia: ZNU, 2025. 119 p.

The purpose of the work is to study modern approaches to the development of cross-platform learning management systems *(LMS),* as well as the creation of a software system that provides access to educational materials, course management, user progress tracking and interaction between students and teachers on different types of devices - *web*, *Android* and *iOS*.

In the process of research, the subject area of *LMS* systems was analyzed, modern web technologies and architectural approaches used in building educational platforms were considered. Based on the analysis, a cross-platform system was developed, which includes a web interface implemented using React, a backend based on *Nest.js* with *REST API* support and a mobile application created using *React Native*.

The developed system was tested at the stages of functional, integration and load testing. The results demonstrate the effectiveness of using a single *JavaScript/TypeScript* technology stack to build a modern *LMS* that can be scaled and expanded according to user needs.

Keywords: *learning management system, LMS, React, Nest.js, React Native, web application, cross-platform development, educational technologies*.

ЗМІСТ

ВСТУП 9
РОЗДІЛ 1 ДОСЛІДЖЕННЯ ПРОБЛЕМИ СТВОРЕННЯ КРОС-ПЛАТФОРМНОЇ СИСТЕМИ УПРАВЛІННЯ НАВЧАННЯМ 16
1.1 Огляд проблеми організації електронного навчання та управління навчальним процесом 16
1.2 Технології для створення систем управління навчанням 18
1.3 Сфери застосування систем управління навчанням 20
1.4 Висновки з розділу 1 22
РОЗДІЛ 2 ДОСЛІДЖЕННЯ ТЕХНОЛОГІЙ ТА ІНСТРУМЕНТІВ ДЛЯ СТВОРЕННЯ КРОС-ПЛАТФОРМНОЇ СИСТЕМИ УПРАВЛІННЯ НАВЧАННЯМ 24
2.1 Загальні вимоги до технологічного стеку крос-платформної LMS 24
2.1.1 Функціональні вимоги 24
2.1.2 Нефункціональні вимоги 25
2.1.3 Узагальнена архітектура системи 26
2.2 Додаткові інструменти та забезпечення якості 27
2.2.1 Контроль версій та командна робота 27
2.2.2 Контейнеризація та розгортання 27
2.2.3 CI/CD та автоматизація 28
2.2.4 Тестування та моніторинг 28
2.3 Типові сценарії використання крос-платформної LMS 28
2.3.1 Робота студента 28
2.3.2 Робота викладача 29
2.3.3 Робота адміністратора 29
2.4 Архітектурні підходи до побудови крос-платформної LMS 30
2.4.1 Шарова та модульна організація системи 30
2.4.2 Монолітна, модульна та мікросервісна архітектури: порівняння 31
2.4.3 Синхронна та асинхронна взаємодія компонентів LMS 32

2.5 Висновки з розділу 2 35
РОЗДІЛ 3 ПРОЄКТУВАННЯ ТА РОЗРОБКА КРОС-ПЛАТФОРМНОЇ СИСТЕМИ УПРАВЛІННЯ НАВЧАННЯМ 36
3.1 Налаштування середовища розробки системи 36
3.1.1 Налаштування середовища для навчання 36
3.1.2 Структура репозиторію проекту 37
3.1.3 Налаштування та запуск бекенд-частини 42
3.1.4 Налаштування та запуск веб-інтерфейсу 44
3.1.5 Налаштування та запуск мобільного застосунку 46
3.1.6 Підсумок 49
3.2 Загальна архітектура системи 49
3.2.1 Компоненти системи та інфраструктура 49
3.2.2 API-шлюз та маршрутизація запитів 51
3.2.3 Доменні сервіси та їх взаємодія 52
3.2.4 Налаштування середовища для впровадження системи 57
3.2.5 Автентифікація та авторизація в системі 58
3.2.6 Клієнтські застосунки та обмін даними з сервером 60
3.2.7 Типові потоки даних у системі 62
3.3 Архітектура серверної частини 64
3.3.1 Реалізація сервісу користувачів 67
3.3.2 Реалізація сервісу курсів 71
3.3.3 Сервіси прогресу та сповіщень 75
3.4 Архітектура та основні екрани веб-клієнта 78
3.4.1 Інтерфейс викладача та адміністратора у веб-клієнті 84
3.5 Архітектура та екран авторизації мобільного клієнта 88
3.5.1 Основний екран та список курсів у мобільному клієнті 91
3.5.2 Перегляд уроків та тестів у мобільному клієнті 94
3.6 Висновки з розділу 3 100
РОЗДІЛ 4 ДОСЛІДЖЕННЯ РЕЗУЛЬТАТІВ ТЕСТУВАННЯ КРОСПЛАТФОРМНОЇ LMS 102

4.1 Методика тестування та експериментальне середовище..... 102
4.2 Функціональне та інтеграційне тестування..... 104
4.3 Навантажувальне тестування (режим нормальної роботи)..... 106
4.4 Стресове та спайк-тестування..... 109
4.4.1 Стрес-тестування (test:stress)..... 109
4.4.2 Спайк-тестування (test:spike)..... 110
4.4.3 Ручне тестування веб- та мобільного клієнтів..... 111
4.5 Висновки з розділу 4..... 112
ВИСНОВКИ..... 116
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ..... 117

ВСТУП

Актуальність теми

У сучасних умовах цифрової трансформації освіти навчальні заклади, приватні школи, корпоративні університети та онлайн-платформи стикаються з необхідністю організації ефективного, гнучкого та масштабованого навчального процесу. Зростання обсягів навчального контенту, поява нових форм взаємодії між викладачами й студентами, а також потреба в безперервному навчанні вимагають застосування інструментів, що здатні забезпечувати централізоване керування освітніми матеріалами, відстеження прогресу користувачів та підтримку різних форматів навчання.

Системи управління навчанням (Learning Management Systems, LMS) є ключовим інструментом для розв’язання цих задач. Вони дозволяють створювати й структуризувати курси, організовувати доступ до навчальних матеріалів, проводити тестування та контролювати успішність. Однак значна частина існуючих LMS-платформ характеризується високою вартістю впровадження, складністю кастомізації, прив’язкою до певних постачальників або застарілими технологічними рішеннями. Такі системи нерідко є монолітними, важкими у підтримці та потребують залучення різнопрофільних команд розробників для супроводу веб-версії, серверної частини та мобільних клієнтів.

Однією з актуальних тенденцій у галузі розробки програмного забезпечення є перехід до використання єдиного технологічного стеку для фронтенду, бекенду та мобільних додатків. Використання JavaScript/TypeScript як основної мови програмування дозволяє поєднати в межах одного проєкту веб-інтерфейс (React), серверну частину (Nest.js) та мобільні застосунки (React Native). Такий підхід сприяє зменшенню кількості технологій, що застосовуються, спрощує обмін знаннями між членами команди та дає можливість уникнути дублювання бізнес-логіки в різних компонентах системи.

Застосування єдиного стеку технологій безпосередньо впливає на економічні аспекти розробки: зменшується потреба в окремих спеціалістах під різні мови програмування, скорочується час на впровадження нових функцій, спрощується оновлення та рефакторинг коду. Це особливо важливо для освітніх організацій та невеликих команд розробників, які не мають можливості підтримувати велику кількість різнорідних технологій. Крім того, уніфікація технологічного стеку позитивно відображається на якості програмного продукту, оскільки дає змогу застосовувати однакові підходи до проєктування, тестування й документування всіх компонентів системи.

Окремої уваги потребує питання доступності навчальних матеріалів на різних типах пристроїв. Сучасні користувачі очікують, що вони зможуть працювати з курсами як через браузер на комп’ютері чи ноутбуку, так і за допомогою мобільних телефонів та планшетів. Крос-платформна LMS, яка забезпечує єдиний користувацький досвід на вебі й у мобільних застосунках, сприяє підвищенню залученості студентів, зменшенню бар’єрів входу в навчальний процес та покращенню загальної ефективності освітньої діяльності.

У цьому контексті особливо актуальним є створення крос-платформної системи управління навчанням, розробленої на базі React, Nest.js та React Native з використанням єдиного стеку JavaScript/TypeScript. Такий підхід дозволяє не лише реалізувати всі ключові функції LMS (керування курсами, тестування, відстеження прогресу), але й забезпечити економію фінансових та людських ресурсів за рахунок технологічної уніфікації, полегшення підтримки та можливості поетапного розширення системи.

Таким чином, тема розробки крос-платформної LMS-системи на основі єдиного технологічного стеку є актуальною з точки зору сучасних вимог до цифрової освіти, тенденцій розвитку програмної інженерії та необхідності оптимізації витрат на створення й підтримку освітніх програмних продуктів.

Мета і завдання дослідження

Метою дослідження є розробка та дослідження крос-платформної системи управління навчанням (LMS), створеної з використанням єдиного технологічного стеку JavaScript/TypeScript на базі React, Nest.js та React Native, яка забезпечує доступ до навчальних матеріалів, керування курсами, організацію тестування та відстеження прогресу користувачів через веб- та мобільні інтерфейси, а також сприяє зниженню витрат на розробку та супровід системи.

Для досягнення поставленої мети у роботі необхідно розв’язати такі завдання:

1. Проаналізувати предметну область систем управління навчанням та визначити основні вимоги до сучасних LMS.
2. Провести огляд існуючих рішень (комерційних і відкритих) та виявити їхні переваги й недоліки з точки зору функціональності, технологічної основи та вартості володіння.
3. Дослідити архітектурні підходи до побудови крос-платформних систем, зокрема на базі єдиного технологічного стеку JavaScript/TypeScript.
4. Проаналізувати можливості фреймворків React, Nest.js та React Native для реалізації веб-інтерфейсу, серверної логіки та мобільного застосунку LMS.
5. Спроєктувати архітектуру крос-платформної LMS, визначити основні модулі, ролі користувачів, моделі даних та взаємодію компонентів системи.
6. Реалізувати серверну частину LMS на базі Nest.js з підтримкою реєстрації, автентифікації, авторизації, керування курсами, навчальними матеріалами та прогресом користувачів.
7. Розробити веб-інтерфейс для студентів, викладачів і адміністраторів засобами React з урахуванням вимог до зручності використання (UX/UI).
8. Створити мобільний застосунок на React Native, що забезпечує доступ до основних функцій LMS з мобільних пристроїв.

9. Виконати функціональне, інтеграційне та навантажувальне тестування розробленої системи, проаналізувати результати та оцінити ефективність застосованого підходу.
10. Зробити висновки щодо доцільності використання єдиного технологічного стеку для розробки крос-платформних LMS-систем.

Об’єкт дослідження

Об’єктом дослідження є процес управління навчальними матеріалами, користувачами та освітніми активностями в рамках інформаційної системи управління навчанням.

Предмет дослідження

Предметом дослідження є методи, моделі, програмні засоби та архітектурні підходи до розробки крос-платформної системи управління навчанням з використанням єдиного технологічного стеку JavaScript/TypeScript.

Методи дослідження

Для вирішення поставленої задачі використовуються наступні методи дослідження:

1. Аналіз предметної області – вивчення принципів організації навчального процесу, ролей учасників (студент, викладач, адміністратор) та типових сценаріїв роботи з LMS.
2. Аналіз існуючих рішень – дослідження функціональних можливостей та архітектурних особливостей популярних LMS-платформ (Moodle, Google Classroom, Canvas, Coursera тощо) з метою виявлення вимог до сучасної системи.
3. Аналіз технологій розробки – вивчення можливостей фреймворків React, Nest.js, React Native, а також супутніх бібліотек та інструментів (ORM, системи керування базами даних, засоби аутентифікації).

4. Моделювання та проєктування – побудова UML-діаграм, проєктування структури бази даних, визначення логічних зв’язків між сутностями, формалізація сценаріїв взаємодії користувачів із системою.
5. Розробка програмного забезпечення – реалізація прототипу та повноцінної системи LMS із використанням обраного стеку технологій, застосування принципів модульності та розширюваності.
6. Експериментальне дослідження – проведення функціонального та інтеграційного тестування, перевірка коректності роботи окремих модулів та системи в цілому.
7. Порівняльний аналіз – зіставлення запропонованого рішення з альтернативними підходами з точки зору складності розробки, вартості підтримки та можливостей масштабування.

Наукова новизна одержаних результатів

Наукова новизна одержаних результатів полягає у комплексному застосуванні єдиного технологічного стеку JavaScript/TypeScript для побудови крос-платформної системи управління навчанням, що об’єднує веб-інтерфейс, бекенд-сервіси та мобільний застосунок у межах узгодженої архітектури. У роботі запропоновано підхід до організації логіки LMS, за якого бізнес-правила можуть повторно використовуватися на різних рівнях системи (клієнтському та серверному), що зменшує дублювання коду та спрощує підтримку програмного продукту.

На відміну від традиційних рішень, де фронтенд, бекенд і мобільні застосунки реалізуються з використанням різних мов програмування та фреймворків, запропонований підхід дозволяє розглядати LMS як єдину крос-платформну екосистему. Це відкриває можливість уніфікації підходів до тестування, логування, обробки помилок, а також спрощує інтеграцію нових модулів у систему.

Практичне значення одержаних результатів

Практичне значення одержаних результатів полягає в розробці

крос-платформної системи управління навчанням (LMS), яка забезпечує доступ до навчальних матеріалів, керування курсами та відстеження прогресу користувачів через веб-інтерфейс і мобільний застосунок. Запропонована система побудована на єдиному технологічному стеку JavaScript/TypeScript (React, NestJS, React Native), що дає змогу зменшити витрати на розробку та супровід, спростити впровадження змін і повторне використання бізнес-логіки. Розроблене програмне забезпечення може бути використане у навчальних закладах, корпоративних навчальних центрах та онлайн-школах як основа для організації електронного навчання, а також як прототип для подальшого розширення функціональності (інтеграція з платіжними сервісами, аналітичними модулями, зовнішніми освітніми платформами).

Глосарій

Система управління навчанням (англ. Learning Management System, LMS) — програмний комплекс, що забезпечує створення, зберігання, поширення та оцінювання навчальних матеріалів, а також керування користувачами та їхнім прогресом.

Курс (англ. Course) — структурована сукупність навчальних матеріалів, завдань, тестів та інших активностей, об’єднаних спільною освітньою метою.

Модуль курсу (англ. Module) — логічно завершена частина курсу, що містить певний набір уроків, матеріалів чи завдань.

Мікросервіс (англ. Microservice) — незалежний невеликий сервіс у складі розподіленої системи, який відповідає за окрему бізнес-функцію (наприклад, керування користувачами, курсами чи сповіщеннями) та взаємодіє з іншими сервісами через стандартизований інтерфейс (REST API або брокер повідомлень).

Урок (англ. Lesson) — окрема навчальна одиниця в межах курсу або модуля, яка може включати теоретичні матеріали, приклади, відео та інтерактивні елементи.

Користувач (англ. User) — особа, що взаємодіє з LMS і має певну роль

(студент, викладач, адміністратор тощо).

Роль користувача (англ. User role) — набір прав доступу та дозволів, що визначає можливості користувача в системі (перегляд матеріалів, редагування курсу, керування користувачами тощо).

Фронтенд (англ. Frontend) — клієнтська частина програмної системи, з якою безпосередньо взаємодіє користувач у веб-браузері.

Бекенд (англ. Backend) — серверна частина програмної системи, відповідальна за обробку запитів, реалізацію бізнес-логіки та доступ до бази даних.

REST API (англ. Representational State Transfer Application Programming Interface) — програмний інтерфейс, що дозволяє клієнтським застосункам взаємодіяти з сервером за допомогою HTTP-запитів та уніфікованих ресурсів. JSON (англ. JavaScript Object Notation) — текстовий формат обміну даними, що використовується для передачі структурованої інформації між клієнтом і сервером.

Автентифікація (англ. Authentication) — процес перевірки тотожності користувача, зазвичай на основі логіна та пароля або інших облікових даних.

Авторизація (англ. Authorization) — процес надання або обмеження прав доступу користувача до певних ресурсів чи функцій системи.

РОЗДІЛ 1 ДОСЛІДЖЕННЯ ПРОБЛЕМИ СТВОРЕННЯ КРОС-ПЛАТФОРМНОЇ СИСТЕМИ УПРАВЛІННЯ НАВЧАННЯМ

1.1 Огляд проблеми організації електронного навчання та управління навчальним процесом

Стрімкий розвиток інформаційних технологій, масове поширення високошвидкісного доступу до мережі Інтернет та популяризація онлайн-форм навчання принципово змінили спосіб організації освітнього процесу. Якщо раніше домінувала традиційна очна форма, то сьогодні дедалі частіше застосовуються змішані (blended learning) та повністю дистанційні моделі. Це стосується як формальної освіти (школи, університети), так і неформальної (онлайн-курси, корпоративне навчання, самоосвіта).

Разом із новими можливостями з’являється низка проблем:

- фрагментація навчальних матеріалів між різними каналами: електронна пошта, месенджери, локальні файли, хмарні сховища, соцмережі;
- ускладнений контроль успішності: результати тестів, домашніх завдань, проектів та відвідуваності зберігаються у різних системах або взагалі в ручному вигляді;
- відсутність єдиного інформаційного простору, де студенти, викладачі та адміністрація бачать узгоджену картину навчального процесу;
- відсутність аналітики щодо ефективності курсів, навантаження викладачів, активності студентів;
- проблеми з безпекою та конфіденційністю даних, коли персональна інформація студентів, оцінки та навчальні результати неконтрольовано розповсюджуються через неавтоматизовані канали.

Ці проблеми посилюються тим, що сучасний студент або слухач курсу звик працювати з різних пристроїв: ноутбука, планшета, смартфона. Відповідно, система управління навчанням має забезпечувати:

- доступ із будь-якого місця та пристрою (web + mobile);

- єдиний акаунт для роботи з усіма курсами та матеріалами;
- сталу якість досвіду (UX) незалежно від форм-фактора пристрою.

У такому контексті системи управління навчанням (Learning Management System, LMS) стають центральною ланкою цифрової освітньої інфраструктури. LMS виконує роль «операційної системи» для освітнього процесу, де зосереджуються:

- створення та структура курсів;
- розподіл ролей і прав доступу;
- зберігання та доставка навчального контенту;
- тестування, оцінювання, формування звітності й аналітики;
- комунікація між учасниками (форум, чат, повідомлення).

На рисунку 1 умовно зображено основних учасників освітнього процесу та їхню взаємодію з системою управління навчанням.

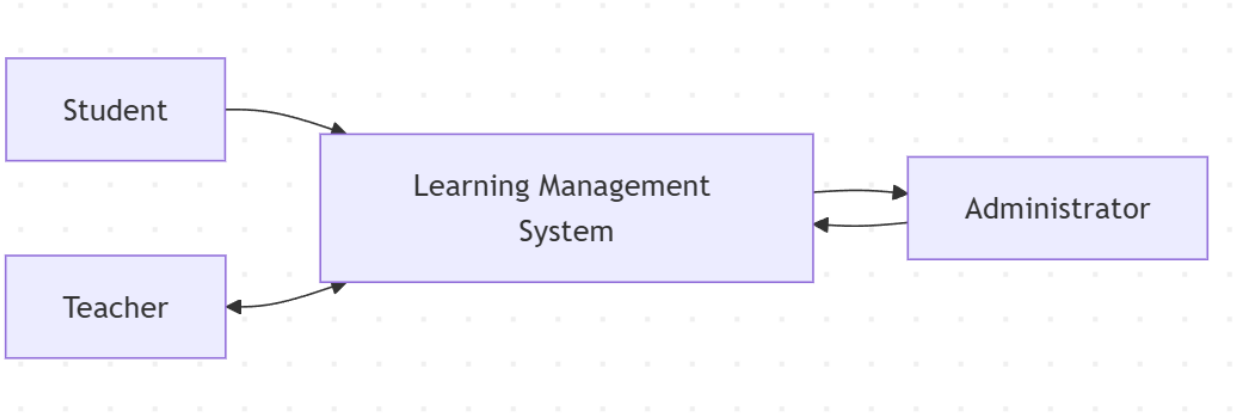


Рисунок 1 — Узагальнена схема взаємодії користувачів із системою управління навчанням

Незважаючи на широкий спектр існуючих рішень, проблема залишається актуальною з кількох причин:

1. Гнучкість під потреби конкретної організаці – багато готових систем надто універсальні й важкі, їх складно адаптувати під специфічні процеси окремого факультету, компанії або онлайн-школи без значних вкладень.

2. Потреба в кросплатформності – сучасному користувачу недостатньо лише веб-інтерфейсу.
3. Вартість розробки та супроводу – при використанні різних мов програмування та фреймворків для фронтенду, бекенду та мобільних клієнтів зростає загальна вартість володіння системою.

Отже, актуальним є завдання розробки крос-платформної LMS, яка з одного боку забезпечує базові сценарії навчання, а з іншого — демонструє економічно доцільну архітектуру на основі єдиного технологічного стеку.

1.2 Технології для створення систем управління навчанням

За час існування електронного навчання сформувались різні підходи до побудови систем управління навчанням, що відрізняються рівнем складності, гнучкістю та можливістю масштабування. Найпоширенішими з них є:

- класичні веб-додатки з серверною генерацією HTML (PHP, Java, ASP.NET);
- монолітні платформи з мінімальним поділом на фронтенд та бекенд;
- модульні системи з плагінною архітектурою, де основна платформа доповнюється розширеннями;
- сучасні SPA/SSR-рішення з чітким розділенням фронтенду та бекенду.

Кожен підхід має свої плюси і мінуси, однак спільною проблемою для більшості «старих» рішень є те, що вони створювалися в епоху, коли:

- не було масової потреби в мобільних додатках;
- не існувало усталених підходів до розподілених систем і мікросервісів;
- JavaScript ще не сприймався як повноцінний «повноцикловий» стек для клієнта й сервера.

Сьогодні в екосистемі JavaScript/TypeScript доступні інструменти, які дозволяють:

- створювати сучасний веб-інтерфейс (React, Next.js);
- реалізовувати структурований серверний застосунок (NestJS);

- розробляти крос-платформні мобільні клієнти (React Native);
- описувати спільні моделі даних на TypeScript, які використовуються і на клієнті, і на сервері.

У таблиці 1 наведено порівняння основних підходів до розробки LMS за критеріями стеку технологій та витрат на підтримку.

Таблиця 1 – Порівняння підходів до розробки LMS за технологічним стеком

Підхід	Мова/платформа	Фронтенд	Мобільні клієнти	Коментар щодо витрат
Java/Java EE	Java	JSP/JSF/Angular Окремі	Android/iOS-команди	Висока надійність, висока ціна входу
.NET-платформа	C#/.NET	ASP.NET + JS	Нативні / Xamarin / MAUI	Сильна інтеграція, часткова прив’язка до екосистеми
Поліглотні рішення	Різні мови (Java, PHP, Go)	React/Vue/Angular	Нативні / Flutter / React Native	Гнучкість, але багато стеків і команд
Єдиний стек JS/TS (запропонований)	JavaScript/TypeScript	React / Next.js	React Native	Зменшення витрат, одна команда, спільні моделі

У рамках дипломної роботи обрано підхід єдиного технологічного стеку

JavaScript/TypeScript.

Це дозволяє:

- використовувати одну мову програмування на всіх рівнях (клієнт, сервер, мобільний застосунок);
- зменшити витрати на навчання нових розробників;
- спростити обмін знаннями всередині команди (розробники можуть легше переходити між фронтендом і бекендом);
- частково розділити код (наприклад, DTO, валідацію, моделі) між різними частинами системи.

На рисунку 2 умовно зображено основних учасників освітнього процесу та їхню взаємодію з системою управління навчанням.

