Міністерство освіти і науки України
Луцький національний технічний університет
Факультет комп'ютерних та інформаційних технологій
Кафедра інженерії програмного забезпечення

КВАЛІФІКАЦІЙНА РОБОТА
ЗА СТУПЕНЕМ ВИЩОЇ ОСВІТИ «МАГІСТР»

РОЗРОБКА ТА ДОСЛІДЖЕННЯ КРОСПЛАТФОРМНОГО МОБІЛЬНОГО ЗАСТОСУНКУ ДЛЯ ВИВЧЕННЯ ІНОЗЕМНИХ МОВ ІЗ ВИКОРИСТАННЯМ ШТУЧНОГО ІНТЕЛЕКТУ

DEVELOPMENT AND RESEARCH OF A CROSS-PLATFORM MOBILE APPLICATION FOR LEARNING FOREIGN LANGUAGE USING ARTIFICIAL INTELLIGENCE

спеціальність 121 – Інженерія програмного забезпечення

освітня програма «Інженерія програмного забезпечення»

Виконав:
здобувач вищої освіти групи ІПЗм-22
Антонюк П. О.

Керівник:
к.т.н., доцент
Бойко Л. С.

Кваліфікаційну роботу
допущено до захисту

«___» _____ 20__ р.
Гарант освітньої програми:
к.т.н., доцент Суринович О. М.

_____

Луцьк – 2025

ЛУЦЬКИЙ НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ

Факультет *комп'ютерних та інформаційних технологій*
Кафедра *інженерії програмного забезпечення*
Ступінь вищої освіти *_магістр*
Галузь знань: *12 «Інформаційні технології»*
Спеціальність: *121 «Інженерія програмного забезпечення»*
Освітня програма: «*Інженерія програмного забезпечення»*

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____
«___» _____ 202__ р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ ЗДОБУВАЧА
ДРУГОГО (МАГІСТЕРСЬКОГО) РІВНЯ ВИЩОЇ ОСВІТИ

Антонюку Павлу Олександровичу

1. Тема кваліфікаційної роботи: *Розробка та дослідження кросплатформного мобільного застосунку для вивчення іноземних мов із використанням штучного інтелекту*
Керівник роботи: *Бойко Лев Степанович, к.т.н., доцент.*
_
затверджені наказом закладу вищої освіти від *«29» березня 2025 року*
_
№ 190/01-02
2. Строк подання здобувачем вищої освіти кваліфікаційної роботи: *04 грудня 2025 р.*
3. Вихідні дані до роботи: *кросплатформне середовище розробки мобільних застосунків, backend-фреймворк, система керування базами даних, інтеграція з API великих мовних моделей*
4. Зміст розрахунково-пояснювальної записки: *аналіз сучасного стану мобільних освітніх технологій та архітектурних підходів, огляд та порівняння існуючих аналогів, дослідження можливостей інтеграції штучного інтелекту, обґрунтування вибору архітектурних патернів та технологічного стеку, опис функціональних можливостей системи, описання структури системи та взаємодії компонентів, результати тестування продуктивності та надійності, економічне обґрунтування рішення.*
_
5. Перелік графічного матеріалу: *19 рисунків, 20 таблиць*

6. Консультанти розділів роботи:

Розділ	Прізвище, ініціали та посада консультанта	Підпис	
		завдання видав	завдання прийняв
Аналіз предметної області	*Бойко Л. С.*		
Специфікація вимог до розробленої системи	*Бойко Л. С.*		
Розробка об'єкта проектування	*Бойко Л.С.*		
Нормоконтроль	*Повстяна Ю.С.*		
Гарант ОП	*Андрущак І.Є.*		
Показник запозичень тексту	*%*		
Академічна доброчесність	*Бойко Л.С.*		

7. Дата видачі завдання: *«02» квітня 2025р*

8. Календарний план:

№№	Назва етапів кваліфікаційної роботи магістра	Строк виконання етапів роботи	Примітка
1	Провести огляд літературних джерел по темі кваліфікаційної роботи	02.05.2025	
2	Провести аналіз загальної проблеми і вибір напрямків дослідження	24.09.2025	
3	Розробити функціональну схему роботи програмного продукту	01.11.2025	
4	Описати засоби розробки об'єкта проектування	19.11.2025	
5	Практична реалізація об'єкта проектування	26.11.2025	
6	Розробити методику для проведення експерименту	05.11.2025	
7	Провести аналіз результатів експерименту	15.11.2025	
8	Здача чистового варіанту кваліфікаційної роботи на кафедру	04.12.2025	

Здобувач вищої освіти _____ Павло Антонюк

Керівник кваліфікаційної роботи _____ Лев Бойко

АНОТАЦІЯ

Антонюк П. О. Розробка та дослідження кросплатформного мобільного застосунку для вивчення іноземних мов із використанням ШІ. Кваліфікаційна робота магістра спеціальності 121 «Інженерія програмного забезпечення». Луцький національний технічний університет, 2025.

У кваліфікаційній роботі розглянуто розробку кросплатформного мобільного застосунку для вивчення іноземних мов із використанням штучного інтелекту. Спочатку здійснено огляд літератури, де аналізуються сучасні мобільні технології та існуючі рішення на ринку, що дозволило визначити недоліки традиційних систем і можливості для вдосконалення через інтеграцію штучного інтелекту. Далі, на основі результатів досліджень, були сформульовані вимоги до системи, зокрема щодо використання великих мовних моделей для персоналізації навчального процесу. На основі цих вимог була спроектована архітектура системи, що включає клієнтську частину на React Native або Flutter, серверну частину на Spring Boot і базу даних на PostgreSQL. У роботі було розроблено прототип застосунку, який інтегрує технології ШІ для генерування персоналізованих навчальних завдань, а також протестовано його продуктивність за високих навантажень, що підтвердило ефективність запропонованого рішення. Також було проведено порівняльний аналіз із існуючими платформами, що показав переваги розробленої системи в плані персоналізації контенту та оптимізації витрат. Завершальним етапом роботи став економічний аналіз, який продемонстрував можливість забезпечити доступну ціну підписки при збереженні високої маржі.

Ключові слова: мобільний застосунок, штучний інтелект, персоналізоване навчання, кросплатформна розробка, мовні моделі.

ABSTRACT

Antoniuk P. O. Development and Research of a Cross-Platform Mobile Application for Learning Foreign Language Using Artificial Intelligence. Master's Thesis in Software Engineering, Lutsk National Technical University, 2025.

This master's thesis explores the development and evaluation of a cross-platform mobile application for foreign language learning enhanced by artificial intelligence. The research examines the theoretical foundations of personalized learning and analyzes modern mobile development technologies in order to identify limitations of traditional language-learning systems and opportunities for improvement through AI integration.

The study includes a literature review of existing mobile platforms and educational solutions, which revealed insufficient adaptability and personalization in conventional approaches. Based on these findings, system requirements were defined with a focus on the application of large language models to generate personalized learning content and tasks tailored to individual user progress. A modular system architecture was designed, consisting of a cross-platform client application developed using React Native or Flutter, a server-side component implemented with Spring Boot, and a PostgreSQL relational database.

