МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Чорноморський національний університет імені Петра Могили

Факультет комп'ютерних наук

Кафедра інженерії програмного забезпечення

ДОПУЩЕНО ДО ЗАХИСТУ

Завідувач кафедри інженерії

програмного забезпечення

_____ Євген ДАВИДЕНКО

«___» _____ 2026 р.

КВАЛІФІКАЦІЙНА РОБОТА

НА ЗДОБУТТЯ ОСВІТНЬОГО СТУПЕНЯ БАКАЛАВРА

КОРПОРАТИВНА ПЛАТФОРМА ВЗАЄМОДІЇ ПРАЦІВНИКІВ З ФУНКЦІОНАЛЬНИМ МОДУЛЕМ НАВЧАННЯ ТА СТОРОННЬОЮ ІНТЕГРАЦІЄЮ

Спеціальність 121 Інженерія програмного забезпечення

Освітня програма «Інженерія програмного забезпечення»

Здобувачка _____ **Оксана ШУМАКОВА**

«__» _____ 20__ р.

Керівник роботи

старша викладачка

_____ **Марина ФАЛЕНКОВА**

«__» _____ 20__ р.

Миколаїв – 2026

Завдання на виконання кваліфікаційної роботи

Чорноморський національний університет імені Петра Могили

Факультет	Комп'ютерних наук
Кафедра	Інженерії програмного забезпечення
Рівень вищої освіти	Перший (бакалаврський)
Освітній ступінь	Бакалавр
Спеціальність	121 Інженерія програмного забезпечення
Освітня програма	Інженерія програмного забезпечення

ЗАТВЕРДЖУЮ
Завідувач кафедри інженерії
програмного забезпечення
_____ Євген ДАВИДЕНКО
«___» _____ 2026 р.

ЗАВДАННЯ

на кваліфікаційну бакалаврську роботу здобувачку

Шумакову Оксану

1. Тема кваліфікаційної роботи «Корпоративна платформа взаємодії працівників з функціональним модулем навчання та сторонньою інтеграцією» затверджена наказом ректора ЧНУ ім. Петра Могили № 349 від «26» грудня 2025 р.
2. Строк представлення кваліфікаційної роботи «___» _____ 2026 р.
3. Очікуваний результат роботи та початкові дані якщо такі потрібні.

Очікуваним результатом є корпоративна платформа взаємодії працівників з функціональним модулем навчання та сторонньою інтеграцією.

4. Перелік питань, що підлягають розробці: дослідження предметної області та аналіз існуючих аналогів, формування специфікації вимог до програмного забезпечення, визначення архітектури для проєктування програмного

забезпечення, моделювання бізнес-процесів, компонентів та інтерфейсів програмного забезпечення, реалізація серверної та клієнтської частин програмного забезпечення, проведення модульного, інтеграційного та навантажувального тестування, проведення аналізу результатів розробки.

5. Перелік графічних матеріалів:

Презентація

6. Консультанти:

Консультант	**Кафедра (організація)**	**Частина роботи**

Дата видачі завдання «__26__» ___грудня_____ 2025 р.

КАЛЕНДАРНИЙ ПЛАН
виконання кваліфікаційної роботи

Тема: **«Корпоративна платформа взаємодії працівників з функціональним модулем навчання та сторонньою інтеграцією»**

№	Найменування роботи	Початок	Закінчення	Примітки
1.	Розробка та затвердження завдання на виконання КБР	26.12.2025	29.12.2025	*Виконано*
2.	Огляд літератури та аналіз аналогів	02.02.2026	06.02.2026	*Виконано*
3.	Складання календарного плану КБР	09.02.2026	10.02.2026	*Виконано*
4.	Аналіз предметної області та визначення вимог	16.02.2026	20.02.2026	*Виконано*
5.	Розробка проєктних рішень	20.02.2026	23.02.2026	*Виконано*
6.	Моделювання та конструювання ПЗ	23.02.2026	26.02.2026	*Виконано*
7.	Розробка структури бази даних	26.02.2026	09.03.2026	*Виконано*
8.	Кодування, тестування та апробація розробленого ПЗ, аналіз результатів тестування, розробка керівництва користувача	09.03.2026	21.05.2026	*Виконано*
9.	Оформлення КБР та презентації	22.05.2026	26.05.2026	*Виконано*
10.	Відгук керівника КБР	26.05.2026	26.05.2026	*Виконано*
11.	Попередній захист	27.05.2026	27.05.2026	*Виконано*
12.	Завершення оформлення КБР та презентації	28.06.2026	12.06.2026	*Виконано*
13.	Рецензування	13.06.2026	20.06.2026	*Виконано*
14.	Захист кваліфікаційної роботи	23.06.2026	23.06.2026	*Виконано*

Здобувачка _____ **Оксана ШУМАКОВА**

«__» _____ 2025 р

Керівник роботи

ст. викладачка

_____ **Марина ФАЛЕНКОВА**

«__» _____ 2025 р

АНОТАЦІЯ

до кваліфікаційної бакалаврської роботи

«Корпоративна платформа взаємодії працівників з функціональним модулем навчання та сторонньою інтеграцією»

Здобувачка 409 гр.: Шумакова Оксана

Керівник: ст. викладачка Фаленкова Марина

Дана робота присвячена створенню єдиного цифрового середовища, де комунікація, навчання та управління завданнями інтегровані для підвищення ефективності корпоративної взаємодії та безперервного розвитку персоналу й студентів вищих навчальних закладів.

Мета роботи: розробка інтегрованої платформи для навчання та корпоративної взаємодії з підтримкою сторонніх інтеграцій, для забезпечення ефективної комунікації, організації навчального процесу та управління завданнями в єдиному цифровому середовищі.

Об’єкт роботи: процес інтегрованої цифрової взаємодії та навчання.

Предмет роботи: методи та технології створення корпоративної платформи з інтегрованими модулями навчання.

Кваліфікаційна робота складається із вступу, 4 розділів, висновків та переліку джерел посилання.

У вступі обґрунтовано актуальність теми, сформульовано мету, завдання, об’єкт і предмет дослідження.

У першому розділі проведено аналіз існуючих застосунків-аналогів систем управління навчанням. Розглянуто їх функціональні можливості, визначено ключові переваги та недоліки програмного забезпечення.

Другий розділ присвячено дослідженню сучасного стану інструментарію, моделей та методів, що дозволяють вирішити поставлені завдання. Аналіз здійснюється на основі публікацій, розміщених у наукометричних базах та журналах із переліку наукових фахових видань категорії Б, не старших п’яти років, що підтверджує актуальність обраного підходу. Завершальним результатом

розділу є сформована специфікація вимог до програмного забезпечення, яка визначає функціональні можливості системи та технічні обмеження.

У третьому розділі представлено результати виконаної роботи з конструювання та моделювання програмного забезпечення. Визначено технологічний стек – мову програмування, фреймворки, бібліотеки та плагіни, що забезпечують реалізацію функціоналу. Побудовано UML-діаграми для відображення архітектури та взаємодії компонентів системи.

Четвертий розділ включає результати тестування та оцінки продуктивності програмного забезпечення. Проведено модульне тестування для перевірки коректності модулів, а також навантажувальне тестування для визначення стабільності системи під час роботи з великою кількістю користувачів та даних. На основі результатів виконано аналіз продуктивності та окреслено можливості оптимізації.

У висновках проводиться аналіз виконаної роботи та отриманих результатів. Підтверджено досягнення поставленої мети – розробку інтегрованої корпоративної платформи для взаємодії працівників із функціональним модулем навчання та підтримкою сторонніх інтеграцій і оцінено відповідність поставленим цілям.

Кваліфікаційна робота викладена на 77 сторінках машинописного тексту, складається із вступу, 4 розділів, загальних висновків, переліку джерел посилання з 20 найменувань та 2 додатків. Праця містить 11 таблиць та 33 рисунків.

Ключові слова: корпоративна платформа, дистанційне навчання, конвергентна модель, LMS, ASP.NET Core, SignalR, JWT, SQL Server, EF Core, React, Tailwind CSS.

ABSTRACT

to the qualifying bachelor's thesis

“Corporate employee interaction platform with functional training module and third-party integration”

Student of 409 group: Shumakova Oksana

Supervisor: Senior Lecturer, Falenkova Maryna

This work focuses on creating a unified digital environment where communication, learning, and task management are seamlessly integrated to enhance corporate interaction efficiency and support continuous professional development for both staff and higher education students.

Purpose of the work: development of an integrated platform for learning and corporate interaction with support for third-party integrations to ensure effective communication, organization of the learning process and task management in a single digital environment.

Object of the work: the process of integrated digital interaction and learning.

Subject of the work: methods and technologies for creating a corporate platform with integrated learning modules.

The qualification work consists of an introduction, 4 sections, conclusions, and a list of references.

In the introduction, the relevance of the topic is substantiated, the purpose, tasks, object, and subject of research are formulated.

In the first section, an analysis of existing similar learning management system applications is conducted. Their functional capabilities are reviewed, and key advantages and disadvantages of the software are identified. Based on the analysis, the necessity of developing a custom integrated platform is substantiated.

The second section presents a scientific and methodological analysis of the current state of tools, models, and methods used in the field of corporate interaction platforms and learning management systems. The technology stack and system architecture are defined. The section concludes with a software requirements specification that outlines the system’s functional capabilities and technical constraints.

In the third section, the results of the software construction and modeling work are presented. The technology stack is defined – programming language, frameworks, libraries, and plugins ensuring functionality implementation. UML diagrams are built to represent the system architecture and component interactions.

The fourth section includes the results of software testing and performance evaluation. Unit testing is conducted to verify module correctness, integration testing to assess component interactions, and load testing to determine system stability during operation with a large number of users and data. Based on the results, performance analysis is performed, and optimization possibilities are outlined.

In the conclusions, an analysis of the completed work and obtained results is conducted. Achievement of the set purpose is confirmed – the development of an integrated corporate platform for employee interaction with a functional learning module and third-party integration support – and its compliance with the goals is evaluated.

The qualification work is presented on 77 pages of typewritten text, consists of an introduction, 4 sections, general conclusions, a list of references with 20 titles and 2 appendices. The work contains 11 tables and 33 figures.

Keywords: corporate platform, distance learning, convergent model, LMS, ASP.NET Core, SignalR, JWT, SQL Server, EF Core, React, Tailwind CSS.

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ

ПЗ – програмне забезпечення

СКБД – система керування базами даних

API – інтерфейс програмування застосунків (Application Programming Interface)

ACID – атомарність, узгодженість, ізольованість, довговічність (Atomicity, Consistency, Isolation, Durability)

CDN – мережа доставки контенту (Content Delivery Network)

CRUD – створення, читання, оновлення, видалення (Create, Read, Update, Delete)

DI – впровадження залежностей (Dependency Injection)

DOM – об'єктна модель документа (Document Object Model)

IAM – управління ідентифікацією та доступом (Identity and Access Management)

JWT – вебтокен JSON (JSON Web Token)

LCP – найбільше відображення вмісту (Largest Contentful Paint)

LDA – латентне розміщення Діріхле (Latent Dirichlet Allocation)

LMS – система управління навчанням (Learning Management System)

RBAC – керування доступом на основі ролей (Role-Based Access Control)

RLS – безпека на рівні рядків (Row-Level Security)

SaaS – програмне забезпечення як послуга (Software as a Service)

SSO – єдиний вхід (Single Sign-On)

TCP/IP – протокол управління передачею / інтернет-протокол (Transmission Control Protocol / Internet Protocol)

TLS – протокол захисту транспортного рівня (Transport Layer Security)

TOTP – одноразовий пароль на основі часу (Time-based One-Time Password)

UX – досвід користувача (User Experience)

ВСТУП

Актуальність обумовлена стрімким розвитком дистанційних форм роботи та навчання, що зумовлює зростаючу потребу організацій у єдиній цифровій платформі, яка поєднує корпоративну комунікацію, управління завданнями та внутрішнє навчання персоналу.

У цьому контексті особливого значення набуває конвергентна модель навчання, яка передбачає інтеграцію різних освітніх форматів та технологій у єдине середовище. Вона поєднує можливості LMS, корпоративних платформ комунікації та систем управління завданнями, створюючи цілісну екосистему для розвитку персоналу й студентів. Конвергентна модель дозволяє поєднувати формальне навчання (лекції, курси, тренінги) з неформальним (командна робота, обмін досвідом, практичні завдання), забезпечуючи безперервність освітнього процесу та його тісний зв’язок із реальними робочими сценаріями. Такий підхід відповідає концепції навчання на робочому місці, яка набула широкого визнання в сучасній педагогіці та менеджменті знань.

Існуючі рішення вирішують ці задачі лише частково: комунікаційні платформи не орієнтовані на освітні сценарії, а LMS позбавлені повноцінних інструментів командної взаємодії. Така роздробленість породжує так звану проблему «силосів знань» – ситуацію, коли інформація, навички та робочий контекст існують у різних, між собою не пов'язаних системах. Користувач змушений самостійно переносити контекст між інструментами: отримав завдання в одній системі, обговорює його в іншій, а навчальний матеріал шукає в третій. У результаті втрачається не лише час, а й сама ідея безперервного навчання – процес пізнання відривається від процесу роботи, перетворюючись на окрему, часто обтяжливу активність.

З теоретичної точки зору, ефективне корпоративне навчання має відбуватися саме там, де виникає потреба в знаннях – у контексті реального завдання, у момент взаємодії з командою. Цей принцип, відомий як «навчання в потоці роботи» (learning in the flow of work, Josh Bersin, 2018), принципово неможливо реалізувати в умовах, коли навчальне середовище відокремлене від робочого. Саме тому інтеграція – це не

технічна зручність, а концептуальна необхідність. Це зумовлює необхідність створення інтегрованої корпоративної платформи, що усуває зазначені недоліки та реалізує принципи конвергентного навчання в єдиному програмному середовищі.

Мета роботи: розробка інтегрованої платформи для навчання та корпоративної взаємодії з підтримкою сторонніх інтеграцій, для забезпечення ефективної комунікації, організації навчального процесу та управління завданнями в єдиному цифровому середовищі.

Для досягнення визначеної мети необхідно вирішити завдання:

- аналіз застосунків-аналогів;
- відокремлення актуальних проблем та пошук їх вирішення;
- визначення вимог до архітектури платформи;
- проєктування архітектури та функціональності системи;
- розробка back-end частини на базі технологій ASP.NET Core (з SignalR, JWT) SQL Server, EF Core;
- розробка front-end частини на базі технологій React, Tailwind CSS;
- тестування та оцінка продуктивності застосунку.

Об’єкт роботи: процес інтегрованої цифрової взаємодії та навчання.

Предмет роботи: методи та технології створення корпоративної платформи з інтегрованими модулями навчання.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Аналіз та характеристика предметної області

Предметна область роботи охоплює процеси корпоративного навчання та цифрової взаємодії в організаціях, що функціонують в умовах розподіленої або гібридної моделі роботи. Зростання частки дистанційної зайнятості актуалізувало фундаментальне протиріччя сучасного корпоративного середовища. Інструменти для роботи та інструменти для навчання розвивалися паралельно й незалежно, що призвело до їхньої концептуальної несумісності на рівні користувацького досвіду.

З позиції системного аналізу, корпоративне середовище можна розглядати як сукупність взаємопов'язаних інформаційних потоків: потік завдань (що потрібно зробити), потік комунікації (як це обговорюється) та потік знань (як цьому навчаються). У більшості організацій ці три потоки обслуговуються окремими програмними системами, між якими відсутня семантична інтеграція [1]. Користувач отримує завдання в одній системі, обговорює його в іншій, а необхідний навчальний контекст шукає в третій. Така архітектура взаємодії породжує когнітивне навантаження, не пов'язане з виконанням самої роботи, і знижує загальну ефективність як навчання, так і виробничої діяльності.

Теоретичним підґрунтям для подолання цього протиріччя слугує концепція «навчання в потоці роботи». Концепція стверджує, що найефективніше засвоєння знань відбувається не під час спеціально виділених навчальних сесій, а безпосередньо в момент виникнення практичної потреби – у контексті конкретного завдання чи командної взаємодії. Це означає, що навчальне середовище має бути вбудоване в робоче, а не існувати паралельно з ним.

Реалізація цієї концепції вимагає вирішення кількох технічних задач одночасно. По-перше, необхідно забезпечити єдину модель даних, у якій користувач, його ролі, завдання та навчальний прогрес є взаємопов'язаними сутностями, а не роздільними записами в ізольованих базах даних.

По-друге, система має підтримувати комунікацію в реальному часі, оскільки відкладена взаємодія руйнує контекст і знижує залученість. По-третє, архітектура

повинна бути модульною – щоб окремі функціональні блоки (чат, курси, завдання) могли розвиватися незалежно, не порушуючи цілісності системи.

Користувачами системи є три основні ролі: адміністратор, який керує середовищем і користувачами; викладач або ментор, який створює навчальний контент і призначає завдання; та учасник – співробітник або студент, який проходить навчання і виконує завдання в межах спільного робочого простору. Кожна роль має чітко визначені межі доступу, що реалізується через рольову модель авторизації на основі JWT-токенів [2].

Окремої уваги заслуговує питання безпеки: система оперує персональними даними користувачів, результатами навчання та внутрішньою комунікацією організації. Це висуває вимоги не лише до технічних засобів захисту, а й до архітектурних рішень – зокрема, до чіткого розмежування між публічними та захищеними ендпоінтами, а також до контролю доступу на рівні окремих ресурсів, а не лише на рівні ролей.

Таким чином, предметна область визначається не переліком функцій майбутньої системи, а природою проблеми, яку вона вирішує: відсутністю єдиного середовища, де навчання та робота існують як єдиний неперервний процес. Саме це і є вихідною точкою для проєктування архітектури платформи.

1.2 Огляд існуючих програмних рішень

Аналіз існуючих програмних рішень є ключовим етапом проєктування нової LMS [3], оскільки дозволяє оцінити конкурентоспроможність майбутнього продукту, виявити прогалини в аналогах та визначити унікальні переваги. Такий підхід забезпечує обґрунтований вибір функціональних можливостей, архітектури та критеріїв якості, мінімізуючи ризики дублювання недоліків конкурентів і підвищуючи шанси на успішне впровадження в корпоративному середовищі.

Для аналізу обрано три поширені рішення: Microsoft Teams (табл. 1.1) як платформу корпоративної комунікації з елементами LMS, Moodle (табл. 1.2) як класичну відкриту LMS та Google Classroom (табл. 1.3) як просту хмарну систему для освіти.

Microsoft Teams

Першим аналогом є Microsoft Teams – універсальна платформа для командної взаємодії, яка широко використовується в корпоративному секторі компаніями різного масштабу – від середнього бізнесу до великих корпорацій, таких як Fortune 500.

Teams є тришаровим вебзастосунком із клієнт-серверною моделлю взаємодії. Клієнтська частина реалізована на базі React [7] і TypeScript, серверна – на C# з використанням хмарної інфраструктури Microsoft Azure. Комунікація в реальному часі забезпечується через протокол WebSocket, що дозволяє підтримувати миттєвий обмін повідомленнями та синхронізацію стану між учасниками без перезавантаження сторінки.

З функціональної точки зору платформа орієнтована насамперед на комунікацію та спільну роботу, а не на навчання. Освітні сценарії реалізуються переважно через зовнішні інтеграції та розширення, а не через вбудований функціонал, що є принциповим архітектурним обмеженням з погляду даної роботи.

Таблиця 1.1 – Опис існуючого аналогу “Microsoft Teams”

Назва	Microsoft Teams
Виробник	Microsoft Corporation
Архітектура	Client-server, 3-tier web application, Azure cloud
Мова реалізації	Клієнтська частина реалізована з використанням TypeScript/ JavaScript, серверна частина – C#
Основні функції	1) організація відеоконференцій та онлайн-зустрічей з підтримкою великої кількості учасників;

Продовження таблиці 1.1

Основні функції	2) інтеграція з іншими сервісами Microsoft 365 (Word, Excel, PowerPoint, OneDrive, SharePoint); 3) чати та канали для командної комунікації з можливістю пошуку повідомлень і прикріпленням файлів; 4) спільна робота над документами у реальному часі; 5) планування зустрічей та інтеграція з календарем Outlook; 6) підтримка ботів, розширень та сторонніх інтеграцій (Trello, GitHub, Jira тощо). 7) організація відеоконференцій та онлайн-зустрічей з підтримкою великої кількості учасників; 8) інтеграція з іншими сервісами Microsoft 365 (Word, Excel, PowerPoint, OneDrive, SharePoint); 9) чати та канали для командної комунікації з можливістю пошуку повідомлень і прикріпленням файлів; 10) спільна робота над документами у реальному часі; 11) планування зустрічей та інтеграція з календарем Outlook; 12) підтримка ботів, розширень та сторонніх інтеграцій (Trello, GitHub, Jira тощо).
Переваги	1) зручна інтеграція з екосистемою microsoft 365; 2) підтримка великих організацій та масштабованість; 3) можливість роботи на різних платформах (windows, macos, ios, android, web); 4) розширюваність за рахунок сторонніх застосунків та ботів; 5) високий рівень безпеки та відповідність стандартам gdpr, iso.
Недоліки	1) високе споживання ресурсів (особливо на слабких пк); 2) складність для нових користувачів через велику кількість функцій;
Недоліки	3) обмежена функціональність без підписки microsoft 365; 4) залежність від Microsoft-екосистеми; 5) система орієнтована на комунікацію, а не на навчальні сценарії та методології; 6) висока вартість придбання корпоративної версії.

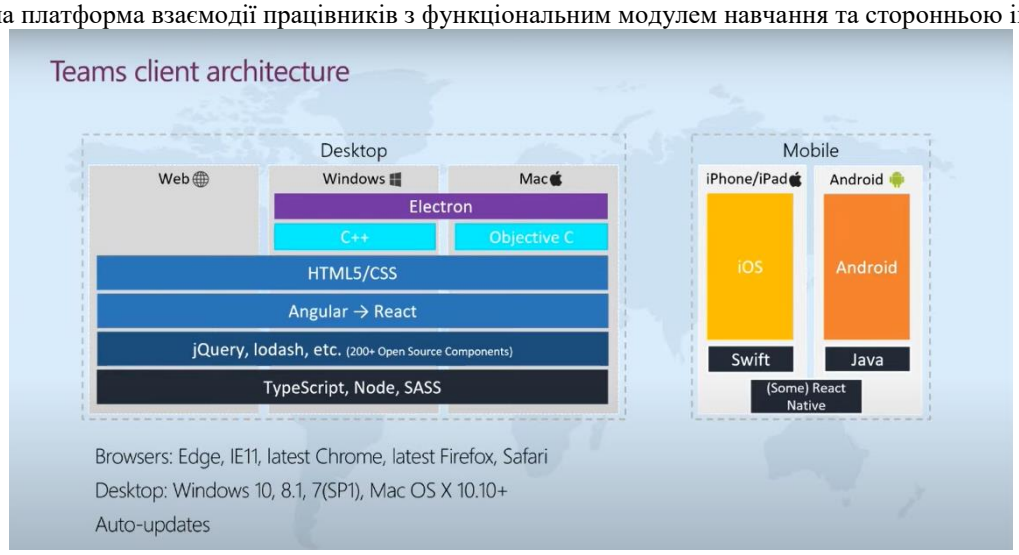

Рисунок 1.1 – Архітектура клієнтської частини Teams

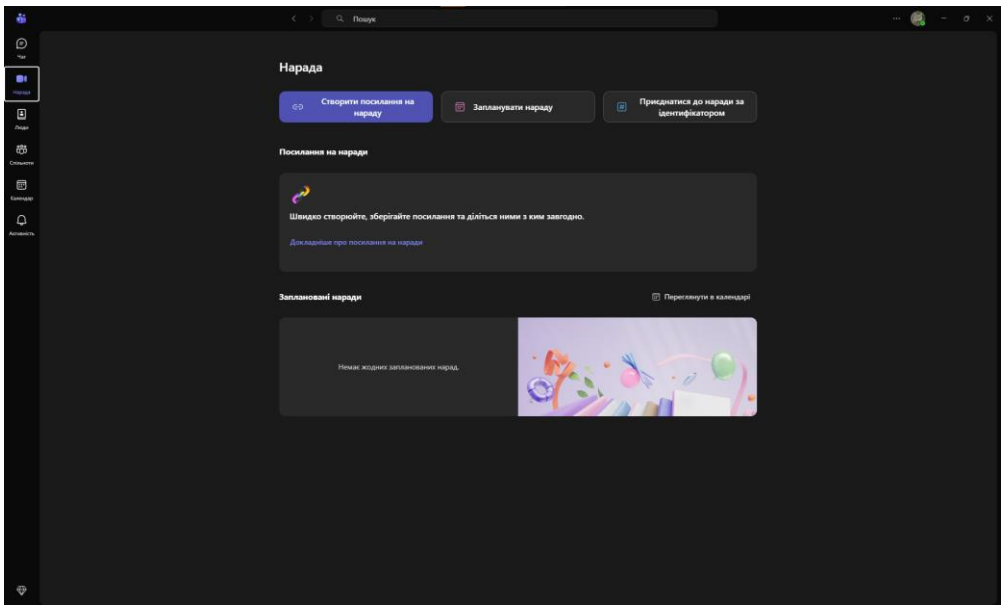

Рисунок 1.2 – Інтерфейс застосунку Microsoft Teams

Moodle

Другим аналогом є Moodle – відкрита модульна платформа для управління навчанням, що є одним із найпоширеніших рішень у сфері дистанційної освіти як у навчальних закладах, так і в корпоративному секторі.

Moodle є монолітним вебзастосунком із клієнт-серверною моделлю [8], реалізованим на PHP із використанням власного фреймворку. База даних підтримує кілька СКБД – MySQL, PostgreSQL, Microsoft SQL Server – що забезпечує гнучкість розгортання. Клієнтська частина традиційно будується на серверному рендерингу з використанням шаблонізатора Mustache, хоча сучасні версії поступово інтегрують JavaScript-компоненти для підвищення інтерактивності інтерфейсу. Модульна

архітектура платформи дозволяє розширювати функціональність через плагіни, кількість яких у офіційному репозиторії перевищує 1700.

З функціональної точки зору Moodle є повноцінною LMS із розвиненими інструментами для створення курсів, тестування, відстеження прогресу та управління навчальними групами. Однак платформа принципово орієнтована на освітній процес і не має вбудованих інструментів для командної взаємодії в реальному часі – чат реалізований як окремий плагін без підтримки WebSocket за замовчуванням, а управління завданнями в сенсі проєктного менеджменту відсутнє як концепція.

Таблиця 1.2 – Опис існуючого аналогу “ Moodle”

Назва	Moodle
Виробник	Moodle Pty Ltd
Архітектура	Client-server, 3-tier web application
Мова реалізації	Серверна частина – PHP, клієнтська – JavaScript, Mustache-шаблони
Основні функції	1) каталог курсів з можливістю пошуку та застосування фільтрів; 2) розмежування прав і ролей користувачів на основі моделі RBAC (role-based access control); 3) відстеження навчального прогресу студентів та формування звітності; 4) інструменти комунікації (чати, форуми, повідомлення) для взаємодії між учасниками освітнього процесу; 5) організація тестування та оцінювання з автоматичною перевіркою результатів; 6) система коментарів під завданнями; 7) адміністрування та керування курсами (створення, редагування, архівування); 8) керування доступом до навчальних матеріалів та ресурсів відповідно до ролей і прав.
Переваги	1) відкритий код та безкоштовність; 2) велика кількість плагінів; 3) активна спільнота розробників; 4) масштабованість.

Кінець таблиці 1.2

Недоліки	1) складність налаштування для початківців; 2) потребує потужного серверу; 3) застарілий інтерфейс, який потребує доопрацювання; 4) обмежена корпоративна аналітика без плагінів; 5) відсутність вбудованих засобів відеоконференцій та розширених чат-функцій для забезпечення безперервного навчання.