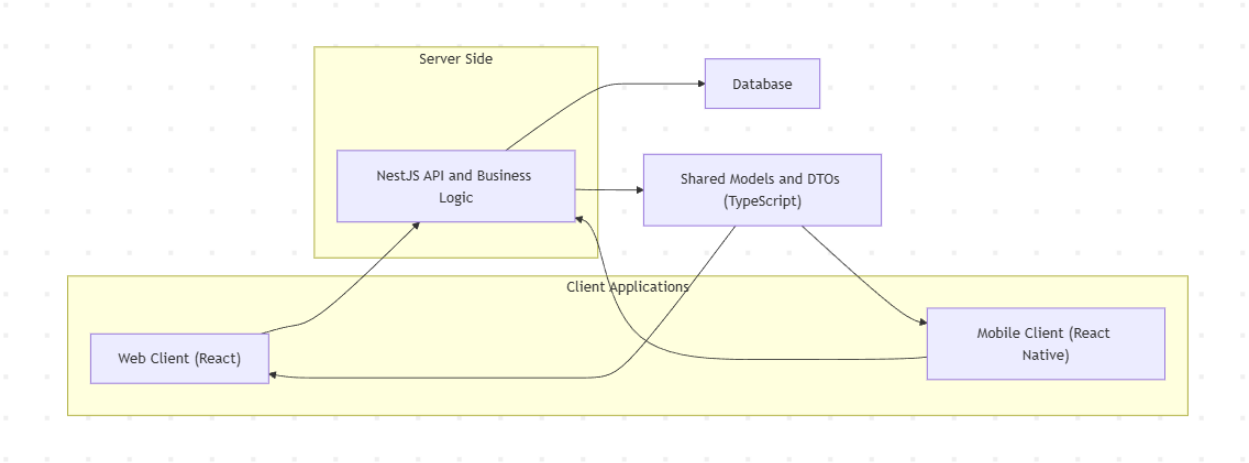


Рисунок 2 — Спрощена архітектура LMS на основі стеку JavaScript/TypeScript

1.3 Сфери застосування систем управління навчанням

LMS є універсальним інструментом, який може застосовуватись у різних контекстах. Основні напрями використання:

1. Формальна освіта (школи, коледжі, університети) –

LMS використовується для:

- створення навчальних курсів та модулів;

- розміщення лекцій, практичних робіт, лабораторних завдань;
- проведення контрольних та екзаменів онлайн;
- ведення електронних журналів та формування підсумкових відомостей;
- організації комунікації між викладачем і студентами.

2. Корпоративне навчання та HR-процеси.

У бізнес-середовищі LMS вирішує такі задачі:

- onboarding нових співробітників (курси «Вступ до компанії», правила, політики);
- регулярне навчання персоналу новим продуктам, сервісам, процесам;
- підготовка до сертифікацій, внутрішні іспити;
- збір статистики про проходження курсів і результати тестів для HR-аналітики.

3. Комерційні освітні платформи та онлайн-школи.

Тут LMS виступає ядром платформи, на основі якої будуються:

- каталог курсів;
- інструменти монетизації (покупка курсів, підписка, промокоди);
- системи рейтингу, відгуків, рекомендацій;
- механізми гейміфікації (бейджі, рівні, досягнення).

4. Самоосвіта та мікронавчання.

Сучасні користувачі очікують можливості проходити короткі модулі «на ходу» — під час поїздки, в перервах між роботою, з мобільного пристрою. LMS, що має повноцінний мобільний клієнт, дозволяє реалізувати мікронавчання, коли студент споживає контент невеликими «порціями».

5. Спеціалізовані сценарії.

Наприклад:

- внутрішні курси для технічної підтримки;
- навчання партнерів та дилерів;
- підготовка волонтерів або учасників великих заходів.

На рисунку 3 наведено узагальнену схему типових сценаріїв

використання LMS.



Рисунок 3 — Типові сценарії використання LMS у формальній, корпоративній та комерційній освіті

Незалежно від галузі, у більшості сценаріїв повторюються ті самі базові вимоги: централізоване зберігання контенту, управління ролями, підтримка тестування, аналітика та, в ідеалі, можливість доступу з різних пристроїв.

1.4 Висновки з розділу 1

У першому розділі було проаналізовано предметну область електронного навчання та визначено ключові вимоги до сучасних систем управління навчанням. На основі проведеного огляду можна зробити такі висновки:

1. Електронне навчання є важливою складовою освітнього процесу, а LMS-системи забезпечують структурування навчального контенту, контроль успішності та підтримку інтерактивної взаємодії.

2. Основні підходи до побудови LMS спираються на модульність, розмежування ролей користувачів, підтримку різних типів навчальних матеріалів та можливість масштабування системи.

3. Сучасні тенденції розвитку LMS показують доцільність використання єдиного технологічного стеку, що дозволяє уніфікувати веб-клієнт, бекенд та

мобільні застосунки. Використання JavaScript/TypeScript на всіх рівнях — ефективний спосіб зменшити витрати на розробку та спростити підтримку.

4. Важливими вимогами до проєктування LMS є: модульна архітектура, наявність чітко визначених ролей користувачів, можливість масштабування, підтримка крос-платформності та забезпечення гнучкості для подальшої адаптації системи.

5. Наведені результати формують основу для розробки власної крос-платформної системи управління навчанням та слугують підґрунтям для архітектурних рішень, що будуть розглянуті у наступних розділах.

РОЗДІЛ 2 ДОСЛІДЖЕННЯ ТЕХНОЛОГІЙ ТА ІНСТРУМЕНТІВ ДЛЯ СТВОРЕННЯ КРОС-ПЛАТФОРМНОЇ СИСТЕМИ УПРАВЛІННЯ НАВЧАННЯМ

2.1 Загальні вимоги до технологічного стеку крос-платформної LMS

У попередньому розділі було показано, що сучасні системи управління навчанням мають забезпечувати безперервний доступ до навчального контенту незалежно від місця перебування студента, типу пристрою та часу доби. Це означає, що LMS повинна коректно працювати:

- у настільних браузерах (Chrome, Firefox, Edge тощо);
- у мобільних браузерах;
- у вигляді окремого мобільного застосунку для Android та iOS.

Крім цього, система має підтримувати декілька ролей користувачів (адміністратор, викладач, студент), інтегруватися з іншими сервісами й залишатися розширюваною у майбутньому. Усе це напряму впливає на вимоги до технологічного стеку.

2.1.1 Функціональні вимоги

Нефункціональні вимоги визначають не стільки функціональність системи, скільки якість її роботи, стабільність та здатність ефективно обслуговувати користувачів у різних умовах. До ключових нефункціональних характеристик крос-платформної LMS належать:

- *Управління користувачами.*
  - реєстрація та авторизація користувачів;
  - розподіл на ролі (адміністратор, викладач, студент);
  - можливість блокування, зміни пароля, редагування профілю.
- *Управління курсами.*
  - створення, редагування, архівація курсів;
  - налаштування структури: модулі, теми, уроки;

- прив’язка викладачів до курсів;
- налаштування доступу для студентів (платний/безоплатний доступ, групи, потоки).

- *Управління навчальним контентом.*
  - додавання текстових матеріалів, презентацій, PDF-файлів;
  - вбудовування відео (посилання на YouTube, власне відеосховище);
  - інтерактивні матеріали (тести, опитування, завдання з відкритою відповіддю).
- *Оцінювання та зворотний зв’язок.*
  - створення тестів різних типів (одна правильна відповідь, декілька правильних, відповідність, відкрита відповідь);
  - налаштування шкал оцінювання;
  - перегляд статистики проходження тестів;
  - можливість залишати коментарі до завдань.
- *Сповіщення та комунікація.*
  - повідомлення про нові завдання та дедлайни;
  - нагадування про прострочені завдання;
  - обговорення в межах курсу (коментарі, форуми, чати).
- *Аналітика та звітність.*
  - перегляд прогресу окремого студента;
  - перегляд загальної статистики курсу;
  - експорт звітів у поширені формати (CSV, Excel, PDF).

2.1.2 Нефункціональні вимоги

Нефункціональні вимоги визначають загальні характеристики системи та впливають на її стабільність, ефективність і безпечність у реальних умовах експлуатації. До основних нефункціональних аспектів розроблюваної LMS належать:

- *Масштабованість***.**

Система повинна підтримувати зростання кількості курсів і користувачів без істотного зниження продуктивності. Це накладає вимоги до вибору СУБД, архітектури бекенду та можливості горизонтального масштабування.

- *Продуктивність*.
  - час відгуку інтерфейсу має залишатися прийнятним навіть при великій кількості запитів;
  - операції, що виконуються часто (відображення списку курсів, завдань), бажано кешувати.
- *Надійність*.
  - забезпечення цілісності даних (транзакції, коректна обробка помилок);
  - відновлення після збоїв (резервне копіювання, механізми відкату).
- *Безпека*.
  - захист персональних даних користувачів;
  - безпечна авторизація (JWT, refresh-токени);
  - захист від типових веб-загроз (SQL-ін’єкції, XSS, CSRF тощо).
- *Крос-платформеність.*
  - автоматична адаптація інтерфейсу під екрани різного розміру (responsive design);
  - підтримка мобільних застосунків, що працюють із тим самим API.
- *Супровідність.*
  - чітка архітектура та модульна структура;
  - наявність документації (опис API, схеми БД, інструкції з розгортання).

2.1.3 Узагальнена архітектура системи

Загальна архітектура крос-платформної LMS охоплює кілька взаємопов’язаних рівнів, кожен з яких виконує свою роль у забезпеченні функціональності, масштабованості та зручності супроводу системи. Їх можна

подати у такому вигляді:

1. *Рівень представлення (клієнт)*
    - веб-інтерфейс на основі React;
    - мобільний застосунок на React Native.
2. *Рівень прикладної логіки (сервер)*
    - REST API на NestJS, що інкапсулює бізнес-логіку.
3. *Рівень доступу до даних*
    - PostgreSQL як основна СУБД;
    - за потреби – Redis для кешування.
4. *Інфраструктурний рівень*
    - система контролю версій;
    - CI/CD-процеси;
    - системи моніторингу та логування.

2.2 Додаткові інструменти та забезпечення якості

2.2.1 Контроль версій та командна робота

Для впорядкування процесу розробки та забезпечення узгодженої роботи команди використовується система контролю версій Git. Вона дає змогу керувати змінами в проєкті та підтримувати структуровану історію розробки.

Репозиторій можна розділити на каталоги:

- backend/ – код NestJS;
- web/ – код React;
- mobile/ – код React Native;
- docs/ – документація, діаграми, SQL-скрипти.

2.2.2 Контейнеризація та розгортання

Для спрощення розгортання системи, уніфікації середовища виконання та забезпечення відтворюваності інфраструктури використовується технологія

контейнеризації Docker.

2.2.3 CI/CD та автоматизація

Для автоматизації процесу розробки використовується CI/CD, реалізований за допомогою GitHub Actions. Під час кожного оновлення коду система автоматично виконує перевірки, збірку застосунку та його тестове розгортання, що забезпечує стабільність і передбачуваність релізів.

2.2.4 Тестування та моніторинг

Для забезпечення стабільності та надійності системи застосовуються юніт- та інтеграційні тести, які перевіряють коректність бізнес-логіки та роботу REST API. Додатково використовується моніторинг, що дозволяє відстежувати помилки й продуктивність у реальному часі через логування та сповіщення.

2.3 Типові сценарії використання крос-платформної LMS

Типові сценарії взаємодії різних категорій користувачів дають можливість оцінити, наскільки система відповідає поставленим вимогам та чи забезпечує вона необхідний рівень зручності, функціональності та доступності. Нижче наведено опис основних сценаріїв роботи ключових ролей LMS.

2.3.1 Робота студента

Роль студента є центральною для будь-якої системи електронного навчання, оскільки саме цей користувач безпосередньо взаємодіє з навчальним контентом, виконує завдання та проходить оцінювання. Сценарій роботи студента демонструє послідовність дій, які він виконує під час навчального процесу.

1. Студент заходить у систему через веб або мобільний застосунок.

2. Переглядає список доступних курсів.
3. Записується на курс (якщо дозволено самостійне зарахування).
4. Проходить уроки, виконує завдання, проходить тести.
5. Отримує сповіщення про нові завдання та дедлайни.
6. Переглядає свій прогрес та підсумкові оцінки.

2.3.2 Робота викладача

Викладач є відповідальним за формування навчального контенту, керування курсами та аналіз успішності студентів. Тому сценарій роботи викладача відображає процес створення навчального матеріалу, адміністрування курсу та взаємодії зі студентами.

1. Викладач створює новий курс.
2. Додає модулі, уроки, завдання та тести.
3. Налаштовує критерії оцінювання.
4. Відстежує активність студентів, переглядає статистику.
5. За потреби – коригує зміст курсу.

2.3.3 Робота адміністратора

Адміністратор відповідає за технічне та організаційне функціонування системи, забезпечує стабільність роботи платформи, керує користувачами та налаштуваннями. Сценарій роботи адміністратора демонструє основні операції, необхідні для підтримки LMS у працездатному стані.

1. Створює облікові записи викладачів.
2. Призначає ролі та права доступу.
3. Контролює загальні налаштування системи.
4. Формує звіти для керівництва (наприклад, про активність користувачів).

2.4 Архітектурні підходи до побудови крос-платформної LMS

Архітектура програмної системи визначає, наскільки просто її можна розвивати, масштабувати та супроводжувати. Для крос-платформної LMS, яка має веб-інтерфейс, мобільний клієнт та серверну частину з базою даних, важливо забезпечити:

- чіткий поділ відповідальностей між компонентами;
- можливість незалежного розвитку окремих модулів;
- мінімізацію зв’язаності між частинами системи;
- підтримку як синхронної (HTTP/REST), так і асинхронної взаємодії (події, повідомлення).

У сучасній практиці розробки LMS найчастіше використовуються шарові, модульні та мікросервісні архітектурні підходи, які можуть комбінуватися між собою.

2.4.1 Шарова та модульна організація системи

Класичний підхід до побудови складних інформаційних систем – *шарова архітектура* (layered architecture), де кожен шар має чітко визначену роль:

- *Шар представлення (presentation layer)* – відповідає за взаємодію з користувачем (веб-інтерфейс, мобільний клієнт, API-шлюз).
- *Прикладний шар (application layer)* – координує виконання бізнес-операцій, обробляє команди та запити, керує транзакціями.
- *Доменний шар (domain layer)* – містить предметну логіку: правила зарахування на курс, обчислення результатів тестів, перевірку дедлайнів тощо.
- *Інфраструктурний шар (infrastructure layer)* – забезпечує роботу з базою даних, зовнішніми сервісами, файловим сховищем, чергами повідомлень.

Такий поділ дозволяє:

- змінювати технології інфраструктури (СУБД, брокер повідомлень, механізм відправки e-mail) без зміни доменної логіки;
- використовувати кілька клієнтів (веб, мобільний застосунок), що звертаються до однакових прикладних сервісів;
- ізолювати бізнес-правила від деталей реалізації.

У сучасних проектах шарова модель часто поєднується з *модульним підходом*, коли система додатково ділиться на доменні модулі: «Користувачі», «Курси», «Тести», «Прогрес», «Нотифікації» тощо. Це полегшує:

- розподіл задач між командами;
- повторне використання модулів;
- поступовий перехід до мікросервісної архітектури за потреби.

2.4.2 Монолітна, модульна та мікросервісна архітектури: порівняння

Історично більшість LMS будувались як *монолітні застосунки*, де весь код (фронтенд, бекенд, бізнес-логіка, робота з БД) розміщувався в одному великому проєкті. Такий підхід спрощує початкову розробку, але ускладнює масштабування й супровід, коли система зростає.

Сучасні підходи передбачають перехід до *модульних монолітів* та *мікросервісів*. У модульному моноліті код розділено на логічні модулі з чіткими межами, але вони все ще розгортаються як один застосунок. У мікросервісній архітектурі кожен модуль перетворюється на окремий сервіс із власною базою даних, інтерфейсом взаємодії та життєвим циклом.

У таблиці 2.6 наведено узагальнене порівняння трьох підходів з точки зору розробки LMS.

Таблиця 2.6 – *Порівняння архітектурних підходів для LMS*

Критерій	**Моноліт**	**Модульний моноліт**	**Мікросервіси**
Початкова складність	Низька	Помірна	Висока
Структурування коду	Часто слабке	Чіткі модулі всередині одного застосунку	Чіткі межі сервісів
Масштабування	Масштабування всього застосунку	Можливе часткове (по компонентах)	Незалежне масштабування окремих сервісів
Ізоляція збоїв	Низька – збій може «покласти» все	Краща, але в межах одного процесу	Висока – падіння одного сервісу не зупиняє інші
Вимоги до інфраструктури	Мінімальні	Мінімальні/помірні	Підвищені (оркестрація, моніторинг, DevOps)
Придатність для невеликих проектів	Висока	Висока	Обмежена
Придатність для великого навантаження	Обмежена	Помірна	Висока

Для навчальних проєктів та перших версій продукту часто обирають модульний моноліт із чітким виділенням доменів. У разі зростання навантаження або розширення функціональності частину модулів можна поступово виділяти в окремі мікросервіси.

2.4.3 Синхронна та асинхронна взаємодія компонентів LMS

Компоненти LMS можуть взаємодіяти між собою *синхронно* та *асинхронно*.

- *Синхронна взаємодія* зазвичай реалізується через HTTP/REST API. Вона підходить для операцій, де користувач очікує миттєвий результат: авторизація, отримання списку курсів, надсилання відповідей на тест.
- *Асинхронна взаємодія* базується на подіях і чергах повідомлень. Коли в системі відбувається певна подія (наприклад, студент завершив тест), замість безпосереднього виклику інших модулів LMS може надіслати повідомлення до брокера, а зацікавлені сервіси його оброблять із затримкою.

Поширеними брокерами повідомлень є *Apache Kafka, RabbitMQ, NATS* тощо. Їх використання в LMS дозволяє:

- розвантажити основні сервіси, виконуючи частину операцій у фоновому режимі (наприклад, формування складних звітів);
- підвищити стійкість до пікових навантажень;
- легше підключати нові сервіси, які «підписуються» на вже наявні події (наприклад, сервіс нотифікацій, що реагує на подію «нове завдання створено»).

Для крос-платформної LMS характерними прикладами подій можуть бути:

- UserRegistered – новий користувач зареєстрований у системі;
- CourseEnrolled – студент записаний на курс;
- AssignmentSubmitted – завдання здано;
- TestCompleted – тест завершено й оцінено;
- NotificationRequested – необхідно надіслати сповіщення.

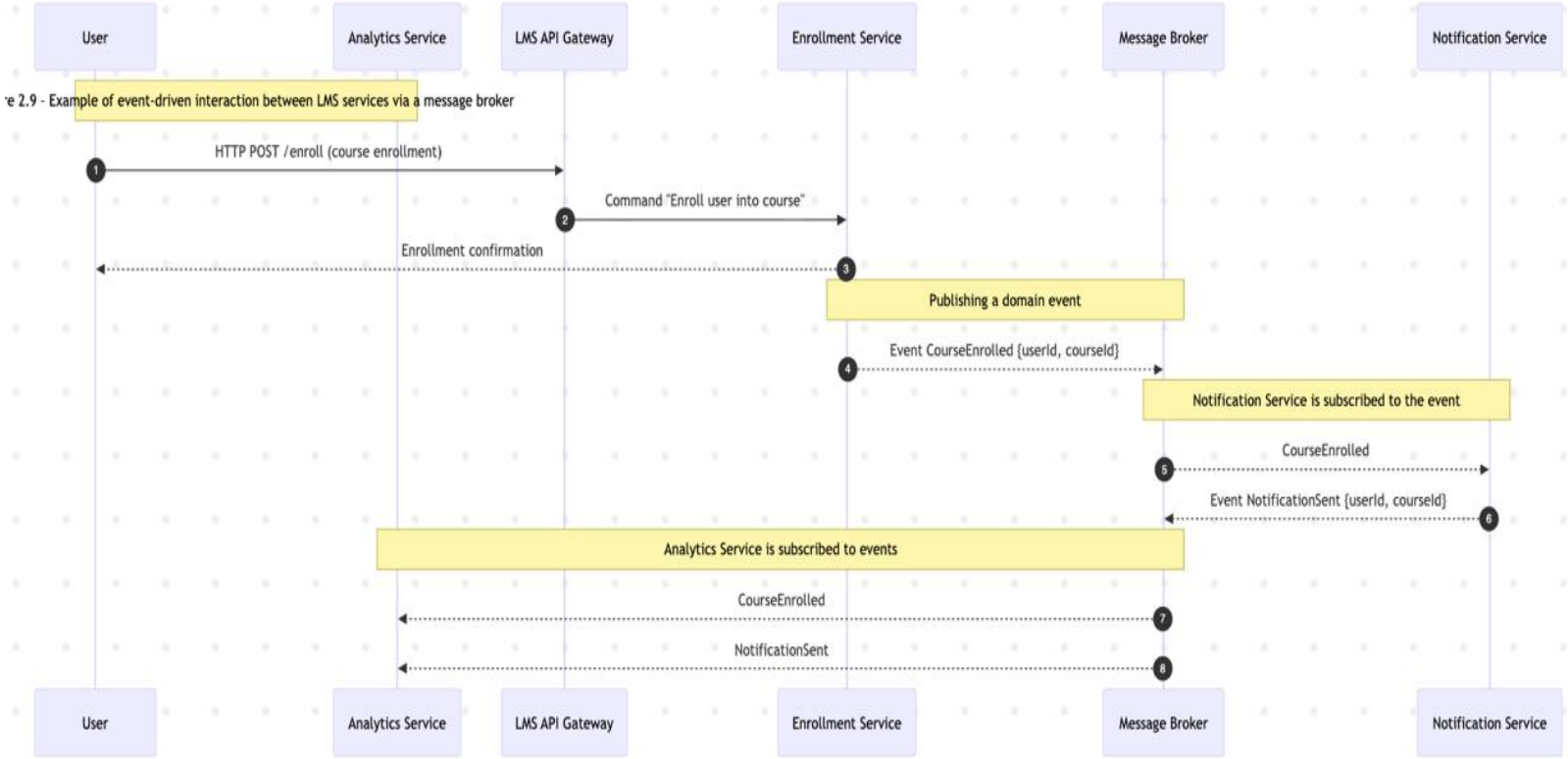


Рисунок 2.9 – *Приклад подійної взаємодії сервісів LMS через брокер повідомлень*

Комбінація синхронних REST-запитів та асинхронної подійної взаємодії дозволяє побудувати гнучку та стійку до навантажень архітектуру LMS, яка з часом може масштабуватися без кардинальної переробки всієї системи.

2.5 Висновки з розділу 2

1. У другому розділі було досліджено технології, стандарти та інструменти, необхідні для створення крос-платформної LMS, а також сформульовано її основні функціональні (ролі користувачів, управління курсами та тестами, прогрес, сповіщення, аналітика) та нефункціональні вимоги (масштабованість, продуктивність, безпека, крос-платформеність, супровідність коду).

2. Розглянуто стандарти електронного навчання SCORM, xAPI та LTI й показано, що, попри їх часткову відсутність у першій версії системи, архітектура має залишатися готовою до подальшої інтеграції.

3. На основі аналізу веб-фреймворків обґрунтовано використання React (разом із Vite) для веб-інтерфейсу, NestJS — для серверної частини з сервісно-орієнтованою побудовою та API-шлюзом, React Native — для мобільного клієнта, а PostgreSQL — як основної реляційної СУБД із можливим кешуванням у Redis.

4. Також розглянуто допоміжні інструменти (Git, Docker, CI/CD, тестування, моніторинг), що забезпечують якість і надійність розробки. У підсумку сформовано цілісний технологічний стек, який у наступному розділі використовується для практичної реалізації LMS.