A functional prototype of the application was implemented and integrated with AI-based modules for adaptive task generation. Performance and load testing confirmed the system's stability and efficiency under increased user load. A comparative analysis with existing platforms demonstrated advantages in personalization and cost optimization. An economic analysis showed the feasibility of an affordable subscription model with sustainable profitability.

Keywords: mobile application, artificial intelligence, personalized learning, cross-platform development, language models.

ЗМІСТ

ВСТУП

Актуальність теми дослідження. У сучасному глобалізованому світі володіння іноземними мовами стало однією з ключових компетенцій, що визначає професійний успіх та можливості особистісного розвитку. За даними Європейської Комісії [1], понад 80 % роботодавців вважають знання іноземних мов критично важливим для працевлаштування, а володіння двома і більше мовами підвищує конкурентоспроможність фахівця на міжнародному ринку праці на 40-60 %.

Традиційні методи вивчення іноземних мов, такі як курси у навчальних закладах або заняття з репетиторами, часто характеризуються високою вартістю (від 20 до 100 USD за годину), негнучким графіком та обмеженою адаптацією до індивідуальних потреб учнів. Водночас, ринок мобільних освітніх технологій (EdTech) демонструє стрімке зростання: за прогнозами аналітичної компанії HolonIQ [2], глобальний ринок цифрової освіти досягне 404 млрд USD до 2025 року, причому сегмент вивчення мов становить близько 28 % цього ринку.

Революційний прорив у галузі штучного інтелекту, зокрема поява великих мовних моделей (Large Language Models, LLM) четвертого покоління, таких як GPT, Google Gemini, Claude та національних розробок на кшталт української моделі LAPA, відкриває принципово нові можливості для персоналізації навчального процесу. Ці моделі здатні генерувати адаптивний навчальний контент у реальному часі, аналізувати помилки учнів з глибоким розумінням контексту, надавати індивідуалізований зворотний зв'язок та підлаштовуватися під темп і стиль навчання кожного користувача.

Однак ефективне впровадження технологій штучного інтелекту в освітні мобільні застосунки вимагає глибокого розуміння архітектурних принципів

побудови таких систем. Неправильний вибір архітектурного рішення може призвести до високих операційних витрат на виклики API великих мовних моделей (від 0,0015 до 0,06 USD за 1000 токенів), низької продуктивності системи (затримки відповіді понад 3 секунд знижують утримання користувачів на 40 %), проблем з масштабованістю при зростанні аудиторії та складнощів у підтримці та розвитку продукту.

Існуючі дослідження в галузі архітектури освітніх застосунків зазвичай фокусуються або на традиційних підходах без інтеграції ШІ, або розглядають інтеграцію LLM поверхово, не приділяючи достатньої уваги специфічним викликам, таким як управління контекстом діалогу, оптимізація витрат на API, забезпечення консистентності згенерованого контенту, кешування результатів та балансування навантаження між різними моделями. Крім того, недостатньо висвітлені питання кросплатформної розробки з єдиною кодовою базою для адаптивних систем навчання та особливості інтеграції множинних LLM-провайдерів для забезпечення резервування та оптимізації якості контенту.

У контексті України актуальність дослідження посилюється необхідністю цифровізації освіти, підвищення доступності якісних освітніх ресурсів для населення, особливо в умовах воєнного стану, коли традиційні форми навчання часто ускладнені або неможливі, а також розвитком власних технологій штучного інтелекту, зокрема української мовної моделі LAPA, що відкриває можливості для створення локалізованих освітніх рішень.

Таким чином, дослідження архітектурних рішень для побудови мобільних застосунків адаптивного навчання іноземних мов із використанням штучного інтелекту є актуальним науковим завданням, що має як теоретичне, так і практичне значення для розвитку галузі освітніх технологій.

Окремо варто зазначити, що для даного дослідження було використано низку джерел, опублікованих до 2019 року. Залучення таких праць є

обґрунтованим, оскільки вони містять фундаментальні результати у сфері когнітивної психології, педагогіки, теорії навчання та програмної інженерії, на яких базуються сучасні підходи до адаптивного навчання та проєктування програмних систем. Незважаючи на давнішу дату публікації, ці роботи залишаються актуальними, широко цитуються в наукових дослідженнях останніх років та формують теоретичний фундамент, необхідний для розуміння архітектурних, освітніх і технологічних рішень, реалізованих у проєкті.

Мета і завдання дослідження – розробка та дослідження оптимальної архітектурної моделі кросплатформного мобільного застосунку для адаптивного навчання іноземних мов, що інтегрує сучасні фронтенд і бекенд-технології з великими мовними моделями штучного інтелекту, забезпечуючи персоналізований навчальний досвід, високу продуктивність системи та економічну ефективність розробки.

Для досягнення поставленої мети в роботі передбачено виконання кількох взаємопов'язаних завдань. Насамперед потрібно проаналізувати сучасні архітектурні підходи до розробки мобільних застосунків і визначити, які з них є найбільш придатними для освітніх систем, що використовують штучний інтелект. Важливо виявити як їхні переваги, так і потенційні обмеження, особливо у випадках, коли застосунок має працювати зі складними динамічними даними та забезпечувати адаптивність навчання.

Окрему увагу приділено інтеграції великих мовних моделей – таких як GPT, Google Gemini, Claude – у мобільне середовище. Тут необхідно дослідити оптимальні стратегії роботи з API моделей, особливості керування контекстом діалогу та способи оптимізації витрат, пов'язаних із генерацією контенту.

Наступним кроком є розробка архітектурної моделі системи адаптивного навчання. Вона має охоплювати клієнтську частину (мобільний застосунок), серверну логіку, підсистему інтеграції з LLM, модулі роботи з базою даних, а

також інфраструктурні компоненти, які забезпечують надійність та масштабованість системи. Після проєктування архітектури необхідно створити її робочий прототип. Для цього планується використовувати сучасний технологічний стек: NativeScript/Vue.js для мобільного фронтенду, Spring Boot для серверної частини, PostgreSQL для збереження даних та Vertex AI Gemini як основний інструмент генерації навчальних матеріалів.

Після реалізації прототипу важливо провести комплексне тестування. Воно включатиме модульну перевірку окремих компонентів, тестування інтеграцій між модулями, оцінку продуктивності системи під різним навантаженням і, окремо, аналіз якості згенерованого навчального контенту.

Ще одним етапом є порівняльний аналіз створеної системи з існуючими рішеннями на ринку – такими як Duolingo, Babbel чи Memrise. Це дозволить визначити рівень функціональності, адаптивності та продуктивності запропонованої моделі, а також оцінити її економічну ефективність. Завершальним кроком стане формування рекомендацій щодо впровадження і подальшого масштабування системи, включаючи стратегії оптимізації витрат, забезпечення безпеки та конфіденційності користувацьких даних.

Об’єкт та предмет дослідження є процес адаптивного навчання іноземних мов у мобільних застосунках з використанням технологій штучного інтелекту.

Предметом дослідження є архітектурні рішення, методи та технології побудови кросплатформних мобільних застосунків адаптивного навчання іноземних мов з інтеграцією великих мовних моделей штучного інтелекту, включаючи структуру компонентів системи, взаємодію між ними, стратегії управління даними та забезпечення продуктивності.