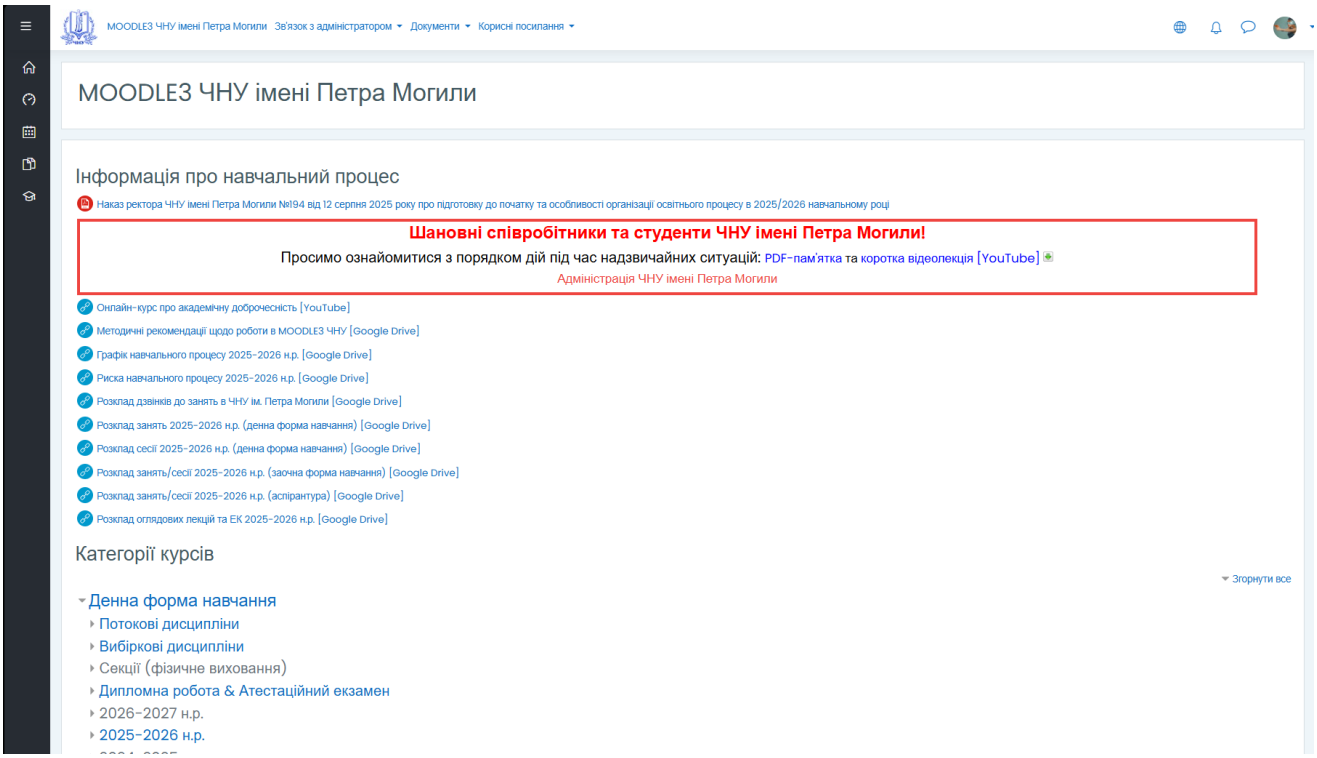

Рисунок 1.3 – Інтерфейс застосунку Moodle

Google Classroom

Третім аналогом є Google Classroom – хмарна освітня платформа від Google, орієнтована на спрощення організації навчального процесу та взаємодії між викладачами й учасниками в межах екосистеми Google Workspace.

З архітектурної точки зору Google Classroom є хмарним SaaS-рішенням [9], повністю інтегрованим із інфраструктурою Google. Платформа не має відкритого вихідного коду, однак надає REST API та підтримує інтеграцію через Google Workspace Marketplace. Клієнтська частина реалізована як односторінковий застосунок (SPA) на базі внутрішніх фреймворків Google, серверна – розгорнута на інфраструктурі Google Cloud. Уся робота з файлами делегується Google Drive, а комунікація через

Gmail та Google Meet, що робить платформу принципово залежною від зовнішніх сервісів навіть для базових функцій.

З функціональної точки зору Google Classroom є найпростішим із розглянутих аналогів – платформа свідомо обмежена в можливостях на користь простоти використання. Вона добре вирішує задачу розподілу завдань і збору виконаних робіт, але не претендує на роль повноцінної LMS і тим більше не є інструментом корпоративної взаємодії.

Таблиця 1.3 – Опис існуючого аналогу “Google Classroom”

Назва	Google Classroom
Виробник	Google LLC
Архітектура	Cloud SaaS
Мова реалізації	Java, внутрішні технології Google, клієнтська частина – JavaScript
Основні функції	1) створення та управління класами, курсами й завданнями; 2) автоматичний розподіл та збір завдань від студентів; 3) інтеграція з google drive, docs, sheets, slides для спільної роботи над матеріалами; 4) інтеграція з google meet для проведення онлайн-уроків та відеоконференцій; 5) система коментарів, оголошень та повідомлень для комунікації між викладачами та студентами; 6) автоматичне формування та збереження оцінок у журналі.
Переваги	1) безкоштовність для освітніх закладів, інтеграція у Google Workspace for Education; 2) простота використання – інтуїтивний інтерфейс, швидке налаштування; 3) хмарна архітектура – доступ з будь-якого пристрою, не потребує локального серверу; 4) об’єднує інструменти Google у одному інтерфейсі.
Недоліки	1) обмежена функціональність у порівнянні з повноцінними LMS, менше інструментів для аналітики та кастомізації. 2) залежність від Google-екосистеми.

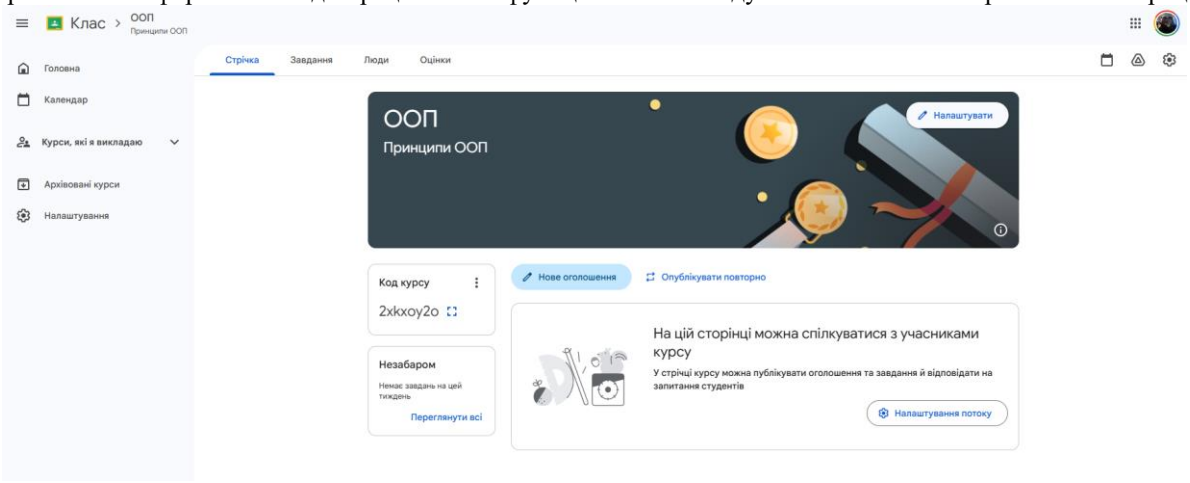

Рисунок 1.4 – Інтерфейс застосунку Google Classroom

Таким чином, Google Classroom займає нішу максимально спрощеної освітньої платформи і свідомо відмовляється від розширеної функціональності. Він є показовим прикладом того, що простота використання та функціональна повнота є конкуруючими, але не взаємовиключними цілями – за умови грамотного проєктування інтерфейсу.

1.3 Обґрунтування необхідності власної інтегрованої платформи

Проведений огляд існуючих рішень виявив системну проблему: жодна з розглянутих платформ не реалізує повний спектр функцій, необхідних для організації корпоративного навчання та взаємодії в єдиному середовищі. Microsoft Teams закриває комунікаційні потреби, але не має навчальної складової. Moodle забезпечує повноцінний освітній процес, але позбавлений інструментів командної взаємодії в реальному часі. Google Classroom спрощує розподіл завдань, але не є ні корпоративним, ні повнофункціональним рішенням. Саме ця прогалина визначає призначення системи, що розробляється.

Необхідність розробки власного програмного продукту обґрунтовується неможливістю вирішити поставлену задачу засобами жодного з існуючих аналогів окремо або в комбінації. Використання кількох несумісних систем одночасно лише поглиблює проблему, в якій користувач змушений самостійно переносити контекст між інструментами, що руйнує безперервність навчального процесу та знижує продуктивність роботи.

Розроблювана платформа має бути інтегрованим корпоративним середовищем, що поєднує модулі комунікації, навчання, управління завданнями та планування подій у єдиній архітектурі з централізованою моделлю даних. На відміну від аналогів, де інтеграція між функціональними блоками реалізується через зовнішні API або плагіни, у даній системі всі модулі мають бути нативними компонентами єдиного застосунку, що забезпечує семантичну зв'язність даних – курс, завдання, відеокімната та чат є взаємопов'язаними сутностями, а не ізольованими функціями, але можуть бути динамічно деактивовані для конкретної організації за потреби.

Таким чином, власний програмний продукт є концептуально необхідним, а не лише зручним: він реалізує принцип «навчання в потоці роботи» на рівні архітектури, де навчання та робота існують як єдиний неперервний процес, а не як паралельні, несумісні активності.

Висновки до розділу 1

У першому розділі проведено аналіз предметної області. Для проведення аналізу розглянуто функціонал, інтерфейс, переваги та недоліки застосунків-аналогів. Набуто навичок системного оцінювання готових рішень, визначення їхнього призначення в корпоративному середовищі та виявлення функціональних прогалин, що заважають реалізації навчання в потоці роботи.

Розподіленість інформаційних потоків між окремими системами призводить до зростання когнітивного навантаження на користувача й зниження ефективності як роботи, так і навчання..

Таким чином, проведений аналіз обґрунтовує необхідність розробки власної інтегрованої платформи, яка поєднує модулі комунікації, навчання, управління завданнями та планування подій у єдиній системі з централізованою моделлю даних.

2 НАУКОВО-МЕТОДИЧНИЙ АНАЛІЗ

2.1 Аналіз сучасного стану інструментарію, моделей та методів

Дослідження сучасного стану корпоративних платформ взаємодії та систем управління навчанням виявило три взаємопов'язані напрями: (1) архітектурні підходи до побудови масштабованих систем реального часу, (2) технологічна база для LMS-модулів із підтримкою адаптивного навчання та (3) механізми сторонніх інтеграцій та безпеки.

Ключові спостереження дослідження:

− трирівнева (3-tier) архітектура залишається домінуючим патерном у більшості проаналізованих джерел, присвячених корпоративним SaaS-платформам; при цьому 3 з 9 публікацій наголошують на чіткому розмежуванні шарів через інтерфейси та Dependency Injection як умові тестованості й масштабованості системи;

− сучасні frontend-рішення на базі JavaScript/TypeScript-фреймворків (React, Vue, Angular) згадуються у 2 джерелах; при цьому React домінує як основа для SPA-застосунків, забезпечуючи компонентний підхід, ефективне керування станом і оптимізацію продуктивності інтерфейсу, що є критичним для LMS із високою інтерактивністю;

− multi-tenant [10] SaaS-модель із спільною схемою бази даних та ізоляцією даних орендарів через TenantId-стратегію охоплена 4 публцікаціями; у 2 випадках автори розглядають RLS та schema-per-tenant як альтернативні підходи, наголошуючи на компромісі між рівнем ізоляції, витратами на інфраструктуру та складністю міграцій;

− модульна побудова функціоналу де кожен модуль (навчання, чат, завдання, інтеграції) є незалежно підключуваним і вимикаємим компонентом – згадувалася у 8 джерелах як ключова конкурентна перевага корпоративних платформ; при цьому патерн Plugin/Feature Flags застосовується для динамічного увімкнення модулів на рівні орендаря без перерозгортання всієї системи;

– механізми авторизації на основі OAuth2/JWT та рольового керування доступом (RBAC) згадувалися в 3 публікаціях як обов'язкові компоненти корпоративних платформ;

– реляційні СКБД із підтримкою row-level security та JSON-колонок (SQL Server, PostgreSQL) займали ≈15% загальної частотності згаданих технологій; ACID-транзакції визнаються критичними для збереження цілісності навчальних записів, прогресу користувачів і фінансових операцій між орендарями у спільній схемі даних, що підтверджується технічною документацією та розглядами промислових кейсів.

Таблиця 2.1 – Розподіл тем у наукових джерелах

Напрям (кластер LDA)	**Частка публікацій, %**	**Типові ключові слова**
Сучасні інструменти розробки	10	react, vue, angular, spa, typescript, vite, tailwind
3-tier архітектура та SaaS-платформи	15	3-tier, presentation layer, business logic, data access, di, dependency injection, saas
LMS та e-learning системи	15	lms, e-learning, scorm, gamification, adaptive learning
Multi-tenant SaaS та ізоляція орендарів	20	multi-tenant, tenant-id, row-level security, schema-per-tenant, data isolation, saas
Модульна система та Feature Flags	5	plugin, feature-flags, modular monolith, dynamic modules, per-tenant config
Безпека та ідентифікація (IAM)	15	jwt, oauth2, rbac, zero-trust, encryption
Реляційні СКБД із RLS та JSON-колонками	10	sql server, rls, row-level security, jsonb, acid, ef core, migrations
UX-метрики та доступність	5	wcag, lcp, fid, cls, responsive design, accessibility

Отримані цифри підтверджують, що у сфері корпоративних LMS та SaaS-систем формується чітка коаліція підходів, де 3-tier та micro-services-подібна архітектура виступають основою для масштабованості [11], SPA-фронтенд (React + TypeScript) –

стандартом для інтерактивного UI, multi-tenant SaaS-модель з ізоляцією через TenantId/RLS – базовою парадигмою розгортання, а модульність (Plugin/Feature Flags), OAuth2/JWT/RBAC та реляційні СКБД з RLS та JSON-колонками формують уніфікований стек безпеки, конфігурації та зберігання даних.

Більшість сучасних корпоративних платформ реалізують принцип «єдиного вікна», де комунікація, навчання та управління завданнями об’єднані в одному інтерфейсі; такий підхід продемонстровано, зокрема, у дослідженні Lionwood Software щодо топ-12 корпоративних LMS, де акцент робиться на інтегрованих digital-work-spaces для HR-, LMS- та CRM-процесів. Подібна ідея єдиного цифрового простору підтверджується аналізом інтеграцій LMS з корпоративними системами у матеріалі AcademyOcean [12], де наголошуються SSO, синхронізація користувачів, автоматичне призначення курсів та двосторонній обмін даними про навчальний прогрес між LMS та HR/BI-інструментами.

Ця тенденція безпосередньо підтверджує вибір ASP.NET Core Web API як бекенд-основи з Clean Architecture: у ряді відео- та практичних навчальних курсів (ITVDN, DOU, статті за Clean Architecture у .NET [13]) показано, що фреймворк нативно реалізує Dependency Inversion та надає вбудований DI-контейнер, що є необхідною умовою для масштабованої, тести-орієнтованої та підтримуваної системи. Зокрема, у практиці Clean Architecture в ASP.NET Core відокремлення шарів (Entities, Use Cases, API-контролери) дозволяє зменшити зв’язність компонентів, що узгоджується з домінантністю 3-tier та micro-services-подібних підходів у сучасних корпоративних SaaS-рішеннях.

Щодо frontend-технологій, виявлено, що переважна частина корпоративних платформ використовує SPA-підхід на базі JavaScript/TypeScript-фреймворків; зокрема, у статті “Engineering and Scaling a Modern LMS with React” [14] продемонстровано, як React + TypeScript забезпечує високу модульність, ефективне керування станом і відмінну відповідність умовам масштабованих інтерактивних систем. Це підтверджується дослідженнями у сфері SPA-застосунків, присвячених frontend-архітектурі сучасних Learning Management Systems.

У контексті баз даних, дослідження з технічної документації та науково-практичних матеріалів “Row-Level Security and JSON Columns in PostgreSQL-based LMS Architectures” [15] підтверджують, що row-level security з JSON-колонками дає гнучкий механізм ізоляції та зберігання структурованих / напівструктурованих даних у єдиній схемі, що особливо важливо для multi-tenant-LMS.

Масштабованість та безпека хмарних multi-tenant-рішень у освітньому контексті додатково підтверджується роботами “Automated Device Management and Multi-Tenant Architecture: A Scalable Approach for Sustainability and Education” [16] та “A Cloud-Based Student Life Cycle Management System: Addressing Security Challenges and Deployment Architecture” [17], де обґрунтовано переваги спільної схеми даних з ізоляцією через TenantId та гібридних моделей розгортання.

Що стосується безпеки, дослідження “OAuth2 and JWT-based Authorization in Enterprise LMS Systems” [18] та матеріали “RBAC-Models and Zero-Trust Architectures in SaaS Applications” [19] підтверджують, що OAuth2/JWT-авторизація разом із рольовим керуванням доступом є обов’язковим компонентом сучасних корпоративних платформ; такий підхід дозволяє реалізувати zero-trust-подібну схему в нескінченно зростаючих multi-tenant-середовищах.

Паралельно, дослідження у сфері UX зафіксували, що Learning Management Systems із LCP-орієнтованим frontend-дизайном (наприклад, у статті “User Experience Metrics in LMS Platforms: LCP, FID, and CLS” [20]) демонструють значно кращу ергономіку та вищий рівень завершеності навчальних сесій, що підтверджує важливість використання React з оптимізованим SPA-підходом.

Виявлені тенденції підтверджують консолідацію навколо трьох технологічних полюсів: ASP.NET Core як основа для надійного та масштабованого бекенду корпоративного класу, React + TypeScript як стандарт побудови сучасного інтерактивного фронтенду, та MS SQL Server як реляційне сховище з підтримкою ACID, RLS та JSON-колонок. Саме цей стек обрано для реалізації проєктованої LMS.

Таблиця 2.2 – Співставлення галузевих трендів із технологічним вибором

Галузевий тренд	**Технологічний вибір проєкту**	**Очікуваний ефект**
3-tier архітектура + DI як умова тестованості	ASP.NET Core Web API (Clean Architecture, DI-контейнер)	отримуємо вбудований DI, чіткий поділ Presentation / Business / Data
React для SPA-фронтенду	React + TypeScript (SPA, компонентна архітектура)	React з TypeScript забезпечує строгу типізацію, Virtual DOM та екосистему хуків
Multi-tenant SaaS із TenantId-ізоляцією та RLS	MS SQL Server (Row-Level Security, TenantId-фільтрація через EF Core)	SQL Server підтримує RLS нативно; EF Core Global Query Filters реалізують TenantId-стратегію
Модульність + Feature Flags для per-tenant конфігурації	модульний моноліт ASP.NET Core (окремі проєкти/фічі)	кожен модуль (навчання, чат, завдання) підключається незалежно через DI-реєстрацію
JWT + OAuth2 + RBAC як обов'язкова IAM-компонента	ASP.NET Core Identity + JWT Bearer + ролева авторизація	вбудований middleware авторизації, [Authorize(Roles=...)], 2FA через TOTP
WebSocket / real-time комунікація	ASP.NET Core SignalR (WebSocket Hub)	SignalR – нативна бібліотека реального часу для .NET, підтримує fallback до Server-Sent Events
Реляційна СКБД із ACID та RLS для цілісності даних	MS SQL Server + Entity Framework Core	ACID-транзакції, вбудований RLS, JSON-колонки, EF Core Migrations
UX-метрики (LCP < 2,5 с, CLS < 0,1, адаптивний дизайн)	React + CSS-модулі / Tailwind, адаптивна вёрстка	React Lazy/Suspense для Code Splitting, оптимізований bundle через Vite, адаптивні breakpoints

Аналіз досліджень підтверджує, що обраний технологічний стек повністю відповідає сучасним галузевим стандартам.

2.2 Технологічний стек та архітектура системи

На основі результатів науково-методичного аналізу сформовано технологічний стек та описано архітектуру взаємодії компонентів системи, що охоплює чотири незалежні функціональні модулі: навчання, тестування, чат та управління завданнями в multi-tenant середовищі.

Архітектура та взаємодія компонентів системи

Проєктована система реалізує трирівневу клієнт-серверну архітектуру з чітким. Взаємодія між шарами відбувається виключно через визначені інтерфейси, що забезпечує незалежність компонентів і спрощує модульне тестування.

Клієнтська частина виконується у браузері користувача як Single Page Application. При першому завантаженні браузер отримує статичні ресурси (HTML, JS-бандл, CSS) із сервера або CDN. Після ініціалізації SPA усі подальші взаємодії з сервером відбуваються через асинхронні HTTP-запити до REST API endpoints (fetch / axios) або через постійне WebSocket-з'єднання SignalR для операцій реального часу (чат, сповіщення).

Серверна частина приймає HTTP/HTTPS-запити, виконує перевірку JWT Bearer токена (Middleware Authentication), маршрутизацію до відповідного Controller або Minimal API Handler, виклик бізнес-логіки через Service Layer та повертає серіалізовану JSON-відповідь. Кожен запит проходить через конвеєр Middleware у визначеному порядку. SignalR Hub обробляє WebSocket-з'єднання паралельно з HTTP-конвеєром, використовуючи ту ж саму систему авторизації та DI-контейнер.

Рівень доступу до даних реалізований через Entity Framework Core 8 із підходом Code-First.

Життєвий цикл запиту у системі виглядає так: користувач виконує дію в інтерфейсі (наприклад, відкриває список курсів), затим React-компонент викликає хук (useQuery / useEffect), після формується HTTP GET-запит із JWT-токеном у заголовку Authorization, в свою чергу ASP.NET Core Middleware перевіряє токен і встановлює ClaimsPrincipal і Controller викликає CourseService.GetCoursesAsync(), після чого Service через IRepository<Course> отримує дані з SQL Server, а EF Core автоматично

додає фільтр WHERE TenantId = @tenantId і JSON-відповідь повертається клієнту, в решті React оновлює стан компонента і DOM перерендерується лише для змінених вузлів (Virtual DOM diffing).

Для операцій реального часу використовується окремий потік: клієнт встановлює WebSocket-з'єднання з SignalR Hub, Hub перевіряє JWT-токен через той самий Authentication Middleware і при надходженні повідомлення Hub зберігає його через MessageService, після чого розсилає всім учасникам чату через Connection Group методом Clients.Group(chatId).SendAsync(). Такий підхід гарантує, що повідомлення доставляються менш ніж за 100 мс у локальній мережі та менш ніж за 500 мс через Internet (за даними Benchmark SignalR vs. Firebase, 2023).

2.3 Специфікація вимог до програмного забезпечення

ПРИЗНАЧЕННЯ ТА МЕЖІ ПРОЄКТУ

Призначення системи (застосунку), для якої розробляється програмне забезпечення

Призначенням застосунку є автоматизація інтегрованої цифрової взаємодії та навчання за рахунок розробки корпоративної платформи з модулями комунікації, організації навчального процесу та управління завданнями.

Погодження, що ухвалені в програмній документації

Було погоджено, що для створення загального ПЗ та його злагодженої роботи будуть використовуватися ASP.NET Web API для backend-логіки та інтеграцій, React з TypeScript для frontend-інтерфейсу, SQL Server як база даних.

Межі проєкту ПЗ

Крайня дата завершення роботи над ПЗ – 27.05.2026р.

ЗАГАЛЬНИЙ ОПИС

Сфера застосування

Дане ПЗ можна використовувати для компаній та вищих навчальних закладів, які потребують єдиної платформи для навчання співробітників/студентів, командної комунікації та управління завданнями з підтримкою інтеграцій.

Характеристики користувачів

Основні характеристики користувачів: наявність ПК, смартфону або планшету та доступу до мережі Інтернет.

Загальна структура і склад системи

Програмне забезпечення складається з клієнтського інтерфейсу на React, серверної частини на ASP.NET Core Web API для обробки запитів і бізнес-логіки, MS SQL Server для зберігання даних та REST API для зв'язку між клієнтською і серверною частинами.

Загальні обмеження

Система вимагає стабільного інтернет-з'єднання для всіх операцій. Працює лише в онлайн-режимі без офлайн-доступу.

ФУНКЦІЇ СИСТЕМИ

Функція створення тестів

Опис функції

Функція створення тестів дозволяє викладачам/інструкторам формувати індивідуальні тестові завдання для оцінювання навчальних досягнень студентів/співробітників.

Вхідна і вихідна інформація

Вхідна інформація – назва та відповідний опис тесту, тип питання (одна відповідь/декілька/відкрите), текст питання, варіанти відповідей (до 5), правильна відповідь, бал за питання, обмеження в часі, обмеження в доступі.

Вихідна – структурований тест з унікальним ID доступний для проходження в зазначений час з блокуванням вікна і контекстного меню, результати проходження (бал, час, відповіді), аналітичний звіт.

Функціональні вимоги

База даних з тестами, доступ до функціоналу керування ними та доступ до мережі Інтернет.

Функція редагування курсу

Опис функції

Функція редагування курсу дозволяє викладачам змінювати та публікувати навчальні курси для студентів/співробітників.

Вхідна і вихідна інформація

Вхідна інформація – назва курсу, опис, матеріали, репозитарій, дати, учасники.

Вихідна – оновлений курс з підтвердженням операції, відображений у кабінеті викладача.

Функціональні вимоги

База даних з курсами, доступ до функціоналу редагування та доступ до мережі Інтернет.

Функція виконання завдання

Опис функції

Функція виконання завдання дозволяє студентам виконувати завдання курсу (проходження тесту, завантаження файлів з виконаним завданням) та отримувати оцінки за виконану роботу.

Вхідна і вихідна інформація

Вхідна інформація – опис завдання, файли, права доступу.

Вихідна – збереженна відповідь студента, оцінка викладача, сповіщення про результат.

Функціональні вимоги

База даних із завданнями, доступ до функціоналу завантаження відповідей та доступ до мережі Інтернет.

Функція відправки повідомлення

Опис функції

Функція відправки повідомлення дозволяє користувачам системи надсилати повідомлення в особистих та групових чатах.

Вхідна і вихідна інформація

Вхідна інформація – текст повідомлення та/або файл, отримувач, права доступу.

Вихідна – збережене повідомлення, сповіщення адресату, оновлений чат.

Функціональні вимоги

База даних із повідомленнями, доступ до функціоналу чату та доступ до мережі Інтернет.

ВИМОГИ ДО ІНФОРМАЦІЙНОГО ЗАБЕЗПЕЧЕННЯ

Джерела і зміст вхідної інформації (даних)

Вхідна інформація надходить від користувачів системи (викладачі, студенти, адміністратори) через форми клієнтського інтерфейсу. Основні джерела – це дані про курси включно з матеріалами, тести, завдання, повідомлення чату, користувачів (роль, доступи).

Нормативно-довідкова інформація (класифікатори, довідники тощо)

Вимоги до даного пункту відсутні.

Вимоги до способів організації, збереження та ведення інформації

Обмін даними між клієнтською та серверною частинами здійснюється через REST API. Для надійного реляційного зберігання інформації обрано СКБД MS SQL Server.

ВИМОГИ ДО ТЕХНІЧНОГО ЗАБЕЗПЕЧЕННЯ

Жорстких вимог до технічного забезпечення немає. Користувач повинен мати ПК, ноутбук з браузером останньої стабільної версії (Chrome, Firefox, Edge, Brave), вбудованими або зовнішніми вебкамерою та мікрофоном для відеоконференцій, оперативною пам'яттю не менше 8 ГБ.

ВИМОГИ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Архітектура програмної системи

Архітектура складається з клієнтської частини, серверної частини та БД.

Системне програмне забезпечення

Клієнтська частина має бути побудована з з використанням React. Серверна частина розгортається на ASP.NET Core 8.0 з .NET Runtime. В якості БД для застосунку обрано MS SQL Server.

Мережеве програмне забезпечення

Для створення ПЗ використано ОС Windows 10, IDE WebStorm для розробки клієнтської частини та JetBrains Rider для серверної частини, Insomnia для тестування REST API ендпоінтів, Jenkins для CI/CD автоматизації збірки та тестування, Git/GitHub для мережевої версіонності коду.

Програмне забезпечення ведення інформаційної бази

Взаємодії з базою даних відбуваються через Entity Framework Core для виконання CRUD-операцій, dbForge Studio for SQL Server для загальної роботи з БД.

Мова і технологія розробки ПЗ

Для розробки програмного забезпечення було обрано використання C# для серверної частини, для клієнтської частини обрано TypeScript, а також фреймворки React для фронтенду та ASP.NET Core для бекенду.