РОЗДІЛ 3 ПРОЄКТУВАННЯ ТА РОЗРОБКА КРОС-ПЛАТФОРМНОЇ СИСТЕМИ УПРАВЛІННЯ НАВЧАННЯМ

У цьому розділі виконується практична реалізація крос-платформної системи управління навчанням (LMS) на основі обраного технологічного стеку JavaScript/TypeScript. Описано налаштування середовища розробки, загальну архітектуру системи, структуру бази даних, а також реалізацію серверної частини на базі NestJS, веб-інтерфейсу на React та мобільного клієнта на React Native. Окрему увагу приділено взаємодії між компонентами системи, організації бізнес-логіки та зберіганню навчальних даних.

Результатом розділу є працюючий програмний комплекс, що включає бекенд-сервіси, веб-додаток та мобільний застосунок, об’єднані єдиною архітектурою та єдиним API. У наступному розділі буде проведено тестування розробленої системи та аналіз її характеристик.

3.1 Налаштування середовища розробки системи

Для реалізації крос платформної LMS було обрано єдиний технологічний стек JavaScript/TypeScript. Це дозволило використовувати спільні підходи до організації коду на всіх рівнях системи та спростило взаємодію між веб додатком, серверною частиною та мобільним клієнтом.

3.1.1 Налаштування середовища для навчання

Для коректної розробки та тестування крос-платформної LMS було підготовлено робоче середовище, яке забезпечує необхідну продуктивність і стабільність роботи інструментів розробника. Основні характеристики обладнання виглядали так:

- багато-ядерний процесор класу Intel Core i5 / i7 або еквівалент AMD;
- не менше 16 ГБ оперативної пам’яті;
- твердотільний накопичувач (SSD) для прискорення роботи з

репозиторієм; операційна система сімейства Windows 10 / 11 або одна з сучасних Linux-дистрибуцій.

Для коректної роботи усіх компонентів системи були встановлені такі програмні засоби:

- Node.js (версія не нижче 18.x) — для запуску бекенд-сервісів на NestJS, веб-додатку на React/Vite та мобільного застосунку на React Native (через Expo);
- Yarn або npm — для керування залежностями JavaScript/TypeScript-проєктів; PostgreSQL — як основна реляційна система керування базами даних;
- Docker та Docker Compose (опційно) — для підняття інфраструктурних сервісів (PostgreSQL, Redis, Kafka) у вигляді контейнерів;
- середовище розробки (IDE) — Visual Studio Code або WebStorm з підтримкою TypeScript та інтеграцією з Git.

Наявність цих інструментів забезпечує можливість локально запускати всі три частини системи (бекенд, веб-клієнт, мобільний клієнт) та відлагоджувати їхню взаємодію.

3.1.2 Структура репозиторію проекту

Уся кодова база системи об’єднана в єдиний Git-репозиторій, що спрощує організацію розробки, контролю версій, повторне використання коду та узгодження змін між різними частинами системи. Репозиторій має чітку модульну структуру та складається з трьох основних кореневих каталогів:

- *backend/* — серверна частина, побудована на основі NestJS та реалізована як монорепозиторій, що включає декілька доменних мікросервісів і центральний API-шлюз;
- *frontend/* — веб-інтерфейс системи у вигляді односторінкового застосунку (SPA), реалізованого на React з використанням Vite;
- *mobile/* — мобільний застосунок на базі React Native (Expo), який взаємодіє з тією ж серверною API-логікою, що й веб-клієнт.

Така структура дозволяє чітко розділити відповідальність між компонентами, водночас зберігаючи їх у межах спільного репозиторію, що значно спрощує командну розробку та підтримку системи.

У каталозі *backend/* розташований NestJS-монорепозиторій, який містить такі підкаталоги:

- *backend/apps/api-gateway* — API-шлюз, що надає REST-інтерфейс для веб- і мобільного клієнтів та взаємодіє з внутрішніми сервісами через Kafka;
- *backend/apps/user* — сервіс керування користувачами та автентифікацією;
- *backend/apps/course* — сервіс для роботи з курсами, модулями та уроками;
- *backend/apps/progress* — сервіс відстеження прогресу користувачів у межах навчальних програм;
- *backend/apps/notification* — сервіс формування та надсилання сповіщень;
- *backend/libs/core* — набір спільних модулів: конфігураційні файли, базові сутності, утиліти загального призначення, middleware, фільтри обробки винятків та інші елементи, що використовуються кількома сервісами.

Лістинг 1 — Структура каталогу backend

```
backend/
  apps/
    api-gateway/
      app.module.ts
      main.ts
      core/
    user/
      core/
    course/
      core/
    progress/
      core/
```

```
    notification/
      core/
  libs/
    core/
      src/
        configuration/
        shared-kernel/
        rest/
      tsconfig.lib.json
```

Структура каталогу *frontend/* відображає поділ веб-інтерфейсу відповідно до доменних областей та функціональних модулів системи. Основні підкаталоги мають таке призначення:

- *frontend/src/pages* — сторінки для різних ролей користувачів (автентифікація, панель студента, панель викладача, панель адміністратора);
- *frontend/src/components* — багаторазово використовувані інтерфейсні компоненти: формові елементи, модальні вікна, таблиці, навігаційні панелі тощо;
- *frontend/src/api* — модулі для роботи з REST-інтерфейсом бекенду (наприклад: auth.api.ts, course.api.ts, user.api.ts, progress.api.ts);
- *frontend/src/stores* — сховища стану на основі бібліотеки Zustand (модулі керування автентифікацією, даними курсів, тестами, інформацією про користувача, UI-станом);
- *frontend/src/routes* — конфігурація маршрутизації (файл AppRoutes.tsx), де визначено доступні маршрути, доступ за ролями та правила переходів між сторінками.

Нижче наведено структуру каталогу *frontend/src*, що ілюструє організацію вихідного коду інтерфейсу:

Лістинг 2 — Структура каталогу frontend/src

```
frontend/src/
  App.tsx
  main.tsx
  routes/
    AppRoutes.tsx
  pages/
    auth/
    student/
    instructor/
    admin/
  components/
    ProtectedRoute.tsx
    Notifications.tsx
  api/
    client.ts
    auth.api.ts
    course.api.ts
  stores/
    auth.store.ts
    course.store.ts
  theme/
  utils/
```

Каталог *mobile/* містить вихідний код мобільного клієнта, який розроблено на базі React Native (Expo) і організовано аналогічно до структури веб-додатку. Основні елементи:

- *mobile/App.tsx* — кореневий компонент застосунку, де ініціалізуються провайдери стану, тема та навігація;
- *mobile/src/navigation* — конфігурація стекової та таб-навігації (файли RootNavigator.tsx, MainTabNavigator.tsx, а також типи параметрів маршрутів);

- *mobile/src/screens* — екрани мобільного застосунку (автентифікація, список курсів, перегляд курсу, відображення уроків, проходження тестів, редагування профілю тощо);
- *mobile/src/api* — модулі доступу до REST API, структуровані подібно до веб-версії;
- *mobile/src/stores* — сховища стану користувача та теми інтерфейсу.

Структура каталогу наведена нижче:

Лістинг 3 — Структура каталогу mobile/src

```
mobile/src/
  navigation/
    RootNavigator.tsx
    MainTabNavigator.tsx
  screens/
    LoginScreen.tsx
    CoursesScreen.tsx
    CourseDetailScreen.tsx
    LessonViewScreen.tsx
    TestTakingScreen.tsx
  api/
    client.ts
    auth.api.ts
    course.api.ts
  stores/
    auth.store.ts
    theme.store.ts
  config/
    api.config.ts
  utils/
```

3.1.3 Налаштування та запуск бекенд-частини

Серверна частина LMS побудована на фреймворку NestJS з використанням TypeORM для доступу до бази даних PostgreSQL, Redis для кешування та Kafka як брокера повідомлень між доменними сервісами. Для полегшення розгортання інфраструктурних компонентів застосовується Docker Compose.

Налаштування бекенд-частини включає такі кроки:

1. Перехід до каталогу бекенду та встановлення залежностей:
   cd backend
   yarn install
2. Підняття інфраструктурних сервісів (PostgreSQL, Redis, Kafka та допоміжних контейнерів) за допомогою Docker Compose:
   yarn compose:db
3. Застосування міграцій бази даних для кожного доменного сервісу.

У файлі backend/package.json визначено відповідні скрипти:

- *migration:run:user* — застосування міграцій для сервісу користувачів;
- *migration:run:course* — застосування міграцій для сервісу курсів;
- *migration:run:progress* — застосування міграцій для сервісу прогресу;
- *migration:run:notification* — застосування міграцій для сервісу сповіщень.

Лістинг 4 — Фрагмент файлу backend/package.json із командами міграцій

```
"scripts": {
  "compose:db": "docker-compose up -d postgres-user
postgres-course postgres-progress postgres-notification
redis zookeeper kafka",
  "start:all": "rimraf dist && concurrently \"yarn
start:dev api-gateway\" \"yarn start:dev user\" \"yarn
start:dev course\" \"yarn start:dev progress\" \"yarn
start:dev notification\"",
  "migration:run:user": "ts-node -r tsconfig-
```

```
paths/register apps/user/migration-data-source.ts
migration:run",
      "migration:revert:user": "ts-node -r tsconfig-
paths/register apps/user/migration-data-source.ts
migration:revert",
      "migration:generate:user": "ts-node -r tsconfig-
paths/register apps/user/migration-data-source.ts
migration:generate apps/user/migrations/$npm_config_name",
      "migration:run:course": "ts-node -r tsconfig-
paths/register apps/course/migration-data-source.ts
migration:run",
      "migration:revert:course": "ts-node -r tsconfig-
paths/register apps/course/migration-data-source.ts
migration:revert",
      "migration:generate:course": "ts-node -r tsconfig-
paths/register apps/course/migration-data-source.ts
migration:generate
apps/course/migrations/$npm_config_name",
      "migration:run:progress": "ts-node -r tsconfig-
paths/register apps/progress/migration-data-source.ts
migration:run",
      "migration:revert:progress": "ts-node -r tsconfig-
paths/register apps/progress/migration-data-source.ts
migration:revert",
      "migration:generate:progress": "ts-node -r tsconfig-
paths/register apps/progress/migration-data-source.ts
migration:generate
apps/progress/migrations/$npm_config_name",
      "migration:run:notification": "ts-node -r tsconfig-
paths/register apps/notification/migration-data-source.ts
migration:run",
      "migration:revert:notification": "ts-node -r
tsconfig-paths/register apps/notification/migration-data-
```

```
source.ts migration:revert",
      "migration:generate:notification": "ts-node -r
tsconfig-paths/register apps/notification/migration-data-
source.ts migration:generate
apps/notification/migrations/$npm_config_name"
    }
```

4. Запуск усіх бекенд-сервісів у режимі розробки:

   yarn start:all

   API-шлюз стає доступним за адресою: http://localhost:3000

3.1.4 Налаштування та запуск веб-інтерфейсу

Веб-клієнт LMS реалізовано як SPA на базі React + Vite. Для виконання HTTP-запитів використовується Axios, а для керування кешем на боці клієнта — @tanstack/react-query.

Кроки запуску:

1. Перехід у каталог frontend та встановлення залежностей:

   ```
   cd frontend
   yarn install
   ```

2. Налаштування конфігурації середовища у .env:

```
VITE_API_BASE_URL=http://localhost:3000
VITE_API_KEY=your_api_key
```

3. Запуск у режимі розробки:

   ```
   yarn dev
   ```

Після цього веб-додаток доступний за адресою: http://localhost:5173

HTTP-клієнт створено в модулі frontend/src/api/client.ts.

Лістинг 5 — Фрагмент файлу backend/package.json із командами міграцій

```
import axios, { AxiosError } from "axios";
import type { AxiosInstance, InternalAxiosRequestConfig }
from "axios";
import { useAuthStore } from "../stores";
```

```
const API_BASE_URL =
  import.meta.env.VITE_API_BASE_URL ||
"http://localhost:3000";
const API_KEY = import.meta.env.VITE_API_KEY;

export const apiClient: AxiosInstance = axios.create({
  baseURL: `${API_BASE_URL}/api`,
  headers: {
    ...(API_KEY && { "X-API-KEY": API_KEY }),
  },
  timeout: 10000,
});

apiClient.interceptors.request.use(
  (config: InternalAxiosRequestConfig) => {
    const token = useAuthStore.getState().accessToken;
    if (token && config.headers) {
      config.headers.Authorization = `Bearer ${token}`;
    }

    if (
      config.headers &&
      !config.headers["Content-Type"] &&
      !(config.data instanceof FormData)
    ) {
      config.headers["Content-Type"] = "application/json";
    }

    return config;
  },
  (error: AxiosError) => {
    return Promise.reject(error);
  }
);

apiClient.interceptors.response.use(
  (response) => response,
  async (error: AxiosError) => {
    const originalRequest = error.config as
InternalAxiosRequestConfig & {
      _retry?: boolean;
    };

    const isAuthEndpoint =
originalRequest.url?.includes("/auth/");

    if (
      error.response?.status === 401 &&
      !originalRequest._retry &&
```

```
      !isAuthEndpoint
    ) {
      originalRequest._retry = true;

      try {
        const success = await
useAuthStore.getState().refreshAccessToken();

        if (!success) {
          useAuthStore.getState().clearAuth();
          window.location.href = "/login";
          return Promise.reject(error);
        }

        const token = useAuthStore.getState().accessToken;
        if (originalRequest.headers && token) {
          originalRequest.headers.Authorization = `Bearer
${token}`;
        }
        return apiClient(originalRequest);
      } catch (refreshError) {
        useAuthStore.getState().clearAuth();
        window.location.href = "/login";
        return Promise.reject(refreshError);
      }
    }

    return Promise.reject(error);
  }
);

export default apiClient;
```

3.1.5 Налаштування та запуск мобільного застосунку

Мобільний клієнт створено на основі React Native з використанням платформи Expo. Для зберігання токенів застосовується expo-secure-store.

Кроки запуску:

1. Перехід у каталог mobile та встановлення залежностей:

cd mobile

yarn install

2. Налаштування параметрів підключення у mobile/src/config/api.config

3. Запуск Expo-середовища:

cd mobile

yarn install

Мобільний клієнт використовує власний Axios-клієнт із токен-refresh логікою.

Лістинг 6 — Модуль mobile/src/api/client.ts (повний фрагмент)

```
import axios, {
  AxiosError,
  AxiosInstance,
  InternalAxiosRequestConfig,
} from "axios";
import * as SecureStore from "expo-secure-store";
import { API_BASE_URL, API_TIMEOUT, API_KEY } from
"../config/api.config";

export const apiClient: AxiosInstance = axios.create({
  baseURL: `${API_BASE_URL}/api`,
  headers: {
    "Content-Type": "application/json",
    ...(API_KEY && { "X-API-KEY": API_KEY }),
  },
  timeout: API_TIMEOUT,
});

// Request interceptor - adds token to every request
apiClient.interceptors.request.use(
  async (config: InternalAxiosRequestConfig) => {
    const token = await
SecureStore.getItemAsync("accessToken");
    if (token && config.headers) {
      config.headers.Authorization = `Bearer ${token}`;
    }
    return config;
  },
  (error: AxiosError) => {
    return Promise.reject(error);
  }
);

// Response interceptor - automatically refreshes token on
401
apiClient.interceptors.response.use(
  (response) => response,
  async (error: AxiosError) => {
    const originalRequest = error.config as
InternalAxiosRequestConfig & {
      _retry?: boolean;
    };
```

```
    const isAuthEndpoint =
originalRequest.url?.includes("/auth/");

    if (
      error.response?.status === 401 &&
      !originalRequest._retry &&
      !isAuthEndpoint
    ) {
      originalRequest._retry = true;

      try {
        const refreshToken = await
SecureStore.getItemAsync("refreshToken");
        if (!refreshToken) {
          throw new Error("No refresh token");
        }

        // Refresh token
        const response = await axios.post(
          `${API_BASE_URL}/api/auth/refresh-token`,
          {
            refreshToken,
          }
        );

        const { accessToken, refreshToken: newRefreshToken
} = response.data;

        // Save new tokens
        await SecureStore.setItemAsync("accessToken",
accessToken);
        await SecureStore.setItemAsync("refreshToken",
newRefreshToken);

        if (originalRequest.headers) {
          originalRequest.headers.Authorization = `Bearer
${accessToken}`;
        }
        return apiClient(originalRequest);
      } catch (refreshError) {
        await SecureStore.deleteItemAsync("accessToken");
        await SecureStore.deleteItemAsync("refreshToken");
        return Promise.reject(refreshError);
      }
    }

    return Promise.reject(error);
  }
```

```
);

export default apiClient;
```

3.1.6 Підсумок

Після виконання всіх наведених кроків бекенд-сервіси, веб-додаток та мобільний застосунок можуть бути одночасно запущені на локальному середовищі та взаємодіяти через єдиний API-шлюз. Це забезпечує узгодженість логіки, уніфіковану систему авторизації та можливість повноцінного тестування всієї LMS.

3.2 Загальна архітектура системи

3.2.1 Компоненти системи та інфраструктура

Розроблена система управління навчанням має багаторівневу архітектуру, у якій кожен компонент відповідає за свою частину функціональності. До складу системи входять такі основні компоненти.

API-шлюз (API Gateway) - реалізований у вигляді застосунку backend/apps/api-gateway на базі NestJS. Приймає всі HTTP-запити від веб- та мобільного клієнтів, відповідає за маршрутизацію запитів до внутрішніх сервісів, застосовує глобальні *middleware*, інтерсептори та фільтри винятків. API-шлюз також виконує перевірку API-ключа та автентифікацію користувачів.

Сервіс користувачів (User Service) - застосунок backend/apps/user. Відповідає за управління обліковими записами користувачів, їхніми ролями та сесіями. Реалізує функції реєстрації, входу в систему, оновлення токенів, верифікації електронної пошти та скидання пароля. Дані зберігаються в таблицях user та session бази даних PostgreSQL.також виконує перевірку API-ключа та автентифікацію користувачів.

Сервіс курсів (Course Service) - Застосунок backend/apps/course. Інкапсулює логіку роботи з навчальним контентом: курсами, модулями,

уроками, категоріями та тестами. Забезпечує створення й редагування курсів, організацію структури модулів і уроків, керування категоріями, питаннями тестів та варіантами відповідей. Основні таблиці бази даних: *course, course_module, lesson, category, course_categories, test, question, option, test_attempt, test_attempt_answer, test_attempt_session.*

Сервіс прогресу (Progress Service) - Застосунок backend/apps/progress. Відповідає за облік навчального прогресу користувачів. Зберігає інформацію про відсоток опанування курсу, статуси проходження уроків і тестів, кількість спроб та найкращий результат. Використовуються таблиці *user_course_progress, lesson_state, test_state.*

Сервіс сповіщень (Notification Service) - Застосунок *backend/apps/notification*. Займається зберіганням і обробкою сповіщень для користувачів (про нові курси, наближення дедлайнів, результати тестів тощо). Дані зберігаються в таблиці notification, де фіксуються тип сповіщення, заголовок, текст, додаткові дані та статус прочитання.

Спільна бібліотека ядра (Core Library) - Каталог backend/libs/core. Містить спільні для всіх сервісів компоненти: конфігураційні налаштування, базовий клас сутності (BaseEntity), інтерсептори та фільтри винятків, спільні DTO та утиліти, middleware для перевірки токенів та API-ключа, а також модулі інтеграції із зовнішніми сервісами (наприклад, Firebase).

Реляційна база даних PostgreSQL - Використовується як основне сховище структурованих даних системи. Кожен доменний сервіс має власні сутності та набір міграцій, при цьому всі вони можуть розміщуватися в одній або в кількох схемах однієї інстанції PostgreSQL. Доступ до бази даних здійснюється за допомогою ORM-бібліотеки TypeORM.

Брокер повідомлень Kafka - Забезпечує обмін повідомленнями між API-шлюзом і доменними сервісами, а також обробку асинхронних подій (оновлення прогресу, формування та надсилання сповіщень). Використання Kafka дозволяє розвантажити API-шлюз та чітко розмежувати відповідальність між сервісами.

Кеш та допоміжні інфраструктурні сервіси - Як кешуючий шар використовується Redis для зберігання тимчасових даних, реалізації обмеження частоти запитів (throttling) або певних частин користувацького стану. Запуск PostgreSQL, Redis та Kafka в середовищі локальної розробки здійснюється за допомогою Docker Compose.

Веб-клієнт (Web Client) - Каталог frontend/. Односторінковий веб-додаток на React (Vite), що надає інтерфейс для студентів, викладачів та адміністраторів: перегляд і проходження курсів, редагування навчального контенту, керування користувачами, категоріями та тестами. Взаємодіє з API-шлюзом через REST API.

Мобільний клієнт (Mobile Client) - Каталог mobile/. Мобільний застосунок на React Native (Expo), орієнтований насамперед на студентів. Забезпечує доступ до основних функцій LMS з мобільних пристроїв: перегляд курсів і модулів, проходження уроків та тестів, перегляд власного прогресу та отримання сповіщень. Використовує ті самі REST-ендпоінти API-шлюзу, що й веб-клієнт.

3.2.2 API-шлюз та маршрутизація запитів

Усі зовнішні HTTP-запити до LMS проходять через API-шлюз, реалізований у застосунку backend/apps/api-gateway на базі NestJS. API-шлюз забезпечує єдину точку входу для веб- та мобільного клієнтів і виконує такі основні функції:

- Приймає всі запити за єдиним префіксом: http://localhost:3000/api/
- застосовує глобальні *middleware* (зокрема, перевірку службового API-ключа, переданого в заголовку X-API-KEY, та логування запитів);
- виконує автентифікацію користувача за JWT-токеном через TokenAuthMiddleware, який звертається до сервісу користувачів (USER_SERVICE) через Kafka-канал auth.verify-token;
- делегує бізнес-запити відповідним доменним сервісам (user, course, progress, notification) у режимі «запит-відповідь» поверх брокера

повідомлень Kafka;

- уніфікує обробку помилок за допомогою глобальних інтерсепторів та фільтрів, що перетворюють винятки мікросервісів у стандартизовані HTTP-відповіді.

Конфігурація API-шлюзу виконується у файлах main.ts та app.module.ts застосунку api-gateway, де:

- створюється HTTP-сервер;
- реєструються контролери (наприклад, /auth, /course, /progress, /notification);
- налаштовуються Kafka-клієнти для кожного доменного сервісу;
- підключаються глобальні фільтри, інтерсептори та middleware.

3.2.3 Доменні сервіси та їх взаємодія

Бізнес-логіка LMS розділена між кількома доменними сервісами, які працюють як окремі NestJS-застосунки в каталозі backend/apps. Кожен сервіс відповідає за власну предметну область та має окремі модулі, контролери, схеми БД та канали обміну повідомленнями.

User Service (backend/apps/user)

Відповідає за роботу з користувачами, ролями, автентифікацією та сесіями.

Основні функції:

- перевірка JWT-токена,
- видача токенів доступу,
- оновлення токенів,
- керування сесіями користувача.