У процесі дослідження були застосовані як теоретичні, так і практичні методи, що дозволили всебічно опрацювати тему та отримати обґрунтовані результати. На теоретичному рівні основою став системний аналіз, за

допомогою якого вдалося дослідити сучасні архітектурні патерни мобільних застосунків і зрозуміти їх сильні та слабкі сторони у контексті інтеграції штучного інтелекту. Значну роль також відіграв порівняльний аналіз, що дав можливість оцінити різні технологічні стеки, фреймворки та великі мовні моделі, а також окреслити відмінності між підходами до проєктування складних освітніх систем.

Для структуризації предметної області та побудови логічної архітектури застосунку використовувався метод декомпозиції, завдяки якому складну систему було розділено на функціональні модулі з чітко окресленими зонами відповідальності. Доповненням до нього став метод абстрагування, що допоміг зосередитись на ключових властивостях системи та сформувати концептуальні моделі на різних рівнях деталізації. У моделюванні та описі сутностей, а також їхніх взаємозв'язків застосовувався об'єктно-орієнтований аналіз і проєктування. Окремим напрямом теоретичної роботи був аналіз вимог, який дозволив формалізувати як функціональні, так і нефункціональні характеристики системи відповідно до потреб користувачів та цілей проєкту.

Практична частина дослідження спиралася на інструменти та методики, які допомагають перевіряти й вдосконалювати архітектурні рішення на реальних прикладах. Одним із ключових інструментів стало прототипування, що дозволило швидко оцінити життєздатність обраних технічних підходів. Розробка здійснювалася ітеративно, завдяки чому функціональність системи нарощувалася поступово, із можливістю швидко реагувати на результати тестування чи зміну вимог.

Для оцінки якості роботи окремих компонентів використовувалося модульне тестування з такими інструментами, як JUnit і Mockito на бекенді та Jest на фронтенді. Перевірка коректної взаємодії між частинами системи здійснювалася в рамках інтеграційного тестування, що включало тестування

REST API, роботу з базою даних і взаємодію з AI-сервісами. Оцінювання продуктивності проводилося під різними рівнями навантаження за допомогою Apache JMeter, що допомогло визначити вузькі місця та межі пропускної здатності системи.

Для підтвердження прийнятих архітектурних рішень та оцінки якості згенерованого штучним інтелектом навчального контенту застосовувалися експертні методи. Отримані результати додатково опрацьовувалися статистичним аналізом, який дав змогу розрахувати основні продуктивні метрики та оцінити їхню значущість. Завершальним етапом став бенчмаркінг, у межах якого розроблену систему було порівняно з популярними аналогами, що дозволило визначити її конкурентні переваги та можливі напрями вдосконалення.

У результаті виконання кваліфікаційної роботи магістра було отримано низку важливих наукових результатів, які поглиблюють підходи до розробки сучасних мобільних освітніх систем із використанням штучного інтелекту. Одним із ключових здобутків стала розробка гібридної архітектурної моделі кросплатформного мобільного застосунку, яка поєднує клієнт-серверну логіку з елементами мікросервісного підходу та спеціалізованим шаром інтеграції зі штучним інтелектом. Така структура забезпечує високу модульність і масштабованість системи, а також дозволяє динамічно обирати оптимальну велику мовну модель для різних типів навчальних завдань.

Під час роботи також було вдосконалено методику інтеграції великих мовних моделей у мобільні освітні продукти. Запропонований механізм автоматичного формування JSON Schema з типізованих структур даних, а також контекстно-чутливе генерування промптів із урахуванням мови навчання, походження студента та рівня володіння мовою, дозволили значно підвищити надійність генерації контенту. Додаткове впровадження

інтелектуального механізму повторних запитів із експоненційною затримкою та валідацією структури відповіді забезпечило стабільність роботи системи на рівні, що суттєво перевищує традиційні підходи.

Окремим результатом є створення алгоритму адаптивної генерації навчальних завдань, який враховує індивідуальну динаміку навчання користувача. До уваги береться його історія помилок, часові особливості використання застосунку та когнітивні характеристики засвоєння матеріалу. Завдяки цьому вдалося досягти високого рівня персоналізації контенту, що помітно перевищує ефективність статичних курсів.

Удосконалення торкнулися й архітектури кросплатформної розробки. Запропонована monorepo-структура з виділеним спільним шаром для бізнес-логіки, API-сервісів, управління станом і типізації даних суттєво зменшує дублювання коду та забезпечує консистентність реалізації між різними платформами. Це дозволяє підтримувати єдину логіку системи, одночасно оптимізуючи процеси розробки.

Ще одним важливим внеском стало впровадження декларативного підходу до конфігурування навчального контенту на основі YAML-специфікацій. Такий підхід робить можливим створення та модифікацію структури курсів без прямого втручання в програмний код, що відкриває процес для методистів та викладачів і суттєво скорочує час розробки нових матеріалів.

Практичне значення отриманих результатів полягає у фактичній можливості застосування розробленої системи в реальних умовах сучасної освітньої практики. У межах роботи було створено повноцінний прототип кросплатформного мобільного застосунку для вивчення іноземних мов з інтеграцією Google Vertex AI Gemini, який демонструє працездатність

запропонованої архітектури та може бути використаний для пілотного впровадження в освітніх закладах.

Проведені розрахунки доводять і економічну доцільність запропонованого рішення. Модель підписки дає можливість утримувати вартість користування на рівні 4.99 USD на місяць із цільовою маржею близько 40 %, що робить продукт суттєво доступнішим порівняно з найпоширенішими аналогами на ринку, такими як Duolingo чи Babbel. Під час розробки було значно оптимізовано витрати на роботу з великими мовними моделями: впровадження багаторівневої системи кешування (пам'ять, Redis та PostgreSQL) дозволило суттєво скоротити кількість дорогих викликів до LLM, зберігши при цьому якість персоналізованого навчального контенту.

Висока продуктивність системи підтверджена результатами тестування: середній час відповіді API становить близько 280 мілісекунд, а 95-й перцентиль не перевищує критичного порогу в 500 мілісекунд. Система стабільно працює за умов одночасної активності сотні користувачів без помітної деградації сервісу. Важливим практичним результатом є також створення умов для розвитку національних освітніх технологій.

У роботі сформовано методичні рекомендації щодо масштабування системи від MVP до enterprise-рівня, включаючи горизонтальне масштабування в Kubernetes, load balancing через Nginx та асинхронну обробку подій через Kafka/RabbitMQ. Це мінімізує ризики розвитку продукту та забезпечує його стабільність при зростанні навантаження.

Результати дослідження мають практичне застосування для навчальних закладів, що впроваджують інноваційні методи вивчення іноземних мов, для EdTech-компаній, які шукають готові архітектурні моделі, та для науково-дослідних установ, що вивчають можливості AI в освіті. Розроблена модель

може слугувати референсною архітектурою для розробників програмного забезпечення при адаптації до конкретних проєктів.