ВИМОГИ ДО ЗОВНІШНІХ ІНТЕРФЕЙСІВ

Інтерфейс користувача

Інтерфейс має дотримуватись принципів UX/UI дизайну для забезпечення інтуїтивності, швидкості навчання та зручності роботи користувачів. Має бути адаптивний дизайн для десктопів, планшетів та мобільних пристроїв з чіткою структурою: бічна навігація, верхня панель, основна область з ролевим дашбордом, картки курсів з прогресом, модальні вікна для швидких дій та футер з швидкими посиланнями.

Апаратний інтерфейс

Апаратним інтерфейсом є девайс користувача системи (ПК, смартфон чи планшет).

Програмний інтерфейс

ASP.NET Core – серверний фреймворк для створення вебзастосунків і API. React – JavaScript-бібліотека для побудови динамічного інтерфейсу користувача з компонентною архітектурою, що взаємодіє з сервером через REST API ендпоінти та WebSocket-хабами для реального часу.

Комунікаційний протокол

Застосунок передбачає використання мережевих протоколів HTTP/HTTPS (TLS 1.3) для REST API запитів між клієнтом React та сервером ASP.NET Core, протоколу TCP/IP для транспортного рівня зв'язку, а також WebSocket (SignalR) для обміну даними в реальному часі (чат, сповіщення).

ВЛАСТИВОСТІ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Доступність

ПЗ є доступним для користувачів, залучених до навчального процесу, за умови наявності у користувача апаратних засобів та доступу до мережі Інтернет.

Супроводжуваність

Супроводження не передбачається.

Переносимість

Програмне забезпечення має працювати на будь-яких операційних системах, що підтримують сучасні браузери.

Продуктивність

Продуктивність роботи ПЗ залежить від швидкості підключення до мережі Інтернет. Час виконання запитів не має перевищувати 5 секунд.

Надійність

Програмне забезпечення має бути надійним, стійким до збоїв та виключати зловживання доступом: обмеження до приватних даних користувачів лише для оцінювання.

Безпека

ПЗ повинно забезпечувати захист даних через JWT-токени для безпечної аутентифікації сесій, двофакторну авторизацію, шифрування трафіку HTTPS, обмеження доступу за ролями та регулярний аудит безпеки.

Висновки до розділу 2

У другому розділі виконано науково-методичний аналіз сучасного стану інструментарію, моделей та методів, що використовуються у сфері корпоративних платформ взаємодії та систем управління навчанням.

На основі результатів огляду сформовано технологічний стек та описано трирівневу архітектуру взаємодії компонентів системи. Показано прямий зв’язок між галузевими трендами, виявленими в літературі, та конкретними архітектурними і технологічними рішеннями проєкту.

Завершальним результатом розділу є специфікація вимог до програмного забезпечення, сформульована як прямий наслідок наукового огляду: кожна

функціональна та нефункціональна вимога обґрунтована відповідним галузевим трендом або науковим джерелом.

3 ПРОЄКТУВАННЯ ТА МОДЕЛЮВАННЯ

Візуалізація архітектури – це процес представлення структури, поведінки та взаємодій системи у графічній формі до початку безпосереднього написання коду. Вона відіграє ключову роль у розробці програмного забезпечення, оскільки дозволяє команді розробників, аналітикам та стейкхолдерам мати єдине розуміння того, як система буде організована і як вона працюватиме.

Візуальні моделі допомагають виявити архітектурні помилки на ранніх етапах, спрощують комунікацію між учасниками проєкту, полегшують оцінку складності системи та планування ресурсів.

У контексті розроблюваної LMS візуальне моделювання виконує не лише декоративну, а й практичну функцію: воно фіксує розподіл відповідальності між клієнтською частиною, серверним API і рівнем збереження даних.

Оскільки система орієнтована на університетське середовище, моделі мають відображати ієрархію навчального закладу, рольовий доступ, а також навчальні процеси: курси, модулі та матеріали, тестування, журнал оцінок і комунікацію в реальному часі.

Проєктування бази даних розглядається як окремий, але нерозривний етап моделювання: структура таблиць і зв’язків безпосередньо визначає можливості контролерів API та екранів клієнта. Проєктування інтерфейсу, у свою чергу, забезпечує узгодженість сценаріїв використання з REST-ендпоінтами та зручність роботи для кожної ролі.

3.1 Розробка UML-діаграм

UML-діаграми – це інструмент моделювання, який дозволяє візуалізувати структуру та поведінку системи на різних рівнях абстракції: від концептуального рівня – те, що система робить, до логічного – те, як вона організована та реалізаційного – як

вона технічно впроваджена. Використання стандартної нотації UML забезпечує однозначне розуміння архітектури між розробниками, аналітиками та стейкхолдерами.

Усі діаграми UML поділяються на дві основні категорії: структурні та поведінкові. Структурні діаграми призначені для візуалізації статичної структури системи – її складових елементів та взаємозв'язків між ними. До цієї категорії належать:

– діаграма класів відображає класи системи, їхні атрибути, методи та взаємозв'язки; є основою документації та стартовою точкою для реалізації коду в об'єктно-орієнтованих мовах програмування;

– діаграма компонентів описує фізичну структуру системи та її компоненти, демонструючи, які модулі входять до системи і як вони взаємодіють між собою;

– діаграма об'єктів зосереджується на конкретних об'єктах у системі та їхніх взаємодіях, представляючи миттєвий стан системи у певний момент часу;

– діаграма пакетів відображає логічну структуру системи на рівні модулів і залежностей між ними, дозволяючи розбити систему на шари та підсистеми для зменшення зв'язності компонентів і спрощення супроводу;

– діаграма розгортання визначає фізичне обладнання, на якому працює програмна система, та відображає, як програмне забезпечення розподіляється між фізичними вузлами інфраструктури.

Поведінкові діаграми UML призначені для опису динамічної поведінки системи – її взаємодії з навколишнім середовищем та внутрішніх процесів. До цієї категорії належать:

– діаграма прецедентів (випадків використання) відображає взаємодію акторів із системою через набір функціональних вимог, показуючи, хто може використовувати систему і які дії виконувати;

– діаграма взаємодії (послідовності) показує взаємодію між об'єктами системи в певній послідовності подій, зосереджуючись на передачі повідомлень між компонентами в рамках конкретного сценарію використання;

– діаграма діяльності візуалізує потік виконання дій у системі, відображаючи послідовність операцій, умовні переходи, паралельні потоки та розподіл відповідальності між акторами.

У контексті розроблюваної LMS побудовано повний набір UML-діаграм, що охоплюють усі перелічені типи. Це дозволяє всебічно описати як статичну архітектуру системи – структуру сутностей, контролерів, модулів та фізичного розгортання, – так і її динамічну поведінку: сценарії автентифікації, проходження тестів студентами та інші ключові процеси платформи.

Діаграма прецедентів

Для визначення функціональних можливостей LMS та ролей, які взаємодіють із нею, побудовано діаграму прецедентів, на якій визначено трьох основних акторів системи: студента, викладача та адміністратора (рис. 3.1).

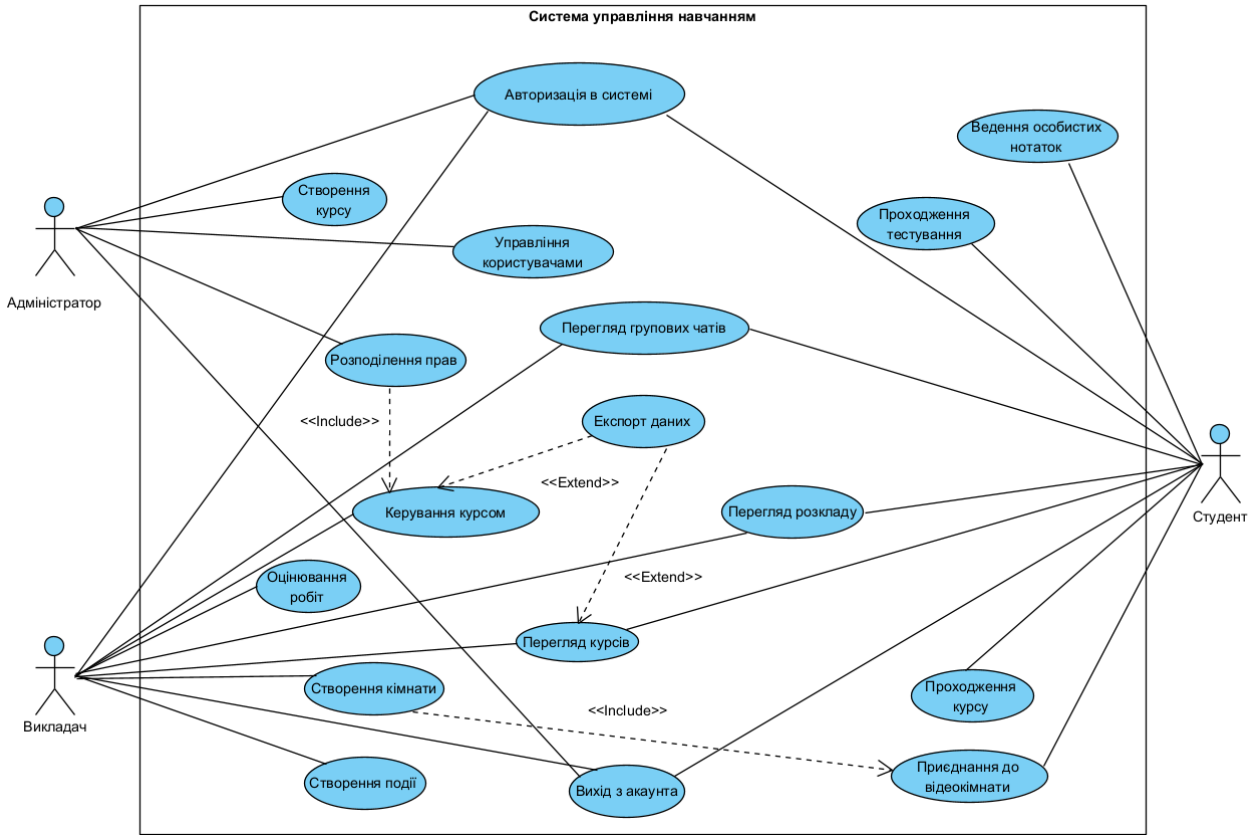

Рисунок 3.1 – Діаграма прецедентів

Студент взаємодіє з прецедентами перегляду курсів, проходження тестів та перегляду власних оцінок. Викладач має доступ до функцій створення курсів, управління навчальними матеріалами та виставлення оцінок. Адміністратор охоплює прецеденти керування обліковими записами користувачів та налаштування платформи.

Діаграма класів

Для моделювання структури сутностей LMS побудовано діаграму класів (рис. 3.2). Вона охоплює основні об'єкти системи – користувачів, курси, завдання, оцінювання та комунікації – і визначає атрибути, методи та зв'язки між ними. Така модель слугує основою для реалізації серверної частини застосунку та узгодження структури бази даних із бізнес-логікою.

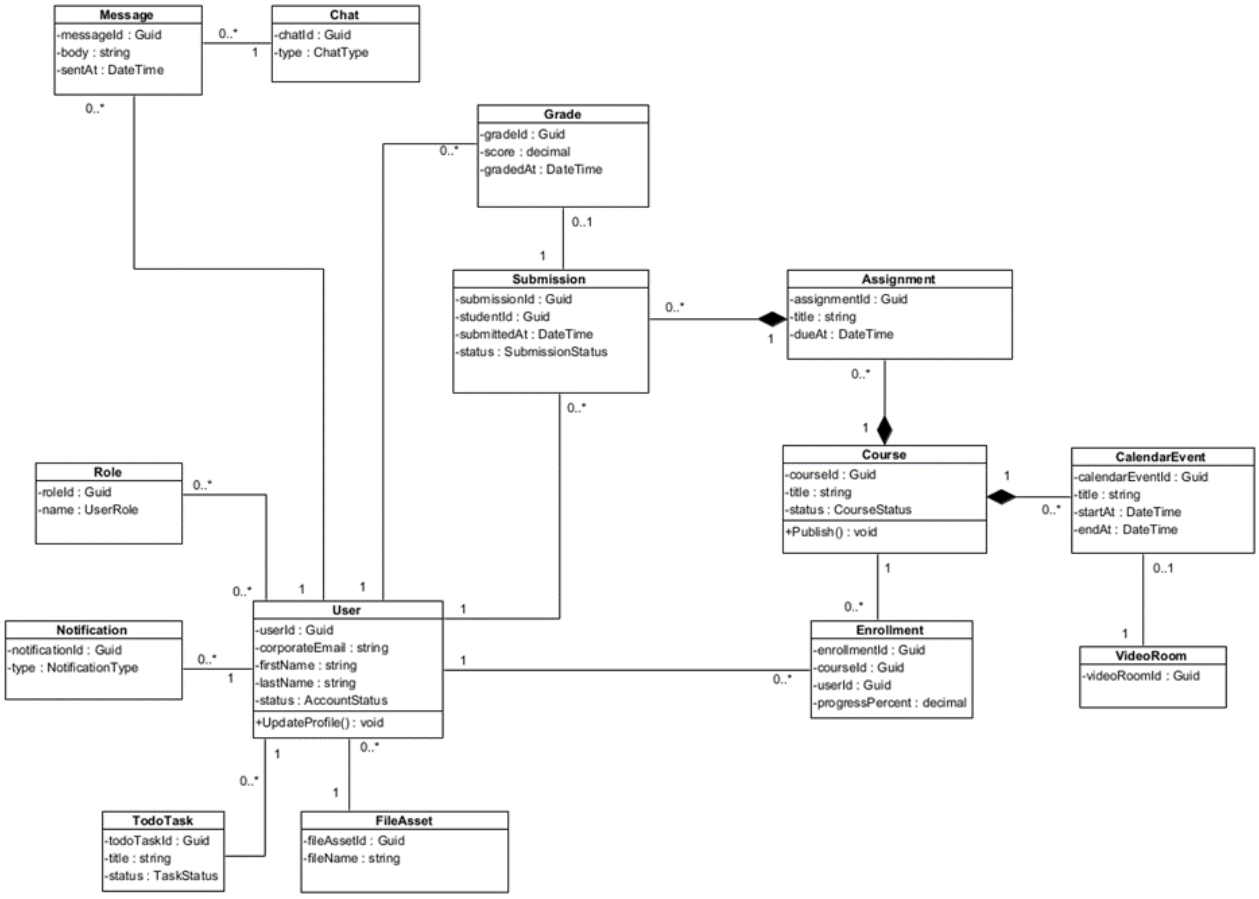

Рисунок 3.2 – Діаграма класів

Система LMS моделюється набором сутностей, що описують користувачів, навчальні курси, навчальні активності, комунікації та допоміжні об’єкти (сповіщення, файли, події календаря тощо).

Базовою сутністю є клас User – обліковий запис користувача. Він містить ідентифікатор userId, службову пошту corporateEmail, ім’я та прізвище (firstName, lastName), а також стан облікового запису status : AccountStatus. Для користувача передбачена операція UpdateProfile().

Для керування правами використовується клас Role, який описує роль у системі (roleId, name : UserRole). Між User і Role встановлено зв’язок, що дозволяє призначати ролі користувачам (на діаграмі показано асоціацію між цими класами).

Навчальний контент організований навколо класу Course – курс містить courseId, назву title та стан status : CourseStatus, а також має дію Publish(). Користувачі приєднуються до курсів через клас Enrollment (зарахування): він містить enrollmentId, посилання на курс (courseId) і користувача (userId) та показник прогресу progressPercent. Таким чином, Enrollment виступає проміжною сутністю, яка фіксує участь користувача в конкретному курсі та його прогрес.

Оцінювання й виконання завдань реалізовані через зв’язку класів Assignment, Submission та Grade. Клас Assignment описує завдання в межах курсу: assignmentId, назва title і дедлайн dueAt. Кожне завдання може мати багато здач (Submission). Клас Submission представляє факт здачі роботи студентом: submissionId, ідентифікатор студента studentId, дата здачі submittedAt та стан status : SubmissionStatus. Клас Grade зберігає оцінювання: gradeId, значення score, дату виставлення gradedAt. На діаграмі показано зв’язок між Submission та Grade, який відображає, що здача може бути оцінена (зазвичай це одна оцінка на одну здачу або відсутність оцінки до моменту перевірки).

Для комунікації в системі передбачено модуль чатів: клас Chat (поля chatId, type : ChatType) та клас Message (поля messageId, body, sentAt). Один чат містить багато повідомлень, а кожне повідомлення належить конкретному чату.

Діаграма класів (Контролери)

Окремо змодельовано структуру контролерів серверної частини LMS (рис. 3.3). Діаграма відображає ієрархію успадкування та залежності між контролерами і сервісними інтерфейсами, що дозволяє оцінити організацію серверної логіки на рівні API.

Рисунок 3.3 – Діаграма класів (Контролери)

На діаграмі показано базовий клас BaseApiController, від якого наслідуються спеціалізовані контролери (AuthController, UsersController, CoursesController, AssignmentsController, SubmissionsController, GradesController, EnrollmentsController). Також діаграма демонструє залежності контролерів від відповідних сервісних інтерфейсів і DTO-об'єктів, через які передаються дані запитів.

Діаграма послідовності

Для ілюстрації динаміки системи побудовано три діаграми послідовності: для варіантів використання «Авторизація в системі» (рис. 3.4), «Керування курсом» (рис. 3.5) та «Проходження тестування» (рис. 3.6). Кожна діаграма відображає часову послідовність обміну повідомленнями між актором, об’єктами клієнтської частини, REST API, контекстом даних Entity Framework і Microsoft SQL Server.

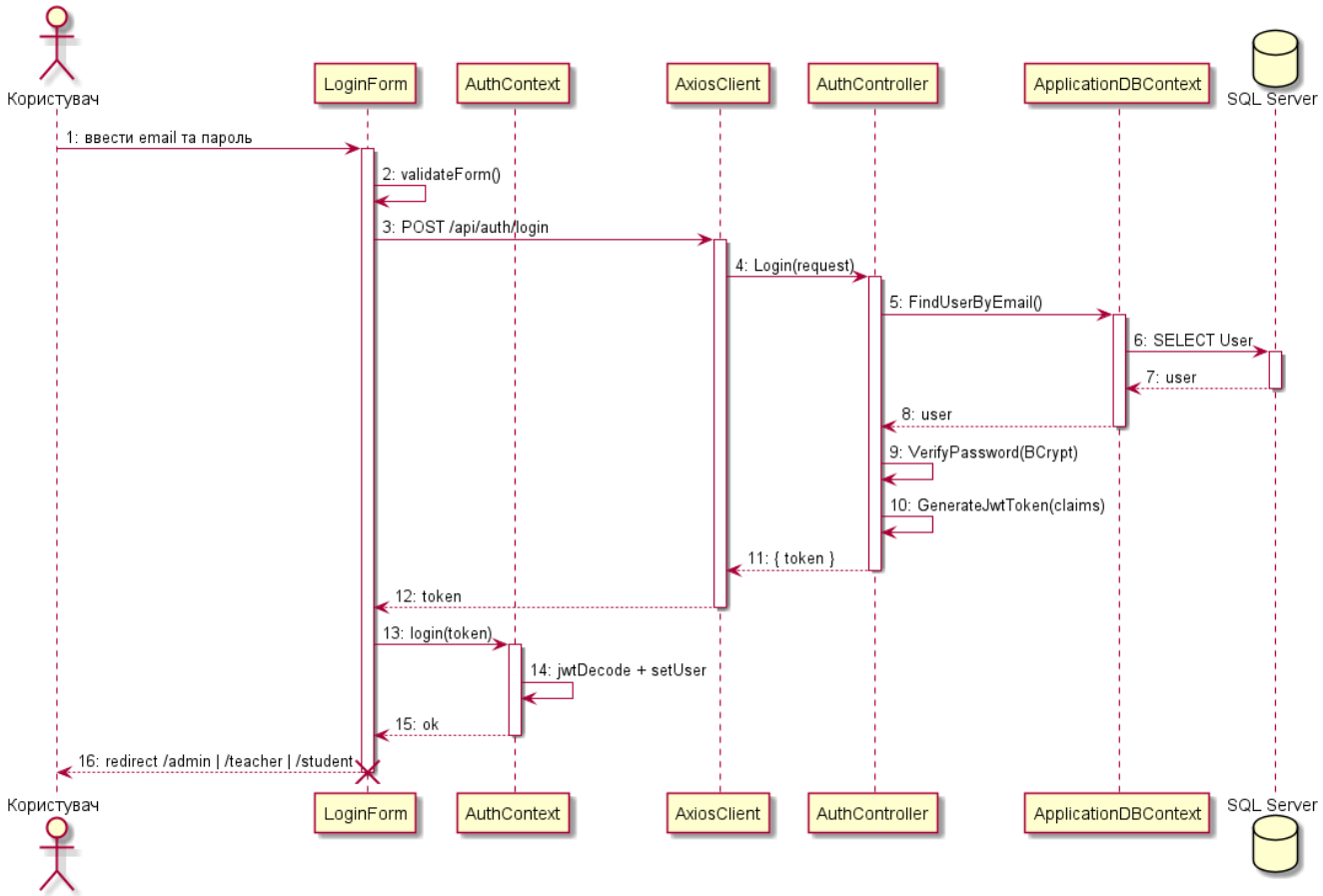

Рисунок 3.4 – Діаграма послідовності для варіанту використання «Авторизація в системі»

На рисунку 3.4 зображено сценарій входу користувача (адміністратора, викладача або студента) у LMS. Актор «Користувач» передає облікові дані об’єкту LoginForm, який рефлексивно виконує validateForm() і через AxiosClient надсилає синхронний запит POST /api/auth/login до AuthController. Контролер звертається до ApplicationDBContext, який виконує пошук запису в SQL Server, після чого повертає дані користувача.

Далі AuthController рефлексивно перевіряє пароль за допомогою BCrypt і формує JWT-токен із claims (ідентифікатор, email, роль, ім’я, по батькові, прізвище).

Токен повертається клієнту; об’єкт AuthContext зберігає його в localStorage, декодує та встановлює стан сесії. Завершальним повідомленням користувач перенаправляється в зону відповідної ролі (/admin, /teacher або /student). Діаграма демонструє типову трирівневу взаємодію «клієнт – API – база даних» і центральну роль JWT у захисті подальших запитів.

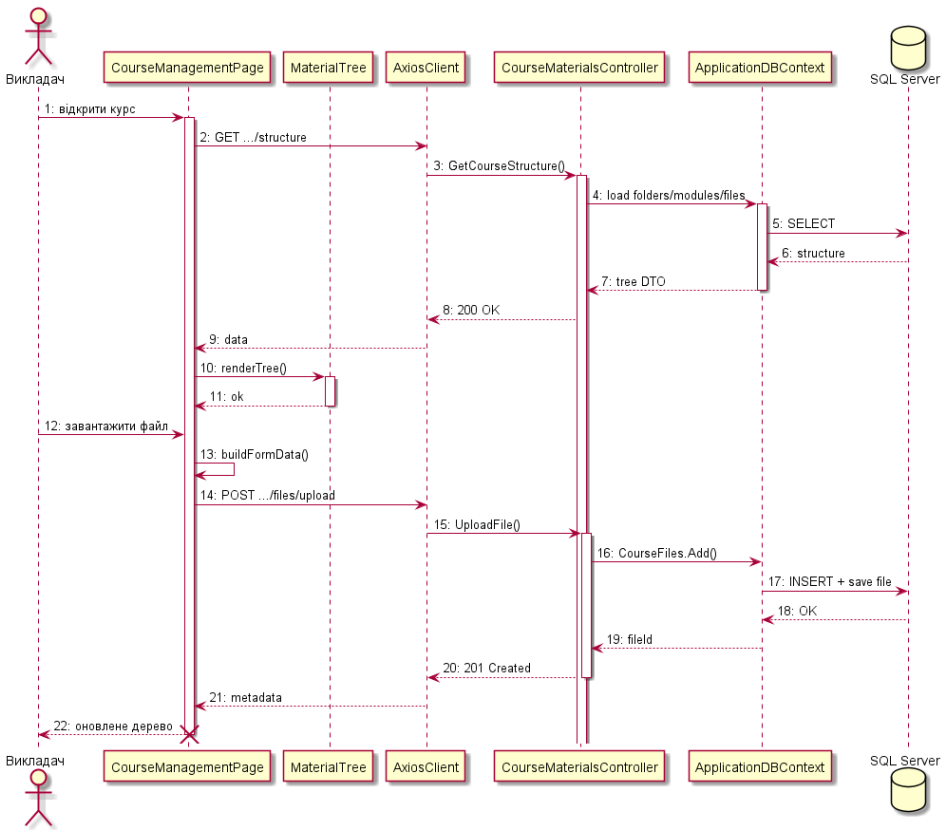

Рисунок 3.5 – Діаграма послідовності для варіанту використання «Керування курсом»

На рисунку 3.5 моделюється робота викладача зі змістом курсу. Актор «Викладач» відкриває CourseManagementPage, яка через AxiosClient запитує структуру

курсу (GET .../structure) у CourseMaterialsController. Контролер завантажує папки, модулі та файли через ApplicationDBContext з SQL Server і повертає дерево матеріалів; компонент MaterialTree відображає його на екрані.

У другій фазі сценарію викладач завантажує навчальний файл: сторінка рефлексивно формує FormData, надсилає multipart-запит POST .../files/upload, контролер зберігає файл на диску сервера та метадані в базі. Зворотна відповідь з ідентифікатором файлу оновлює інтерфейс. Діаграма ілюструє, як викладач керує контентом курсу через REST API без прямого доступу до бази даних з браузера.

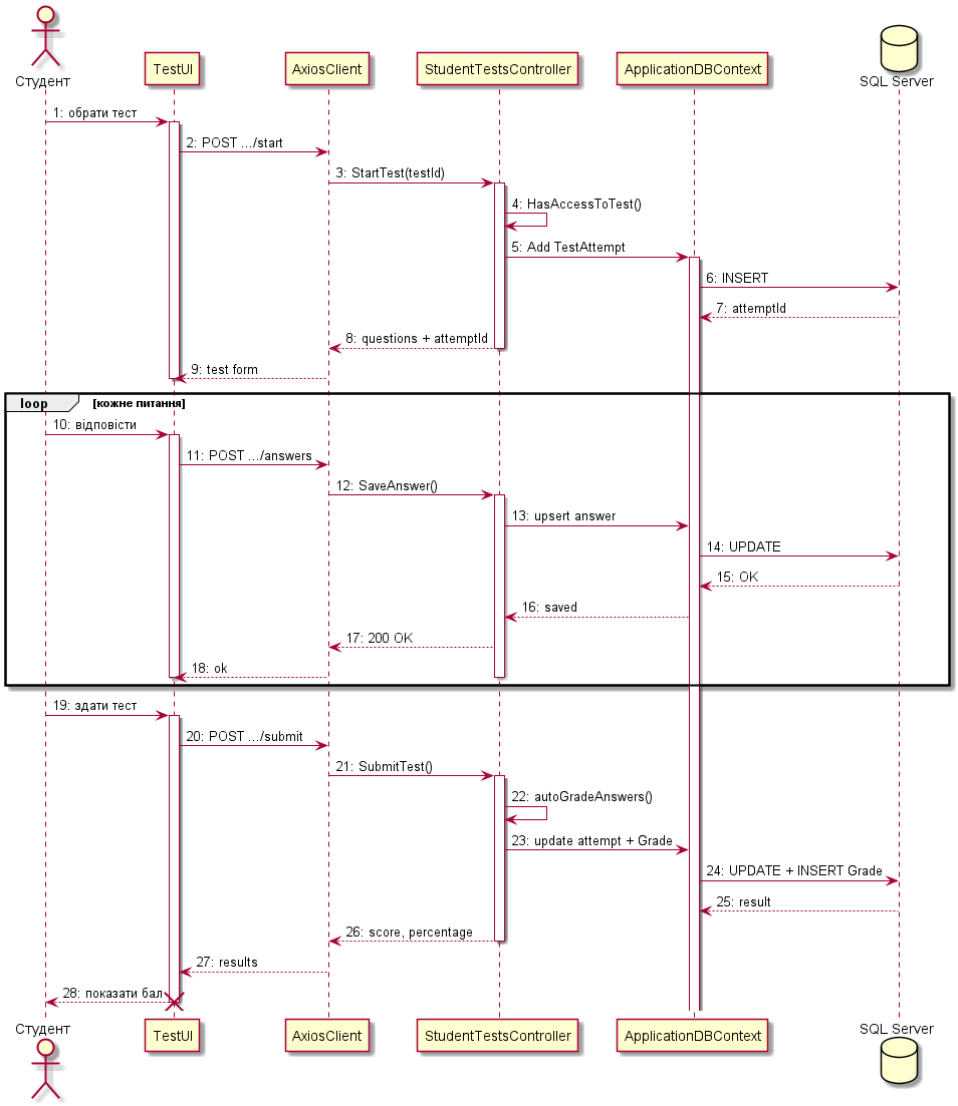

Рисунок 3.6 – Діаграма послідовності для варіанту використання «Проходження тестування»

На рисунку 3.6 відображено повний сценарій здачі тесту студентом у LMS. Актор «Студент» обирає тест у TestUI, після чого клієнт через AxiosClient надсилає запит POST /api/studenttests/{testId}/start до StudentTestsController.