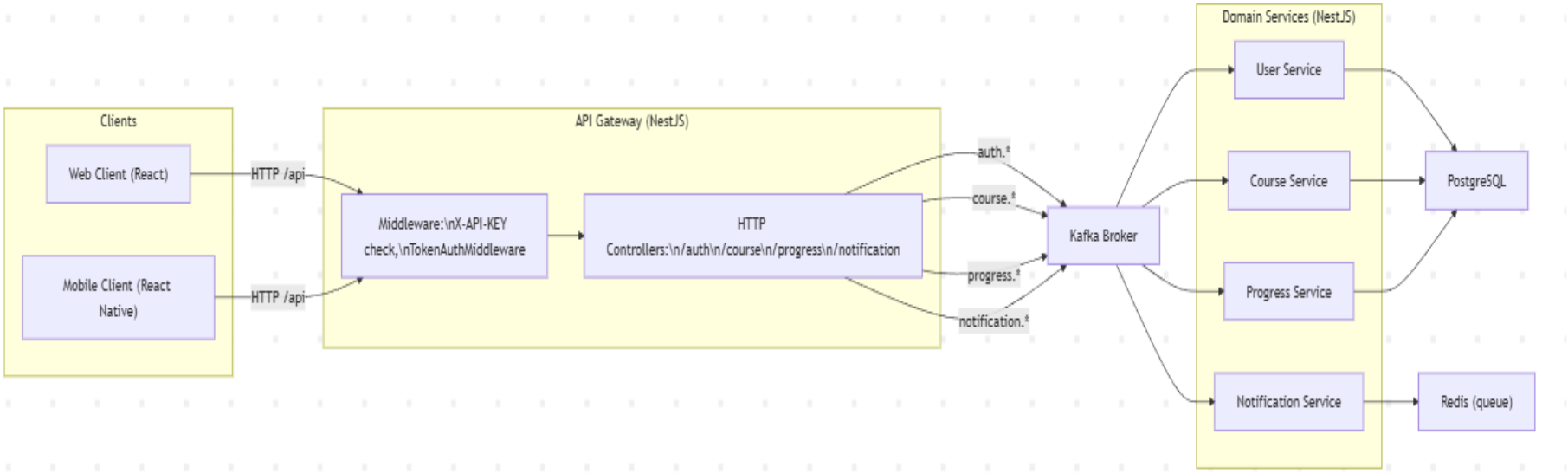


Рисунок 3.1 — *Схема маршрутизації запитів через API-шлюз*

Лістинг 3.8 — Обробка повідомлення auth.verify-token у сервісі користувачів

```
// backend/apps/user/core/components/auth/primary-
adapters/controllers/auth.controller.ts

@Controller()
export class AuthController {
 constructor(
 private readonly tokenValidationService:
TokenValidationService,
  ) {}

  @MessagePattern('auth.verify-token')
  async verifyToken(@Payload() payload: { token: string })
{
    const result = await
this.tokenValidationService.validateToken({
      token: payload.token,
    });

    return {
      id: result.user.id,
      sessionId: result.sessionId,
      role: result.user.role,
      email: result.user.email,
      phoneNumber: result.user.phoneNumber,
      firstName: result.user.firstName,
      lastName: result.user.lastName,
      image: result.user.image,
      onboard: result.user.onboard,
    };
  }
}
```

Course Service (backend/apps/course)

Обробляє домен навчального контенту:

курси, модулі, уроки, категорії, тести, питання, варіанти відповідей, спроби проходження тестів. Канали: course.*, test.*.

Лістинг 3.9 — Обробка повідомлення course.list.all у сервісі курсів

```
// backend/apps/course/core/components/course/primary-
adapters/course.kafka.controller.ts

@Controller()
```

```
export class CourseController {
  constructor(
    private readonly listAllCoursesUseCase:
ListAllCoursesUseCase,
  ) {}

  @MessagePattern('course.list.all')
  async listAll(
    @GetUser() user: User,
    @PageQuery() pageQuery: NormalizedPageQuery,
    @Payload()
    payload?: {
      authorId?: string;
      categoryIds?: string[];
      level?: CourseLevel;
    },
  ): Promise<Paginated<Course>> {
    return this.listAllCoursesUseCase.execute({
      page: pageQuery.page,
      limit: pageQuery.limit,
      search: pageQuery.search,
      authorId: payload?.authorId,
      categoryIds: payload?.categoryIds,
      level: payload?.level,
      userId: user?.id,
    });
  }
}
```

Лістинг 3.10 — Обробка події progress.lesson.completed у сервісі прогресу

```
// backend/apps/progress/core/components/progress/primary-
adapters/progress.kafka.controller.ts

@Controller()
export class ProgressController {
  constructor(private readonly svc:
ProgressAggregateService) {}

  @MessagePattern('progress.lesson.completed')
  @UseGuards(AuthGuard)
  @ApiBearerAuth(API_JWT_TOKEN_NAME)
  async lessonCompleted(
    @Payload() payload: { userId: string; courseId: string;
lessonId: string },
  ) {
    return this.svc.markLessonCompleted(payload);
  }
}
```

Лістинг 3.11 — Створення сповіщень у сервісі сповіщень

```
//
backend/apps/notification/core/components/notification/prim
ary-adapters/notification.kafka.controller.ts

@Controller()
export class NotificationController {
  constructor(private readonly service:
NotificationService) {}

  @MessagePattern('notification.create')
  async create(
    @Payload()
    payload: {
      userId: string;
      type: string;
      title: string;
      body: string;
      data?: any;
    },
  ) {
    return this.service.createNotification(payload);
  }
}
```

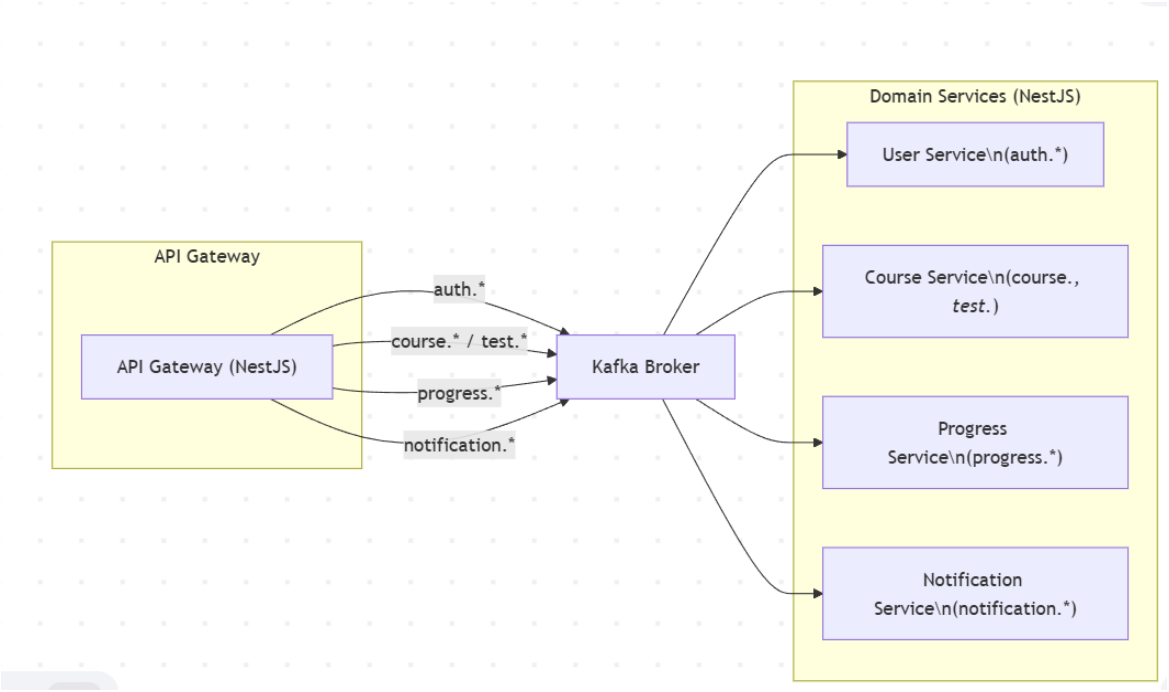


Рисунок 3.2 — *Схема роботи брокера*

3.2.4 Налаштування середовища для впровадження системи

Дані в системі LMS зберігаються в реляційній базі даних *PostgreSQL*. Структура бази даних спроєктована на основі доменної моделі та реалізована через сутності *TypeORM* у сервісах *user*, *course*, *progress* та *notification*. Для аналізу моделі даних використано утиліту *backend/tools/generate-er.ts,* яка на основі метаданих TypeORM створює ER-діаграми у форматі *Mermaid* (*backend/er-diagrams/*.mmd*). На їх основі побудовано інтегральну ER-діаграму бази даних LMS.

Основні групи таблиць:

Користувачі та сесії:

- *user* — інформація про користувача (e-mail, роль, профільні дані, стани верифікації).
- *session* — активні сесії / refresh-токени, прив'язані до користувача (зв'язок *«один-до-багатьох»* між *user* та *session*).

Навчальний контент:

- *course* — курси (назва, опис, рівень, автор, тривалість, зображення, банер, статус публікації).
- *course_module* — модулі курсу (назва, опис, порядок, посилання на курс).
- *lesson* — уроки (назва, опис, контент, тривалість, порядок, статус публікації, ознака прев'ю, посилання на модуль, URL відео).
- *category* та зв'язуюча таблиця *course_categories* — категорії курсів і зв'язок *«багато-до-багатьох»* між курсами та категоріями.

Тестування

- *test* — тести, прив'язані до модулів (часові обмеження, поріг проходження, кількість спроб, режими випадкового порядку).
- *question* — питання тесту (тип, текст, зображення, порядок, ваговий коефіцієнт, додаткові параметри).
- *option* — варіанти відповідей (текст, ознака правильності, порядок).
- *test_attempt, test_attempt_answer, test_attempt_session* — спроби

проходження тестів, відповіді в межах спроби та проміжні сесії тестування.

Прогрес навчання:

- *user_course_progress* — агрегований прогрес користувача по кожному курсу
  (кількість завершених елементів, загальна кількість, відсоток, статус *not_started / in_progress / completed*).
- *lesson_state* — стани уроків (користувач, урок, статус, тривалість перегляду).
- *test_state* — стани тестів (користувач, тест, найкращий бал, кількість спроб, ознака проходження, посилання на останню спробу).

Сповіщення:

- *notification* — сповіщення для користувачів (тип, заголовок, текст, додаткові дані, пріоритет, категорія, статус прочитання).

Ключові зв’язки між таблицями:

- користувач (user) має багато записів у таблицях session, user_course_progress, lesson_state, test_state, test_attempt, notification;
- курс (*course*) має багато модулів *(course_module),* а через них — уроки (*lesson*) і тести (*test*);
- курс (*course)* пов’язаний із категоріями (*category*) через зв’язуючу таблицю *course_categories*;
- тест (*test*) має багато питань (*question*), а кожне питання — багато варіантів (*option*);
- таблиці прогресу (*user_course_progress, lesson_state, test_state*) логічно пов’язані з таблицями *user, course, lesson, test* через зовнішні ключі.

3.2.5 Автентифікація та авторизація в системі

У розробленій LMS автентифікація побудована на *механізмі JSON Web Token (JWT)***.**

Після успішного входу *User Service* генерує пару токенів:

- *accessToken* — короткочасний токен, який використовується для підпису кожного запиту;
- *refreshToken* — довготривалий токен для оновлення сесії.

Refresh-токени зберігаються у таблиці *session*, прив’язані до користувача за полем *userId*, що дає змогу централізовано керувати активними сесіями.

Веб-клієнт зберігає токени у *localStorage*, а мобільний застосунок — у *SecureStore*.

Під час доступу до захищених ресурсів клієнт додає заголовок:

Authorization: Bearer <access_token>

На рівні API Gateway цей заголовок перевіряється middleware TokenAuthMiddleware, яке через Kafka-канал auth.verify-token викликає User Service для валідації токена.

Якщо токен валідний — у запит додається інформація про користувача.

У разі помилки повертається 401 Unauthorized.

Авторизація реалізована на основі ролей, що зберігаються у полі role таблиці *user* (наприклад: *Student*, *Teacher*, *Admin*).

На backend застосовуються guard’и та декоратори ролей, які контролюють доступ до маршрутів.

На frontend — компонент ProtectedRoute, який приховує недоступні маршрути для поточної ролі.

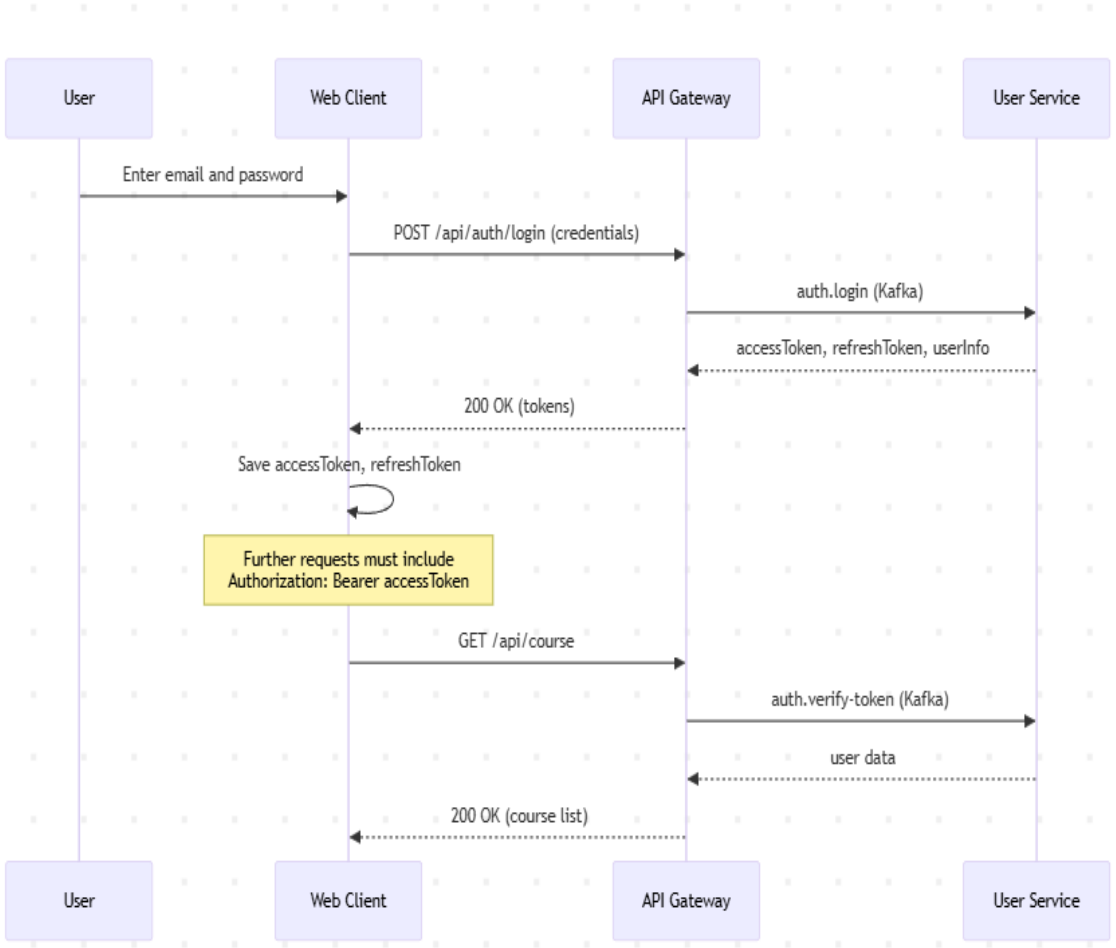


Рисунок 3.3 — Sequence-діаграма процесу автентифікації

3.2.6 Клієнтські застосунки та обмін даними з сервером

У системі LMS реалізовано два основні клієнтські застосунки: веб-клієнт на React (frontend/) та мобільний клієнт на React Native (mobile/). Обидва використовують спільний REST API, що надається API-шлюзом за префіксом */api/*, тому логіка роботи з даними є повністю уніфікованою між платформами.

Веб-клієнт побудований як односторінковий застосунок (SPA) на базі React із використанням Vite. Маршрутизація організована через бібліотеку *react-router-dom* (конфігурація знаходиться в *AppRoutes.tsx*), а доступ до серверних ресурсів здійснюється через модуль *apiClient* (Axios) у поєднанні з *React Query* для кешування й оптимізації мережевих запитів. Для контролю стану автентифікації застосовується сховище Zustand (*useAuthStore*), а компонент *ProtectedRoute* обмежує доступ до сторінок залежно від ролі

користувача та наявності валідного токена.

Мобільний клієнт реалізований на React Native (Expo) і використовує *@react-navigation* для створення стекової та таб-навігації (компоненти *RootNavigator* та *MainTabNavigator*). Для взаємодії з API застосовується окремий *apiClient*, який автоматично додає JWT-токен із SecureStore до заголовка *Authorization* кожного запиту. У разі отримання відповіді 401 мобільний клієнт виконує запит на оновлення токенів через ендпоінт */auth/refresh-token*, що забезпечує прозоре продовження сеансу користувача.

Основні сценарії мобільного застосунку — перегляд курсів, деталей курсу, уроків і тестів — використовують ті самі REST-ендпоінти, що й веб-клієнт.

На рисунку 3.6 схематично показано взаємодію веб-клієнта, мобільного клієнта та API-шлюзу.

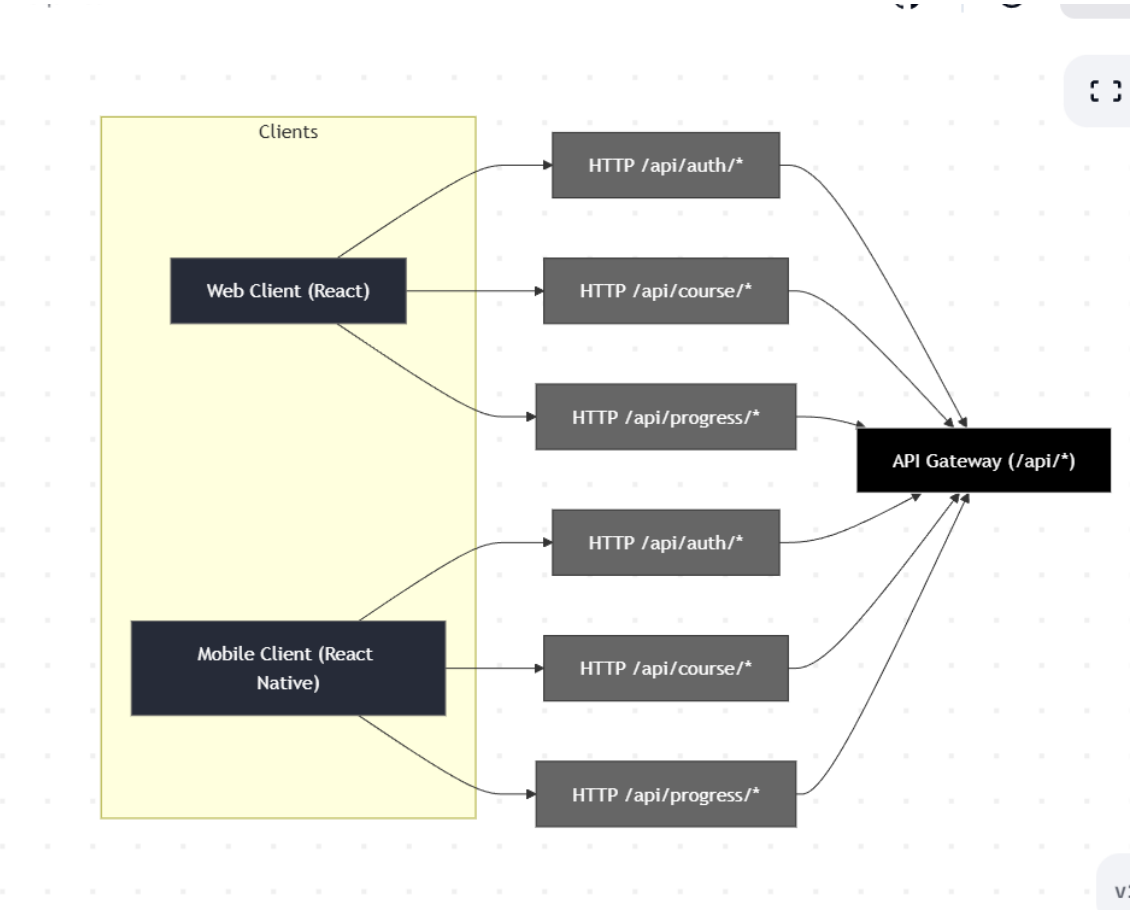


Рисунок 3.4 — Взаємодія веб-клієнта, мобільного клієнта та API-шлюзу

3.2.7 Типові потоки даних у системі

Для демонстрації роботи архітектури розглянемо типовий сценарій взаємодії студента з навчальним контентом: відкриття уроку, його завершення та оновлення прогресу в системі.

Користувач у веб- або мобільному клієнті відкриває курс і обирає урок. Клієнт надсилає запит до API-шлюзу (GET */api/course/lessons/:id*), який перенаправляє його до Course Service через Kafka-подію *course.lesson.get*. Сервіс курсів отримує дані уроку з бази даних і повертає їх назад до API-шлюзу, після чого клієнт отримує відповідь *200 OK* із вмістом уроку.

Після завершення перегляду клієнт надсилає запит позначити урок як завершений (POST */api/course/lessons/:id/complete*). API-шлюз транслює цей запит до Progress Service у вигляді Kafka-події *progress.lesson.completed*. Сервіс прогресу оновлює записи в таблицях *user_course_progress* та *lesson_state*, після чого повертає підтвердження через API-шлюз у вигляді відповіді *200 OK*.

Аналогічний механізм використовується під час завершення тесту: Course Service генерує подію *progress.test.attempt.finished*, на основі якої Progress Service оновлює *test_state* та агрегований прогрес курсу. За необхідності система може надіслати користувачу сповіщення через Notification Service.

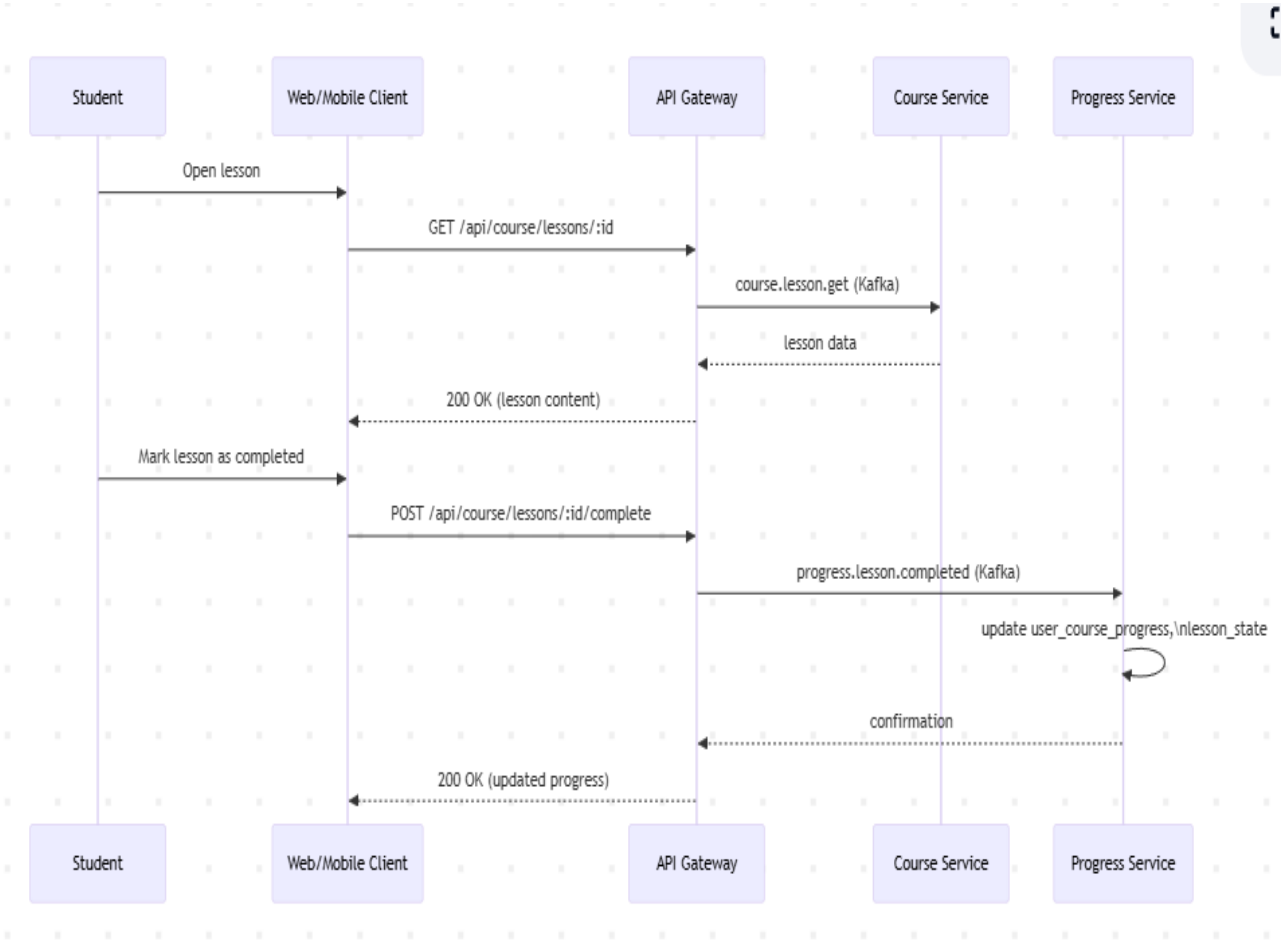


Рисунок 3.5 — Послідовність дій при проходженні уроку й оновленні прогресу

3.3 Архітектура серверної частини

Серверна частина LMS реалізована у вигляді NestJS-монорепозиторію каталозі *backend/*. Код поділено на окремі застосунки в каталозі *apps/* (API-шлюз, доменні сервіси *user*, *course*, *progress*, *notification*) та спільну бібліотеку *libs/core*, яка містить базові сутності, конфігурацію, інтерфейси та загальні утиліти.