Автором самостійно проведено детальний аналіз архітектурних підходів до розробки мобільних систем та можливостей інтеграції великих мовних моделей, на основі якого сформована архітектура програмного комплексу. Реалізовано працездатний прототип із серверною частиною на Spring Boot, клієнтською частиною на Vue.js/NativeScript та інтеграцією з Google Vertex AI Gemini. Система пройшла комплексне тестування (модульне, інтеграційне, навантажувальне) та порівняльний аналіз з аналогами. Науковий керівник здійснював консультування лише з питань методології та структурування матеріалу.

Апробація результатів дослідження. Основні положення та результати магістерської роботи апробовано на XII Міжнародній науково-практичній конференції «Main Trends in Science, Teaching and Modern Learning» (Гамбург, Німеччина, 18-21 листопада 2025 р.) [3] та XIII Міжнародній науково-практичній конференції «Innovative Directions for Improving Science, Research and Practice» (Краків, Польща, 25-28 листопада 2025 р.) [4]. Результати дослідження опубліковано у виданні «Студентський науковий вісник ЛНТУ» [5].

Робота складається зі вступу, трьох розділів, висновків, списку використаних джерел та додатків.

РОЗДІЛ 1
АНАЛІЗ ПРОБЛЕМАТИКИ ЗА ТЕМОЮ РОБОТИ ТА ПОСТАНОВКА ЗАВДАНЬ ДОСЛІДЖЕННЯ

1.1 Огляд і аналіз предметної області проблеми, результатів існуючих теоретичних та експериментальних досліджень

У XXI столітті мобільні технології стали ключовим драйвером цифрової трансформації суспільства, бізнесу та освіти. Розвиток мобільних екосистем, зростання обчислювальних можливостей смартфонів та доступність мобільного інтернету сформували нову парадигму створення і споживання інформаційних продуктів. Комплексні мобільні застосунки перестали бути допоміжними інструментами–сьогодні вони є основною платформою доступу користувачів до сервісів, контенту та навчання.

За даними GSMA Intelligence, у 2024 році кількість унікальних користувачів мобільного зв'язку перевищила 5,6 млрд, а кількість підключень до мобільних пристроїв сягнула 8,8 млрд USD [6]. У свою чергу, за прогнозами Statista, кількість користувачів смартфонів у 2025 році перевищить 6,3 млрд USD, а обсяг глобального ринку мобільних застосунків досягне 613 млрд USD [7]. Така динаміка підтверджує, що мобільні технології залишаються наймасовішою цифровою платформою у світі.

Ці зміни суттєво вплинули на вимоги до розробки мобільного ПЗ. Бізнес і освітні платформи прагнуть одночасної присутності в екосистемах iOS та Android, які разом охоплюють понад 99 % ринку мобільних операційних систем [8]. Традиційна нативна розробка вимагає підтримки окремих кодових баз на Swift/Kotlin, подвоєних витрат на тестування і розгортання, що підвищує загальну вартість володіння продуктом. У цих умовах особливої актуальності набувають кросплатформні технології, які дозволяють створювати мобільні

застосунки на єдиній кодовій базі та забезпечувати одночасну підтримку двох основних мобільних платформ.

Разом із цим освітня сфера демонструє стрімке зростання інтересу до персоналізованих мобільних рішень. Дослідження HolonIQ показують, що ринок EdTech у 2024 році перевищив 300 млрд USD, а одним із ключових напрямів розвитку стала адаптивна освіта, що базується на аналітиці поведінки користувачів та алгоритмах штучного інтелекту [9]. Поява потужних моделей штучного інтелекту, зокрема GPT та Gemini, створила нові можливості для прийняття рішень у реальному часі, автоматичного формування індивідуальних траєкторій навчання та підвищення ефективності освітнього процесу.

У цьому контексті розробка кросплатформних мобільних застосунків для адаптивного вивчення іноземних мов – актуальна і перспективна науково-практична задача. Поєднання кросплатформної інженерії, сучасних мобільних фреймворків, методів аналізу даних та генеративного ШІ дозволяє створювати масштабовані, доступні та інтелектуальні освітні продукти.

1.2 Сучасний стан ринку мобільних технологій та кросплатформної розробки

Мобільний сегмент продовжує залишатися домінуючим у глобальному технічному ландшафті. За даними App Annie та Data.ai, у 2023 році користувачі витратили понад 5 трильйонів годин у мобільних застосунках, а середньодобовий час використання смартфона в ряді країн перевищив 4 - 5 годин [10]. Мобільні платформи стали основним каналом цифрової взаємодії: понад 70 % інтернет-трафіку сьогодні припадає саме на мобільні пристрої [11].

Зростання мобільного ринку зумовлене поєднанням кількох ключових тенденцій, які сформували сприятливе середовище для розвитку мобільних

технологій. Важливу роль відіграло поступове зниження вартості смартфонів та широка доступність високошвидкісного мобільного інтернету – передусім мереж 4G і 5G. Це зробило сучасні мобільні пристрої доступними для значно більшої частини населення. Додатковим чинником стало стрімке поширення фінтех-рішень, що забезпечили користувачам зручні сервіси для платежів, банківських операцій і цифрової ідентифікації. Потужний вплив на розвиток мобільних екосистем має і прогрес хмарних платформ, таких як Firebase, AWS чи Supabase, які спростили створення масштабованих застосунків та зменшили технічний поріг для розробників. Окремо варто відзначити інтеграцію штучного інтелекту в повсякденні мобільні сценарії, що відкрила можливості для персоналізації, автоматизації та підвищення якості користувацького досвіду, стимулюючи подальше зростання ринку.

Популярність мобільних платформ сьогодні визначається насамперед домінуванням Android та iOS, які фактично формують двополярну структуру глобального ринку мобільних операційних систем. За даними StatCounter за 2024 рік, Android утримує 71,4 % світового ринку, тоді як частка iOS становить 27,9 % [12]. Такий розподіл вказує не лише на значну перевагу Android у кількісному вимірі, а й на специфіку їхніх аудиторій: Android залишається основною платформою для країн, що розвиваються, де ключову роль відіграють доступні за ціною пристрої, тоді як iOS демонструє стабільні позиції у сегменті преміальних смартфонів та ринках з вищою купівельною спроможністю.

Подібна структура ринку суттєво впливає на підхід до розробки мобільних застосунків, зокрема розширюючи потребу в кросплатформних рішеннях, які дозволяють ефективно охоплювати обидві екосистеми без значних витрат на підтримку окремих кодових баз. Розподіл часток між Android та iOS у глобальному масштабі подано на рисунку 1.1, що наочно ілюструє тенденції та взаємну динаміку цих платформ.

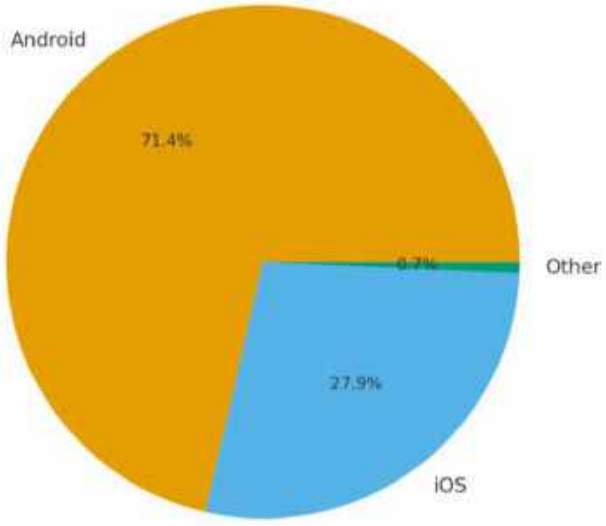

Рисунок 1.1 – Розподіл частки мобільних операційних систем у світі, 2024 р.