Контролер рефлексивно виконує перевірку доступу HasAccessToTest(), створює запис TestAttempt через ApplicationDBContext і зберігає його в SQL Server; у відповідь повертається ідентифікатор спроби та перелік питань тесту. Далі в циклі для кожного питання студент вводить відповідь, TestUI викликає POST /api/studenttests/attempts/{attemptId}/answers, а контролер зберігає або оновлює TestAttemptAnswer у базі даних, повертаючи підтвердження збереження.

Після натискання «Здати тест» виконується POST /api/studenttests/attempts/{attemptId}/submit: StudentTestsController рефлексивно запускає autoGradeAnswers(), обчислює бал і за умови автоматичного оцінювання закритих питань створює запис Grade у журналі курсу.

Результат (score, percentage) повертається на клієнт і відображається студенту; після завершення сценарію об’єкт TestUI знищується. Діаграма показує, що в реалізованій системі контроль знань і оцінювання реалізовані через підсистему тестів (Test, TestAttempt, TestAttemptAnswer, Grade), а не через окремі сутності «завдання–здача».

Діаграма діяльності

Для опису алгоритму виконання окремих варіантів використання побудовано діаграму діяльності, яка відображає послідовність дій адміністратора при створенні навчального курсу, умови прийняття рішень та паралельну обробку пов’язаних операцій у LMS.

На діаграмі використано оператор розгалуження для перевірки коректності введених даних і оператор паралельного розділення (fork/join) для одночасного формування запису курсу та зарахування навчальних груп. Діаграма діяльності для варіанту використання «Створення курсу» наведена на рисунку 3.7.

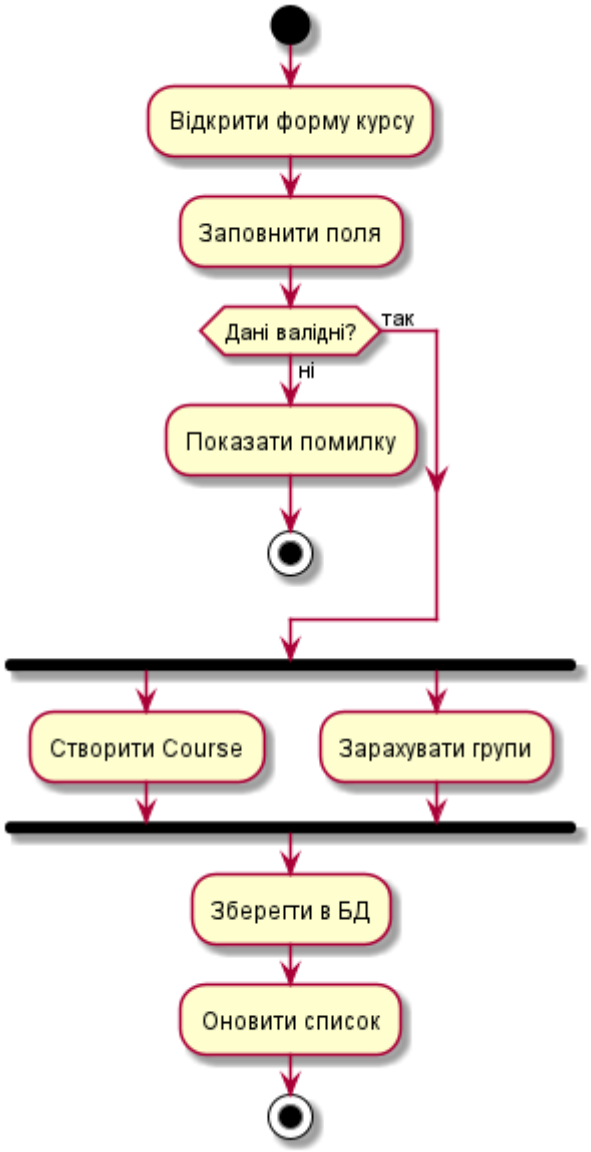

Рисунок 3.7 – Діаграма діяльності «Створення курсу»

На рисунку 3.7 зображено алгоритм створення нового курсу адміністратором у LMS. Процес починається з відкриття форми курсу та заповнення обов’язкових полів (назва, опис, дати, кафедра, викладач тощо). Далі виконується умовна перевірка валідності даних: якщо поля заповнені некоректно, користувачу показується повідомлення про помилку і процес завершується.

Якщо дані валідні, активується паралельний потік обробки. У першій гілці формується та створюється сутність Course, у другій – підготовлюється зарахування навчальних груп (або окремих студентів) на курс. Після об’єднання паралельних гілок виконується збереження даних у базі через ApplicationDBContext (запис курсу та записів CourseEnrollment). Завершальними кроками є оновлення списку курсів у

інтерфейсі адміністратора та відображення підтвердження успішного створення. Діаграма ілюструє, що створення курсу в LMS – це не лише додавання одного запису в таблицю, а комплексна операція з валідацією, паралельною підготовкою пов'язаних даних і синхронізацією з базою даних.

Діаграма розгортання

Проєктування програмної системи LMS не обмежується лише логічною структурою модулів і взаємодією об'єктів: для повного опису архітектури необхідно також зафіксувати, на якому фізичному або віртуальному обладнанні розміщуються компоненти застосунку та якими протоколами вони обмінюються даними. Для цього розроблено діаграму розгортання (рис. 3.8).

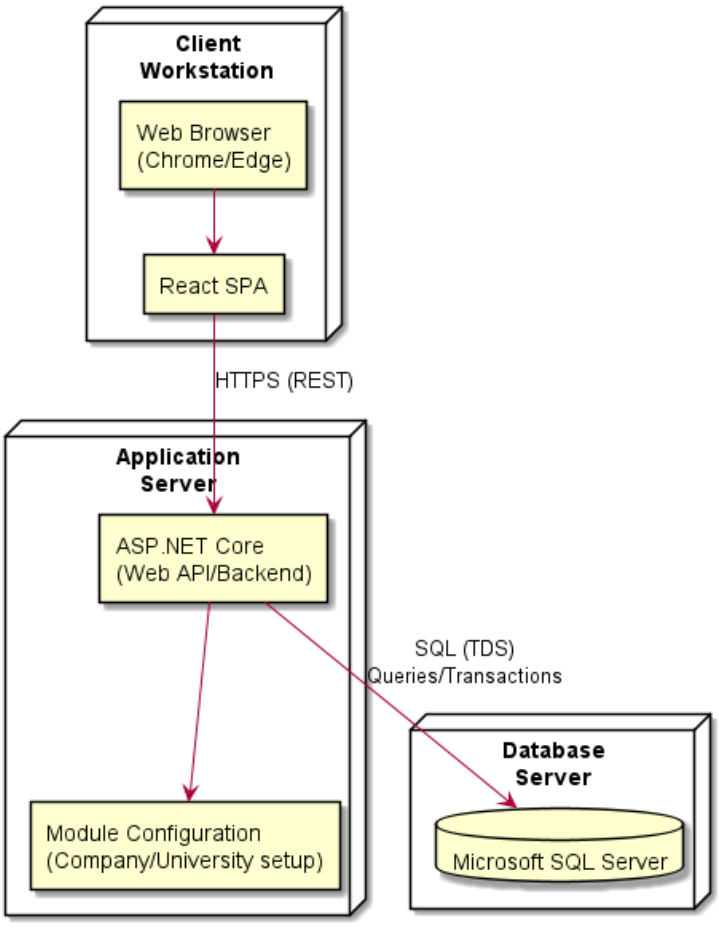

Рисунок 3.8 – Діаграма розгортання

Діаграма на рисунку 3.8 демонструє фізичну архітектуру розгортання LMS у трирівневій клієнт-серверній моделі. На вузлі Client Workstation розміщено компонент Web Browser у середовищі якого виконується React SPA – односторінковий інтерфейс

застосунку. Браузер забезпечує відображення UI та виконання клієнтського JavaScript-коду.

На вузлі Application Server розгорнуто ASP.NET Core, який приймає HTTP-запити від клієнта, реалізує бізнес-логіку (автентифікація, курси, тести, оцінки, повідомлення) та звертається до компонента Module Configuration (Company/University setup) для роботи з конфігурацією модулів залежно від типу організації (університет або компанія).

На вузлі Database Server працює Microsoft SQL Server, де зберігаються користувачі, курси, навчальні матеріали, спроби тестів, оцінки та інші доменні сутності.

Зв’язок між React SPA і ASP.NET Core здійснюється по протоколу HTTPS – клієнт надсилає JSON-запити з JWT-токеном у заголовку Authorization. Зв’язок між сервером застосунку і СКБД – по протоколу SQL, через який виконуються запити та транзакції Entity Framework Core, що забезпечує цілісність і узгодженість даних у системі.

Діаграма пакетів

У об’єктно-орієнтованій системі класи утворюють її структурний кістяк, однак у реальних застосунках, як LMS, їхня кількість швидко зростає: контролери API, моделі даних, компоненти інтерфейсу, сервіси доступу до БД, допоміжні утиліти.

Для наочної організації такої множини елементів у UML використовують пакети – засіб групування, який об’єднує класи, компоненти та інші конструкції в логічні одиниці високого рівня. Кожен клас належить лише одному пакету, а самі пакети можуть бути вкладеними, утворюючи ієрархію від загальних шарів архітектури до конкретних підсистем. Пунктирні залежності між пакетами показують, хто від кого залежить при виконанні функцій, не перевантажуючи діаграму деталями реалізації кожного класу.

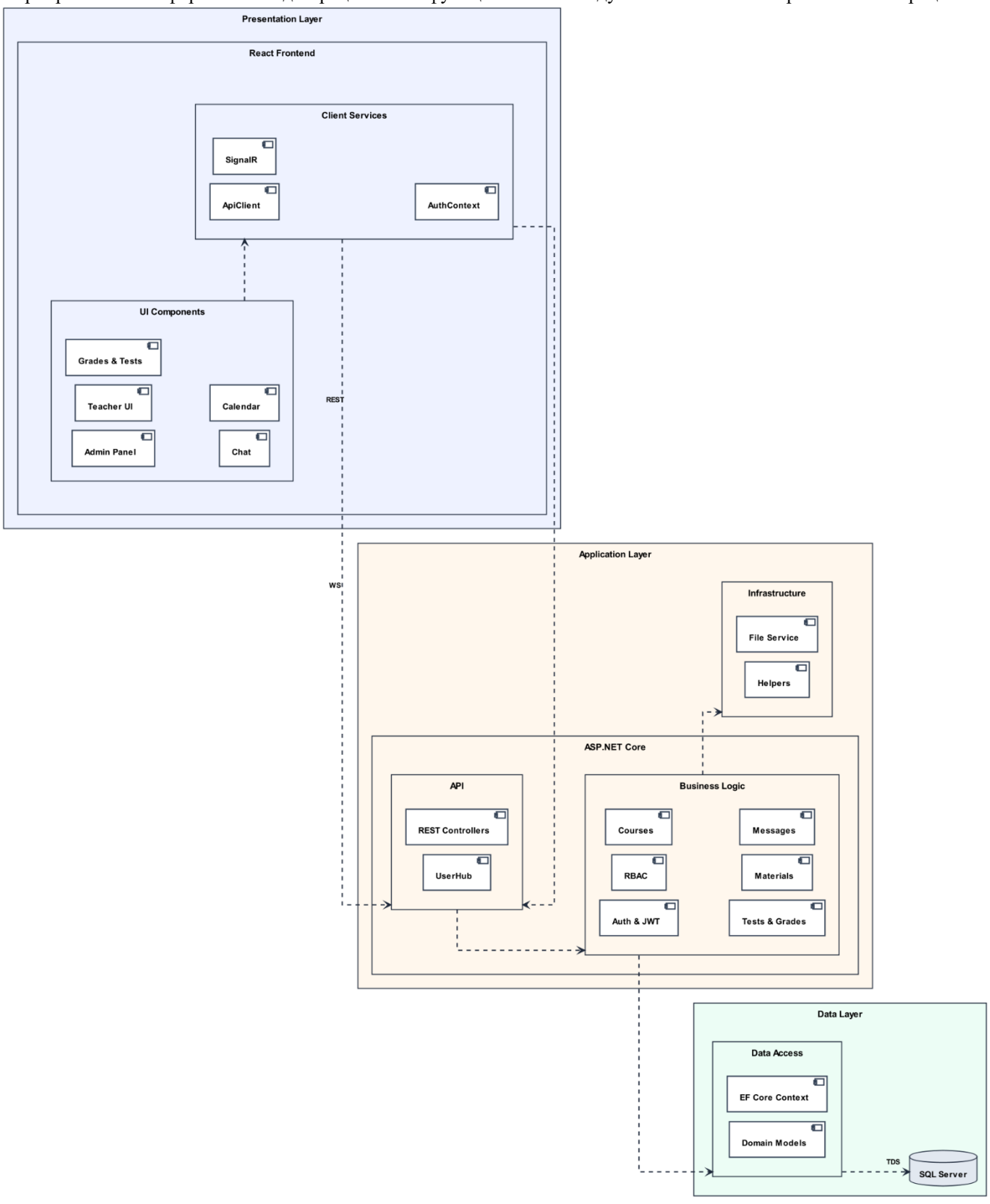

Рисунок 3.9 – Діаграма пакетів

Діаграма пакетів (рис. 3.9) зображено логічну тришарову структуру LMS, організовану за принципом розділення відповідальностей, де шар Presentation Layer містить пакет React Frontend, у складі якого виділено UI Components (панель

адміністратора, інтерфейс викладача, журнал оцінок і тести, чат, календар) та Client Services (ApiClient на базі axios, клієнт SignalR для обміну повідомленнями в реальному часі, AuthContext для зберігання стану автентифікації). Компоненти інтерфейсу залежать від клієнтських сервісів, які, у свою чергу, звертаються до серверної частини.

Шар Application Layer об’єднує пакет ASP.NET Core з підпакетами API (REST Controllers і SignalR-хаб UserHub) та Business Logic (автентифікація та JWT, рольовий доступ RBAC, курси та зарахування, тести й оцінки, навчальні матеріали, повідомлення). Окремо виділено пакет Infrastructure із File Service для роботи з файловим сховищем та допоміжними класами (PasswordHelper, EncryptionHelper). Залежності спрямовані від API до бізнес-логіки, далі – до інфраструктури та шару даних.

Шар Data Layer включає Data Access з ApplicationDBContext і доменними моделями, а також SQL Server як СКБД. Зв’язок між шарами позначено протоколами REST і WebSocket на стороні клієнт–сервер та TDS між сервером застосунку і базою даних.

На етапі аналізу вимог до LMS побудовано діаграму прецедентів, яка визначає основних акторів системи – адміністратора, викладача та студента – і перелік функцій, доступних кожній ролі (авторизація, керування курсами, тестування, оцінювання, чат, календар, відеокімнати тощо). Діаграма прецедентів дає узагальнене уявлення про функціональність платформи, однак для детального опису сценаріїв використання доцільно розгорнути окремі варіанти використання у вигляді текстових специфікацій.

У даному підрозділі розроблено п’ять основних сценаріїв, які охоплюють ключові функціональності системи та відповідають найбільш типовим навчальним процесам: керування змістом курсу викладачем, виконання завдання студентом і отримання оцінки, обмін повідомленнями між студентом і викладачем, адміністрування облікових записів студентів та участь у відеоконференції курсу. Деталізовані специфікації наведено в таблицях 3.1-3.5.

Таблиця 3.1 – Сценарій редагування курсу викладачем

Сценарій 1: Редагування та публікація навчального курсу	
Передумови: Викладач авторизований у системі (наявний валідний JWT-токен); викладач призначений відповідальним за курс або має до нього доступ	
Учасники:	Викладач, Система
Тригер	Викладач обирає курс у модулі «Курси» і переходить до сторінки керування курсом
Опис процесу	1. викладач переходить до модуля «Курси» і натискає «Редагувати курс»; 2. система формує запит із даними форми та передає його до Application Layer через HTTP-інтерфейс; 3. application layer здійснює валідацію даних відповідно до бізнес-правил (перевірка коректності та наявності обов'язкових полів); 4. дані передаються до Data Access Layer, де створюється або оновлюється відповідний запис у базі даних; 5. система генерує підтвердження успішного виконання операції та відображає оновлений курс у кабінеті викладача.
Альтернативний потік	Якщо викладач не має прав на курс або дані форми некоректні, API повертає помилку (403/400), і система показує повідомлення без збереження змін
Постумова	Структура курсу оновлена в БД; студенти з відповідним доступом можуть переглядати опубліковані матеріали

Таблиця 3.2 – Сценарій виконання завдання студентом та отримання оцінки

Сценарій 2: Виконання завдання та отримання оцінки	
Передумови: Студент авторизований у системі (валідний JWT); студент зарахований на курс; тест доступний за розкладом та налаштуваннями TestAccess	
Учасники:	Студент, Викладач, Система
Тригер	Студент обирає доступний тест у навчальному модулі курсу

Кінець таблиці 3.2

Опис процесу	1. студент відкриває завдання з курсу; 2. Application Layer звертається до бізнес-логіки, яка перевіряє права доступу студента; 3. Data Access Layer отримує з бази даних опис завдання/файли та передає його студенту; 4. студент завантажує відповідь (файл/текст); 5. Application Layer передає дані до бізнес-логіки, яка створює новий запис у Data Access Layer для збереження відповіді у базі даних; 6. викладач отримує сповіщення і перевіряє роботу; 7. викладач виставляє оцінку, система зберігає її у БД; студенту приходить сповіщення про отриману оцінку.
Альтернативний потік	Якщо немає доступу, минув дедлайн або вичерпано спроби API повертає 403/400, тест не розпочинається. Якщо вичерпано час (TimeLimitMinutes) – спроба завершується зі статусом TimeExpired
Постумова	Спроба тесту збережена в БД; за умов сценарію в журналі курсу з’являється запис Grade

Таблиця 3.3 – Сценарій відправки повідомлення викладачеві

Сценарій 3: Відправка повідомлення викладачеві	
Передумови: Студент і викладач авторизовані в системі; між ними дозволено особисте спілкування в межах LMS	
Учасники:	Студент, Викладач, Система
Тригер	Студент відкриває розділ «Особисті чати» і обирає викладача зі списку

Кінець таблиці 3.3

Опис процесу	1. Presentation Layer відображає список доступних контактів викладачів; 2. студент обирає викладача та відкриває чат; 3. студент вводить текст повідомлення та натискає «Відправити»; 4. Application Layer приймає запит та передає його до бізнес-логіки для перевірки прав доступу та валідності повідомлення; 5. бізнес-логіка зберігає повідомлення у Data Access Layer в сховищі повідомлень; 6. система формує сповіщення та доставляє повідомлення викладачу у Presentation Layer; 7. викладач отримує повідомлення та може відповісти, повторюючи ті ж кроки.
Альтернативний потік	При розриві WebSocket-з'єднання клієнт намагається перепідключитися; історія повідомлень залишається доступною через REST
Постумова	Повідомлення збережене в БД і відображене в чаті обох користувачів

Таблиця 3.4 – Сценарій видалення студента з системи адміністратором

Сценарій 4: Видалення студента	
Передумови: Адміністратор авторизований у системі; студент існує в БД	
Учасники:	Адміністратор, Система
Тригер	Адміністратор відкриває модуль «Студенти» в адмін-панелі

Кінець таблиці 3.4

Опис процесу	1. Presentation Layer відображає список зареєстрованих студентів академічної групи в адмін-панелі; 2. адміністратор обирає студента для видалення; 3. натискає кнопку «Видалити студента»; 4. Application Layer формує запит і передає його до бізнес-логіки для перевірки прав адміністратора та існування студента у системі; 5. Бізнес-логіка ініціює видалення запису студента та пов’язаних даних через Data Access Layer у базі даних; 6. Presentation Layer оновлює список студентів групи, відображаючи актуальний стан.
Альтернативний потік	Якщо студента не знайдено – 404; якщо видалення блокується зв’язаними даними – 500/повідомлення про помилку
Постумова	Обліковий запис студента видалено (або операція відхилена з поясненням); список в адмін-панелі актуалізовано

Таблиця 3.5 – Сценарій участі студента у відеоконференції курсу

Сценарій 5: Участь студента у відеоконференції	
Передумови: Студент авторизований у системі; зарахований на курс; у календарі курсу запланована відеоконференція (або викладач відкрив відеокімнату)	
Учасники:	Адміністратор, Система
Тригер	Наближається час заняття або студент відкриває розділ «Відеокімнати» / «Календар»

Кінець таблиці 3.5

Опис процесу	1. система надсилає push/email-сповіщення користувачу про наближення пари (за 5-10 хв до часу); 2. Application Layer обробляє сповіщення та звертається до бізнес-логіки, яка перевіряє статус події в календарі; 3. користувач переходить до курсу в навчальному модулі; 4. Application Layer запитує у бізнес-логіки права доступу до відеокімнати; Data Access Layer отримує дані події з БД; 5. у вказаний час кімната автоматично відкривається (доступ надається за роллю та токеном JWT);
Опис процесу	6. бізнес-логіка генерує унікальний токен доступу та передає його Application Layer для клієнтського застосунку; 7. користувач доєднується до відеоконференції; 8. Data Access Layer записує участь у лог активності та прогрес курсу в БД; 9. після завершення викладач/керівник закриває кімнату; користувач отримує сповіщення з записом/матеріалами (за наявності).
Альтернативний потік	Якщо студент не зарахований на курс або кімната ще не відкрита – доступ блокується з відповідним повідомленням
Постумова	Студент брав участь у відеозанятті; за наявності реалізації – фіксується факт участі та/або додаються матеріали до курсу

3.3 Структура бази даних

Проєктування бази даних є невід'ємною частиною розробки LMS, оскільки саме рівень збереження даних визначає, як у системі фіксуються користувачі, навчальні курси, матеріали, результати тестування, оцінки та повідомлення.

Для LMS важливо не лише зберігати інформацію, а й забезпечити її цілісність, нормалізацію та логічну узгодженість з предметною областю: ієрархія закладу, рольовий поділ користувачів, зв'язки, тощо.

Логічну структуру бази даних доцільно представити у вигляді ER-діаграми, яка показує основні сутності предметної області, їхні атрибути та типи зв'язків (один-до-багатьох, багато-до-багатьох через проміжні таблиці). ERD доповнює діаграму класів UML: якщо діаграма класів орієнтована на об'єктно-орієнтовану реалізацію в коді, то ERD фокусується на табличній організації даних і зовнішніх ключах, що безпосередньо впливають на коректність SQL-запитів, транзакцій і обмежень цілісності (у т.ч. унікальність email, індекси на назви факультетів/кафедр/груп, обмеження каскадного видалення в OnModelCreating).

Центральною таблицею облікових записів є Users з дискримінатором TPH: записи студентів і викладачів зберігаються в тій самій таблиці з різними значеннями поля Discriminator, що спрощує спільні операції (email, ПІБ, пароль) і водночас дозволяє мати специфічні поля на рівні похідних типів. Для Email встановлено унікальний індекс; аналогічно унікальність забезпечено для назв факультетів, пар «назва кафедри + факультет» та «назва групи + кафедра», що запобігає дублюванню в організаційній ієрархії.

Організаційний блок включає таблиці Faculties, Departments, Groups. Студент пов'язаний із групою зовнішнім ключем GroupId (видалення групи обмежено політикою Restrict). Викладач пов'язаний із кафедрою; курс посилається на кафедру та викладача-відповідального. Таблиця CourseEnrollments реалізує зв'язок «багато-до-багатьох» між студентами та курсами з атрибутом дати зарахування.

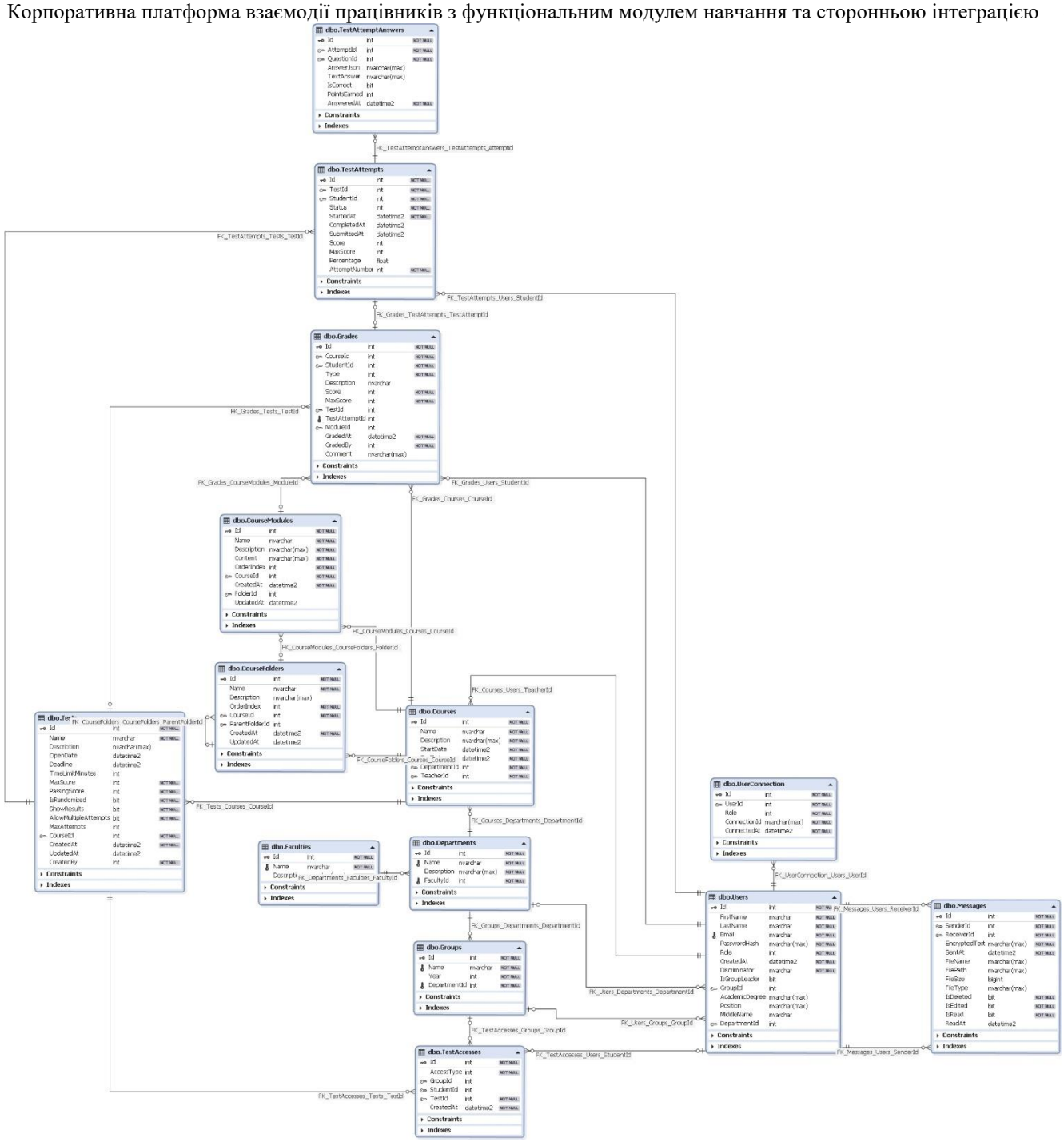

Рисунок 3.10 – ER-діаграма БД

Підсистема оцінювання та контролю знань включає Tests (параметри часу, прохідний бал, кількість спроб), Questions, TestAccess, TestAttempts, TestAttemptAnswers. Питання та деякі залежні записи видаляються каскадно разом із тестом. Таблиця Grades є універсальним журналом оцінок: вона пов’язана з курсом і студентом, може посилатися на тест, спробу тесту або модуль, містить тип оцінки, бали, коментар і ідентифікатор викладача, який виставив оцінку. Зв’язок «одна спроба – одна оцінка» між TestAttempts і Grades моделюється як один-до-одного.