Архітектура кожного сервісу побудована згідно з типовим для NestJS шаровим підходом:

- *Module* — об’єднує контролери, сервіси, репозиторії, підключає зовнішні модулі (TypeORM, Kafka-клієнти).
- *Controller* — приймає HTTP-або Kafka-повідомлення, викликає бізнес-логіку сервісів та use case’ів.
- *UseCase/Service* — інкапсулює бізнес-логіку домену.
- *Repository/Adapter* — відповідає за доступ до бази даних через TypeORM, реалізує порти доменної логіки.
- *Entity* — описує структуру таблиць та зв’язки в базі даних.

Спільний базовий клас сутності *BaseEntity* у *libs/core* містить стандартні поля *id*, *createdAt*, *updatedAt*, які успадковують усі доменні сутності.

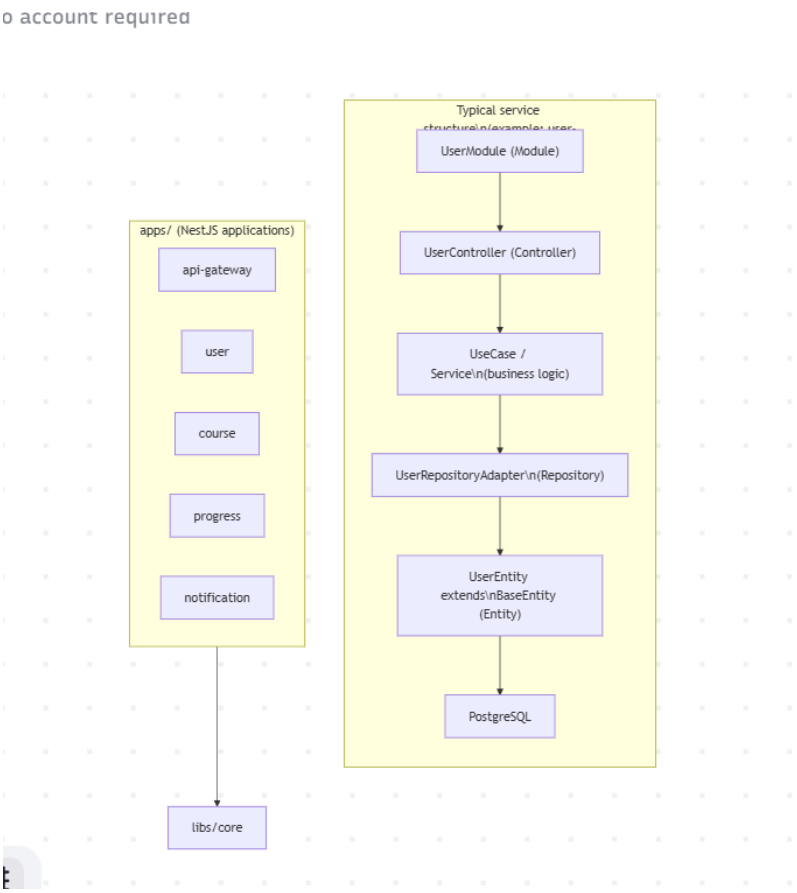


Рисунок 3.6 — *Внутрішня архітектура серверної частини LMS*

Лістинг 3.12 — Модуль UserModule у сервісі користувачів

```
// backend/apps/user/core/components/user/user.module.ts

import { AuthModule } from
'@apps/user/core/components/auth/auth.module';
import { Module, forwardRef } from '@nestjs/common';
import { ClientsModule, Transport } from
'@nestjs/microservices';
import { TypeOrmModule } from '@nestjs/typeorm';

import { SessionEntity } from '../auth/secondary-
adapters/postgres/data/session.entity';

import { CreateUserUseCase } from
'./application/usecases/create-user.usecase';
import { DeleteUserUseCase } from
'./application/usecases/delete-user.usecase';
import { EditUserUseCase } from
'./application/usecases/edit-user.usecase';
import { GetAllUsersUseCase } from
'./application/usecases/get-all-users.usecase';
```

```
import { GetUserByEmailUseCase } from
'./application/usecases/get-user-by-email.usecase';
import { GetUserByIdUseCase } from
'./application/usecases/get-user-by-id.usecase';
import { UserRepositoryType } from './ports/user-
repository.port';
import { UserController } from './primary-
adapters/user.kafka.controller';
import { UserEntity } from './secondary-
adapters/postgres/data/user.entity';
import { UserRepositoryAdapter } from './secondary-
adapters/postgres/repository/user-repository.adapter';

@Module({
  imports: [
    TypeOrmModule.forFeature([UserEntity, SessionEntity]),
    forwardRef(() => AuthModule),
    ClientsModule.register([
      {
        name: 'COURSE_SERVICE',
        transport: Transport.KAFKA,
        options: {
          client: {
            brokers: (process.env.KAFKA_BROKERS ||
'localhost:9092').split(','),
            clientId: 'user-service-course-client',
          },
          consumer: {
            groupId: 'user-service-course-consumer',
          } as any,
          producer: {
            allowAutoTopicCreation: true,
          },
        },
      },
    ]),
  ],
  controllers: [UserController],
  providers: [
    CreateUserUseCase,
    EditUserUseCase,
    DeleteUserUseCase,
    GetAllUsersUseCase,
    GetUserByEmailUseCase,
    GetUserByIdUseCase,
    {
      provide: UserRepositoryType,
      useClass: UserRepositoryAdapter,
    },
```

```
  ],
  exports: [
    UserRepositoryType,
    GetUserByEmailUseCase,
    GetUserByIdUseCase,
  ],
})
export class UserModule {}
```

Лістинг 3.13 — Базовий клас сутності BaseEntity

```
// backend/libs/core/src/configuration/base.entity.ts

import {
  CreateDateColumn,
  PrimaryGeneratedColumn,
  UpdateDateColumn,
} from 'typeorm';

export class BaseEntity {
  @PrimaryGeneratedColumn('uuid')
  id: string;

  @CreateDateColumn({ name: 'created_at' })
  createdAt: string;

  @UpdateDateColumn({ name: 'updated_at' })
  updatedAt: string;
}
```

Ці приклади демонструють типовий підхід до побудови модулів у серверній частині системи: модулі підключають сутності та залежності, контролери відповідають за транспортний рівень, use case’и реалізують бізнес-логіку,а репозиторії забезпечують роботу з базою даних через спільні доменні сутності.

3.3.1 Реалізація сервісу користувачів

Сервіс користувачів (*backend/apps/user*) відповідає за зберігання та обробку даних користувачів, їхніх ролей і сесій, а також виконує основні операції автентифікації: реєстрацію, вхід до системи, перевірку токенів та їх оновлення. Дані зберігаються в таблицях *user* та *session*, структура яких

визначена відповідними сутностями *UserEntity* та *SessionEntity*. Уся бізнес-логіка сервісу зосереджена в use case-класах (наприклад, *CreateUserUseCase*, *EditUserUseCase*, *GetUserByEmailUseCase*), що взаємодіють із базою даних через адаптер репозиторію *UserRepositoryAdapter*.

Сутність *UserEntity* успадковує типові поля ідентифікатора та часових міток від базового класу *BaseEntity*. Вона містить повний набір властивостей користувача: ім’я, прізвище, номер телефону, e-mail, захешований пароль, роль, стан онбордингу, ознаку верифікації електронної пошти та службові токени для підтвердження e-mail і скидання пароля.

Лістинг 3.14 — Сутність UserEntity

```
// backend/apps/user/core/components/user/secondary-
adapters/postgres/data/user.entity.ts

import { Column, Entity, OneToMany } from 'typeorm';

import { BaseEntity } from
'backend/libs/core/src/configuration/base.entity';
import { Fillable } from 'backend/libs/core/src/shared-
kernel/common/decorators/fillable.decorator';
import { Role } from 'backend/libs/core/src/shared-
kernel/common/enums/role.enum';

import { SessionEntity } from
'@apps/user/core/components/auth/secondary-
adapters/postgres/data/session.entity';

@Entity('user')
export class UserEntity extends BaseEntity {
  @Column({ name: 'first_name', type: 'varchar', length:
50, nullable: true })
  firstName: string;

  @Column({ name: 'last_name', type: 'varchar', length: 50,
nullable: true })
  lastName: string;

  @Column({
    name: 'phone_number',
    type: 'varchar',
    length: 20,
    unique: true,
```

```
    nullable: true,
  })
  phoneNumber: string;

  @Column({ type: 'varchar', length: 255, unique: true })
  email: string;

  @Column({ type: 'varchar', length: 255 })
  password: string;

  @Column({ type: 'varchar', nullable: true })
image: string;

  @Column({ type: 'varchar', length: 20, default:
Role.Student })
  role: Role;

  @Column({ type: 'boolean', default: false })
  onboard: boolean;

  @Column({ name: 'email_verified', type: 'boolean',
default: false })
  emailVerified: boolean;

  @Column({
    name: 'email_verification_token',
    type: 'varchar',
    length: 255,
    nullable: true,
  })
  emailVerificationToken: string;

  @Column({
    name: 'reset_password_token',
    type: 'varchar',
    length: 255,
    nullable: true,
  })
  resetPasswordToken: string;

  @Column({
    name: 'reset_password_expires',
    type: 'timestamp with time zone',
    nullable: true,
  })
  resetPasswordExpires: Date;

  @OneToMany(() => SessionEntity, (session) =>
session.user)
```

```
  sessions: SessionEntity[];

  @Fillable
  public static fromObject(_obj: Partial<UserEntity>):
UserEntity {
    return new UserEntity();
  }
}
```

Створення нового користувача виконується через use case *CreateUserUseCase*, який приймає DTO з даними реєстрації, будує доменну модель *User* та зберігає її через репозиторій. За замовчуванням новий користувач отримує роль *Student*, а стани онбордингу та верифікації e-mail ініціалізуються значеннями *false*.

Лістинг 3.15 — Use case створення користувача CreateUserUseCa

```
//
backend/apps/user/core/components/user/application/usecases
/create-user.usecase.ts

import { Injectable, Inject } from '@nestjs/common';

import { Role } from 'backend/libs/core/src/shared-
kernel/common/enums/role.enum';
import { UseCase } from 'backend/libs/core/src/shared-
kernel/common/interface/usecase';

import {
  UserRepositoryType,
  UserRepository,
} from '../../ports/user-repository.port';
import { User } from '../data/user';

export interface CreateUserUseCaseParams {
  email: string;
  password: string;
  phoneNumber?: string;
  firstName?: string;
  lastName?: string;
  role?: Role;
  onboard?: boolean;
  emailVerified?: boolean;
  emailVerificationToken?: string;
}
```

```
@Injectable()
export class CreateUserUseCase
  implements UseCase<CreateUserUseCaseParams, User>
{
  constructor(
    @Inject(UserRepositoryType)
    private readonly userRepository: UserRepository,
  ) {}

  public async execute(params: CreateUserUseCaseParams):
Promise<User> {
    const user = User.build({
      email: params.email,
      password: params.password,
      phoneNumber: params.phoneNumber,
      firstName: params.firstName,
      lastName: params.lastName,
      role: params.role ?? Role.Student,
      onboard: params.onboard ?? false,
      emailVerified: params.emailVerified ?? false,
      emailVerificationToken:
params.emailVerificationToken,
    });

    return await this.userRepository.save(user);
  }
}
```

Ці приклади демонструють реалізацію сервісу користувачів відповідно до принципів NestJS та підходів DDD: сутність описує схему таблиці, use case інкапсулює бізнес-логіку, а модуль *UserModule* (див. лістинг 3.12) поєднує сутності, бізнес-логіку, контролери та репозиторії в єдину модульну структуру.

3.3.2 Реалізація сервісу курсів

Сервіс курсів (backend/apps/course) відповідає за управління навчальними курсами, їхньою структурою (модулі та уроки), категоріями та тестами. Центральною доменною сутністю є CourseEntity, яка описує основні атрибути курсу: назву, опис, автора, рівень складності, тривалість, зображення, банер та статус публікації. Курси пов’язані з категоріями через

таблицю course_categories, а також містять набір модулів (CourseModuleEntity), у межах яких організовано уроки й тести.

Лістинг 3.16 — Сутність CourseEntity

```
// backend/apps/course/core/components/course/secondary-
adapters/postgres/data/course.entity.ts

import { BaseEntity } from
'backend/libs/core/src/configuration/base.entity';
import { Fillable } from 'backend/libs/core/src/shared-
kernel/common/decorators/fillable.decorator';
import { CourseLevel } from 'backend/libs/core/src/shared-
kernel/common/enums/course-level.enum';
import {
  Column,
  Entity,
  Index,
  ManyToMany,
  JoinTable,
  OneToMany,
} from 'typeorm';

import { CategoryEntity } from
'../../../../category/secondary-
adapters/postgres/data/category.entity';
import { CourseModuleEntity } from '../../../../course-
module/secondary-adapters/postgres/data/course-
module.entity';

@Entity('course')
export class CourseEntity extends BaseEntity {
  @Column({ name: 'title', type: 'varchar', length: 255 })
  title: string;

  @Column({ name: 'description', type: 'text' })
  description: string;

  @Index()
  @Column({ name: 'author_id', type: 'uuid' })
  authorId: string;

  @Column({
    name: 'level',
    type: 'enum',
    enum: CourseLevel,
    enumName: 'course_level',
```

```
  })
  level: CourseLevel;

  @Column({ name: 'duration', type: 'int' })
  duration: number;

  @Column({ name: 'image_url', type: 'varchar', length:
2048, nullable: true })
  imageUrl: string;

  @Column({ name: 'banner_url', type: 'varchar', length:
2048, nullable: true })
  bannerUrl: string;

  @Column({ name: 'is_published', type: 'boolean', default:
false })
  isPublished: boolean;

  @Column({ name: 'is_public', type: 'boolean', default:
true })
  isPublic: boolean;

  @Column({
    name: 'deleted_at',
    type: 'timestamp with time zone',
    nullable: true,
  })
  deletedAt: string;

  @ManyToMany(
    () => CategoryEntity,
    (category: CategoryEntity) => category.courses,
    { cascade: false },
  )
  @JoinTable({
    name: 'course_categories',
    joinColumn: { name: 'course_id', referencedColumnName:
'id' },
    inverseJoinColumn: { name: 'category_id',
referencedColumnName: 'id' },
  })
  categories: CategoryEntity[];

  @OneToMany(
    () => CourseModuleEntity,
    (module: CourseModuleEntity) => module.course,
    { cascade: false },
  )
  modules: CourseModuleEntity[];
```

```
  @Fillable
  public static fromObject(_obj: Partial<CourseEntity>):
CourseEntity {
    return new CourseEntity();
  }
}
```

На HTTP-рівні керування курсами здійснюється через контролер API-шлюзу CourseController (backend/apps/api-gateway/core/components/course/course.controller.ts), який приймає запити від клієнтів і делегує їх обробку сервісу курсів через Kafka. Наприклад, створення нового курсу (POST /api/course) транслюється у повідомлення course.create, яке обробляється відповідним use case'ом у Course Service.

Лістинг 3.17 — Створення курсу через API-шлюз (фрагмент CourseController)

```
// backend/apps/api-
gateway/core/components/course/course.controller.ts
(фрагмент)

@ApiOperation({ summary: 'Create course' })
@ApiBody({ type: CreateCourseDto })
@ApiOkResponse({ type: CourseOutput })
@Post()
async create(
  @Headers('authorization') auth: string,
  @Body() dto: CreateCourseDto,
) {
  try {
    const token = auth?.replace(/^Bearer\s+/i, '');
    return await lastValueFrom(
      this.courseClient
        .send('course.create', {
          ...dto,
          headers: token ? { authorization: `Bearer
${token}` } : undefined,
        })
        .pipe(timeout(10000)),
    );
  } catch (error) {
    throwHttpException(error, 'Failed to create course');
```

```
  }
}
```

Цей приклад демонструє типовий патерн взаємодії між компонентами системи: API-шлюз приймає HTTP-запит, перетворює його у Kafka-повідомлення (course.create) і надсилає його до сервісу курсів. На стороні Course Service відповідний use case створює записи в таблиці course та пов’язаних таблицях (категорії, модулі тощо), забезпечуючи цілісність структури курсу.

3.3.3 Сервіси прогресу та сповіщень

Сервіс прогресу (*backend/apps/progress*) відповідає за облік навчальної активності користувачів: проходження курсів, уроків і тестів. Він агрегує інформацію про кількість завершених елементів, загальну кількість та відсоток опанування курсу в таблиці *user_course_progress*, а також веде детальні стани уроків і тестів у таблицях *lesson_state* та *test_state*. Ключовою доменною сутністю для відображення зведеного прогресу курсу є *CourseProgressEntity*.

Лістинг 3.18 — Сутність CourseProgressEntity

```
//
backend/apps/progress/core/components/progress/secondary-
adapters/postgres/data/course-progress.entity.ts

import { Column, Entity, Index, Unique } from 'typeorm';
import { BaseEntity } from
'backend/libs/core/src/configuration/base.entity';

@Entity('user_course_progress')
@Unique('uq_user_course', ['userId', 'courseId'])
export class CourseProgressEntity extends BaseEntity {
  @Index()
  @Column({ name: 'user_id', type: 'uuid' })
  userId: string;

  @Index()
  @Column({ name: 'course_id', type: 'uuid' })
  courseId: string;
```

```
  @Column({ name: 'completed_items', type: 'int', default:
0 })
  completedItems: number;

  @Column({ name: 'total_items', type: 'int', default: 0 })
  totalItems: number;

  @Column({ name: 'percentage', type: 'float', default: 0
})
  percentage: number;

  @Column({
    name: 'status',
    type: 'varchar',
    length: 20,
    default: 'not_started',
  })
  status: 'not_started' | 'in_progress' | 'completed';
}
```

Оновлення прогресу здійснюється на основі подій, що надходять до ProgressController. Події на кшталт progress.lesson.completed або progress.test.attempt.finished обробляються сервісом ProgressAggregateService, який розраховує кількість завершених елементів, перерахує відсоток проходження курсу та оновлює записи у таблицях user_course_progress, lesson_state і test_state.

Сервіс сповіщень (backend/apps/notification) відповідає за зберігання та доставку сповіщень користувачам. Він використовує таблицю notification, яка містить тип сповіщення, заголовок, текст, додаткові дані, пріоритет та ознаку прочитання. Це дозволяє відображати сповіщення у веб- та мобільних клієнтах, а також у майбутньому підключати додаткові канали доставки (e-mail, push-сповіщення тощо).

Лістинг 3.19 — Сутність NotificationEntity

```
//
backend/apps/notification/core/components/notification/seco
ndary-adapters/postgres/data/notification.entity.ts
```

```
import {
  Entity,
  PrimaryGeneratedColumn,
  Column,
  CreateDateColumn,
  UpdateDateColumn,
} from 'typeorm';

@Entity('notification')
export class NotificationEntity {
  @PrimaryGeneratedColumn('uuid')
  id!: string;

  @Column({ name: 'user_id', type: 'uuid' })
  userId!: string;

  @Column({ type: 'varchar', length: 50 })
  type!: string;

  @Column({ type: 'varchar', length: 255 })
  title!: string;

  @Column({ type: 'text' })
  body!: string;

  @Column({ type: 'jsonb', nullable: true })
  data?: any;

  @Column({ name: 'is_read', type: 'boolean', default:
false })
  isRead!: boolean;

  @Column({ name: 'read_at', type: 'timestamp with time
zone', nullable: true })
  readAt?: Date;

  @Column({ type: 'varchar', length: 20, default: 'normal'
})
  priority!: string;

  @Column({ type: 'varchar', length: 50, nullable: true })
  category?: string;

  @CreateDateColumn({ name: 'created_at', type: 'timestamp
with time zone' })
  createdAt!: Date;

  @UpdateDateColumn({ name: 'updated_at', type: 'timestamp
with time zone' })
```

```
  updatedAt!: Date;
}
```

Разом ці два сервіси забезпечують повний цикл обробки навчальної активності:

- зберігають деталізовану інформацію про проходження уроків і тестів,
- підтримують агрегований прогрес курсу,
- формують і зберігають сповіщення, пов’язані з подіями навчання.

Детальні сценарії використання прогресу та сповіщень у веб- і мобільному клієнтах, а також приклади їх тестування, розглянуто у відповідних підрозділах.

3.4 Архітектура та основні екрани веб-клієнта

Веб-клієнт LMS реалізований як односторінковий застосунок (SPA), побудований на основі React із використанням Vite як інструмента збирання. Архітектура клієнтської частини організована за доменним принципом і структурована в каталозі frontend/src таким чином:

- pages/ — повноекранні сторінки для різних ролей (автентифікація, панель студента, панель викладача, адміністративна панель).
- components/ — багаторазово використовувані UI-компоненти (форми, таблиці, модальні вікна, елементи навігації).
- api/ — модулі HTTP-доступу до бекенду (auth.api.ts, course.api.ts, progress.api.ts, user.api.ts), що працюють поверх apiClient (Axios + інтерцептори).
- stores/ — сховища стану на базі Zustand (auth.store.ts, course.store.ts, learning.store.ts), які інкапсулюють локальний та кешований стан інтерфейсу.
- routes/ — конфігурація маршрутів (AppRoutes.tsx), де визначено, які сторінки доступні публічно, а які — лише для автентифікованих користувачів певної ролі.

Головний компонент App.tsx налаштовує середовище застосунку:

підключає провайдери React Query, тему Material UI, а також ініціалізує маршрутизацію через BrowserRouter. Маршрути структуровано відповідно до ролей користувачів у компоненті AppRoutes, що забезпечує контроль доступу через компонент ProtectedRoute.