Це означає, що для охоплення цільової аудиторії компаніям потрібно підтримувати обидві платформи. Тому вибір технологічного підходу до розробки (нативний, гібридний, кросплатформний) безпосередньо впливає на бюджет, терміни та масштабованість продукту.

Перехід до кросплатформної розробки зумовлений низкою чинників, які активно підкреслюють аналітики Forrester та Gartner [13, 14]. Однією з ключових причин є можливість працювати з єдиною кодовою базою, що дозволяє суттєво знизити витрати на розробку – у середньому на 30-50 % – та скоротити час виходу продукту на ринок. Крім економічної ефективності, важливим аргументом є швидше оновлення застосунків і впровадження нових функцій, оскільки зміни синхронно застосовуються для всіх платформ.

Кросплатформний підхід також сприяє підтримці єдиного дизайну та узгодженого користувацького досвіду на різних пристроях, що спрощує забезпечення візуальної цілісності продукту. Можливість швидкого створення

прототипів та ефективного масштабування MVP робить такі технології привабливими для стартапів і компаній, які прагнуть швидко тестувати бізнес-гіпотези. Не менш важливим є й те, що сучасні кросплатформні фреймворки підтримують популярні мови програмування, зокрема TypeScript, Dart і JavaScript, та мають великі екосистеми, що полегшує пошук бібліотек і розширень.

Серед найпоширеніших рішень у цій сфері особливе місце займають Flutter, React Native та NativeScript. Кожен із цих фреймворків має власні архітектурні підходи, окремі механізми рендерингу та специфічні способи інтеграції з нативними API, що формує унікальні переваги й визначає сферу їхнього оптимального застосування.

Розробка мобільних застосунків сьогодні представлена кількома основними архітектурними підходами, що відрізняються принципами формування інтерфейсу, способом доступу до нативних можливостей пристрою та моделлю виконання коду. Вибір архітектури значною мірою впливає на продуктивність застосунку, вартість розробки, час на підтримку і можливості масштабування.

Нативна розробка передбачає створення окремих застосунків для Android та iOS із використанням офіційних інструментів і мов програмування, притаманних кожній платформі. Для екосистеми Android це насамперед Kotlin і Java, Android SDK та набір бібліотек Jetpack, тоді як розробка під iOS традиційно базується на Swift, інструментах SwiftUI або UIKit та середовищі XCode. Такий підхід забезпечує найглибшу інтеграцію з платформою та дає можливість максимально повно використовувати її можливості.

Головною перевагою нативного підходу є найвища продуктивність, оскільки код виконується без додаткових проміжних прошарків, безпосередньо у середовищі операційної системи. Це дозволяє застосункам ефективно

працювати навіть у сценаріях з підвищеним навантаженням або вимогами до обробки великих обсягів даних у реальному часі. Крім того, нативна розробка забезпечує повний доступ до всіх можливостей пристрою, включаючи найновіші апаратні функції та API. Підтримка технологій на кшталт ARKit, ML Kit чи Secure Enclave стає доступною розробникам одразу після їхнього офіційного релізу, що дозволяє створювати інноваційні рішення без затримок. Ще однією суттєвою перевагою є природна відповідність застосунків офіційним UI/UX-гайдам платформ, що позитивно впливає на користувацьке сприйняття.

Водночас нативна розробка має і свої обмеження. Найсуттєвішим з них є подвоєння роботи, адже для підтримки Android і iOS необхідно розробляти та обслуговувати дві незалежні кодові бази, залучаючи команди різної спеціалізації. Це підвищує витрати на розробку, ускладнює процес тестування та подовжує загальний цикл реалізації нових функцій. Нативні підходи менш придатні до швидкого прототипування, що може бути критичним на ранніх етапах проєкту або у стартап-середовищі.

Як зазначають аналітики Gartner, повністю нативний підхід є найбільш обґрунтованим у тих випадках, коли вимоги до продуктивності та доступу до апаратних можливостей є ключовими, або коли продукт глибоко інтегрується з екосистемними сервісами Apple чи Google [15].

Гібридні архітектури, представлені такими рішеннями, як Apache Cordova та ранні версії Ionic, передбачають використання HTML, CSS і JavaScript для створення інтерфейсу, який виконується всередині спеціального контейнера WebView. Для взаємодії з можливостями пристрою в таких системах застосовуються плагіни, що забезпечують доступ до нативних API, компенсуючи обмеження браузерного середовища. У свій час цей підхід швидко набув поширення, оскільки дозволяв веб-розробникам без значних

додаткових витрат адаптувати свої навички під мобільні платформи та створювати застосунки у звичному технологічному стеку.

Серед переваг гібридних рішень можна відзначити низький поріг входу, можливість швидкого початку розробки та легке залучення фахівців, що працюють із веб-технологіями. Крім того, такий підхід забезпечував певну уніфікацію компонентів між веб-версіями продуктів і їх мобільними адаптаціями, що зменшувало витрати на підтримку інтерфейсів.

Однак із розвитком мобільних платформ і зростанням вимог до продуктивності недоліки гібридних застосунків стали дедалі очевиднішими. Головною проблемою залишається обмежена продуктивність, особливо у випадках складної анімації, високонавантажених графічних сцен чи інтерактивних елементів. Значна залежність від WebView спричиняє непередбачувані поведінкові відмінності між пристроями та версіями ОС, а доступ до нативних API часто відстає від можливостей, які пропонують Android чи iOS після оновлень. Усе це обмежує застосовність гібридного підходу в сучасних продуктивних мобільних системах.

Через згадані обмеження гібридні архітектури сьогодні переважно розглядаються як технологія попереднього покоління і поступилися місцем більш ефективним кросплатформним фреймворкам, що пропонують кращу продуктивність, ширший доступ до нативних можливостей і сучасніші парадигми розробки.

Кросплатформні рішення дозволяють розробляти застосунки з однією кодовою базою, які компілюються або транспілюються під дві платформи. Найбільш поширені фреймворки:

- Flutter (Google);
- React Native (Meta);
- NativeScript (OpenJS Foundation);

Кожен із них має унікальні архітектурні принципи.

Flutter ґрунтується на використанні власного високопродуктивного графічного рушія Skia, який забезпечує незалежний від платформи рендеринг інтерфейсу та не покладається на системні UI-компоненти Android чи iOS. Завдяки цьому всі елементи інтерфейсу повністю контролюються самим фреймворком, що гарантує однаковий зовнішній вигляд застосунку на різних пристроях. Код на мові Dart компілюється у нативний ARM або x86-код за технологією Ahead-of-Time, що позитивно впливає на продуктивність і зменшує затримки під час виконання.