Комунікаційний модуль представлений таблицею Messages із зовнішніми ключами на відправника та одержувача (SenderId, ReceiverId); для них застосовано DeleteBehavior.NoAction, оскільки каскадне видалення користувачів могло б порушити цілісність при складній мережі зв’язків. Додатково в системі можуть зберігатися службові дані з’єднань для SignalR.

З точки зору цілісності даних у проєкті свідомо обмежено агресивне каскадне видалення для зв’язків, що утворюють цикли, і залишено каскад лише там, де він безпечний: питання тесту, відповіді спроби, деякі записи доступу. Фізичні файли зберігаються поза BLOB-полями основних таблиць – у файловій системі сервера, а в БД зберігаються лише шляхи та метадані, що зменшує навантаження на СКБД і спрощує резервне копіювання.

3.4 Прототипування інтерфейсу користувача

Інтерфейс користувача є ключовим рівнем LMS, через який адміністратор, викладач і студент взаємодіють із функціями системи. При проєктуванні дотримано принципів узнаваності, послідовності та мінімізації когнітивного навантаження: однакові патерни карток, кнопок, таблиць і форм повторюються на різних сторінках; акцентні елементи (основні дії, статуси, попередження) виділені кольором і типографікою; вторинні операції винесені в контекстні меню або бічні панелі.

Макети ключових екранів побудовано у середовищі Figma як інтерактивні прототипи. Вони узгоджені з архітектурою клієнтської частини (SPA на React, бібліотека Mantine, маршрутизація за ролями) і з API-контрактами серверної частини. Прототипи охоплюють точку входу в систему, кабінет викладача, сценарій проходження тесту студентом, модуль чатів, адміністративну панель та інтерфейс відеоконференцій.

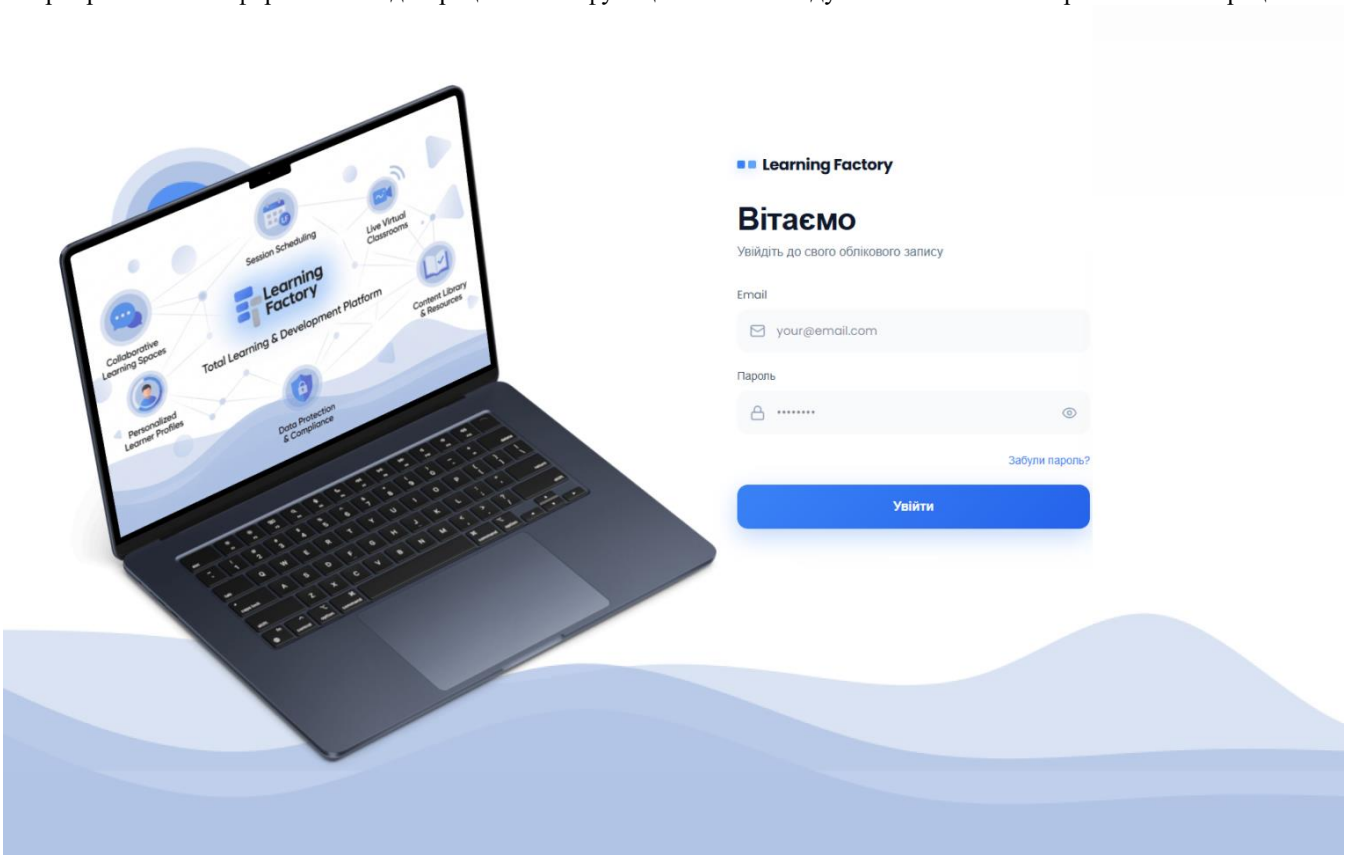

Рисунок 3.11 – Форма реєстрації користувача

На рисунку 3.11 представлено сторінку авторизації користувача в системі. Сторінка розділена на дві функціональні зони: візуальну презентаційну панель зліва та інтерактивну форму входу справа. Форма авторизації містить текстові поля для введення ідентифікатора користувача (Email) та секретного ключа (Пароль). Передбачено функцію відновлення доступу через посилання «Забули пароль?».

Основна кнопка «Увійти» виділена акцентним синім кольором для полегшення візуального орієнтування користувача. Ліворуч розміщено стилізовану ілюстрацію з ноутбуком, на екрані якого відображено екосистему платформи «Learning Factory», включаючи модулі планування сесій, віртуальні класи, чат в режимі реального часу, тощо.

Рисунок 3.12 демонструє основний робочий простір користувача з роллю «Викладач». Бокова панель зліва містить розділи: Панель управління, Курси, Особисті чати, Групи, Відеоруми, Календар та Нотатки. Центральна частина відображає поточну активність (активну лекцію), список курсів із прогресом студентів та перелік завдань на перевірку. Окремий віджет праворуч показує детальний розклад занять на сьогодні.

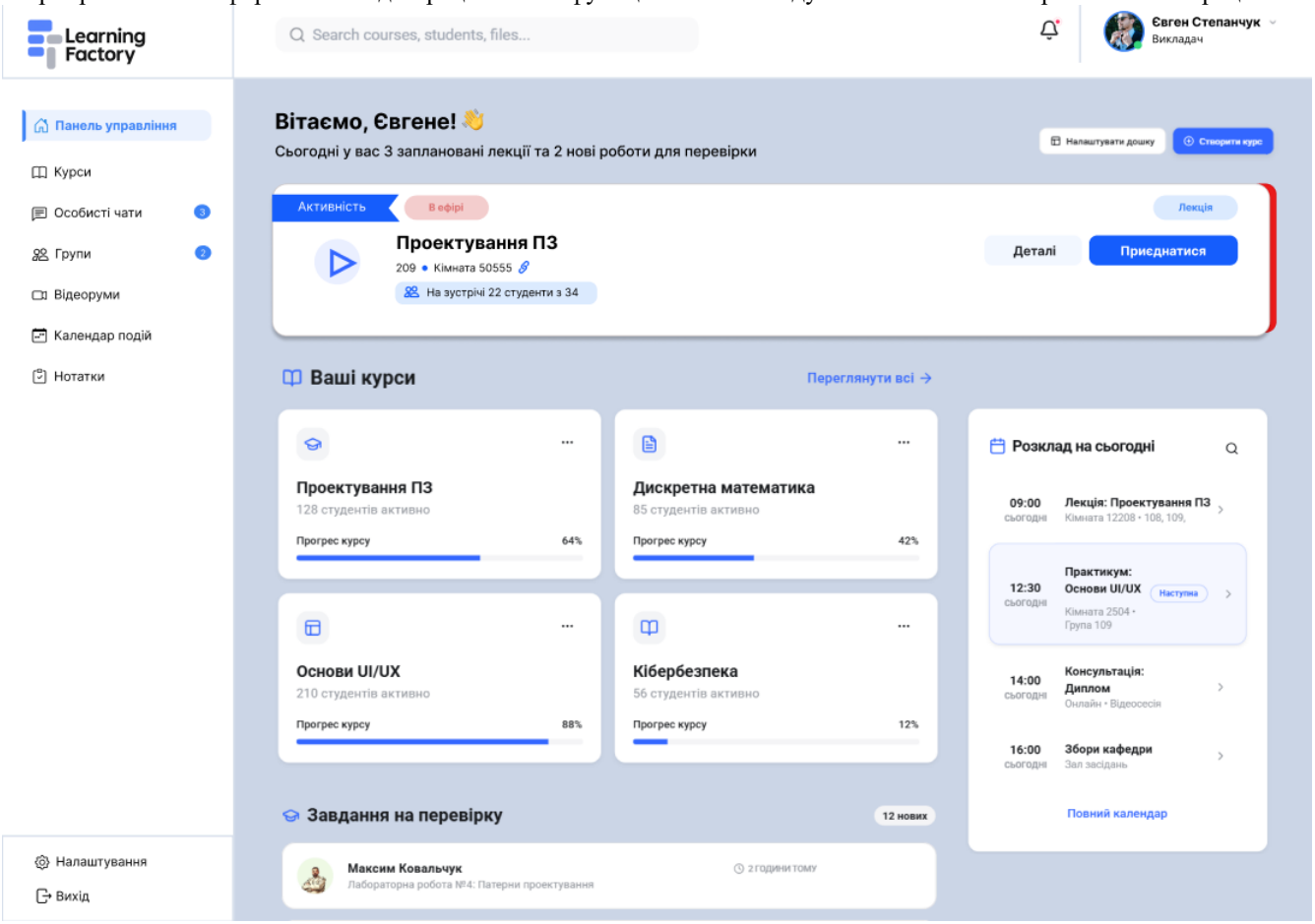

Рисунок 3.12 – Прототип інтерфейсу викладача

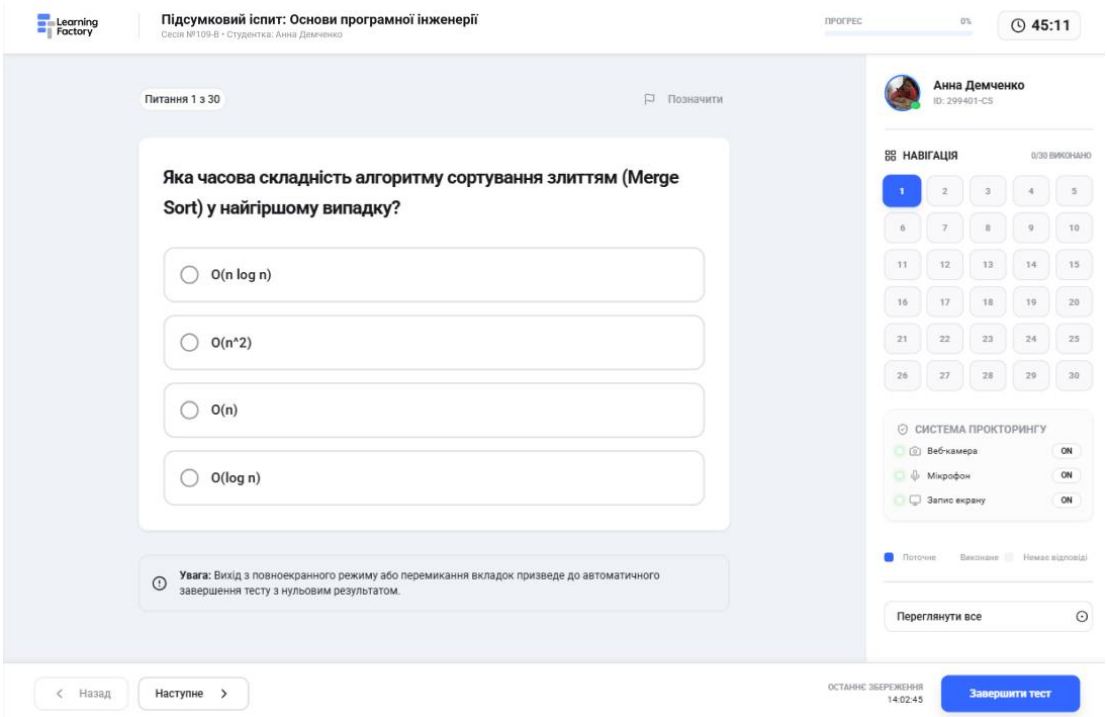

Рисунок 3.13 – Прототип інтерфейсу проходження тестування

На рисунку 3.13 відображено інтерфейс проходження тесту студентом. У центральній зоні розташовано поточне питання, варіанти відповідей (один або кілька правильних залежно від типу питання) та елементи керування спробою: збереження відповіді, перехід до наступного питання, завершення тесту.

Праворуч наведено панель навігації по питаннях із візуальним позначенням уже відповідених, пропущених і поточного пунктів. У верхній частині екрана показано назву тесту, обмеження за часом (за наявності) та зворотний відлік.

Після завершення тесту студенту відображається підсумковий бал і статус проходження (за умови налаштування ShowResults). Для відкритих типів питань передбачено окремий стан очікування перевірки викладачем. Макет передбачає зрозумілі попередження при спробі вийти з тесту до завершення або при вичерпанні часу.

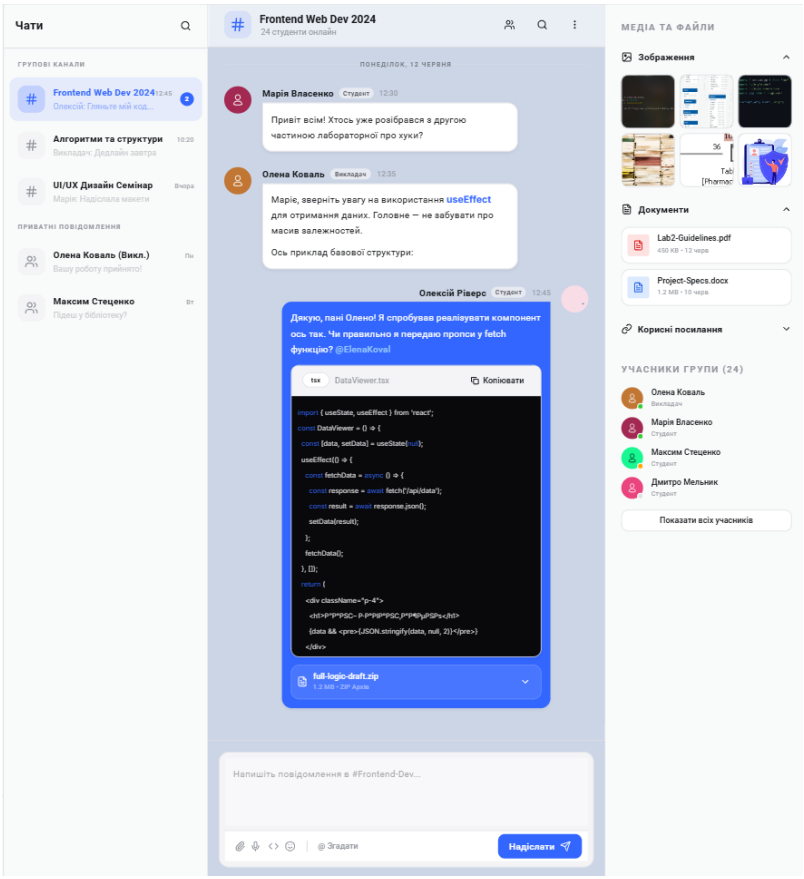

Рисунок 3.14 – Прототип інтерфейсу модуля чатів

Рисунок 3.14 описує інтерфейс для текстового спілкування та обміну файлами. Зліва представлено список активних чатів та приватних повідомлень. Центральна

частина містить історію повідомлень з можливістю вставки програмного коду та перегляду вкладень. Праворуч винесено блок із медіафайлами, документами та списком учасників групи.

На рисунку 3.15 показано Адмін-панель керування системою. Верхня частина містить картки з ключовими показниками (метриками): кількість активних користувачів, завантаження сховища, пропускна здатність мережі та системні інциденти. Візуалізовано графік трендів використання сховища за останні 6 місяців. Винесено окремий блок, який відображає системні логи в режимі реального часу (резервне копіювання, помилки входу, оновлення системи).

Інтерфейс адміністратора спроєктовано як інформаційну панель моніторингу й керування системою. Таке представлення дає адміністратору швидкий огляд стану системи без переходу в окремі реєстри. Блок швидких дій скорочує шлях до типових операцій: додавання викладача, студента, курсу чи перегляду або експорту звітів.

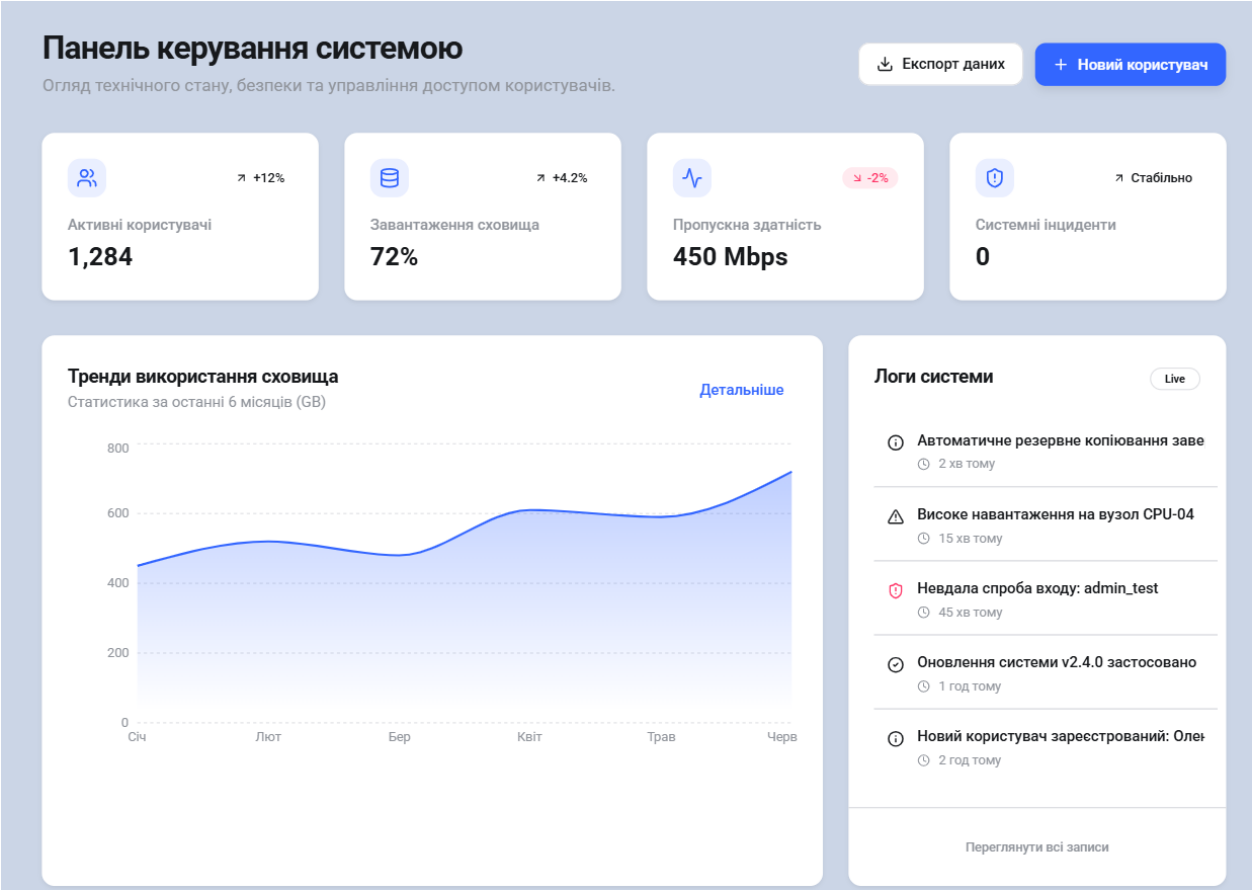

Рисунок 3.15 – Прототип фрагменту адмін-панелі

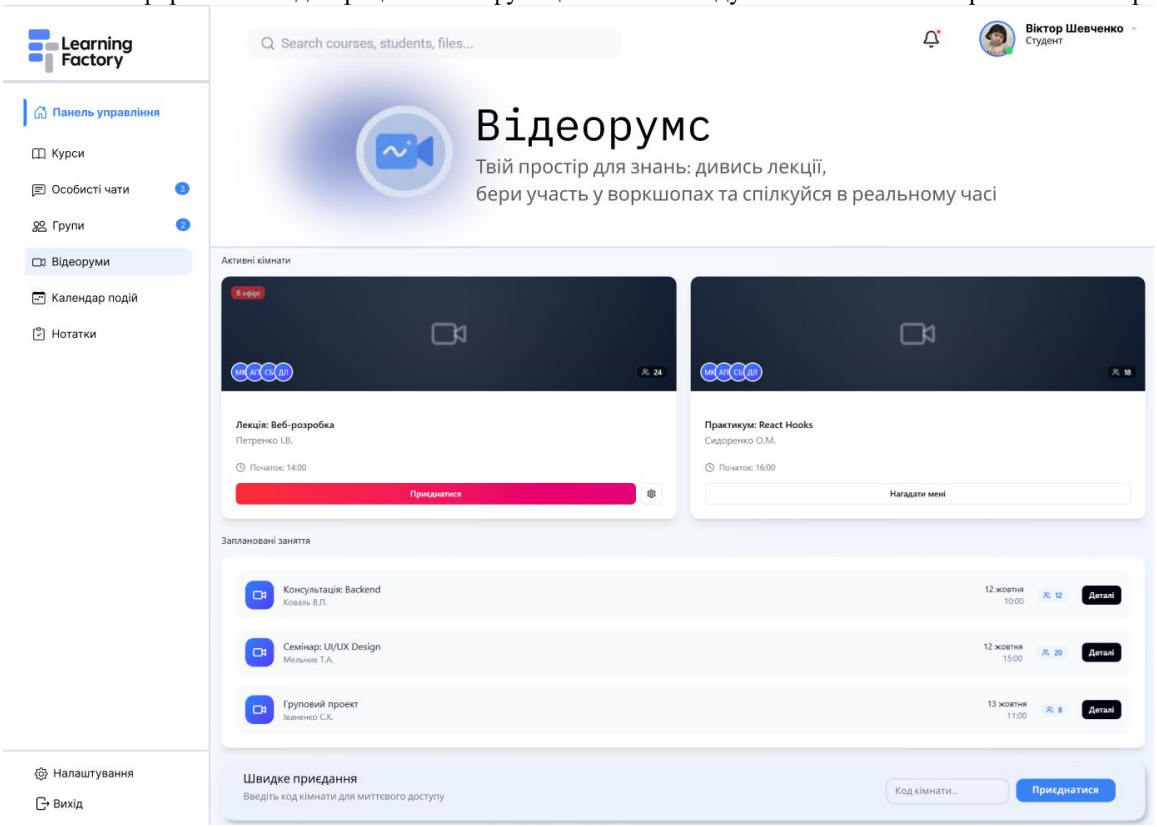

Рисунок 3.16 – Прототип інтерфейсу модулю з кімнатами для проведення конференціій

Рисунок 3.16 представляє інтерфейс онлайн-занять і консультацій. На екрані відображаються картки активних або запланованих відеокімнат із назвою, часом проведення, кількістю учасників і кнопкою швидкого приєднання. Порожній стан інтерфейсу містить підказку створити першу кімнату – це знижує поріг входу для нового викладача.

Нижче наведено розклад майбутніх трансляцій (лекції, практичні, консультації) із зазначенням дати та формату проведення. Передбачено поле для введення коду кімнати, що дозволяє студентам підключитися без пошуку в довгому списку. Модуль доповнює очний і дистанційний формат навчання та логічно пов’язаний із розділами «Календар» і «Курси» кабінету викладача.

Інтерфейс відеорумів виконано в тому ж стилі, що й інші екрани LMS: карткова сітка, акцентна кнопка створення, статусні бейджі (активна / завершена сесія).

Висновки до розділу 3

В рамках розділу виконано комплексне проєктування та моделювання LMS для університетського середовища на основі об’єктно-орієнтованого підходу та стандарту UML. Побудовані діаграми прецедентів, класів, контролерів, взаємодії, діяльності, розгортання та пакетів узгоджені з фактичною архітектурою застосунку: клієнт React, сервер ASP.NET Core Web API, база Microsoft SQL Server, автентифікація JWT і обмін повідомленнями через SignalR. Доменна модель відображає специфіку закладу вищої освіти (факультети, кафедри, групи), рольовий поділ користувачів і повний цикл навчальної діяльності – від зарахування на курс до тестування та виставлення оцінок.

Окремо спроєктовано структуру бази даних із застосуванням TPH для користувачів, нормалізованими сутностями курсів, матеріалів, тестів і журналу оцінок, а також обґрунтованими політиками цілісності та індексації. Проєктування інтерфейсу забезпечило рольові кабінети з єдиним технологічним стеком на клієнті та централізованим доступом до API через axios, що спростило підтримку безпеки та узгодженість даних між екранами.

Результати моделювання створюють цілісну основу для реалізації та подальшого розширення системи. Виявлення структури та поведінки на етапі проєктування дозволило зменшити ризик архітектурних помилок.

4 РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ

Розробка корпоративних платформ вимагає комплексного підходу, який враховує не лише технічні аспекти, але й специфічні потреби бізнесу, вимоги до безпеки, масштабованості та інтеграції з існуючими корпоративними системами. Особливістю даного проєкту є поєднання соціальної взаємодії працівників з освітніми функціями та можливостями інтеграції з зовнішніми сервісами, що створює унікальні виклики для архітектури та реалізації.

Процес реалізації структурований за принципом поступового ускладнення: від початкової конфігурації проєкту та налаштування середовища розробки до створення складних інтерактивних компонентів та інтеграції з зовнішніми API. Такий підхід дозволяє забезпечити стабільну основу для подальшого розширення функціональності та додавання нових модулів.

Архітектурні рішення базуються на принципах модульності, переусобуваності компонентів та чіткого розділення відповідальностей. Це забезпечує можливість паралельної роботи команди розробників над різними частинами системи та спрощує подальшу підтримку і розвиток платформи.

Особлива увага приділяється питанням оптимізації продуктивності, оскільки корпоративні платформи часто обробляють великі обсяги даних та обслуговують значну кількість одночасних користувачів. Використання сучасних React-паттернів та інструментів дозволяє досягти високої продуктивності навіть при складних взаємодіях між компонентами.

4.1 Реалізація серверної частини

Початкова фаза розробки корпоративної платформи включає створення структури проєкту, налаштування середовища розробки та конфігурацію основних інструментів. Правильна ініціалізація проєкту закладає фундамент для ефективної розробки та забезпечує дотримання best practices протягом усього життєвого циклу застосунку.

Тож серверну частину реалізовано як вебпроєкт ASP.NET Core 8.0. Точка входу – Program.cs, де зібрано «каркас» (рис. 4.1) застосунку: реєстрація контролерів, підключення Swagger для документування API, налаштування JWT-автентифікації, політики CORS для клієнта на http://localhost:5173, реєстрація ApplicationDBContext із провайдером SQL Server, підключення SignalR і конвеєр middleware. При старті застосунку автоматично застосовуються міграції EF Core, що узгоджує схему БД із моделлю даних без ручного виконання скриптів (див. Додаток А).