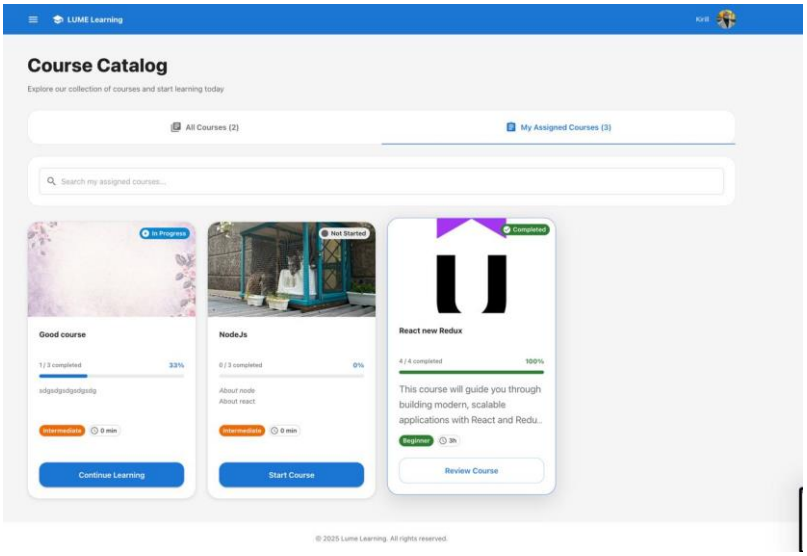


Рисунок 3.7 — *Перегляд назначених курсів*

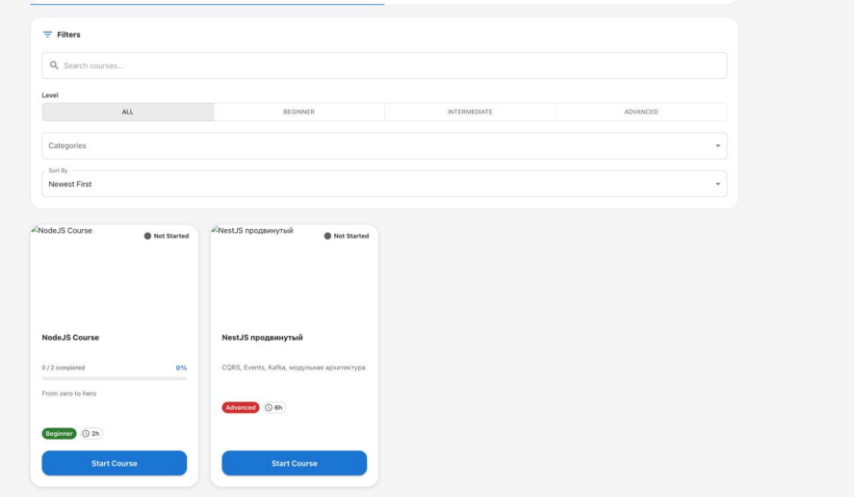


Рисунок 3.8 — *Перегляд публічних курсів*

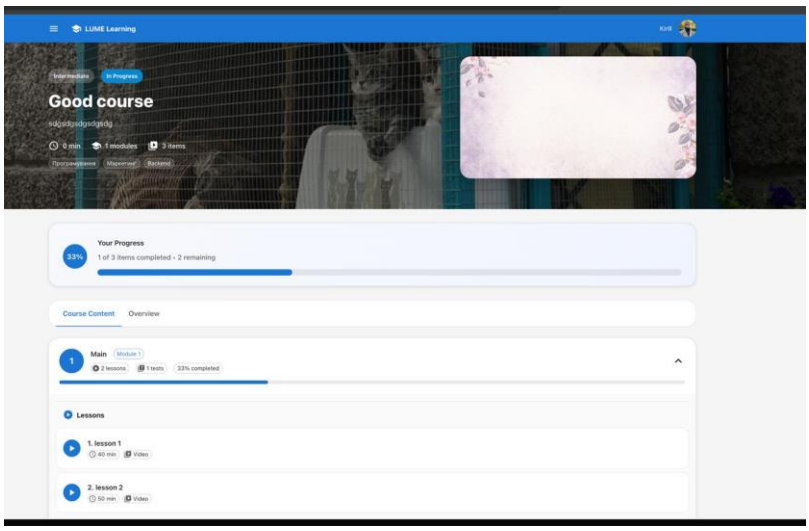


Рисунок 3.9 — *Перегляд курсу з модулями та уроками*

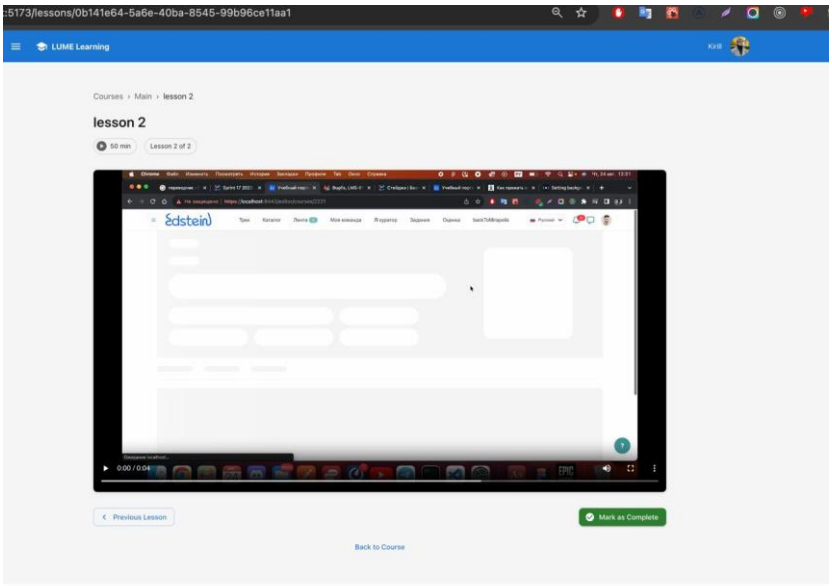


Рисунок 3.10 — *Перегляд уроку у курсі*

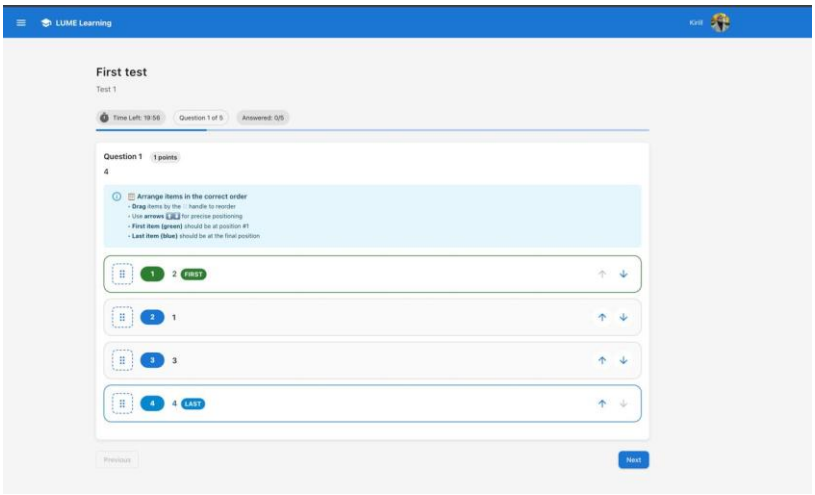


Рисунок 3.11 — *Перегляд тесту order-muny*

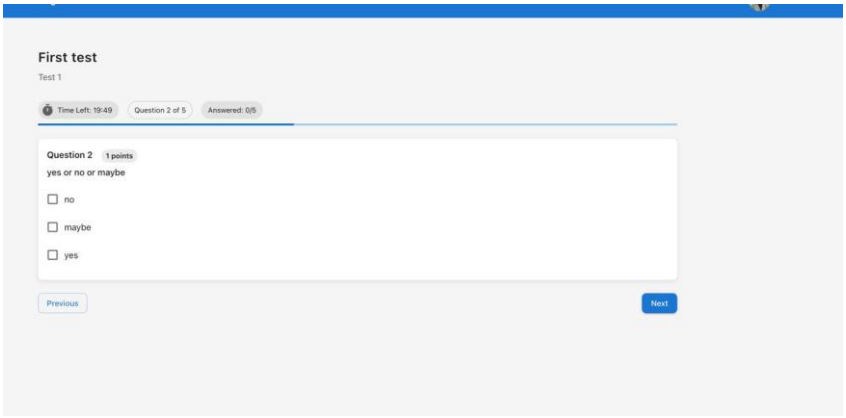


Рисунок 3.12 — *Перегляд тесту з декількома відповідями*

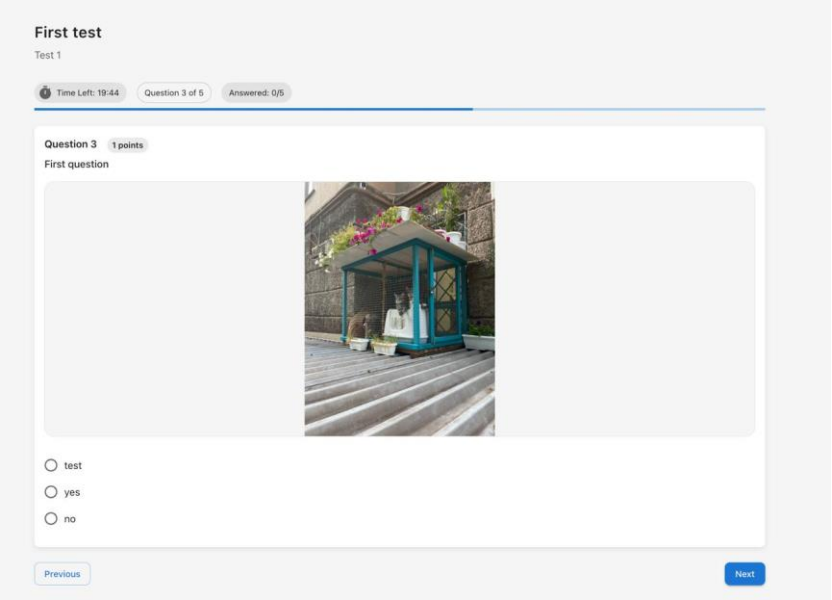


Рисунок 3.13 — *Перегляд тесту з одною відповіддю та картинкою*

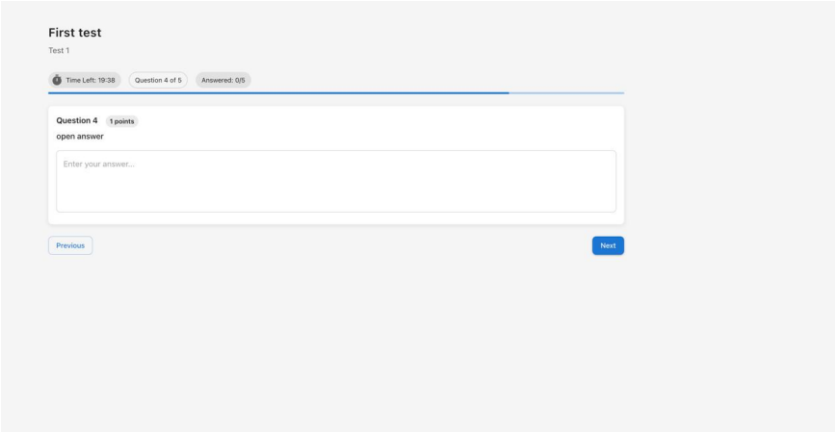


Рисунок 3.14 — *Перегляд з відкритою відповіддю*

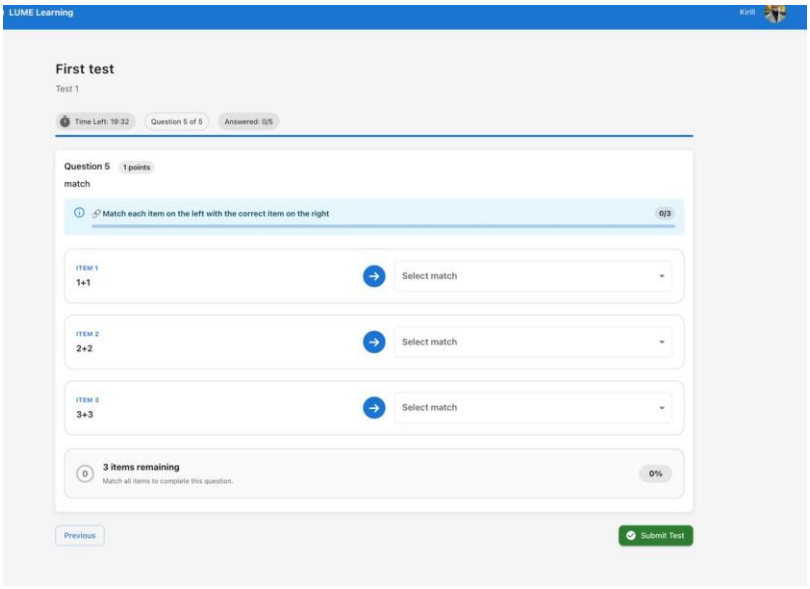


Рисунок 3.15 — *Перегляд тесту, де треба поставити відповідність*

Маршрутизація веб-клієнта описана у файлі AppRoutes.tsx, де визначено:

- публічні сторінки: логін, реєстрація, відновлення пароля;
- сторінки для авторизованих користувачів: профіль, каталог курсів, перегляд курсу, уроків і тестів;
- сторінки для викладачів і адміністраторів (опущено для стислості).

Лістинг 3.20 — Фрагмент конфігурації маршрутів AppRoutes.tsx

```
// frontend/src/routes/AppRoutes.tsx (фрагмент)

export const AppRoutes: React.FC = () => {
  return<Routes>
      {/* Публічні маршрути */}
      <Route path="/login" element={<LoginPage />} />
      <Route path="/register" element={<RegisterPage />} />
      <Route path="/forgot-password"
element={<ForgotPasswordPage />} />

      {/* Профіль (будь-який автентифікований користувач)
*/}
      <Route
        path="/profile"
        element={
          <ProtectedRoute>
            <ProfilePage />
          </ProtectedRoute>
        }
      />

      {/* Маршрути студента */}
      <Route
        path="/courses"
        element={
          <ProtectedRoute
            allowedRoles={[UserRole.STUDENT,
UserRole.TEACHER, UserRole.ADMIN]}
          >
            <CourseCatalog />
          </ProtectedRoute>
        }
      />
      <Route
        path="/courses/:courseId"
        element={
          <ProtectedRoute
            allowedRoles={[UserRole.STUDENT,
UserRole.TEACHER, UserRole.ADMIN]}
          >
            <CourseView />
          </ProtectedRoute>
        }
      />
      <Route
        path="/lessons/:lessonId"
        element={
```

```
          <ProtectedRoute
            allowedRoles={[UserRole.STUDENT,
UserRole.TEACHER, UserRole.ADMIN]}
          >
            <LessonView />          </ProtectedRoute>
        }
      />

      {/* ...інші маршрути викладача та адміністратора
опущено */}
    </Routes>
  );
}
```

3.4.1 Інтерфейс викладача та адміністратора у веб-клієнті

Веб-клієнт LMS містить окремі інструменти для викладачів і адміністраторів, які реалізують можливості управління навчальним контентом та користувачами. Доступ до цих інструментів обмежено ролями користувачів і надається через компонент ProtectedRoute, що перевіряє роль у Zustand-сховищі автентифікації.

Інтерфейс викладача

Для викладачів реалізовано розширений інтерфейс редагування курсів. Основний інструмент — CourseEditorNew, який дозволяє повністю управляти структурою та вмістом курсу. На одному екрані відображаються:

- загальна інформація про курс (назва, опис, рівень, тривалість, статус публікації);
- список модулів курсу;
- уроки й тести кожного модуля;
- категорії курсу.

Інтерфейс дозволяє:

- створювати, перейменовувати та видаляти модулі;
- додавати нові уроки та тести;
- редагувати існуючі елементи;
- змінювати категорії курсу;
- переміщувати уроки та тести між модулями.

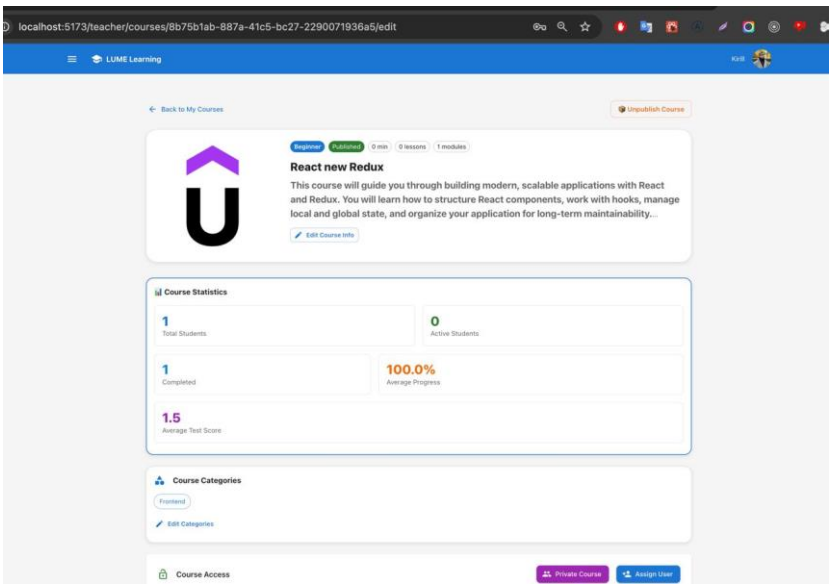


Рисунок 3.16 — *Редактор курсу для викладача з переліком модулів, уроків і тестів*

Інтерфейс адміністратора

Інтерфейс адміністратора забезпечує можливість повного контролю над роботою системи, надаючи інструменти для керування користувачами, налаштування доступів та підтримання загальної безпеки платформи.

1. Управління користувачами.

Адміністратор має доступ до таблиці всіх зареєстрованих користувачів. Для кожного користувача відображаються:

- e-mail,
- роль,
- статус облікового запису,
- дата створення.

Доступні операції:

- змінити роль користувача,
- заблокувати або активувати обліковий запис,
- видалити користувача.

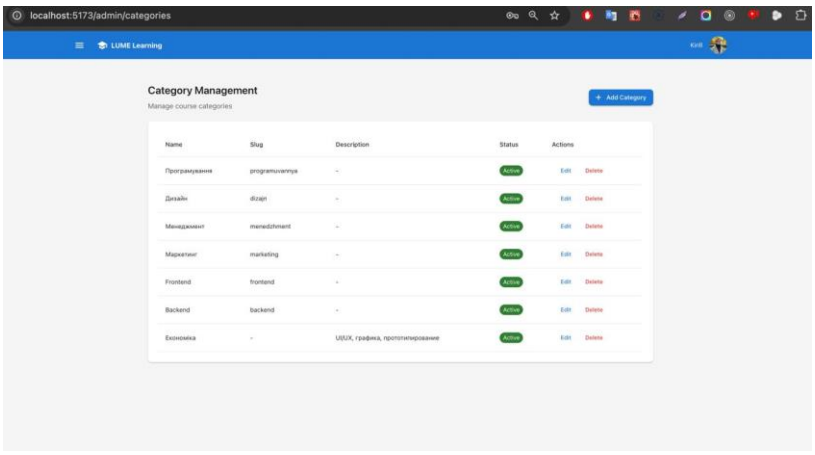


Рисунок 3.17 — *Адміністративна панель керування користувачами*

2. Управління категоріями курсів.

Для впорядкування навчального контенту та зручної навігації в системі передбачено спеціальний інструмент для роботи з категоріями курсів. Окремий інтерфейс CategoryManagement дозволяє:

- створювати нові категорії курсів,
- перейменовувати існуючі,
- деактивувати та активувати категорії,
- переглядати кількість курсів у кожній категорії.

Категорії використовуються як у каталозі курсів, так і в інтерфейсі викладача для групування та фільтрації курсів.

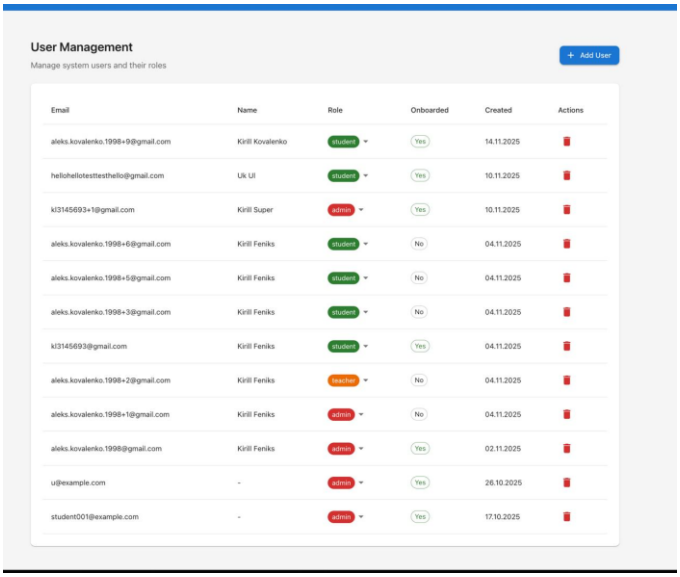


Рисунок 3.18 – *Керування категоріями курсів у адміністративному інтерфейсі*

Обмеження доступу за ролями. Правила доступу до інтерфейсів викладача та адміністратора реалізовано через компонент ProtectedRoute, який перевіряє роль користувача, отриману після автентифікації.

Наведений нижче фрагмент показує:

- редактор курсу доступний лише Teacher або Admin;
- управління користувачами та категоріями доступне лише Admin.

Лістинг 3.22 — Обмеження доступу до маршрутів викладача та адміністратора

```
// frontend/src/routes/AppRoutes.tsx (фрагмент)

<Route
  path="/teacher/courses/:courseId/edit"
  element={
    <ProtectedRoute allowedRoles={[UserRole.TEACHER,
UserRole.ADMIN]}>
      <CourseEditorNew />
    </ProtectedRoute>
  }
/>

<Route
  path="/admin/users"
  element={
    <ProtectedRoute allowedRoles={[UserRole.ADMIN]}>
      <UserManagement />
    </ProtectedRoute>
  }
/>

<Route
  path="/admin/categories"
  element={
    <ProtectedRoute allowedRoles={[UserRole.ADMIN]}>
      <CategoryManagement />
    </ProtectedRoute>
  }
/>
```

3.5 Архітектура та екран авторизації мобільного клієнта

Мобільний клієнт LMS реалізовано на базі React Native (Expo) у каталозі mobile/. Структура mobile/src організована за доменними областями та відображає логіку модульності:

- navigation/ — конфігурація навігації (RootNavigator, MainTabNavigator), де визначено стек авторизації та основний стек застосунку.
- screens/ — екрани логіну, реєстрації, списку курсів, деталей курсу, перегляду уроку та проходження тесту.
- api/ — модулі доступу до REST API (auth.api.ts, course.api.ts, progress.api.ts), які працюють поверх спільного apiClient з автоматичним додаванням JWT-токенів.
- stores/ — сховища стану на базі Zustand (auth.store.ts, theme.store.ts), що відповідають за автентифікацію та тему інтерфейсу.

Головний компонент App.tsx підключає провайдери SafeAreaProvider, React Query та тему React Native Paper, після чого рендерить компонент RootNavigator. Саме RootNavigator визначає, який набір екранів показувати — авторизаційний або основний — залежно від того, чи міститься валідний access-токен у SecureStore.