Серед ключових переваг Flutter вирізняється надзвичайно високою продуктивністю, стабільністю та передбачуваністю UI незалежно від платформи, а також розвиненою системою віджетів, яка дозволяє створювати інтерфейси практично будь-якої складності. Додатковою сильною стороною є підтримка кількох цільових платформ – окрім мобільних систем, Flutter забезпечує можливість розробки веб-версій і десктопних застосунків, що робить цей фреймворк універсальним інструментом.

Разом із тим Flutter має низку недоліків, які важливо враховувати на етапі вибору технологій. Типовий застосунок не має «нативного» вигляду за замовчуванням, оскільки UI малюється власними засобами рушія, що може вимагати додаткових зусиль для досягнення стилістики, звичної користувачам певної платформи. Застосунки на Flutter зазвичай мають більший розмір порівняно з нативними аналогами, а складні кастомізації або робота з низькорівневими можливостями системи нерідко потребують високої кваліфікації та глибокого розуміння внутрішніх механізмів фреймворку.

Попри ці обмеження, Flutter продовжує залишатися одним із найпопулярніших інструментів мобільної розробки. За результатами опитування Stack Overflow Developer Survey 2024 року він входить до трійки

найпоширеніших фреймворків у цій сфері [16], що підтверджує його актуальність, активну підтримку спільноти та високий попит на ринку.

React Native базується на використанні JavaScript, а побудова інтерфейсу відбувається через спеціальний міст (bridge), який забезпечує взаємодію між JavaScript-рушієм і нативними компонентами платформи. Такий механізм дозволив поєднати гнучкість вебтехнологій із продуктивністю нативних елементів, що зробило фреймворк одним із найпопулярніших рішень для створення мобільних застосунків. На практиці це означає, що логіка застосунку виконується у середовищі JavaScript, тоді як відтворення UI передається на рівень Android чи iOS, завдяки чому розробники можуть використовувати звичні інструменти екосистеми JavaScript, включно з великою кількістю бібліотек та інструментів.

Однією з помітних переваг React Native є можливість швидкого оновлення коду без необхідності повного перескладання застосунку за допомогою технологій на кшталт CodePush. Крім того, фреймворк забезпечує значну гнучкість у розширенні можливостей застосунку – за потреби розробники можуть створювати власні нативні модулі, інтегруючи логіку на Swift, Objective-C, Kotlin або Java.

Попри свої сильні сторони, React Native має і низку обмежень. Найвідомішим із них є той самий міст (bridge), який, хоча і забезпечує взаємодію між шарами системи, може уповільнювати роботу застосунку в сценаріях, де інтерфейс вимагає значної кількості швидких і синхронних оновлень. Це особливо відчутно у складних UI-взаємодіях або при роботі з великою кількістю анімацій. Крім того, різна поведінка тих самих компонентів на різних платформах іноді ускладнює забезпечення сталості користувацького досвіду. У деяких випадках окремі функції вимагають додаткових нативних доповнень, що збільшує обсяг роботи.

Meta у звітах за 2023 - 2024 роки підтверджує активний перехід фреймворку до нової архітектури Fabric, яка суттєво зменшує залежність від bridge і покращує продуктивність, завдяки чому React Native отримує новий етап розвитку та стає більш конкурентоспроможним у порівнянні з іншими кросплатформними технологіями [17].

NativeScript є кросплатформним фреймворком, який поєднує виконання логіки на TypeScript або JavaScript із повністю нативним рендерингом інтерфейсу. На відміну від більшості інших рішень, він забезпечує прямий доступ до API Android та iOS без використання проміжного мосту, що дозволяє працювати з можливостями платформи практично на тому ж рівні, що й нативні застосунки. Завдяки цьому UI формується з використанням нативних компонентів, а не емуляцій чи WebView, що позитивно впливає на продуктивність і відповідність системним UX-гайдам.

Однією з важливих переваг NativeScript є гнучкість у виборі інструментів для побудови застосунку: розробники можуть використовувати Angular, Vue або React, що робить технологію доступнішою для тих, хто вже працює з популярними вебфреймворками. Прямий доступ до нативних API дозволяє розширювати функціональність без потреби створювати додаткові модулі або використовувати сторонні обгортки, що забезпечує глибоку інтеграцію з операційною системою та стабільність роботи застосунку.

Разом з тим NativeScript має певні обмеження. Фреймворк менш популярний у порівнянні з Flutter чи React Native, що зумовлює менший розмір спільноти та повільніший розвиток екосистеми. Крім того, для ефективної роботи з ним розробнику необхідно добре розуміти нативні API Android та iOS, адже багато завдань передбачають взаємодію з низькорівневими можливостями платформ.

Попри ці виклики, NativeScript залишається важливим інструментом у корпоративному середовищі. Його активно застосовують у медичних, фінансових та освітніх системах, де ключовими вимогами є стабільність, надійність та глибока інтеграція з нативними функціями пристрою [18].

Згідно з результатами StackOverflow Survey 2024 року, кросплатформні фреймворки залишаються одними з ключових інструментів мобільної розробки. Найбільшого поширення набули Flutter, React Native та NativeScript, хоча їх використання залежить від вимог проєкту та сфери застосування. Порівняльний розподіл популярності цих технологій подано на рисунку 1.2, який демонструє лідерство Flutter та стабільний інтерес до інших сучасних рішень.

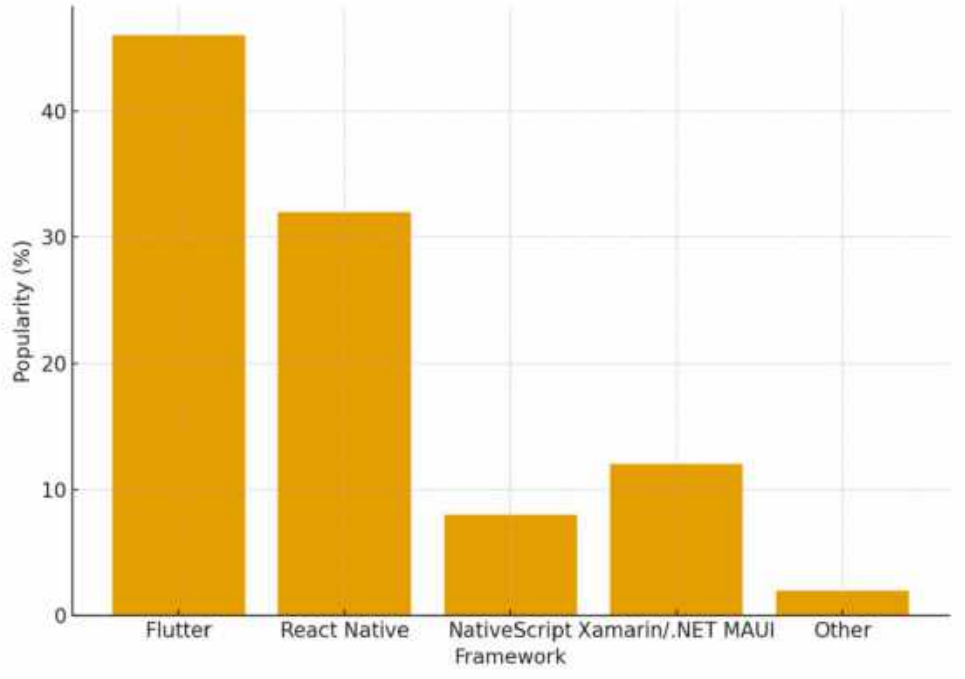

Рисунок 1.2 – Популярність кросплатформних мобільних фреймворків серед розробників у 2024 р. (за StackOverflow Survey)

Аналітика Gartner і Forrester підтверджує, що кросплатформна розробка є оптимальним вибором у більшості бізнес-кейсів, окрім високопродуктивних застосунків (ігри, AR/VR, медичні пристрої) [15; 19].