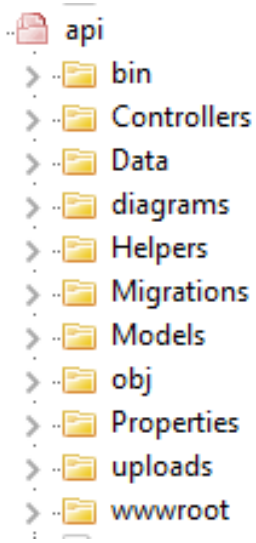

Рисунок 4.1 – Каркас серверної частини застосунку

Шар даних і предметна модель

Доменна модель описана класами каталогу api/Models і відображена в ApplicationDBContext. Центральною сутністю є User з полями ПІБ, email, хеш пароля та роллю. Для спрощення зберігання та запитів застосовано успадкування TPH: Student і Teacher зберігаються в одній таблиці з дискримінатором Discriminator, що дозволяє мати спільні атрибути користувача й водночас зберігати специфічні поля нащадків (наприклад, прив’язку студента до академічної групи).

Організаційна структура університету представлена сутностями Faculty, Department і Group; для них у OnModelCreating задано унікальні індекси, що запобігає дублюванню назв на одному рівні ієрархії.

Навчальний блок охоплює Course, CourseEnrollment (зарахування студентів), CourseModule, CourseFolder, CourseFile, ModuleResource та MaterialAccess для

керування доступом до матеріалів курсу. Підсистема контролю знань реалізована сутностями Test, Question, TestAccess, TestAttempt, TestAttemptAnswer; журнал успішності – Grade з типами оцінок (Test, Assignment, LabWork тощо).

Окремо передбачено модуль здачі робіт: Assignment (назва, дедлайн, максимальний бал, обмеження розміру та розширень файлів, кількість спроб), AssignmentAccess (обмеження доступу за групою або студентом) і AssignmentSubmission (завантажені файли та статус перевірки).

Комунікаційний шар включає Message для особистих повідомлень, CourseGroupChat з пов’язаними Message для групових каналів курсу, а також допоміжні сутності для опитувань, системних сповіщень і відеокімнат. У конфігурації контексту задано політики каскадного видалення та обмеження Restrict там, де видалення батьківського запису могло б пошкодити цілісність даних (наприклад, зв’язок студента з групою).

Автентифікація та захист API

Єдиним публічним ендпоінтом автентифікації є POST /api/auth/login. Контролер AuthController знаходить користувача за email, перевіряє пароль через PasswordHelper.VerifyPassword, який використовує алгоритм BCrypt, і формує JWT із claims: ідентифікатор, email, роль, ім’я, по батькові та прізвище.Решта API захищена атрибутом [Authorize]; операції студента, викладача та адміністратора додатково обмежуються [Authorize(Roles = "...")].

У Program.cs налаштовано валідацію токена (issuer, audience, термін дії, підпис HMAC-SHA256), а для SignalR-з’єднань передбачено передачу JWT через query-параметр access_token на маршруті /userHub, оскільки WebSocket не завжди дозволяє стандартно передати заголовок Authorization.

Лістинг коду 4.1 – Фрагмент генерації JWT (AuthController.cs)

```
var claims = new[]
{
    new Claim(ClaimTypes.NameIdentifier, user.Id.ToString()),
    new Claim(ClaimTypes.Email, user.Email),
    new Claim(ClaimTypes.Role, user.Role.ToString()),
    new Claim("FirstName", user.FirstName),
    new Claim("MiddleName", user.MiddleName ?? string.Empty),
    new Claim("LastName", user.LastName),
};

var token = new JwtSecurityToken(
    issuer: _configuration["Jwt:Issuer"],
    audience: _configuration["Jwt:Audience"],
    claims: claims,
    expires:
DateTime.UtcNow.AddDays(Convert.ToDouble(_configuration["Jwt:ExpiryInDa
ys"])),
    signingCredentials: credentials
);
```

Перевірка ролі на сервері ілюструється, зокрема, AdminController, де доступ до адміністративних даних дозволено лише користувачам з роллю Admin. На рівні бізнес-логіки контролери курсів, тестів і завдань додатково перевіряють власність ресурсу: викладач може змінювати лише ті курси, де він призначений TeacherId, а студент — лише ті тести й завдання, на курси яких він зарахований через CourseEnrollment і які не обмежені записами TestAccess або AssignmentAccess.

Контролери та REST API

API організовано за REST-принципом: кожна предметна область має контролер із базовим маршрутом api/[controller]. Серед ключових – CoursesController, CourseMaterialsController, TestsController і StudentTestsController, AssignmentsController – створення тек завдань, завантаження робіт студентом, ручне оцінювання викладачем з можливістю виставити бал нижче MaxScore, GradesController, CourseGroupChatsController – групові канали курсу, VideoRoomsController, MessagesController, StudentsController і TeachersController.

Для розробки та інтеграційного тестування підключено Swagger UI з підтримкою Bearer-токена. Версіонування схеми БД ведеться через міграції Entity Framework Core; при старті застосунку виконується context.Database.Migrate(), після чого сервіс

CourseProvisioningService автоматично створює для курсів відеокімнати, групові чати та репозиторії, а TestDataSeeder – початкові тестові дані для демонстрації функціоналу.

Реалізація тестування та журналу оцінок

Створення тесту викладачем передбачає збереження метаданих (назва, дедлайн, ліміт часу, максимальний і прохідний бал) і колекції питань типу SingleChoice з JSON-представленням варіантів і правильної відповіді. Студент отримує список доступних тестів через GET /api/studenttests/available з урахуванням зарахування на курс, вікна відкриття/закриття та правил доступу.

Запуск спроби – POST /api/studenttests/{testId}/start: якщо вже існує незавершена спроба зі статусом InProgress, сервер відновлює сесію замість створення дубліката; інакше створюється новий TestAttempt. Відповіді зберігаються інкрементально – SaveAnswer, а при SubmitTest для закритих питань виконується автоматичне порівняння з CorrectAnswerJson і формується запис у Grades, який одразу відображається в журналі викладача та студента.

Реалізація завдань

Модель Assignment дозволяє викладачу створити «теку» здачі робіт у межах курсу, модуля або папки матеріалів, задати DueDate, MaxScore (наприклад, 10 балів для лабораторної), обмеження на розмір файлу та дозволені розширення, а також кількість спроб і правило пізньої здачі.

Студент завантажує файл через POST /api/assignments/{id}/submit; викладач переглядає подання і через POST /api/assignments/submissions/{submissionId}/grade виставляє фактичний Score, який може бути меншим за MaxScore, супроводжуючи рішення коментарем у полі Comment. Результат синхронізується з сутністю Grade, що забезпечує єдиний журнал оцінок для тестів і письмових робіт.

Захист даних і файлове сховище

Текст особистих повідомлень у БД зберігається в полі EncryptedText сутності Message; перед записом і після читання застосовується EncryptionHelper на основі AES із ключем довжиною 256 біт з конфігурації Encryption:Key. Файли курсів, чатів і зданих робіт зберігаються у файловій системі сервера (wwwroot/uploads, каталоги курсів), а в базі даних лишаються метадані (шлях, ім’я, тип, розмір). Статичні файли

обслуговуються через UseStaticFiles(). Політика CORS дозволяє запити з http://localhost:5173 з credentials, що необхідно для cookie-aware запитів axios на клієнті.

Відеоруми

Підсистема відеоконференцій побудована за гібридною схемою: LMS зберігає метадані та керує доступом, а сама відеосесія відбувається на зовнішній платформі Jitsi Meet. Для кожного курсу при провізіонінгу автоматично створюється запис VideoRoom з унікальним RoomCode, статусом Scheduled та датами з Course.StartDate / EndDate. URL приєднання формується як https://meet.jit.si/university-lms-{RoomCode} – публічний інстанс Jitsi, не потребує власного медиа-сервера на етапі прототипу.

Контролер VideoRoomsController надає REST API: GET /api/videorooms/teacher і GET /api/videorooms/student – списки кімнат зарахованих/викладаних курсів; GET /api/videorooms/join/{roomCode} – пошук кімнати за кодом; PUT /api/videorooms/{id} – зміна статусу (Live, Scheduled, Ended), назви та часу (лише викладач). Статус «В ефірі» змінюється викладачем у модальному вікні налаштувань; клієнт відображає відповідний бейдж і рожеву кнопку «Приєднатися». Технологічний стек відео: WebRTC для медіа-потоків; ASP.NET Core – лише оркестрація, авторизація JWT і персистентність у SQL Server.

Real-time комунікація

Окрім відео через Jitsi, real-time у LMS забезпечує ASP.NET Core SignalR. Хаб UserHub зареєстрований на /userHub, захищений [Authorize]; JWT передається через query-параметр access_token (налаштування в JwtBearerEvents.OnMessageReceived), оскільки WebSocket не завжди передає заголовок Authorization. При OnConnectedAsync сервер зберігає пару «userId – ConnectionId» для адресної доставки особистих повідомлень; при відключенні запис видаляється.

Особистий чат використовує SignalR для миттєвого оновлення стрічки; текст повідомлень у БД зберігається зашифрованим. Групові канали курсу працюють переважно через REST із polling/React Query, доповнені push-подіями за потреби; розширений функціонал включає reply, upload файлів, контекстне меню, скарги в канал системних сповіщень адміна та автовизначення мови коду нейромережею. Таким

чином real-time охоплює два рівні: текстовий чат (SignalR + шифрування) і відеолекції, обидва інтегровані в єдину навігацію LMS.

ШІ-компонент: автовизначення мови програмного коду

У групових чатах курсу передбачено автоматичне розпізнавання мови лістингу для подальшої підсвітки синтаксису. Реалізовано двошарову нейромережу CodeLanguageNeuralNet: вхідний шар формується з ознак ключових слів і структурних характеристик тексту, прихований шар містить 24 нейрони, вихідний – 12 класів мов (Python, C#, JavaScript, TypeScript, C++, Java, SQL, HTML, CSS, JSON, Go, Rust).

Лістинг коду 4.2 – Фрагмент класифікації (CodeLanguageNeuralNet.cs)

```
public (string Language, float Confidence) Predict(string content)
{
    if (string.IsNullOrWhiteSpace(content))
        return ("plaintext", 0f);

    var features = ExtractFeatures(content);
    var output = Forward(features);
    var best = 0;
    for (var i = 1; i < output.Length; i++)
        if (output[i] > output[best]) best = i;

    return (Languages[best], output[best]);
}
```

Мережа навчається один раз при старті сервера на вбудованих зразках коду; результат класифікації повертає мову та рівень впевненості. Клас CodeLanguageDetector інтегрує нейромережу з евристиками (розширення файлу, маркери блоків коду) і викликається при обробці повідомлень у CourseGroupChatsController, а також через окремий ендпоінт CodeLanguageController для діагностики.

4.2 Реалізація клієнтської частини

Клієнтську частину реалізовано як односторінковий застосунок на React 18 із збірником Vite, що забезпечує швидкий hot reload під час розробки та оптимізовану production-збірку. Стилізація поєднує Tailwind CSS, компоненти Mantine і власні CSS-модулі для shell-інтерфейсів викладача та студента (наприклад, teacherShell.css), що

дозволяє дотримуватися єдиного візуального макету – бокова панель, верхній header, карткові секції контенту.

Кореневий компонент App.jsx обгортає застосунок провайдерами: ThemeProvider: світла та темна теми, QueryClientProvider – TanStack React Query для кешування серверних даних, MantineProvider, AuthProvider, SidebarProvider і BrowserRouter.

Публічний маршрут – /login; після успішного authApi.login JWT зберігається в localStorage, декодується через jwt-decode, і компонент ProtectedRoute допускає до вкладених маршрутів лише користувачів із дозволеною роллю, перенаправляючи інших на /unauthorized (див. ліст. коду 4.3).

Лістинг коду 4.3 – Захист маршрутів на клієнті (ProtectedRoute.jsx)

```
if (!user) {
  return <Navigate to="/login" replace />;
}
if (!allowedRoles.includes(user.role)) {
  return <Navigate to="/unauthorized" replace />;
}
return <Outlet />;
```

Кореневий компонент App.jsx обгортає застосунок провайдерами: ThemeProvider, QueryClientProvider, MantineProvider, AuthProvider, SidebarProvider і BrowserRouter для маршрутизації. Публічний маршрут – /login; після успішного authApi.login токен зберігається в localStorage, роль зчитується з JWT, і ProtectedRoute (див. Лістинг коду 4.3) перенаправляє на /admin, /teacher або /student.

Авторизація на клієнті

На відміну від класичних корпоративних схем із refresh-токенами, у поточній версії LMS застосовано спрощену, але достатню для навчального середовища модель: один JWT з терміном дії, заданим у конфігурації сервера (Jwt:ExpiryInDays), зберігається в localStorage і автоматично додається до кожного HTTP-запиту через interceptor axios (Authorization: Bearer …).

Стан користувача тримає React Context: при перезавантаженні сторінки токен зчитується зі сховища, декодується, і інтерфейс відновлює сесію без повторного входу, доки токен валідний. Реєстрація публічно недоступна – облікові записи створює адміністратор через адмін-панель, що відповідає політиці централізованого управління доступом в освітній організації.

Організація інтерфейсу за ролями

Маршрути розділено на три незалежні оболонки:

– адміністратор – /admin/*, керує викладачами, студентами, курсами, звітами та системними сповіщеннями;

– викладач – /teacher/*, має dashboard, список курсів, сторінку керування курсом з матеріалами, тестами, завданнями та журналом оцінок, групові канали, відеоруми, календар і особисті чати;

– студент – /student/*, який отримує аналогічну оболонку з dashboard, переглядом курсів, груповими каналами, відеорумами, списком тестів, сторінкою здачі завдань і журналом оцінок; проходження тесту винесено на окремий повноекранний маршрут /student/test/:testId з таймером, сіткою навігації по питаннях і з системою прокторингу.

Взаємодія з API

Усі запити інкапсульовано в модулі client/src/services/api.js: окремі об’єкти authApi, courseMaterialsApi, testsApi, studentTestsApi, gradesApi, assignmentsApi, videoRoomsApi, courseGroupChatsApi тощо. TanStack Query використовується для асинхронного завантаження даних, кешування і інвалідації після мутацій, наприклад, після завершення тесту оновлюється журнал оцінок, після відправки роботи – список подань у викладача.

Компоненти форм: TestForm та AssignmentForm працюють через controlled inputs Mantine і useForm, а для групового чату виділено підкомпоненти GroupChatMessage, GroupChatCompose, FilePreviewModal з підсвіткою коду через highlight.js.

Real-time на клієнті

Особистий чат підключається до SignalR-хаба з передачею токена. Підключення виконується через @microsoft/signalr з токеном із localStorage; оновлення стрічки особистих і групових чатів відбувається без повного перезавантаження сторінки, що підвищує відчуття «живої» навчальної платформи порівняно з класичним REST-only підходом.

4.3 Тестування програмного забезбечення

Тестування є невід'ємною частиною життєвого циклу розробки будь-яких систем. Для LMS це особливо важливо: платформа об'єднує критичні сценарії – автентифікацію, збереження оцінок, здачу робіт, проходження тестів і обмін даними в реальному часі. Помилка в будь-якому з цих модулів може призвести до некоректних академічних результатів, втрати даних або несанкціонованого доступу.

Модульне тестування

Модульне тестування (англ. unit testing) перевіряє коректність роботи окремих модулів у ізоляції від зовнішніх залежностей: без реального SQL Server, без браузера, без живого SignalR-з'єднання. Це найнижчий і найшвидший рівень автоматизованої перевірки; він дозволяє виявити дефекти на ранній стадії – у хелперах, сервісах і контролерах до інтеграції з UI.

Серверна частина оформлена як окремий проєкт api.Tests (xUnit, .NET 8.0), доданий до solution api.sln. Для ізоляції даних використовується Entity Framework Core InMemory – контекст ApplicationDBContext створюється у пам'яті для кожного тесту, без підключення до production БД.

Допоміжний клас TestDbContextFactory формує типовий набір сутностей (факультет, кафедра, група, викладач, студент, курс, тест, завдання) і дозволяє повторно використовувати seed у тестах контролерів. Для контролерів, що потребують

автентифікованого контексту, ControllerTestHelper імітує JWT claims (NameIdentifier, Role) без реального токена.

Клієнтська частина тестується фреймворком Vitest (v3.2.6). На першому етапі покрито модуль codeUtils.js – утиліти групового чату (нормалізація мови, парсинг code fence, форматування розміру файлу, розпізнавання зображень, парсинг @mentions). Це відповідає принципу тестувати «чисті» функції без React-компонентів і без HTTP.

Таблиця 4.1 – Об’єкти модульного тестування та сценарії

Модуль / клас	**Фреймворк**	**Що перевіряється**
PasswordHelper	xUnit	BCrypt-хешування, верифікація пароля, зворотна сумісність з plaintext
EncryptionHelper	xUnit	AES round-trip шифрування та дешифрування повідомлень
CodeLanguageDetector	xUnit	Визначення мови з розширення файлу, code fence, евристики, автокласифікація коду в каналі
CodeLanguageNeuralNet	xUnit	MLP-класифікація зразків Python і C#
StudentTestsController	xUnit	Старт тесту, відновлення InProgress, здача та запис у Grades, блокування повторного проходження
AssignmentsController	xUnit	Оцінювання роботи балом нижче MaxScore, валідація меж балу, Forbid для стороннього викладача
Grade (персистентність)	xUnit	Збереження оцінки за спробою тесту в InMemory БД
codeUtils.js	Vitest	6 сценаріїв утиліт групового чату

Контролери тестуються напряму (без Kestrel і HTTP pipeline), що є типовим підходом для unit-тестів ASP.NET Core: перевіряється бізнес-логіка метода, а не мережевий стек.

Лістинг коду 4.4 – Покриття коду тестами (EncryptionHelperTests)

```
public class EncryptionHelperTests
{
    public EncryptionHelperTests()
    {
        TestDbContextFactory.InitializeEncryption();
```

```
    }

    [Fact]
    public void EncryptDecrypt_RoundTripsPlaintext()
    {
        const string message = "Конфіденційне повідомлення";
        var encrypted = EncryptionHelper.Encrypt(message);
        var decrypted = EncryptionHelper.Decrypt(encrypted);

        Assert.NotEqual(message, encrypted);
        Assert.Equal(message, decrypted);
    }

    [Fact]
    public void EncryptMessage_RoundTripsLikeEncrypt()
    {
        const string message = "test";
        var encrypted = EncryptionHelper.EncryptMessage(message);
        var decrypted = EncryptionHelper.Decrypt(encrypted);
        Assert.Equal(message, decrypted);
    }
}
```

Усі автоматизовані модульні тести завершились успішно; регресійні дефекти в покритих модулях на момент тестування не виявлено (див. рис. 4.2-4.3).

```
[xUnit.net 00:00:00.00] xUnit.net VSTest Adapter v2.8.2+699d445a1a (64-bit .NET 8.0.28)
[xUnit.net 00:00:00.13]   Discovering: api.Tests
[xUnit.net 00:00:00.25]   Discovered:  api.Tests
[xUnit.net 00:00:00.25]   Starting:    api.Tests
[xUnit.net 00:00:54.16]   Finished:    api.Tests
  api.Tests test succeeded (56,0s)

Test summary: total: 29; failed: 0; succeeded: 29; skipped: 0; duration: 56,5s
Build succeeded with 26 warning(s) in 63,1s
```

Рисунок 4.2 – Результати модульного тестування Backend-частини

```
> client@0.0.0 test
> vitest run


 RUN  v3.2.6 E:/University_LMS/University_LMS/client

 ✓ src/Teacher/groupChat/codeUtils.test.js (6 tests) 6ms
   ✓ codeUtils > normalizeLang maps aliases 2ms
   ✓ codeUtils > detectLanguageFromContent recognizes csharp 0ms
   ✓ codeUtils > parseCodeFence extracts fenced block 0ms
   ✓ codeUtils > formatFileSize formats bytes 0ms
   ✓ codeUtils > isImageFile detects images 0ms
   ✓ codeUtils > parseMentionsFromText finds members 2ms

 Test Files  1 passed (1)
      Tests  6 passed (6)
   Start at  00:00:24
   Duration  1.71s (transform 519ms, setup 0ms, collect 203ms, tests 6ms, environment 0ms, prepare 716ms)
```

Рисунок 4.3 – Результати модульного тестування Frontend-частини

Модульне тестування LMS охоплює 35 автоматизованих тест-кейсів (29 backend + 6 frontend) з pass rate 100 %. Перевірено критичні модулі безпеки (паролі, шифрування), ШІ-детектор мови коду, сценарії проходження тестів та оцінювання робіт. Це забезпечує регресійну стабільність при розробці.

Навантажувальне тестування

Навантажувальне тестування (англ. load testing) дозволяє оцінити поведінку системи в умовах одночасного використання багатьма користувачами. Для LMS це особливо актуально під час сесійних періодів, коли велика кількість студентів одночасно проходять тести або здають роботи.

Тестування проводилось із симуляцією 20 одночасних анонімних підключень до системи протягом 60 секунд. Метою прогону було визначити стабільність API-шару та виявити можливі вузькі місця при пікових навантаженнях.

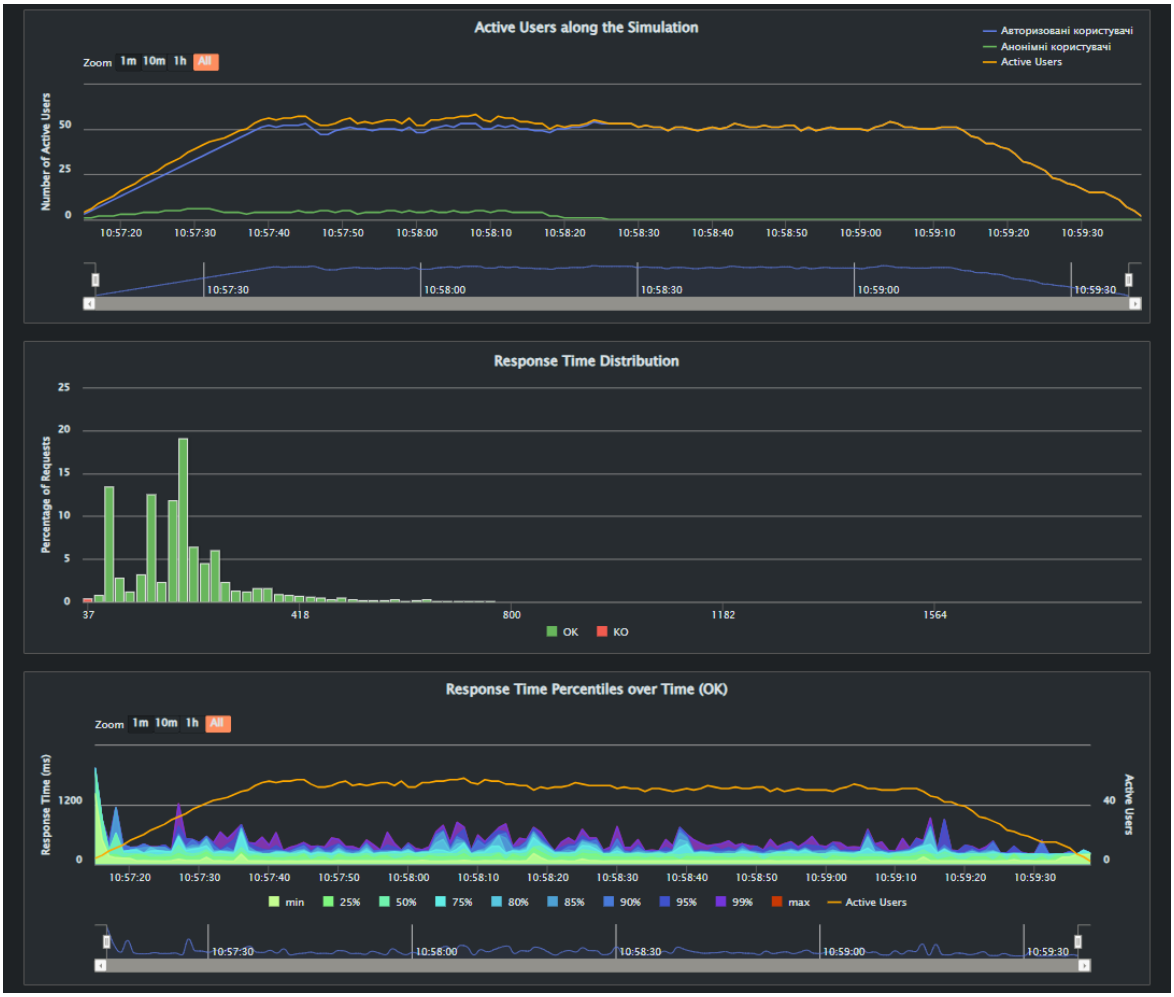

Рисунок 4.4 – Навантажувальний прогін

На рисунку 4.4 одразу 20 анонімних користувачів підключаються до системи протягом 60 секунд. За результатами прогону, система впоралась із навантаженням без критичних збоїв: всі запити отримали відповідь у межах допустимого часу. Навантажувальне тестування підтвердило готовність серверної інфраструктури до реального використання в умовах навчального закладу.

4.4 Результати розробки

Результатом імплементації LMS є повнофункціональний програмний комплекс, що складається з серверної частини на ASP.NET Core 8.0, клієнтського SPA на React і реляційної бази даних SQL Server. Система охоплює три рольові кабінети – адміністратора, викладача та студента, а також реалізує ключові освітні сценарії: керування навчальними курсами, обмін повідомленнями в реальному часі, автоматизоване тестування, здачу та оцінювання робіт, відеоконференції та адміністрування користувачів. У цьому підрозділі наведено візуальну демонстрацію реалізованих екранів інтерфейсу з поясненням їхньої функціональної ролі в загальній архітектурі системи. Кожен екран ілюструє не лише зовнішній вигляд, а й зв’язок із backend-модулями, маршрутизацією React Router і механізмами захисту доступу.

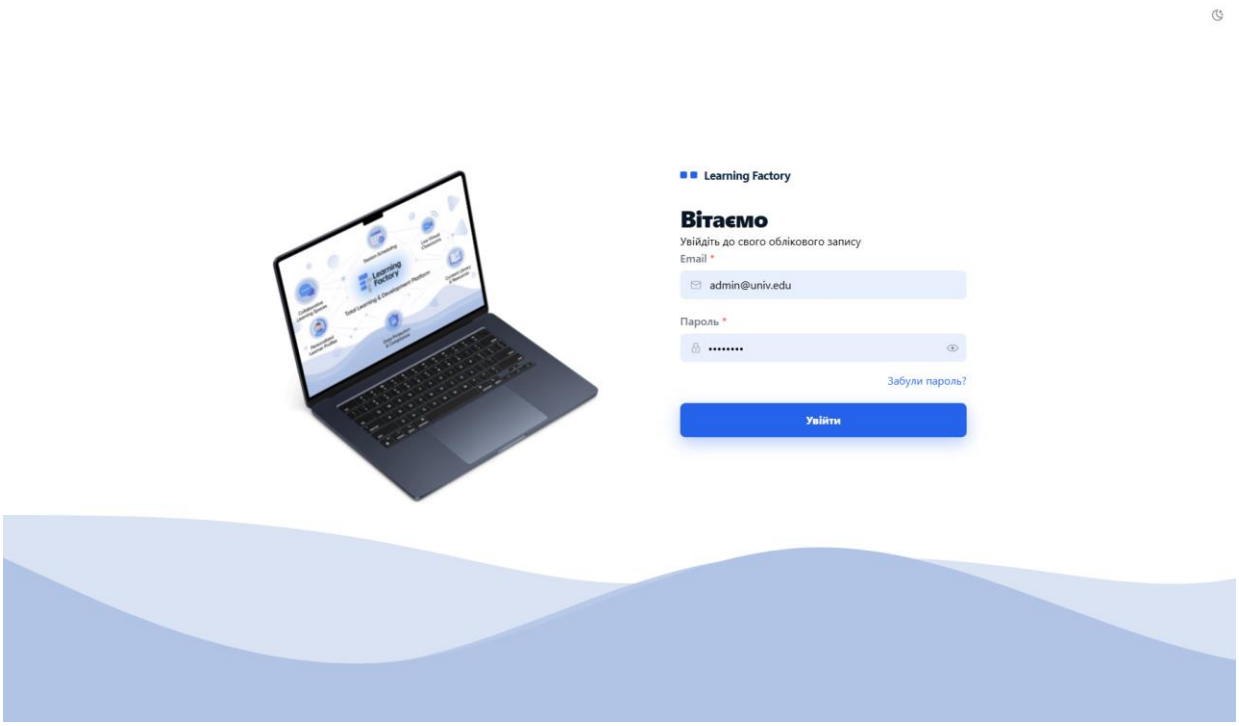

Рисунок 4.5 – Сторінка авторизації

На рисунку 4.5 продемонстровано реалізовану сторінку авторизації. Екран поділено на дві функціональні зони: ліворуч розміщено ілюстративну панель з брендингом платформи «Learning Factory», праворуч – форму входу з полями для Email та Паролю, акцентною кнопкою «Увійти» та посиланням «Забули пароль?» та без публічної реєстрації.