Екран авторизації

Початковим екраном мобільного клієнта є LoginScreen, який відображається до моменту успішної автентифікації користувача. На ньому користувач вводить e-mail і пароль; після підтвердження даних виконуються запити до API-шлюзу, отримані токени зберігаються у SecureStore, а RootNavigator перемикає інтерфейс на основний стек застосунку.

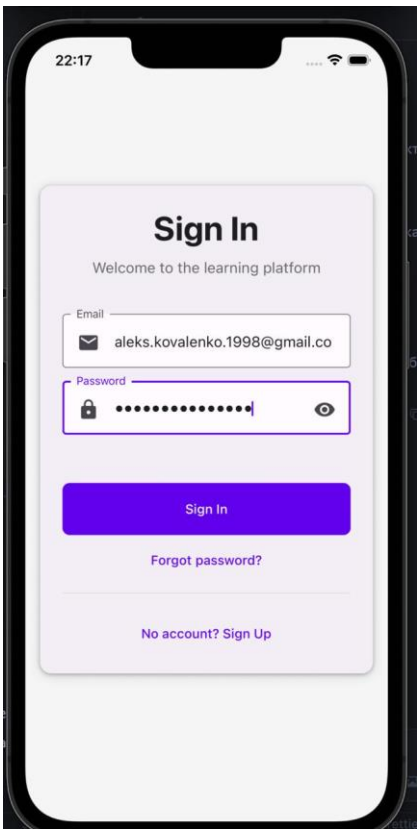


Рисунок 3.19 — *Екран логіну мобільного клієнта*

Логіка вибору між стеком авторизації та основним RootNavigator працює за простим алгоритмом:

- Викликає checkAuth() зі сховища auth.store.ts.
- Перевіряє, чи існують токени у SecureStore.
- Якщо токен дійсний → показує основний стек (курси, уроки, тести).
- Якщо токена немає → показує авторизаційний стек (Login, Register).
- Поки перевірка триває — показує індикатор завантаження.

Нижче наведено спрощений фрагмент RootNavigator, який використовується у мобільному клієнті.

Лістинг 3.23 — Спрощений RootNavigator мобільного клієнта

```
// mobile/src/navigation/RootNavigator.tsx (спрощений
фрагмент)

const Stack =
createNativeStackNavigator<RootStackParamList>();
```

```
export function RootNavigator() {
  const { isAuthenticated, checkAuth } = useAuthStore();
  const [initializing, setInitializing] = useState(true);

  useEffect(() => {
    checkAuth().finally(() => setInitializing(false));
  }, []);

  if (initializing) {
    return (
      <View style={{ flex: 1, justifyContent: "center",
alignItems: "center" }}>
        <ActivityIndicator size="large" />
      </View>
    );
  }

  return (
    <NavigationContainer>
      <Stack.Navigator screenOptions={{ headerShown: true
}}>
        {!isAuthenticated ? (
          <>
            <Stack.Screen
              name="Login"
              component={LoginScreen}
              options={{ headerShown: false }}
            />
            <Stack.Screen name="Register"
component={RegisterScreen} />
            {/* інші екрани авторизації опущено */}
          </>
        ) : (
          <>
            <Stack.Screen
              name="Main"
              component={MainTabNavigator}
              options={{ headerShown: false }}
            />
            <Stack.Screen name="CourseDetail"
component={CourseDetailScreen} />
            <Stack.Screen name="LessonView"
component={LessonViewScreen} />
            <Stack.Screen name="TestTaking"
component={TestTakingScreen} />
          </>
        )}
      </Stack.Navigator>
```

```
    </NavigationContainer>
  );
}
```

3.5.1 Основний екран та список курсів у мобільному клієнті

Після успішної автентифікації користувач переходить на головний екран мобільного клієнта, який реалізовано у вигляді нижньої таб-навігації (MainTabNavigator). На цьому екрані зібрано основні розділи мобільного застосунку:

- Home (Dashboard) — огляд загальної активності користувача, останні події та рекомендації курсів.
- Courses — список курсів, доступних для проходження.
- Profile — перегляд і редагування персональних даних.

Така структура дозволяє швидко перемикатися між навчанням, профілем та домашньою панеллю.

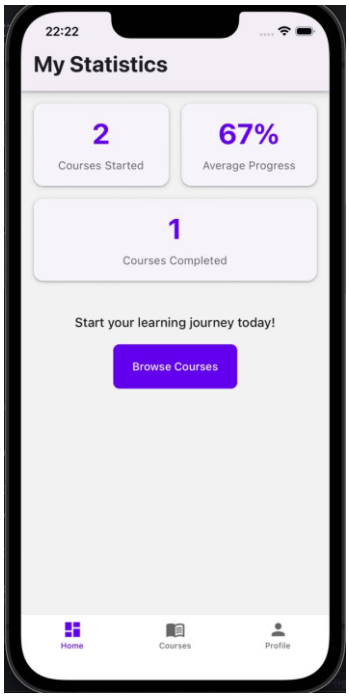


Рисунок 3.20 — *Головний екран мобільного клієнта з табами Home, Courses та Profile*

Екран списку курсів (CoursesScreen)

Екран CoursesScreen відіграє ключову роль у взаємодії студента з навчальною платформою. Він відображає:

- персональний список курсів, на які студент підписаний;
- загальний каталог курсів LMS.
- Для зручності передбачено:
- перемикач між вкладками «Мої курси» та «Усі курси»;
- поле пошуку;
- фільтри за категоріями, рівнем та тривалістю курсу;
- індикатор прогресу для кожного курсу;
- підвантаження курсів «порціями» за допомогою React Query (useInfiniteQuery).

Це робить інтерфейс зручним і адаптивним при роботі з великим обсягом навчального контенту.

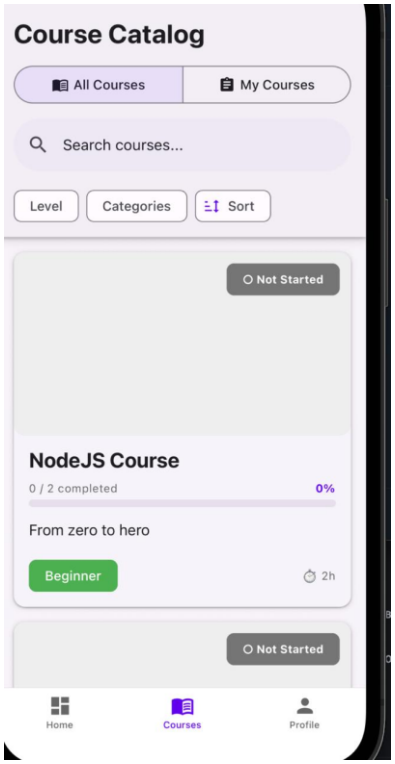


Рисунок 3.21 — *Екран списку курсів з вкладками «Мої» та «Усі курси»*

Лістинг 3.24 — Конфігурація головного таб-навігатора MainTabNavigator

```
// mobile/src/navigation/MainTabNavigator.tsx (фрагмент)
```

```
const Tab = createBottomTabNavigator<MainTabParamList>();

export function MainTabNavigator() {
  return (
    <Tab.Navigator
      screenOptions={{
        headerShown: false,
        tabBarActiveTintColor: "#6200ee",
        tabBarInactiveTintColor: "#666",
      }}
    >
      <Tab.Screen
        name="Dashboard"
        component={DashboardScreen}
        options={{
          title: "Home",
          tabBarIcon: ({ color, size }) => (
            <MaterialCommunityIcons
              name="view-dashboard"
              size={size}
              color={color}
            />
          ),
        }}
      />
      <Tab.Screen
        name="Courses"
        component={CoursesScreen}
        options={{
          title: "Courses",
          tabBarIcon: ({ color, size }) => (
            <MaterialCommunityIcons
              name="book-open-variant"
              size={size}
              color={color}
            />
          ),
        }}
      />
      <Tab.Screen
        name="Profile"
        component={ProfileScreen}
        options={{
          title: "Profile",
          tabBarIcon: ({ color, size }) => (
            <MaterialCommunityIcons name="account"
size={size} color={color} />
          ),
        }}
```

```
        />
      </Tab.Navigator>
    );
}
```

3.5.2 Перегляд уроків та тестів у мобільному клієнті

Після вибору курсу зі списку мобільного інтерфейсу студент переходить до екрана деталей курсу, звідки може відкривати окремі уроки та тести. Мобільний клієнт надає зручний адаптивний інтерфейс для навчання, а також автоматично оновлює прогрес після завершення кожного елемента.

Екран перегляду уроку (LessonViewScreen)

Екран LessonViewScreen є одним із центральних елементів мобільного клієнта. Він відображає:

- назву уроку;
- короткий опис;
- теоретичний контент (текстовий або змішаний);
- відеоматеріали (вбудовані або зовнішні).

Якщо урок містить відео-посилання на YouTube або Vimeo, користувачу пропонується відкрити відео у браузері чи нативному застосунку. У випадку локального файлу — використовується вбудований програвач (наприклад, Expo Video).

У нижній частині екрана розташована кнопка “Mark as Complete”. Після натискання:

- викликається відповідний ендпоінт бекенду (POST /api/course/lessons/:id/complete);
- оновлюється кеш прогресу через React Query;
- інтерфейс автоматично переходить до наступного уроку або повертається на екран курсу, якщо урок був останнім.

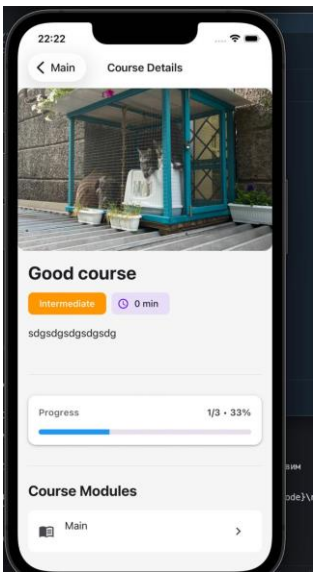


Рисунок 3.22 — *Екран прогресу курсів*

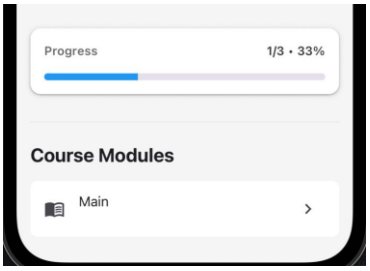


Рисунок 3.23 — *Екран прогресу курсів, 2 частина*

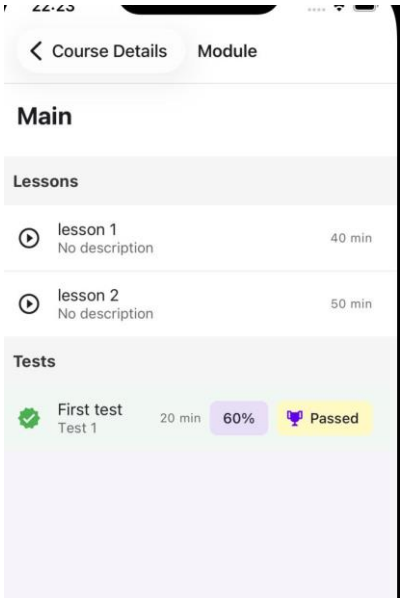


Рисунок 3.24 — *Екран наявних уроків*

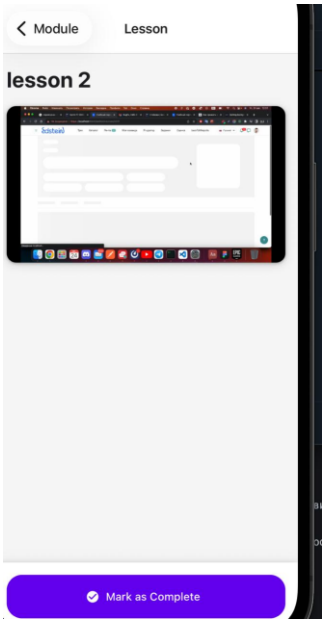


Рисунок 3.25 — *Екран перегляду курсів*

Екран проходження тесту (TestTakingScreen)

Екран TestTakingScreen адаптований під мобільний формат і відображає: текст питання, варіанти відповідей, тип питання (одна відповідь, кілька, текстова відповідь тощо) та індикатор прогресу (поточне питання / загальна кількість).

Після проходження тесту користувач отримує короткий підсумок результату. Тестові дані надсилаються на бекенд, де обробляються у сервісі тестування та прогресу.

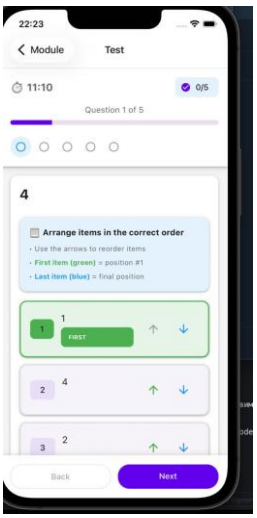


Рисунок 3.26 — *Екран проходження тестів*



Рисунок 3.27 — *Екран вибору відповіді*

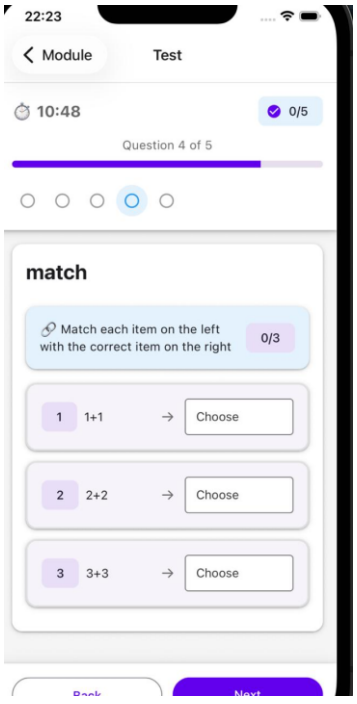


Рисунок 3.28 — *Екран вибору відповідностей*

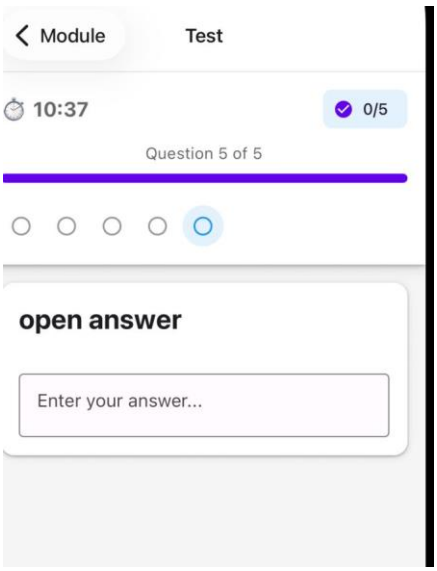


Рисунок 3.29 — *Екран написання розгорнутої відповіді*

Лістинг 3.26 — Обробка завершення уроку в LessonViewScreen

```
// mobile/src/screens/LessonViewScreen.tsx (фрагмент)

export function LessonViewScreen({ route, navigation }:
Props) {
  const { lessonId, courseId, moduleId } = route.params;
  const queryClient = useQueryClient();

  const { data: lesson, isLoading } = useQuery({
    queryKey: ['lessons', lessonId],
    queryFn: () => courseApi.getLesson(lessonId),
  });

  const { data: allLessons = [] } = useQuery({
    queryKey: ['modules', moduleId, 'lessons'],
    queryFn: () => courseApi.getAllLessons(moduleId),
    enabled: !!moduleId,
  });

  const sortedLessons = useMemo(
    () => [...allLessons].sort((a, b) => a.order -
b.order),
    [allLessons],
  );

  const nextLesson = useMemo(() => {
    const currentIndex = sortedLessons.findIndex((l) =>
l.id === lessonId);
    if (currentIndex >= 0 && currentIndex <
sortedLessons.length - 1) {
```

```
      return sortedLessons[currentIndex + 1];
    }
    return null;
  }, [sortedLessons, lessonId]);

  const completeMutation = useMutation({
    mutationFn: () => courseApi.completeLesson(lessonId,
courseId),
    onSuccess: () => {
      queryClient.invalidateQueries({ queryKey:
['progress', 'course', courseId] });
      queryClient.invalidateQueries({ queryKey:
['progress', 'lessons', courseId] });
    },
  });

  useEffect(() => {
    if (completeMutation.isSuccess) {
      const timer = setTimeout(() => {
        if (nextLesson) {
          navigation.replace('LessonView', { lessonId:
nextLesson.id, courseId, moduleId });
        } else {
          navigation.navigate('CourseDetail', { courseId
});
        }
      }, 1500);
      return () => clearTimeout(timer);
    }
  }, [completeMutation.isSuccess, nextLesson, navigation,
courseId, moduleId]);

  const handleComplete = () => {
    completeMutation.mutate();
  };

  return (
    <View style={styles.container}>
      {/* контент уроку */}
      <Surface style={styles.footer} elevation={4}>
        <Button
          mode="contained"
          onPress={handleComplete}
          loading={completeMutation.isPending}
          disabled={completeMutation.isPending ||
completeMutation.isSuccess}
          icon="check-circle"
        >
          {completeMutation.isSuccess ? 'Completed!' :
```

```
'Mark as Complete'}
        </Button>
      </Surface>
    </View>
  );
}
```

Він демонструє:

- завантаження уроку через React Query;
- визначення наступного уроку в модулі;
- виклик мутації для завершення уроку;
- автоматичну навігацію після завершення.

3.6 Висновки з розділу 3

У цьому розділі здійснено повний цикл проєктування та реалізації крос-платформної системи управління навчанням (LMS), що складається з бекенд-частини на NestJS, веб-клієнта на React та мобільного клієнта на React Native (Expo). Вибір єдиного технологічного стеку JavaScript/TypeScript дозволив забезпечити узгодженість модулів системи, спростити обмін даними та мінімізувати витрати на підтримку.

Архітектура серверної частини була побудована за принципами сервісно-орієнтованого підходу. До її складу входять окремі доменні сервіси — Auth/User Service, Course Service, Progress Service, Notification Service, — які взаємодіють через API-шлюз та брокер повідомлень Kafka. Такий підхід забезпечує модульність, ізольованість доменної логіки, можливість горизонтального масштабування та гнучкого розширення системи. Реляційна база даних PostgreSQL, спроєктована відповідно до сутностей LMS (користувачі, курси, модулі, уроки, тести, спроби тестування, прогрес, сповіщення), гарантує цілісність і структурованість даних.

У межах цього розділу були реалізовані ключові сценарії освітнього процесу: реєстрація та автентифікація користувачів, перегляд та проходження курсів, автоматичне оновлення прогресу, формування тестів та обробка

результатів, надсилання сповіщень. Окрему увагу приділено побудові потоку подій та обміну повідомленнями між сервісами, що дозволяє ефективно оновлювати прогрес студента в реальному часі.

Веб-клієнт забезпечує три групи користувачів — студентів, викладачів та адміністраторів — інструментами для роботи з навчальним контентом, редагуванням курсів, модерацією користувачів та управлінням категоріями. Мобільний клієнт адаптує основні можливості для студентів, надаючи повноцінний доступ до уроків, тестів і прогресу у мобільному форматі. Реалізація окремої мобільної навігації, оптимізованих екранів та інтеграції з API забезпечує зручність використання та безперервність навчання.

Таким чином, розроблена система демонструє практичну придатність сервісно-орієнтованої архітектури та стеку JavaScript/TypeScript для створення сучасної, масштабованої та крос-платформної LMS. У наступному розділі буде проведено тестування програмного забезпечення та оцінено його відповідність функціональним і нефункціональним вимогам.

РОЗДІЛ 4 ДОСЛІДЖЕННЯ РЕЗУЛЬТАТІВ ТЕСТУВАННЯ КРОСПЛАТФОРМНОЇ LMS

4.1 Методика тестування та експериментальне середовище

У цьому розділі наведено результати перевірки працездатності та продуктивності розробленої системи управління навчанням. Тестування проводилося для підтвердження коректної роботи основних бізнес-сценаріїв (автентифікація, керування курсами, проходження уроків і тестів, відстеження прогресу) та оцінки поведінки системи під навантаженням.

У роботі використовувалися такі види тестування:

- функціональне – перевірка REST-ендпоінтів і клієнтських екранів;
- інтеграційне / end-to-end – перевірка наскрізних сценаріїв, що охоплюють кілька сервісів (user, course, progress, notification) та клієнтів (web, mobile);
- навантажувальне – моделювання реальних режимів використання за допомогою інструмента k6 (smoke-тести, сценарні тести, load-тести, stress-тести, spike-тести).

Навантажувальні сценарії зосереджені в окремому пакеті backend/load-tests.

Основні команди запуску наведено у лістингу 4.1.

```
"scripts": {
  "test:smoke": "k6 run --env TEST_MODE=true
tests/smoke.test.js",
  "test:load": "k6 run --env TEST_MODE=true
tests/load.test.js",
  "test:stress": "k6 run --env TEST_MODE=true
tests/stress.test.js",
  "test:spike": "k6 run --env TEST_MODE=true
tests/spike.test.js",
  "test:auth": "k6 run --env TEST_MODE=true
tests/scenarios/auth.test.js",
  "test:course": "k6 run --env TEST_MODE=true
tests/scenarios/course.test.js",
  "test:test-passing": "k6 run --env TEST_MODE=true
```

```
tests/scenarios/test-passing.test.js",
  "test:full-flow": "k6 run --env TEST_MODE=true
tests/scenarios/full-flow.test.js"
}
```

Тестування виконувалося на локальному середовищі розробки (ПК з процесором рівня Intel Core i5/i7, 16 ГБ ОЗП, SSD, ОС Windows 10/11).

Бекенд-частина (API-шлюз та доменні сервіси) розгортається як набір NestJS-застосунків, які використовують PostgreSQL, Redis та Kafka (через Docker Compose).

Веб-клієнт (React/Vite) і мобільний клієнт (React Native/Expo) взаємодіють із системою через єдиний REST API: http://localhost:3000/api

На рисунку 4.1 наведено приклад консольного звіту k6 для smoke-тесту (npm run test:smoke), де відображаються кількість ітерацій, частка помилок, частка успішних перевірок та середній час відповіді.