Розробка мобільних застосунків супроводжується низкою технічних викликів, які впливають на вибір технологій і архітектурних рішень. Одним із основних факторів є значна фрагментація пристроїв, особливо в екосистемі Android, що ускладнює забезпечення стабільної роботи застосунку на різних моделях і версіях операційної системи. Додаткові труднощі створює відмінність у поведінці інтерфейсів між платформами, що потребує ретельного врахування UX-особливостей кожної з них. Важливо також забезпечити оптимізацію енергоспоживання та гарантувати безпеку персональних даних користувачів, оскільки ці аспекти є критичними для сучасних мобільних рішень. Суттєвою є й складність підтримки синхронізації даних у реальному часі, особливо за умов нестабільного мережевого середовища. Окреме місце займає інтеграція функцій штучного інтелекту, що вимагає додаткових обчислювальних ресурсів і продуманих механізмів взаємодії з AI-сервісами. У комплексі ці виклики підкреслюють важливість правильного вибору архітектурного підходу, який забезпечить масштабованість та довгострокову підтримку застосунку.

Розробка інтелектуальних мобільних застосунків, що підтримують персоналізоване навчання, потребує комплексного поєднання клієнтських і серверних технологій, систем аналітики, методів машинного навчання та сучасних архітектурних підходів. У цьому підрозділі систематизовано основні технологічні компоненти, що застосовуються під час створення кросплатформних мобільних мовних застосунків із підтримкою адаптивності.

У сучасній мобільній розробці вебтехнології дедалі частіше використовуються як основа для побудови клієнтської логіки та інтерфейсу.

Одним із найбільш перспективних рішень у цьому напрямі є комбінація TypeScript, Vue.js та NativeScript, яка поєднує переваги веб-екосистеми з нативними можливостями мобільних платформ. TypeScript, створений компанією Microsoft, додає до JavaScript систему статичної типізації, що підвищує надійність коду, полегшує підтримку великих проєктів і дає змогу виявляти помилки ще на етапі компіляції. Мова має тісну інтеграцію з інструментами статичного аналізу та популярними IDE, а результати Stack Overflow Developer Survey 2024 підтверджують високий рівень її популярності, адже TypeScript стабільно входить до топ-5 найулюбленіших мов розробників [11].

Vue.js, який використовується як декларативний шар для побудови інтерфейсу, вирізняється реактивністю, простою компонентною моделлю та швидким порогом входу. Його гнучка архітектура та розвинена екосистема роблять фреймворк зручним для створення модульних UI-рішень, що легко масштабуються. У контексті мобільної розробки Vue працює в тісній інтеграції з NativeScript, який відповідає за трансформацію компонентів Vue у повністю нативні елементи інтерфейсу.

NativeScript виступає основою всього технологічного стеку, оскільки надає прямий доступ до API Android та iOS без використання додаткових мостів чи WebView. Завдяки цьому застосунки отримують нативну продуктивність і можливість працювати зі всіма системними функціями пристрою, включно з сенсорами, камерами та платформними службами. Підтримка кількох популярних фреймворків – Angular, Vue та React – робить NativeScript універсальним інструментом, здатним адаптуватися під різні потреби команд розробників. У сукупності така комбінація технологій виявляється особливо ефективною для мобільних освітніх застосунків, де

важливими є стабільність, швидкодія, та гнучка оптимізація під специфічні навчальні сценарії.

Для підтримки адаптивного навчання необхідна надійна серверна інфраструктура, яка забезпечує збирання та обробку даних користувачів, збереження прогресу й формування персоналізованих рекомендацій. Важливо також гарантувати стабільну й передбачувану комунікацію між клієнтом і сервером, оскільки саме вона визначає швидкість реакції системи та якість взаємодії з користувачем. У базі цього підходу лежить необхідність працювати з великим обсягом динамічних даних, оперативно формувати відповіді та забезпечувати стійку роботу системи в умовах масштабування.

Одним із ключових елементів серверної частини є PostgreSQL – потужна об'єктно-реляційна система керування базами даних, яка підтримує транзакційну цілісність за моделлю ACID і характеризується високим рівнем розширюваності. Завдяки підключеним модулям, зручній роботі з форматом JSONB та можливості масштабування в кластерних конфігураціях ця СУБД добре підходить для зберігання як структурованих, так і гнучких даних, що є типовим сценарієм для адаптивних освітніх систем. Рейтинг DB-Engines за 2024 рік підтверджує стабільно високу популярність PostgreSQL, яка входить до трійки найпоширеніших СУБД у світі [20].

Не менш важливим компонентом є Spring Framework, а особливо Spring Boot, який сьогодні є фактичним стандартом у створенні високонавантажених бекенд-рішень на Java. Його модульна архітектура, підтримка мікросервісних підходів, інтегрована система безпеки Spring Security та зручність побудови REST API роблять Spring Boot оптимальним вибором для серверної частини освітнього застосунку. Фреймворк також легко взаємодіє з AI-сервісами через HTTP або gRPC, що є критичним у контексті адаптивного навчання. За даними JetBrains Developer Ecosystem Survey 2023 року, Spring Boot використовується

більш ніж у 60 % корпоративних Java-проєктів [21], що свідчить про його надійність і зрілість. У структурі системи REST API виконує роль базового механізму інтеграції між клієнтом та сервером, забезпечуючи обмін прогресом, статистикою, метаданими вправ та токенами автентифікації завдяки своїй простоті, масштабованості та ефективній роботі з JSON-форматом.

Інтеграція великих мовних моделей стала одним із центральних напрямів розвитку сучасних систем адаптивного навчання, оскільки саме LLM забезпечують можливість створення динамічного, персоналізованого навчального контенту. Моделі, такі як GPT, Gemini чи Claude здатні генерувати вправи, пояснення, діалоги й тести, а також класифікувати рівень складності завдань відповідно до потреб студента. Вони аналізують стиль навчання, помилки та темп роботи користувача, формуючи рекомендації, які покращують індивідуальну траєкторію опанування матеріалу. Завдяки глибокому розумінню контексту сучасні моделі можуть працювати з великими фрагментами текстів і підтримувати цілі навчальні сесії, що відкриває можливості для створення гнучких, змістовних та реалістичних навчальних сценаріїв [22].

Разом із тим використання LLM має свої технологічні виклики. Оскільки виклики до модельних API є відносно дорогими, інтеграція потребує оптимізації інфраструктури та продуманого підходу до кешування й повторного використання відповідей. Моделі можуть допускати неточності або так звані «галюцинації», тому важливо передбачити механізми валідації та модерації контенту. Значну роль відіграє і швидкість мережі: повільне з'єднання може призводити до затримок у навчанні. Не менш критичними є питання конфіденційності, зокрема відповідність вимогам GDPR, FERPA, COPPA та іншим стандартам захисту персональних даних, які особливо важливі у сфері освіти.