Компонент LoginForm використовує Mantine (TextInput, PasswordInput) і звертається до POST /api/auth/login. Після успішної автентифікації JWT-токен зберігається в localStorage, роль декодується через jwt-decode, і ProtectedRoute перенаправляє користувача до відповідного кабінету: /admin, /teacher або /student.

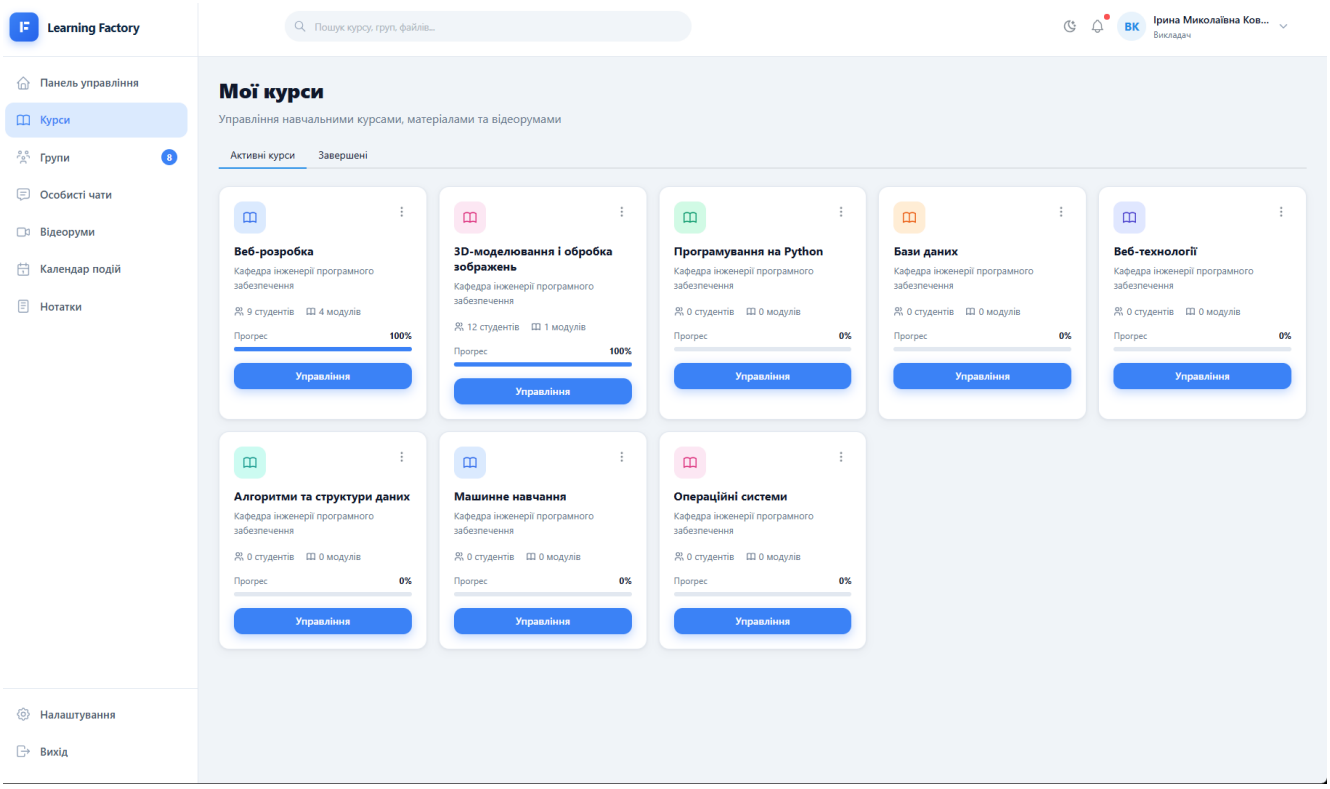

Рисунок 4.6 – Сторінка керування курсами

На рисунку 4.6 представлено сторінку переліку курсів у кабінеті викладача. Курси відображаються у вигляді карток із назвою, коротким описом, кафедрою та кількістю модулів і тестів. Дані надходять з GET /api/coursematerials/teacher/courses.

З цієї сторінки викладач переходить до детального управління курсом (/teacher/courses/:courseId), відеорумів, репозиторію курсу, групових каналів і журналу оцінок. Маршрутизація реалізована в межах захищеної оболонки /teacher/* з перевіркою ролі Teacher на клієнті та [Authorize(Roles = "Teacher")] на сервері.

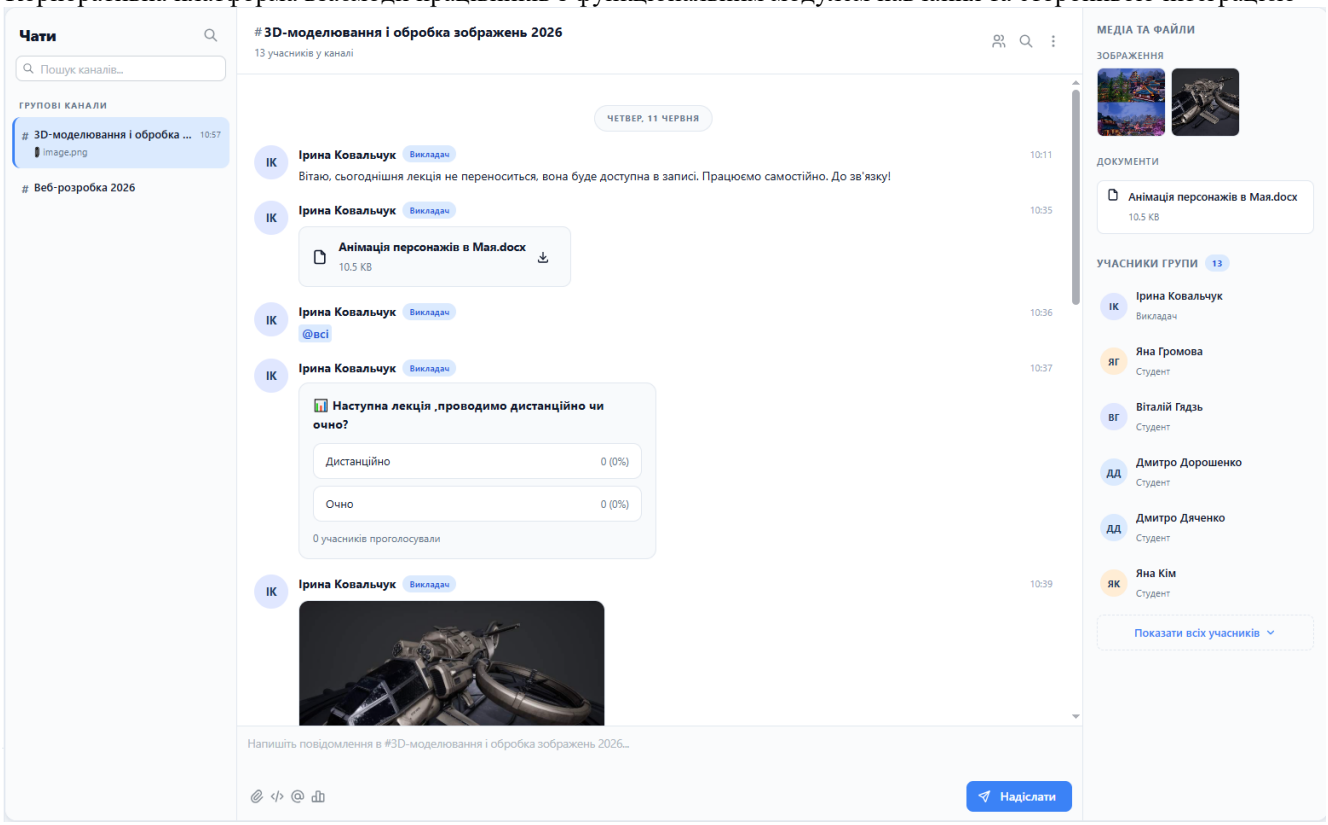

Рисунок 4.7 – Груповий канал курсу

На рисунку 4.7 зображено інтерфейс групового каналу курсу. Центральна область відображає стрічку повідомлень з підтримкою вставки програмного коду, прикріплення файлів та відповідей на конкретні повідомлення. Оновлення стрічки відбувається в реальному часі через SignalR-хаб UserHub без перезавантаження сторінки.

Передбачено контекстне меню для видалення власного повідомлення або надсилання скарги адміністратору. Підсвітка синтаксису – highlight.js на клієнті; мова визначається на сервері модулем CodeLanguageDetector і нейромережею CodeLanguageNeuralNet (див. рис. 4.8).

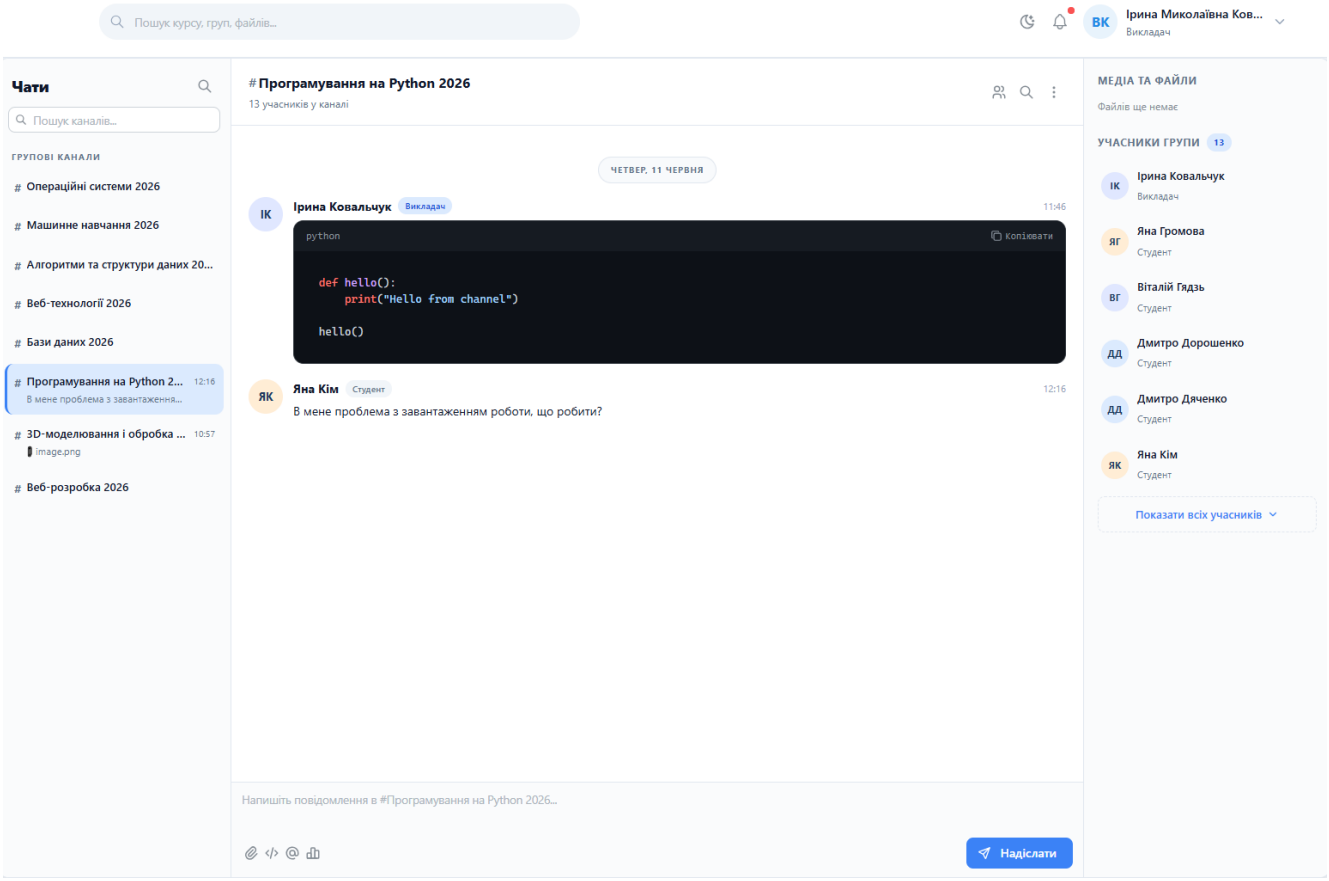

Рисунок 4.8 – Розпізнавання мов програмування

На рисунку 4.8 продемонстровано роботу модуля автоматичного визначення мови програмування у груповому каналі курсу. При надсиланні блоку коду система аналізує його вміст за допомогою двошарової нейромережі CodeLanguageNeuralNet (24 нейрони прихованого шару, 12 класів на виході) та відображає над блоком бейдж із назвою визначеної мови. Детектор підтримує Python, C#, JavaScript, TypeScript, C++, Java, SQL, HTML, CSS, JSON, Go та Rust; якщо мову не вдається визначити з достатньою впевненістю – застосовується позначка plaintext.

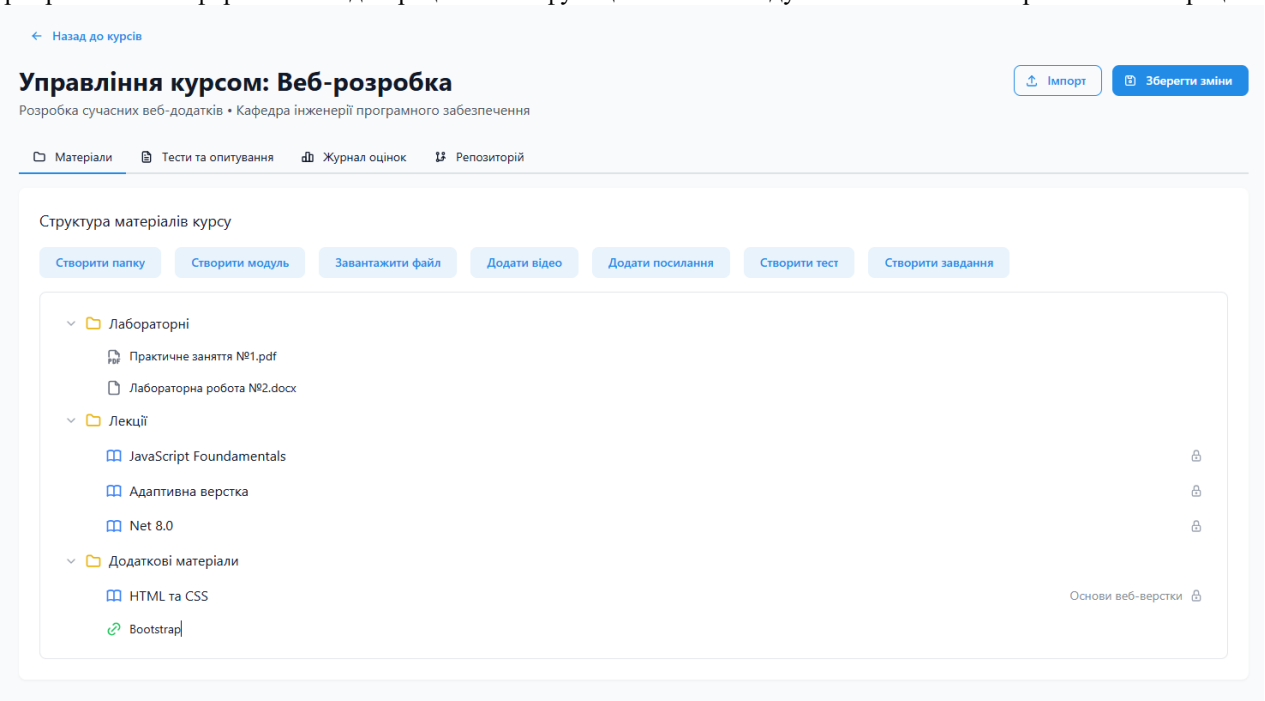

Рисунок 4.9 – Сторінка управління курсом

На рисунку 4.9 показано сторінку управління конкретним курсом у кабінеті викладача. Інтерфейс організовано у вигляді вкладок: структура курсу з модулями, папками та файлами матеріалів; окремі розділи для тестів, завдань та журналу оцінок. Викладач може додавати й редагувати модулі, завантажувати файли, налаштовувати правила доступу через TestAccess і AssignmentAccess, а також переглядати подання студентів і виставляти оцінки. Після кожної мутації TanStack Query інвалідує відповідний кеш, забезпечуючи актуальність відображуваних даних.

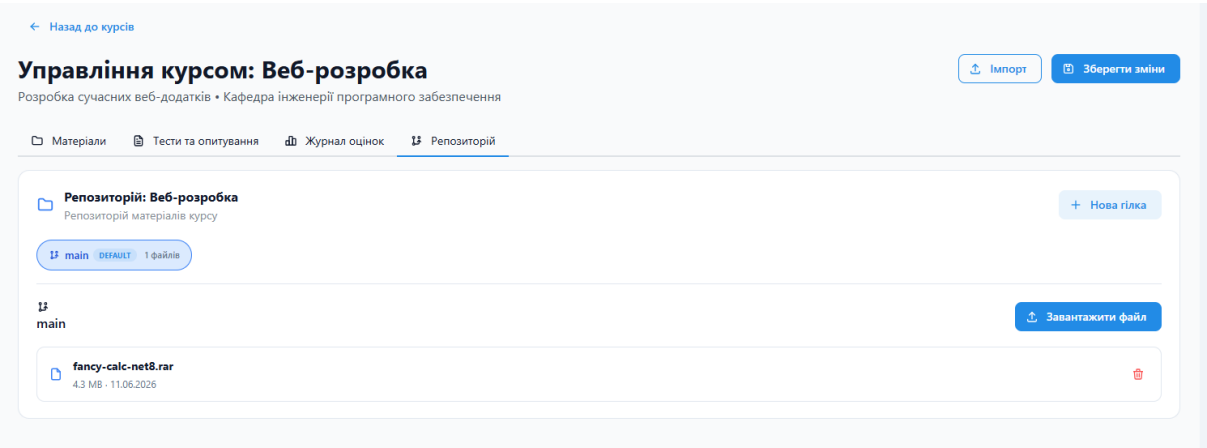

Рисунок 4.10 – Репозиторій курсу в панелі управління курсу

На рисунку 4.10 відображено репозиторій навчальних матеріалів курсу. Ієрархічна структура будується з сутностей CourseModule, CourseFolder та CourseFile; вкладеність передається відступами та іконками типів файлів. Доступ до ресурсів визначається записами MaterialAccess. Файли зберігаються у файловій системі сервера у каталозі wwwroot/uploads, а в базі даних фіксуються лише метадані – шлях, ім’я, тип і розмір. Такий підхід зменшує навантаження на СКБД і спрощує резервне копіювання матеріалів.

Звичайні папки курсу зручні для лекційних PDF і презентацій, але не відображають ітеративну роботу над кодом. Репозиторій закладає основу для сценарію «лабораторія всередині LMS» за аналогією з Git і GitHub: окремі гілки для варіантів рішення, завантаження артефактів збірки, у перспективі – інтеграція з GitHub API (clone, pull request, CI-статус). Це дозволяє викладачу оцінювати не лише фінальний файл, а процес розробки, а студенту – працювати в знайомому git-workflow без виходу з платформи.

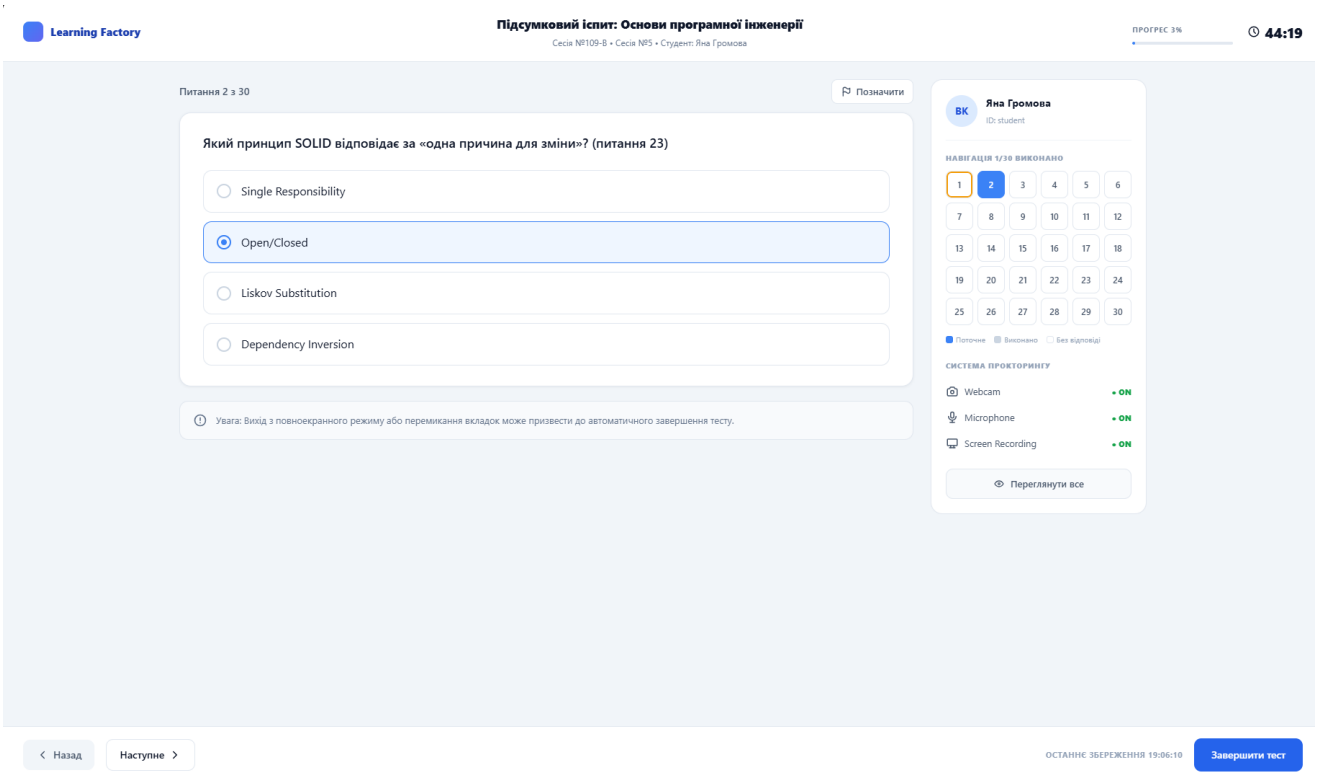

Рисунок 4.11 – Проходження тесту студентом

На рисунку 4.11 наведено інтерфейс проходження тесту студентом. У центральній зоні відображається поточне запитання та варіанти відповідей. Праворуч

розміщено панель навігації по запитаннях з візуальним позначенням відповідених, пропущених і поточного пунктів.

У верхній частині показано назву тесту та зворотний відлік таймера за наявності обмеження. Система прокторингу дозволяє достигнути максимально чесної оцінки знань здобувача. Після завершення тесту сервер автоматично порівнює їх з CorrectAnswerJson і формує запис у журналі Grades, що одразу відображається в журналі і студента, і викладача.

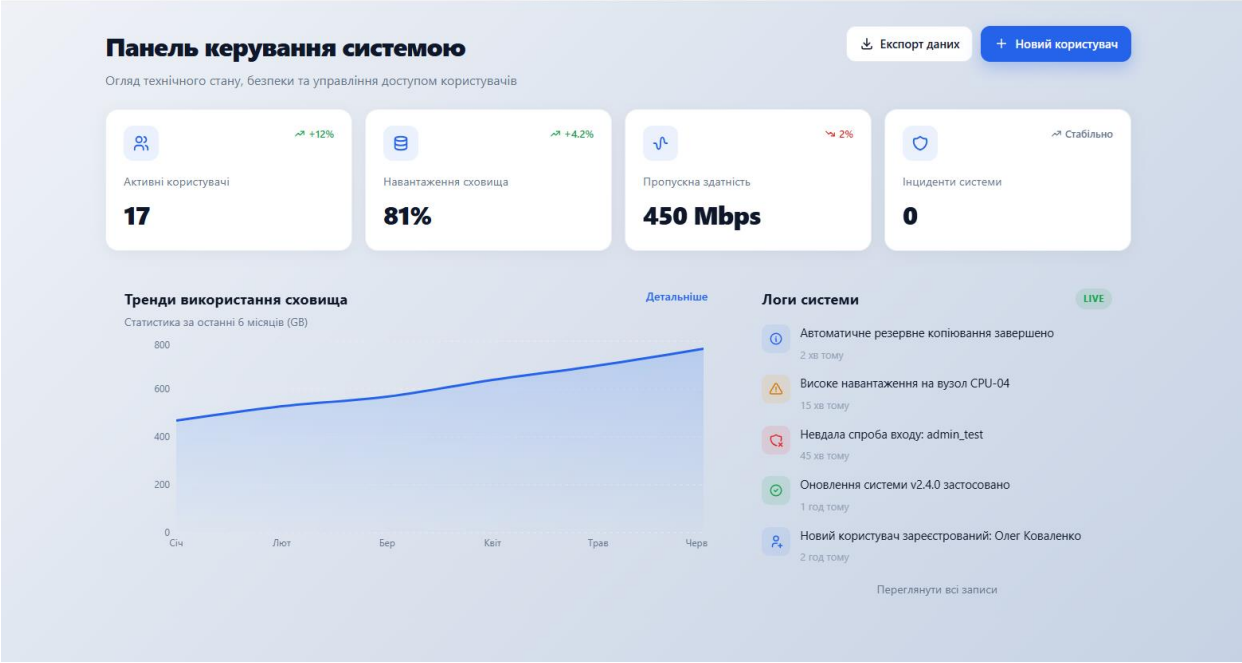

Рисунок 4.12– Інтерфейс адмін-панелі

На рисунку 4.12 представлено кабінет адміністратора. Головний екран панелі адміністратора містить картки статистики (студенти, викладачі, кафедри, курси, групи), графіки (UsersChart, CoursesDistribution, EnrollmentTrend, ActivityChart, PerformanceMetrics), блоки QuickActions, RecentActivity і SystemStatus.

Бокова панель веде до управління викладачами, студентами, курсами, звітів, налаштувань і каналу «Системні сповіщення», куди потрапляють скарги на повідомлення з групових чатів. Доступ обмежено ProtectedRoute з роллю Admin і [Authorize(Roles = "Admin")] на сервері. Адмін-панель – єдина точка моніторингу стану LMS і реагування на інциденти модерації.