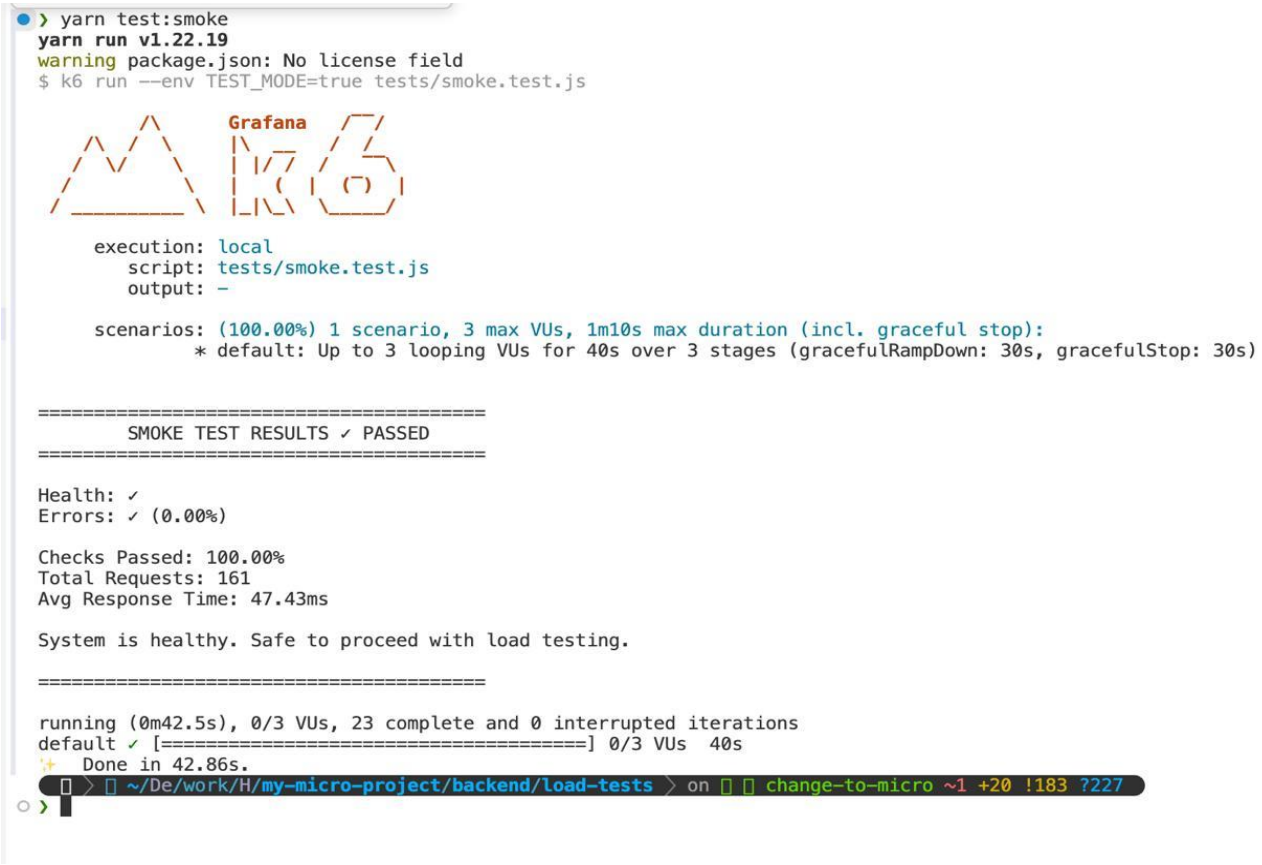


Рисунок 4.1 — Приклад виконання smoke-тесту в k6

4.2 Функціональне та інтеграційне тестування

Метою функціонального та інтеграційного тестування було перевірити, що розроблена LMS коректно реалізує основні вимоги до роботи студента, викладача та адміністратора, а також що окремі сервіси (user, course, progress, notification) узгоджено працюють як єдина система.

Функціональне тестування виконувалося у два етапи:

- ручна перевірка ключових сценаріїв у веб-клієнті (React) та мобільному клієнті (React Native): реєстрація та вхід користувача, зміна ролі, перегляд каталогу курсів, проходження уроків і тестів, оновлення прогресу;
- автоматизовані сценарні тести на основі k6, які одночасно виступають як інтеграційні / end-to-end перевірки.
- До автоматизованих сценаріїв належать такі тести k6:
- Auth Scenario (test:auth) – наскрізний сценарій автентифікації: реєстрація користувача, вхід до системи, отримання поточного профілю, оновлення токена доступу;
- Course Scenario (test:course) – перевірка повного циклу роботи з курсом: створення курсу, читання, оновлення, видалення, перегляд списку курсів;
- Test Passing Scenario (test:test-passing) – перевірка проходження тесту: створення курсу, модуля, тесту, питань і варіантів відповідей, старт спроби, авто-збереження відповідей, отримання сесії тесту, здача тесту та перевірка результату;
- Full Flow Scenario (test:full-flow) – повний навчальний цикл: реєстрація й автентифікація користувача, створення курсу з модулем і уроком, створення тесту з питаннями, проходження тесту, авто-збереження, здача, оновлення прогресу та перевірка даних сервісу прогресу.

Усі сценарні тести виконувалися проти тієї ж інфраструктури, що й у розділі 4.1. Узагальнені результати запуску сценарних тестів наведені в

таблиці 4.1.

Таблиця 4.1 – Результати сценарних (e2e) тестів k6

Сценарій	Кількість ітерацій	Успішні ітерації, %	Помилки HTTP, %	p95 часу відповіді, ms*
Auth Scenario	348	100	0.00	~1318
Course Scenario	135	100	0.00	~482
Test Passing	100	100	0.00	~219
Full Flow	55	100	0.00	~106 (HTTP), ~6280 (тривалість всієї ітерації)**

* значення p95 наведені за консолями k6

** для Full Flow окремо вимірюється p95 часу однієї повної «навчальної» ітерації (≈ 6,3 с)

Як видно з таблиці 4.1, в усіх сценаріях:

- частка успішних ітерацій становить 100 %;
- частка помилкових HTTP-запитів дорівнює 0 % – усі виклики REST-ендпоінтів завершилися зі статусами 2xx;
- основні функціональні ланцюжки (реєстрація, вхід, робота з курсами, проходження уроків і тестів, оновлення прогресу) виконуються без збоїв.

При цьому для окремих сценаріїв (зокрема, Auth та Course) k6 зафіксував перевищення цільових порогів за часом відповіді (thresholds по

http_req_duration), що відноситься до показників продуктивності, а не до функціональної коректності. Детальніше ці аспекти розглянуто у підрозділі 4.3, присвяченому навантажувальному тестуванню.

Отже, функціональне та інтеграційне тестування показало, що розроблена кросплатформна LMS коректно реалізує основні сценарії роботи користувачів, а взаємодія між веб-клієнтом, мобільним клієнтом та серверною частиною відбувається без логічних помилок.

4.3 Навантажувальне тестування (режим нормальної роботи)

Навантажувальне тестування проводилося за допомогою сценарію test:load (файл load.test.js), метою якого було змоделювати «типову» робочу активність системи при поступовому збільшенні кількості користувачів.

Сценарій складається з п’яти стадій навантаження:

- 20 віртуальних користувачів (VU) – розігрів;
- 50 VU – нормальне навантаження;
- 100 VU – пікове навантаження;
- 50 VU – зниження до нормального рівня;
- 0 VU – завершення тесту.

Кожна ітерація виконує:

- автентифікацію (реєстрація + вхід + підвищення ролі до admin);
- один із трьох сценаріїв:
- управління курсом (створення, оновлення, читання);
- створення контенту (курс + модуль + урок);
- читання (списки курсів, деталі курсу, модулі).

Узагальнені результати запуску test:load наведено в таблиці 4.2.

Таблиця 4.2 – *Результати навантаженого тестування*

Показник	Значення
Кількість операцій	1154
Успішні операції	1154 (100 %)
Помилки HTTP	0 (0.00 %)
Загальна кількість HTTP-запитів	8078
Середній час операції	~6.88 с
Середній час HTTP-запиту	~133.84 ms
p95 http_req_duration	~3525 ms
Середній http_reqs (запитів/с)	~38.17 req/s

На рисунку 4.2 наведено фрагмент консольного звіту k6 для виконання npm run test:load.

У звіті видно загальну кількість запитів та ітерацій, відсоток помилок (http_req_failed) – 0 % та перцентилі часу відповіді (http_req_duration p(95)), за якими k6 фіксує перетин порогових значень.



Рисунок 4.2 – *Приклад консольного звіту k6 для навантажувального тесту test:load*

Сценарій пройшов без логічних помилок:

- усі бізнес-операції (автентифікація, робота з курсами, уроками й тестами) завершилися успішно;
- частка помилкових HTTP-запитів становить 0 %.

Разом із тим k6 зафіксував порушення частини порогових значень (thresholds):

- http_req_duration p(95) < 1500 ms – не виконано (фактично ≈ 3525 ms);
- http_reqs rate > 50 req/s – не виконано (фактично ≈ 38 req/s).

Ці відхилення пов’язані з обмеженими ресурсами локального середовища та наявністю «важких» операцій у кожній ітерації (реєстрація нового користувача, bcrypt-хешування пароля, створення курсів/модулів/уроків).

Функціонально система залишається стабільною: помилок 4xx/5xx під навантаженням не спостерігалося, а всі сценарії виконуються до кінця.

Питання продуктивності (значення p95 та кількість HTTP-запитів за секунду) розглядаються як потенційний напрям подальшої оптимізації й масштабування системи.

4.4 Стресове та спайк-тестування

Стресове та спайк-тестування використовувалося для оцінювання поведінки LMS у граничних режимах роботи: при поступовому наближененні до «точки відмови» (stress) та при різких стрибках навантаження (spike). Обидва сценарії реалізовані у вигляді окремих скриптів k6 (stress.test.js, spike.test.js).

4.4.1 Стрес-тестування (test:stress)

Стрес-сценарій моделює поступове збільшення кількості одночасних користувачів до 300 VU з утриманням навантаження на кожному рівні.

На кожній ітерації виконується:

- автентифікація користувача;
- створення курсу;
- створення модуля в курсі.

До певного порогу навантаження система залишається стабільною: усі операції виконуються успішно, помилки HTTP не фіксуються.

При подальшому збільшенні кількості віртуальних користувачів (приблизно від 150–200 VU) у логах починають з’являтися помилки для запиту POST /user/me/make-admin, який у тестовому режимі підвищує роль новоствореного користувача до адміністратора.

Типові помилки:

- 500 Internal Server Error з повідомленням "Timeout has occurred" – сервіс користувачів не встигає обробити всі запити;
- 401 Unauthorized з повідомленням "Invalid or expired token" – частина токенів не встигає коректно валідуватися через затримки та черги.

Таким чином, стрес-тест дозволив виявити перший «вузький» елемент системи під екстремальним навантаженням — допоміжний ендпоінт підвищення ролі користувача. Основні бізнес-операції (робота з курсами та модулями) до цього порогу залишаються працездатними, що підтверджує адекватну стійкість системи у порівнянні з реалістичними сценаріями.

Можливі напрями оптимізації:

- винесення операції «зробити адміністратора» за межі кожної ітерації (попередньо створювати користувача з роллю admin/teacher);
- масштабування сервісу користувачів та оптимізація процесів валідації JWT-токенів.

4.4.2 Спайк-тестування (test:spike)

Спайк-тестування використовується для перевірки поведінки системи під час раптових та інтенсивних стрибків навантаження, а також здатності швидко відновлюватися після пікових значень. У межах цього сценарію моделюються такі етапи:

- від 5 до 200 VU (перший спайк, утримання 20 секунд);
- зниження до 5 VU (відновлення);
- від 5 до 250 VU (другий, ще більший спайк);
- завершення з утриманням високого навантаження.

Кожна ітерація включає швидку автентифікацію та одну або кілька операцій створення курсів під час пікових фаз.

Узагальнені результати запуску test:spike наведені в таблиці 4.3.

Таблиця 4.3 – *Результати запуску test:spike*

Показник	Значення
Загальна кількість запитів	3804
Error Rate (http_req_failed)	6.60 %
Recovery Rate (spike_recovery)	78.74 %
First Spike Errors	140
Second Spike Errors	116
p95 http_req_duration	78.74

Результати спай текстування

k6- сценарій має такі порогові значення:

- http_req_failed < 0.3 (допускається до 30 % помилок під час спайку) – виконано (6.6 %);
- spike_recovery > 0.6 (не менше 60 % успішних операцій) – виконано (78.74 %);
- http_req_duration p(95) < 8000 ms – порушено (≈ 11.8 с).

Таким чином, при різких стрибках навантаження система:

- зберігає прийнятно низьку частку помилок;
- демонструє високий показник відновлення (recovery rate);
- водночас має значну деградацію часу відповіді (p95), що є очікуваним для агресивного пікового навантаження у локальному середовищі розробки.

Сумарний висновок збігається з результатами k6: система стійка до спайків, але потребує оптимізації часу відповіді при екстремальних піках активності.

4.4.3 Ручне тестування веб- та мобільного клієнтів

Окрім автоматизованих тестів навантаження, важливим етапом було проведення ручного тестування клієнтських частин системи, що дозволило

перевірити реальну поведінку інтерфейсів та виявити можливі помилки у взаємодії з користувачем. Окремо від автоматизованих k6-сценаріїв було проведено ручне тестування клієнтських застосунків:

- веб-клієнта (Next/React/Vite) у десктопному браузері;
- мобільного клієнта (React Native/Expo) на смартфоні.
- Для обох інтерфейсів перевірялися типові сценарії:
- реєстрація нового користувача, вхід та вихід із системи;
- перегляд каталогу курсів, відкриття сторінки курсу, модуля та уроку;
- проходження тестів і перегляд отриманих результатів;
- перевірка відображення власного прогресу;
- (для веб-версії) перевірка доступу до різних розділів залежно від ролі (студент / викладач / адміністратор) і коректність адаптивної верстки.

Ручне тестування підтвердило, що веб-інтерфейс і мобільний застосунок узгоджено працюють із єдиним REST API, а основні сценарії взаємодії користувачів із LMS коректно підтримуються на обох платформах.

4.5 Висновки з розділу 4

У цьому розділі було досліджено результати тестування розробленої кросплатформної LMS на рівні окремих сервісів, всієї серверної частини та клієнтських застосунків (web і mobile), а також оцінено поведінку системи під різними видами навантаження. Основні висновки такі:

Логічна коректність серверної частини

Запуск юніт- та інтеграційних тестів NestJS-сервісів (user, course, progress, notification, api-gateway) показав, що всі 68 тестових наборів (345 тестів) завершуються успішно. Це підтверджує коректну реалізацію бізнес-логіки, валідації, доступу до бази даних та обробки помилок на рівні бекенду.

На рисунку 4.3 наведено фрагмент консолі з результатами запуску Jest-тестів.

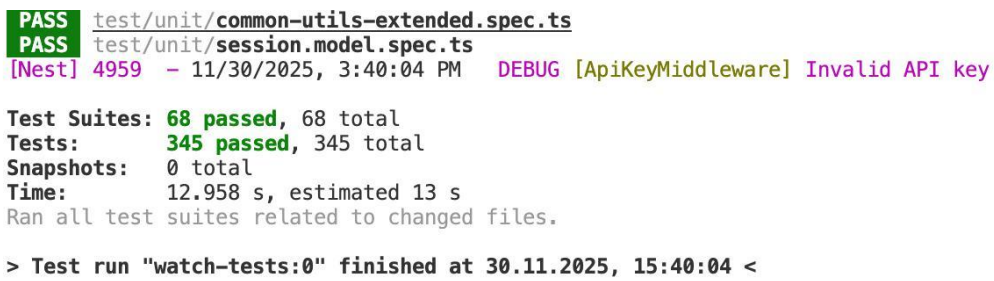


Рисунок 4.3 – *Результати запуску Jest-тестів серверної частини LMS*

```
[Nest] 26258  - 11/26/2025, 12:49:24 PM   DEBUG
[ApiKeyMiddleware] Invalid API key

Test Suites: 68 passed, 68 total
Tests:       345 passed, 345 total
Snapshots:   0 total
Time:        10.557 s
Ran all test suites related to changed files.
```

Функціональна цілісність системи. Сценарні k6-тести (auth, course, test-passing, full-flow) продемонстрували, що основні наскрізні сценарії — реєстрація та автентифікація користувача, робота з курсами, створення і проходження тестів, оновлення прогресу — виконуються без помилок.

Для всіх сценаріїв частка успішних ітерацій становила 100 %, помилки HTTP (4xx/5xx) не виникали.

Коректна робота веб- та мобільного інтерфейсів. Ручне тестування*:*

- веб-клієнта (React),
- мобільного застосунку (React Native),

підтвердило, що обидва інтерфейси узгоджено працюють із єдиним REST API, коректно відображають курси, уроки, тести та прогрес користувача, а також підтримують сценарії для різних ролей (студент, викладач, адміністратор).

Поведінка під типовим навантаженням. У навантажувальному тесті test:load (до 100 одночасних віртуальних користувачів) було виконано:

- 1195 комплексних операцій,
- 8365 HTTP-запитів,

- з 100 % успішністю.

k6 зафіксував, що цільові пороги по часу відповіді (p95) та швидкості обробки запитів у локальному середовищі не досягаються:

- p95 ≈ 3,3 с при цільових 1,5 с;

≈ 39 req/s при цільових 50 req/s.

Водночас error rate = 0 %, тому на функціональну стабільність ці показники не впливають.

Граничні режими: stress та spike. Стрес-тестування показало, що першим «вузьким місцем» стає допоміжний ендпоінт
/user/me/make-admin,
який під великим навантаженням повертає
500 Timeout та 401 Unauthorized.

Спайк-тестування продемонструвало:

- error rate ≈ 6,6 % при допустимих 30 %;
- recovery rate ≈ 78,7 % при вимозі ≥ 60 %;
- p95 часу відповіді ≈ 11,8 с — перевищує поріг, що типово для локального середовища та різких піків навантаження.

Напрями подальшого вдосконалення

Отримані результати свідчать, що розроблена LMS готова до використання в умовах малих та середніх навантажень, а основні бізнес-функції реалізовано коректно.

Подальший розвиток системи може включати:

- оптимізацію найважчих операцій
- (реєстрація, підвищення ролі, масове створення курсів);
- впровадження кешування та оптимізацію запитів до БД;
- масштабування окремих сервісів (особливо сервісу користувачів);
- уточнення порогових значень (SLA) з урахуванням характеристик цільового середовища.

Загальний підсумок

Тестування підтвердило, що запропонований архітектурний підхід та

стек технологій (React, NestJS, React Native, PostgreSQL, Kafka) дозволяють побудувати працездатну, розширювану та кросплатформну LMS, яку за потреби можна оптимізувати й масштабувати відповідно до вимог реальних користувачів.

ВИСНОВКИ

1. Досліджено проблему організації електронного навчання та управління навчальним процесом у крос-платформному середовищі: проаналізовано сучасні LMS-рішення (Moodle, Google Classroom, Canvas тощо), їхні переваги та обмеження з точки зору функціональності, гнучкості та вартості супроводу.
2. Досліджено технології та архітектурні підходи до побудови крос-платформних систем на єдиному стеку JavaScript/TypeScript; обґрунтовано вибір React для веб-інтерфейсу, NestJS для серверної частини, React Native для мобільного застосунку, PostgreSQL як основної СУБД та Kafka як брокера повідомлень.
3. Спроєктовано архітектуру крос-платформної системи управління навчанням у вигляді сервісно-орієнтованої системи з API-шлюзом і доменними сервісами (user, course, progress, notification); визначено ролі користувачів, модулі системи, моделі даних та структуру реляційної бази.
4. Реалізовано серверну частину LMS на базі NestJS: створено API-шлюз, сервіси керування користувачами, курсами, модулями, уроками, тестами, прогресом і сповіщеннями; впроваджено автентифікацію та авторизацію на основі JWT і refresh-токенів, а також взаємодію між сервісами через Kafka.
5. Розроблено веб-інтерфейс системи на основі React/Vite для студентів, викладачів і адміністраторів та мобільний застосунок на React Native/Expo; забезпечено узгоджений користувацький досвід на різних платформах і повторне використання спільних моделей і типів даних.
6. Проведено юніт- та інтеграційне тестування серверної частини (68 тестових наборів, 345 тестів), що підтвердило коректність бізнес-логіки, валідації, доступу до бази даних та обробки помилок на рівні бекенду.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Brusilovsky P. Adaptive Hypermedia and Adaptive Web. User Modeling and User-Adapted Interaction. 2001.

2. Watson J., Watson S. Learning Management Systems: An Overview. Encyclopedia of Information Science and Technology. 2015.

3. Ally M. Mobile Learning: Transforming the Delivery of Education and Training. Athabasca University Press, 2009.

4. Graf S., List B. An evaluation of open source e-learning platforms. IEEE International Conference on Advanced Learning Technologies. 2005.

5. Moodle Documentation. URL: https://moodle.org/ (дата звернення: 12.09.2025).

6. Blackboard LMS Overview. URL: https://www.blackboard.com/ (дата звернення: 15.09.2005).

7. Canvas LMS Documentation. URL: https://www.instructure.com/ (дата звернення: 18.09.2005).

8. React Documentation. Meta Platforms. URL: https://react.dev/ (дата звернення: 25.09.2005)

9. React Native Documentation. Meta Platforms. URL: https://reactnative.dev/ (дата звернення 26.09.2005)

10. NestJS Documentation. URL: https://docs.nestjs.com/ (дата звернення: 30.09.2025)

11. Docker Documentation. URL: https://docs.docker.com/ (дата звернення 03.10.2025)

12. PostgreSQL Documentation. URL: https://www.postgresql.org/docs/ (дата звернення 05.10.2025)

13. Redis Documentation. URL: https://redis.io/docs/ (дата звернення 06.10.2025)

14. TanStack Query Documentation. URL: https://tanstack.com/query/latest (дата звернення: 10.10.2025)

15. GraphQL Foundation. GraphQL Documentation. URL: https://graphql.org/ (дата звернення: 16.10.2025)

16. TypeORM Documentation. URL: https://typeorm.io/ (дата звернення 29.10.2025)

17. Kravets A., Shorikov A., Kultsova M. Digital Education: Challenges and Perspectives. Lecture Notes in Networks and Systems. 2021. (дата звернення:

18. Khalil M., Ebner M. Learning Analytics in MOOCs: A Review of Empirical Research. Educational Technology & Society. 2016.

19. Pappas C. The Future of Learning Management Systems. eLearning Industry. 2022.

20. Sclater N. Learning Analytics Explained. Routledge, 2017.

21. Zare M., Watson S. Cloud-based LMS architecture: benefits and challenges. Journal of Online Learning. 2020.

22. Basak S., Wotto M., Bélanger P. E-learning, M-learning and D-learning: Conceptual review. Journal of Computer Assisted Learning. 2018.

23. IBM. Microservices Architecture Explained. URL: https://www.ibm.com/cloud/learn/microservices

24. Richardson C. Microservices Patterns. Manning Publications, 2018.

25. Google Firebase Documentation. URL: https://firebase.google.com/docs

26. AWS Architecture Blog. Scalable Backend Patterns. URL: https://aws.amazon.com/blogs/architecture/

27. React Native vs Flutter vs Kotlin Multiplatform: Comparative Analysis. IEEE Access. 2023.

28. Holovetskyi R. Analysis of LMS Adoption in Ukrainian Universities. Kyiv: NULP Press, 2021.

29. Zakharov V. Web-oriented technologies for educational systems. 2020.

30. Андрусенко В. Розроблення веб-систем на базі JavaScript-фреймворків. Київ: КНЕУ, 2021.

31. Кравчук М. Технології REST API у сучасних інформаційних системах. Збірник наук. праць КНУ. 2022.

32. Krishnan S. Full-Stack Development with React, Node and PostgreSQL. Apress, 2021.

33. Nesterov A. Cross-platform mobile development: trends and analysis. Journal of Systems Integration. 2020.

34. UX in e-learning platforms: navigation and accessibility. IEEE Education Society. 2021.

35. OWASP Foundation. Secure Development Guidelines. URL: https://owasp.org/www-project-top-

Декларація
академічної доброчесності
здобувача ступеня вищої освіти ЗНУ

Я Коваленко Кирило Максимович, студент _2_ курсу, форми здобуття освіти денної, Інженерного навчально-наукового інституту, спеціальність 121 Інженерія програмного забезпечення, адреса електронної пошти ipz20bd-217@stu.zsea.edu.ua, – підтверджую, що написана мною кваліфікаційна робота на тему: «Особливості створення крос-платформної системи управління навчанням (LMS) на базі Next.js, Nest.js та React Native» відповідає вимогам академічної доброчесності та не містить порушень, що визначені у ст. 42 Закону України «Про освіту», зі змістом яких ознайомлений.

- заявляю, що надана мною для перевірки електронна версія роботи є ідентичною її друкованій версії;
- згоден/згодна на перевірку моєї роботи на відповідність критеріям академічної доброчесності у будь-який спосіб, у тому числі за допомогою інтернет-системи, а також на архівування моєї роботи в базі даних цієї системи.

Дата _____ _____Коваленко Кирило Максимович
Підпис (студент)

Дата _____ _____Скрипник Ірина Анатоліївна
Підпис (Науковий керівник)