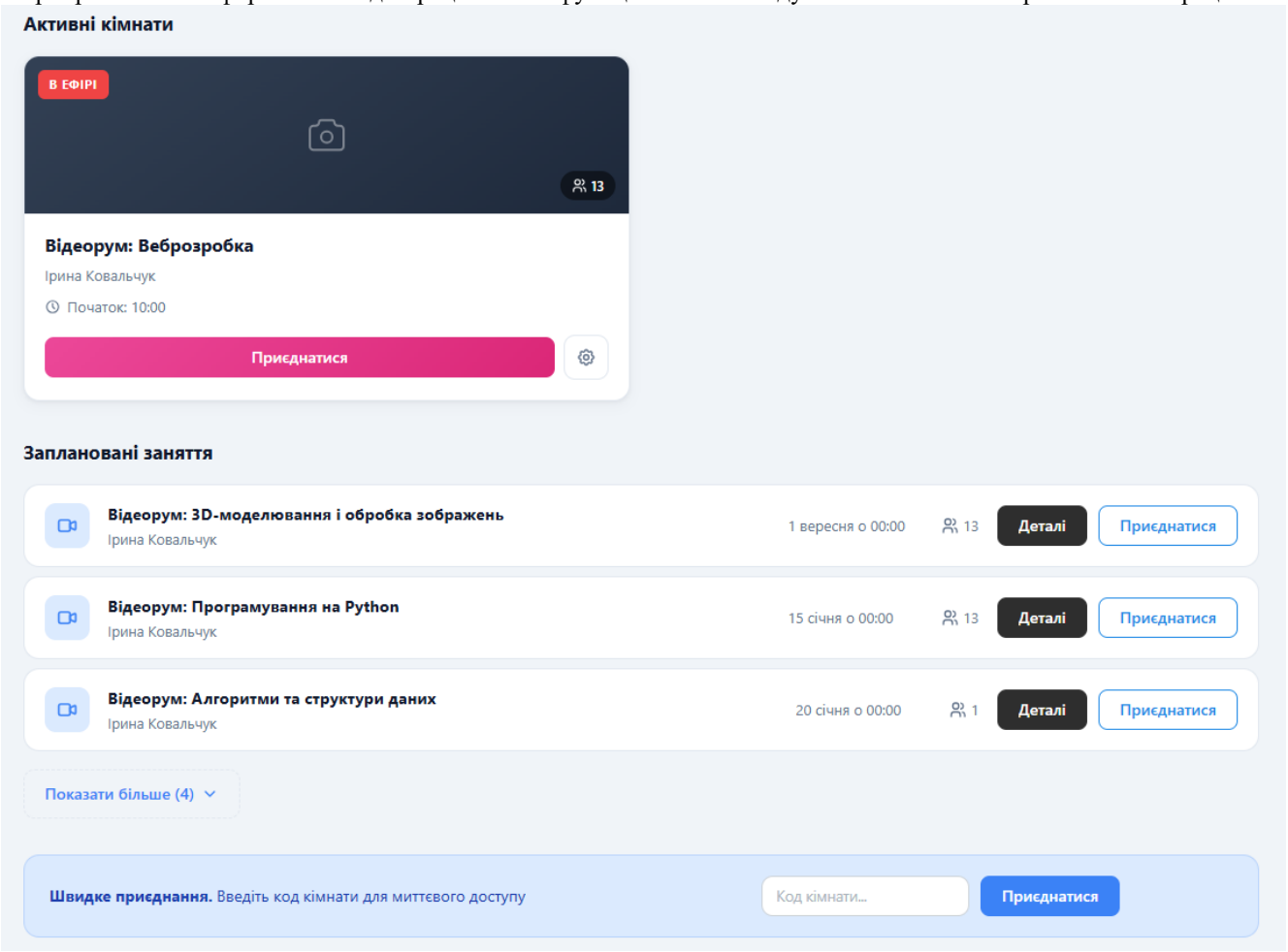

Рисунок 4.13 – Інтерфейс проведення і планування занять

На рисунку 4.13 показано сторінку відеорумів у кабінеті викладача або студента. Секція «Активні кімнати» містить картки з прев'ю, бейджем «В ефірі», кількістю учасників, кнопкою «Приєднатися» та блоком «Швидке приєднання» за кодом кімнати. При натисканні «Приєднатися» відкривається зовнішнє вікно Jitsi Meet – відеоконференція без встановлення окремого ПЗ. Секція «Заплановані заняття» – список майбутніх сесій; при великій кількості кімнат працює згортання «Показати більше». Відеоруми створюються автоматично для кожного курсу сервісом CourseProvisioningService і прив'язані до розкладу курсу.

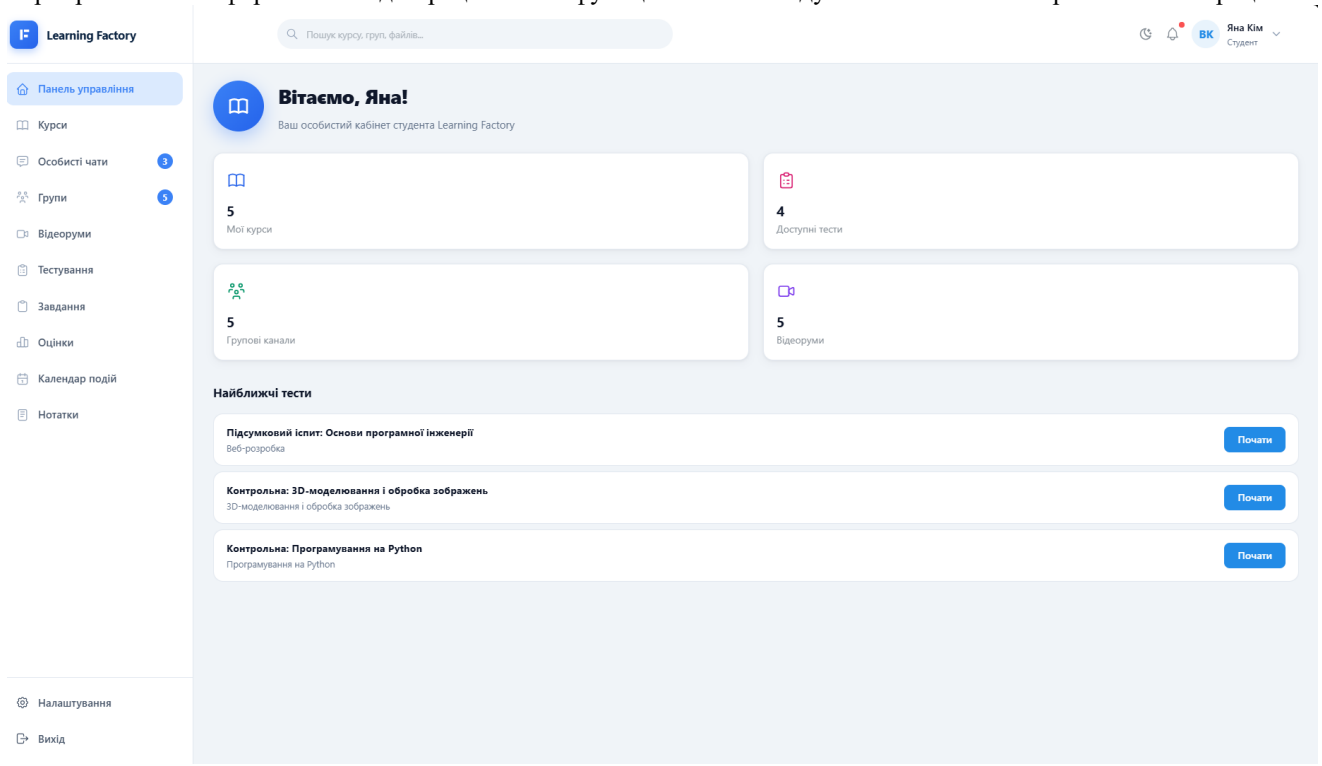

Рисунок 4.14 – Головна сторінка інтерфейсу студента

На рисунку 4.14 зображено головну сторінку особистого кабінету студента. Інтерфейс витримано в єдиній оболонці платформи: ліворуч розміщено бокову панель навігації, у верхній частині – рядок пошуку та елементи керування профілем. Знизу містяться актуальні тести, які користувач ще не пройшов, або доступ до них відкриється найближчим часом.

Висновки до розділу 4

У четвертому розділі детально розглянуто програмну реалізацію LMS. Описано серверну частину на ASP.NET Core 8.0 і клієнтський SPA на React 18: доменну модель, ключові сервіси та компоненти інтерфейсу для трьох рольових кабінетів. Проведено модульне та навантажувальне тестування, яке підтвердило стабільність API-шару в умовах пікового навантаження. Візуальна демонстрація реалізованих екранів засвідчила повну відповідність системи заявленим функціональним вимогам. Таким чином, програмна частина проєкту пройшла цикл «реалізація – тестування – демонстрація» і готова до дослідного розгортання.

ВИСНОВКИ

У кваліфікаційній бакалаврській роботі розроблено інтегровану корпоративну платформу взаємодії працівників із функціональним модулем навчання та сторонньою інтеграцією. Мета роботи досягнута: платформа забезпечує ефективну комунікацію, організацію навчального процесу й управління завданнями в єдиному цифровому середовищі.

У процесі виконання роботи вирішено всі заплановані завдання:

– проведено аналіз застосунків-аналогів (Microsoft Teams, Moodle, Google Classroom), у результаті якого виявлено системну прогалину: жодне з існуючих рішень не реалізує повний спектр функцій корпоративного навчання та взаємодії в єдиній архітектурі – комунікаційні платформи позбавлені навчальної складової, а LMS не мають вбудованих інструментів командної взаємодії в реальному часі;

– ідентифіковано ключову проблему «силосів знань» та обґрунтовано необхідність реалізації принципу «навчання в потоці роботи» через єдину платформу з нативно інтегрованими модулями комунікації, навчання та управління завданнями;

– визначено вимоги до архітектури платформи: трирівнева клієнт-серверна архітектура, multi-tenant SaaS-модель з ізоляцією орендарів через TenantId, RBAC-авторизація на базі JWT, модульність із динамічним підключенням функцій через Feature Flags, підтримка real-time комунікації через WebSocket (SignalR);

– спроєктовано архітектуру та функціональність системи: розроблено повний комплект UML-діаграм (прецедентів, класів, контролерів, послідовності, діяльності, розгортання та пакетів), спроєктовано нормалізовану базу даних з ER-діаграмою та обґрунтованими політиками цілісності, а також прототипи рольових інтерфейсів у Figma;

– розроблено серверну частину на базі ASP.NET Core 8.0 з Clean Architecture, Entity Framework Core, MS SQL Server, SignalR та JWT-автентифікацією. Реалізовано модулі курсів, тестування з автоматичним оцінюванням, завдань із файловою здачею, єдиного журналу оцінок Grade, особистих та групових чатів із AES-шифруванням,

відеоконференцій через Jitsi Meet та ШІ-компонент автовизначення мови програмного коду на базі нейромережі;

– розроблено клієнтську частину як SPA на React 18 із Vite, Tailwind CSS та Mantine UI. Реалізовано три рольові оболонки (адміністратора, викладача, студента) із захистом маршрутів через ProtectedRoute, кешуванням серверного стану через TanStack React Query та real-time оновленнями через SignalR-клієнт;

– проведено модульне та навантажувальне тестування ключових сценаріїв.

Порівняно з розглянутими аналогами розроблена платформа вирізняється нативною інтеграцією навчального модуля, корпоративного чату, управління завданнями та відеоконференцій у єдиній кодовій базі, що усуває проблему «силосів знань» на архітектурному рівні. Додатковими перевагами є вбудований ШІ-компонент автовизначення мови коду та шифрування особистих повідомлень AES-256.

Перспективами подальшого розвитку є: впровадження адаптивного навчання на основі аналітики прогресу користувачів; розширення ШІ-функціоналу; реалізація мобільного застосунку; інтеграція з HR- та BI-системами через REST API; перехід на власний WebRTC-медіасервер для відеоконференцій.

Таким чином, кваліфікаційну роботу виконано у повному обсязі – створено теоретичну базу, спроєктовано і реалізовано мінімально життєздатний продукт, підтверджено його ефективність і значимість в галузі.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Fusco G., Aversano L. An approach for semantic integration of heterogeneous data sources. PeerJ Computer Science. 2020. Vol. 6. Art. e254. DOI: 10.7717/peerj-cs.254.

2. Peyrott S. The JWT handbook. Auth0 Inc. 2016. URL: http://www.jre-training.earl-family.net/NextGen/Documents/jwt-handbook.pdf (Accessed: 10.03.2026).

3. Bradley V. M. Learning Management System (LMS) use with online instruction. ERIC. 2021. Vol. 4, No. 1. pp. 68-92. DOI: https://doi.org/10.46328/ijte.36.

4. Microsoft Teams. Microsoft Corporation. URL: https://www.microsoft.com/uk-ua/microsoft-teams/group-chat-software. (Accessed: 10.03.2026).

5. Moodle Pty Ltd. URL: https://docs.moodle.org/502/en/About_Moodle. (Accessed: 10.03.2026).

6. Google Classroom. URL: https://edu.google.com/intl/ALL_us/workspace-for-education/products/classroom/. (Accessed: 10.03.2026).

7. React Documentation. URL: https://reactjs.org. (Last Access 10.04.2026).

8. Architectural Review of Client-Server Models. URL: https://www.academia.edu/download/111170439/Architectural_Review_of_Client_Server_Models.pdf. (Accessed: 01.04.2026).

9. Software as a Service (SaaS). URL: https://www.investopedia.com/terms/s/software-as-a-service-saas.asp. (Accessed: 01.05.2026).

10. IBM. What Is Multi-Tenant? URL: https://www.ibm.com/think/topics/multi-tenant. (Accessed 01.05.2026).

11. Ji W., Muljana P. S., Romero-Hall E. The Three-Tier Design Process: Streamlined Guidelines for Designing and Developing a Course in a Learning Management System to Promote Effective Learning. College Teaching. 2020. pp. 3-14. DOI: 10.1080/87567555.2020.1865253.

12. Personalization of LMS for Different Audiences: How to Adapt Learning to Specific Business Needs. URL: https://academyocean.com/ua/blog/post/personalizaciya-

lms-dlya-riznih-auditorij-yak-adaptuvati-navchannya-pid-specifichni-biznes-potrebi. (Accessed: 10.04.2026).

13. Priyanka D. Educational Technology and Libraries Supporting Online/Digital Learning With the ASP.NET MVC Framework. AI-Assisted Library Reconstruction. IGI Global. 2024. P. 18. DOI: 10.4018/979-8-3693-2782-1.ch011.

14. Hajdin A. From Architecture to Adoption: Engineering & Scaling a Modern LMS with React. URL: https://gitnation.com/contents/from-architecture-to-adoption-engineering-and-scaling-a-modern-lms-with-react. (Accessed: 10.04.2026).

15. Row-Level Security and JSON Columns in PostgreSQL-based LMS Architectures. URL: https://www.academia.edu/114483238/Architectural_Review_of_Client_Server_Models. (Accessed: 01.05.2026).

16. Rao A. D. Automated Device Management and Multi-Tenant Architecture: A Scalable Approach for Sustainability and Education. AI-Enabled Sustainable Innovations in Education and Business. IGI Global. 2025. P. 24. DOI: 10.4018/979-8-3373-3952-8.ch006.

17. Krishna A., Rath S., Yadav S. A Cloud-Based Student Life Cycle Management System: Addressing Security Challenges and Deployment Architecture. Lecture Notes in Electrical Engineering. 2024. Vol. 1196 LNEE. pp. 295-310. DOI: 10.1007/978-981-97-7862-1_20.

18. OAuth2 and JWT-based Authorization in Enterprise LMS Systems. URL: https://docs.openedx.org/projects/openedx-proposals/en/latest/best-practices/oep-0042-bp-authentication.html. (Accessed: 01.05.2026).

19. RBAC-Models and Zero-Trust Architectures in SaaS Applications. URL: https://www.trustcloud.ai/security-assurance/what-is-zero-trust-security-in-saas-applications-a-practical-implementation-guide. (Accessed: 01.05.2026).

20. User Experience Metrics in LMS Platforms: LCP, FID, and CLS. URL: https://www.corewebvitals.io/core-web-vitals. (Accessed: 01.05.2026).

ДОДАТОК А

Лістинг коду Program.cs

```
using System.Text;
using api.Data;
using api.Hubs;
using Microsoft.AspNetCore.Authentication.JwtBearer;
using Microsoft.AspNetCore.SignalR;
using Microsoft.EntityFrameworkCore;
using Microsoft.IdentityModel.Tokens;
using Microsoft.OpenApi.Models;

var builder = WebApplication.CreateBuilder(args);

builder
    .Services.AddControllers()
    .AddJsonOptions(options =>
    {
        options.JsonSerializerOptions.PropertyNamingPolicy = null;
        options.JsonSerializerOptions.ReferenceHandler = System
            .Text
            .Json
            .Serialization
            .ReferenceHandler
            .IgnoreCycles;
    });

builder.Services.AddEndpointsApiExplorer();
builder.Services.AddSwaggerGen(options =>
{
    options.AddSecurityDefinition(
        "Bearer",
        new OpenApiSecurityScheme
        {
            Name = "Authorization",
            Type = SecuritySchemeType.Http,
            Scheme = "bearer",
            BearerFormat = "JWT",
            In = ParameterLocation.Header,
            Description = "JWT токен (без 'Bearer '). Swagger додасть
Bearer автоматично.",
        }
    );

    options.AddSecurityRequirement(
        new OpenApiSecurityRequirement
        {
            {
                new OpenApiSecurityScheme
                {
                    Reference = new OpenApiReference
                    {
```

```
                            Type = ReferenceType.SecurityScheme,
                            Id = "Bearer",
                        },
                    },
                    Array.Empty<string>()
                },
            }
    );
});

var jwtKey =
    builder.Configuration["Jwt:Key"] ?? throw new
InvalidOperationException("JWT Key not found");

builder
    .Services.AddAuthentication(JwtBearerDefaults.AuthenticationScheme)
    .AddJwtBearer(options =>
    {
        options.TokenValidationParameters = new
TokenValidationParameters
        {
            ValidateIssuer = true,
            ValidateAudience = true,
            ValidateLifetime = true,
            ValidateIssuerSigningKey = true,
            ValidIssuer = builder.Configuration["Jwt:Issuer"],
            ValidAudience = builder.Configuration["Jwt:Audience"],
            IssuerSigningKey = new
SymmetricSecurityKey(Encoding.UTF8.GetBytes(jwtKey)),
        };

        options.Events = new JwtBearerEvents
        {
            OnMessageReceived = context =>
            {
                var accessToken =
context.Request.Query["access_token"];
                var path = context.HttpContext.Request.Path;

                if (!string.IsNullOrEmpty(accessToken) &&
path.StartsWithSegments("/userHub"))
                {
                    context.Token = accessToken;
                }
                return Task.CompletedTask;
            },
        };
    });

builder.Services.AddCors(options =>
{
    options.AddDefaultPolicy(builder =>
    {
```

```
        builder
            .WithOrigins("http://localhost:5173")
            .AllowAnyMethod()
            .AllowAnyHeader()
            .AllowCredentials()
            .WithExposedHeaders("Authorization");
    });
});

builder.Services.AddDbContext<ApplicationDBContext>(options =>
{

options.UseSqlServer(builder.Configuration.GetConnectionString("Default
Connection"));
});

builder.Services.AddScoped<api.Services.CourseProvisioningService>();
builder.Services.AddSingleton<api.Services.CodeLanguageNeuralNet>();

builder.Services.AddSignalR(options =>
{
    options.EnableDetailedErrors = true;
    options.HandshakeTimeout = TimeSpan.FromSeconds(15);
});

api.Helpers.EncryptionHelper.Initialize(builder.Configuration);

var app = builder.Build();

api.Helpers.CodeLanguageDetector.UseNeuralClassifier(

app.Services.GetRequiredService<api.Services.CodeLanguageNeuralNet>()
);

if (app.Environment.IsDevelopment())
{
    app.UseSwagger();
    app.UseSwaggerUI();
}

app.UseHttpsRedirection();
app.UseStaticFiles();
app.UseRouting();

app.UseCors();

app.UseAuthentication();
app.UseAuthorization();

app.UseEndpoints(endpoints =>
{
    endpoints.MapControllers();
    endpoints.MapHub<UserHub>("/userHub");
```

```
});

using (var scope = app.Services.CreateScope())
{
    var services = scope.ServiceProvider;
    try
    {
        var context =
services.GetRequiredService<ApplicationDBContext>();
        context.Database.Migrate();

        var provisioning =
services.GetRequiredService<api.Services.CourseProvisioningService>();
        await provisioning.EnsureTeacherCoursesProvisionedAsync(3);
        await api.Services.TestDataSeeder.SeedAsync(context);
    }
    catch (Exception ex)
    {
        var logger = services.GetRequiredService<ILogger<Program>>();
        logger.LogError(ex, "An error occurred while migrating the
database.");
    }
}

app.Run();
```

ДОДАТОК Б

Лістинг коду GroupChatMessage.jsx

```
import { useState } from 'react';
import { useMutation, useQueryClient } from '@tanstack/react-query';
import hljs from 'highlight.js/lib/core';
import javascript from 'highlight.js/lib/languages/javascript';
import typescript from 'highlight.js/lib/languages/typescript';
import python from 'highlight.js/lib/languages/python';
import csharp from 'highlight.js/lib/languages/csharp';
import cpp from 'highlight.js/lib/languages/cpp';
import xml from 'highlight.js/lib/languages/xml';
import css from 'highlight.js/lib/languages/css';
import sql from 'highlight.js/lib/languages/sql';
import json from 'highlight.js/lib/languages/json';
import java from 'highlight.js/lib/languages/java';
import bash from 'highlight.js/lib/languages/bash';
import go from 'highlight.js/lib/languages/go';
import rust from 'highlight.js/lib/languages/rust';
import { IconCopy, IconFile, IconDownload, IconCheck, IconArrowBackUp }
from '@tabler/icons-react';
import { API_BASE_URL } from '../../config/api';
import { courseGroupChatsApi } from '../../services/api';
import { formatFileSize, isImageFile, parseCodeFence } from
'./codeUtils';

const MentionText = ({ text }) => {
  if (!text) return null;
  const parts = text.split(/(@[\w\u0400-\u04FF]+(?:\s+[\w\u0400-
\u04FF]+)?)/g);
  return parts.map((part, i) =>
    part.startsWith('@') ? (
      <span key={i} className="gc-mention">{part}</span>
    ) : (
      <span key={i}>{part}</span>
    )
  );
};
import 'highlight.js/styles/github-dark.css';

hljs.registerLanguage('javascript', javascript);
hljs.registerLanguage('typescript', typescript);
hljs.registerLanguage('tsx', typescript);
hljs.registerLanguage('jsx', javascript);
hljs.registerLanguage('python', python);
hljs.registerLanguage('csharp', csharp);
hljs.registerLanguage('cpp', cpp);
hljs.registerLanguage('c', cpp);
hljs.registerLanguage('html', xml);
hljs.registerLanguage('xml', xml);
hljs.registerLanguage('css', css);
hljs.registerLanguage('sql', sql);
```

```
hljs.registerLanguage('json', json);
hljs.registerLanguage('java', java);
hljs.registerLanguage('bash', bash);
hljs.registerLanguage('go', go);
hljs.registerLanguage('rust', rust);

const roleLabel = (role) => {
  if (role === 'Teacher' || role === 'Admin') return 'Викладач';
  if (role === 'Student') return 'Студент';
  return 'Учасник';
};

const CodeBlock = ({ code, language, fileName }) => {
  const [copied, setCopied] = useState(false);
  const lang = language || 'plaintext';
  let highlighted = code;
  try {
    if (hljs.getLanguage(lang)) {
      highlighted = hljs.highlight(code, { language: lang }).value;
    } else {
      highlighted = hljs.highlightAuto(code).value;
    }
  } catch {
    highlighted = code.replace(/</g, '&lt;').replace(/>/g, '&gt;');
  }

  const copy = () => {
    navigator.clipboard.writeText(code);
    setCopied(true);
    setTimeout(() => setCopied(false), 2000);
  };

  return (
    <div className="gc-code-block">
      <div className="gc-code-block__head">
        <span>{fileName || `${lang}${lang === 'typescript' ? '
DataViewer.tsx' : ''}`}</span>
        <button type="button" onClick={copy} className="gc-code-
block__copy">
          {copied ? <IconCheck size={14} /> : <IconCopy size={14} />}
          {copied ? 'Скопійовано' : 'Копіювати'}
        </button>
      </div>
      <pre className="gc-code-block__pre">
        <code className={`hljs language-${lang}`}
dangerouslySetInnerHTML={{ __html: highlighted }} />
      </pre>
    </div>
  );
};

const PollBlock = ({ poll, chatId }) => {
  const queryClient = useQueryClient();
```

```
  const [selected, setSelected] = useState(poll.myOptionIds || []);

  const voteMutation = useMutation({
    mutationFn: (optionIds) => courseGroupChatsApi.votePoll(poll.id,
optionIds),
    onSuccess: () => {
      queryClient.invalidateQueries(['group-chat-messages', chatId]);
    },
  });

  const totalVotes = poll.options.reduce((s, o) => s + o.votes, 0) ||
1;

  const toggleOption = (optionId) => {
    let next;
    if (poll.allowMultiple) {
      next = selected.includes(optionId)
        ? selected.filter((id) => id !== optionId)
        : [...selected, optionId];
    } else {
      next = [optionId];
    }
    setSelected(next);
    voteMutation.mutate(next);
  };

  return (
    <div className="gc-poll">
      <div className="gc-poll__question">📊 {poll.question}</div>
      <div className="gc-poll__options">
        {poll.options.map((opt) => {
          const pct = Math.round((opt.votes / totalVotes) * 100);
          const isSelected = selected.includes(opt.id);
          return (
            <button
              key={opt.id}
              type="button"
              className={`gc-poll__option${isSelected ? ' gc-
poll__option--selected' : ''}`}
              onClick={() => toggleOption(opt.id)}
            >
              <div className="gc-poll__option-bar" style={{ width:
`${pct}%` }} />
              <span className="gc-poll__option-text">{opt.text}</span>
              <span className="gc-poll__option-votes">{opt.votes}
({pct}%)</span>
            </button>
          );
        })}
      </div>
      <div className="gc-poll__foot">{poll.totalVoters} учасників
проголосували</div>
    </div>
```

```
  );
};

const FileAttachment = ({ msg }) => {
  const url = msg.fileUrl?.startsWith('http') ? msg.fileUrl :
`${API_BASE_URL}${msg.fileUrl}`;

  if (isImageFile(msg.fileType, msg.fileName)) {
    return (
      <a href={url} target="_blank" rel="noopener noreferrer"
className="gc-msg__image-wrap">
        <img src={url} alt={msg.fileName} className="gc-msg__image" />
      </a>
    );
  }

  return (
    <a href={url} download={msg.fileName} className="gc-msg__file-
card">
      <IconFile size={20} />
      <div>
        <strong>{msg.fileName}</strong>
        <span>{formatFileSize(msg.fileSize)}</span>
      </div>
      <IconDownload size={16} />
    </a>
  );
};

const ReplyQuote = ({ replyTo }) => (
  <div className="gc-msg__reply-quote">
    <strong>{replyTo.senderName}</strong>
    <span>{replyTo.messageType === 'file' ? replyTo.fileName :
replyTo.text}</span>
  </div>
);

const GroupChatMessage = ({
  msg,
  chatId,
  avatarColor,
  getInitials,
  formatMessageTime,
  onContextMenu,
  onReply,
  currentUserId,
  isChatAdmin,
}) => {
  const canDelete = String(msg.senderId) === String(currentUserId) ||
isChatAdmin;
  const codeFence = msg.messageType !== 'file' && msg.text ?
parseCodeFence(msg.text) : null;
  const showAsCode = msg.messageType === 'code' || !!codeFence;
```

```
  const codeText = codeFence ? codeFence.code : msg.text;
  const codeLang = codeFence ? codeFence.language : msg.codeLanguage;

  return (
    <div
      className="gc-msg"
      onContextMenu={(e) => {
        e.preventDefault();
        onContextMenu?.(e, msg, canDelete);
      }}
    >
      <div className="gc-msg__avatar" style={{ background:
avatarColor(msg.senderName) }}>
        {getInitials(msg.senderName)}
      </div>
      <div className="gc-msg__body">
        <div className="gc-msg__head">
          <strong>{msg.senderName}</strong>
          <span className={`gc-msg__role gc-msg__role--
${(msg.senderRole || '').toLowerCase()}`}>
            {roleLabel(msg.senderRole)}
          </span>
          <span className="gc-
msg__time">{formatMessageTime(msg.sentAt)}</span>
          <button type="button" className="gc-msg__reply-btn"
onClick={() => onReply?.(msg)} title="Відповісти">
            <IconArrowBackUp size={14} />
          </button>
        </div>

        {msg.replyTo && <ReplyQuote replyTo={msg.replyTo} />}

        {msg.messageType === 'poll' && msg.poll ? (
          <PollBlock poll={msg.poll} chatId={chatId} />
        ) : (
          <>
            {showAsCode && (
              <CodeBlock code={codeText} language={codeLang} />
            )}
            {!showAsCode && msg.messageType !== 'file' && msg.text && (
              <div className="gc-msg__text"><MentionText
text={msg.text} /></div>
            )}
            {msg.messageType === 'file' && (
              <>
                {msg.text && <div className="gc-msg__text"><MentionText
text={msg.text} /></div>}
                {msg.fileName && <FileAttachment msg={msg} />}
              </>
            )}
          </>
        )}
      </div>
```

```
      </div>
    );
};

export default GroupChatMessage;
```