Для ефективної роботи із великими мовними моделями в EdTech-системах застосовуються різні архітектурні патерни. Найпоширенішим є використання LLM як сервісу через API OpenAI або Google AI, що дозволяє швидко інтегрувати мовні можливості без розгортання власної ML-інфраструктури. Інший підхід – server-side aggregation – передбачає попередню обробку запитів на сервері, комбінування правил адаптивності та оптимізацію промптів для зниження витрат. Третій варіант, так звана гібридна AI-архітектура, поєднує можливості LLM із класичними алгоритмами рекомендацій і оцінювання. За даними HolonIQ AI in Education 2024, понад 45 % компаній EdTech уже інтегрували LLM у свої продукти [23], що підтверджує стратегічне значення цих технологій для майбутнього освітніх платформ.

Адаптивне навчання передбачає постійний збір, обробку та інтерпретацію даних про діяльність студента, оскільки саме ці дані дозволяють системі реагувати на індивідуальні особливості користувача. У процес аналізу включаються журнали взаємодії, час виконання завдань, успішність проходження вправ, частота повторення помилок та унікальні мовні патерни кожного студента. Сукупність цих показників формує профіль навчальної поведінки, який відображає темп опанування матеріалу, рівень складності, що є комфортним для конкретного користувача, та його слабкі місця. Для збору таких даних застосовуються як власні аналітичні модулі, так і спеціалізовані інструменти на кшталт Google Analytics for Firebase, Amplitude чи Mixpanel, що дають змогу відстежувати як технічні параметри взаємодії, так і поведінкові характеристики під час навчання.

Отримані показники надалі використовуються для прийняття рішень у системі адаптивності: вони допомагають визначати поточний рівень знань студента, автоматично змінювати складність вправ, прогнозувати потенційні

труднощі та формувати рекомендації, спрямовані на покращення результатів. Такий підхід забезпечує персоналізований досвід навчання, у якому зміст, формат і навантаження коригуються відповідно до реальних потреб користувача, що є ключовою перевагою сучасних EdTech-рішень.

Освітні застосунки, особливо ті, що працюють із неповнолітніми користувачами, повинні відповідати суворим вимогам міжнародних стандартів захисту даних. До ключових нормативів належать GDPR, який регулює обробку персональної інформації в країнах Європейського Союзу, а також FERPA і COPPA, що визначають правила роботи з даними учнів у Сполучених Штатах. Додатково важливим є дотримання стандарту інформаційної безпеки ISO/IEC 27001. Сукупність цих регуляцій встановлює чіткі правила щодо конфіденційності, обмежень на збирання даних про дітей та прозорості їх використання. В освітніх системах такі вимоги є особливо критичними, адже платформи працюють з індивідуальними профілями учнів, їхнім прогресом, історією навчальної активності та іншими чутливими відомостями.

Для забезпечення відповідності цим вимогам у застосунках впроваджуються різноманітні механізми захисту, які гарантують збереження інформації на всіх етапах її обробки. Зазвичай використовується шифрування даних як під час передачі, так і в стані зберігання, а доступ до ресурсу регулюється за допомогою токенів автентифікації. Системи також регулярно проходять аудит прав доступу, що дозволяє вчасно виявляти та усувати потенційні загрози. Окрему увагу приділяють локальній обробці найбільш чутливих даних, що мінімізує ризики витоку інформації та підтримує відповідність міжнародним стандартам безпеки, які є обов'язковими для сучасних освітніх платформ.

Узагальнюючи викладене, створення сучасного адаптивного мобільного застосунку потребує комплексного поєднання кросплатформних клієнтських

технологій, потужної серверної інфраструктури, генеративних мовних моделей, аналітичних інструментів та механізмів персоналізації, здатних ураховувати індивідуальні поведінкові характеристики користувачів. Лише інтеграція цих складових дає можливість забезпечити високу якість інтерактивного навчання, гарантувати стабільність і масштабованість системи, а також формувати конкурентоспроможні рішення на глобальному ринку EdTech.

Адаптивне навчання є підходом, що передбачає індивідуалізацію освітнього процесу відповідно до рівня, потреб, стилю та динаміки користувача. У мовних мобільних застосунках адаптивність сприяє формуванню персоналізованих траєкторій навчання, підвищує залученість і дозволяє швидше досягти практичних результатів. Наукові дослідження свідчать, що адаптивні системи підвищують ефективність навчання на 25 - 50 % порівняно зі статичними курсами [24].

Адаптивні системи навчання в галузі вивчення іноземних мов ґрунтуються на принципах індивідуалізації та постійного моніторингу навчальної діяльності студента. Зміст матеріалів і завдань формується з урахуванням рівня володіння мовою, темпу засвоєння, типових помилок і сильних чи слабких сторін користувача. Дослідження Carnegie Mellon University свідчать, що такі персоналізовані підходи дозволяють скоротити час досягнення навчальних результатів приблизно на третину [25]. Ефективність адаптивності забезпечується неперервною діагностикою, під час якої система аналізує відповіді студента, тривалість виконання завдань та характер повторюваних помилок, що дає змогу динамічно коригувати траєкторію навчання в реальному часі.

Важливою складовою таких систем є механізм автоматичного підстроювання складності, який змінює рівень завдань відповідно до результатів і навчальної моделі користувача. Як показують дослідження Baker

& Yacef, це сприяє підтриманню мотивації, запобігає перевантаженню або нудьзі та підсилює сталість навчального прогресу [26]. Не менш значущим є персоналізований зворотний зв'язок: замість загальних рекомендацій студент отримує пояснення, підказки та приклади, релевантні його поточним темам і типам помилок, що підвищує ефективність засвоєння матеріалу й робить навчання більш усвідомленим і структурованим.

Адаптивні моделі в мовних освітніх системах ґрунтуються на понятті «моделі студента» – центрального компонента, що описує рівень користувача, історію відповідей, ймовірність засвоєння тем та поведінкові характеристики. Така модель може будуватися на різних підходах: від класичних Item Response Theory та Bayesian Knowledge Tracing до сучасних нейронних методів, зокрема Deep Knowledge Tracing та LLM-based Profiling, які використовують можливості великих мовних моделей для глибшого аналізу навчальної поведінки.

Рекомендаційні алгоритми в мовних застосунках використовують як контентно-орієнтовані, так і колаборативні або змішані підходи, а впродовж 2023-2024 років у сфері EdTech особливо поширилися системи, підсилені LLM. Такі рішення дозволяють автоматично генерувати вправи відповідно до навчального контексту, розпізнавати та пояснювати мовні помилки, оцінювати відкриті відповіді та добирати матеріали, релевантні стилю та темпу навчання конкретного студента. Це забезпечує значно вищу гнучкість у формуванні навчальних траєкторій, ніж традиційні фіксовані курси.

Генеративні моделі, зокрема GPT та Gemini, розширюють адаптивні можливості системи, дозволяючи створювати унікальні навчальні сценарії. Вони можуть моделювати діалогові ситуації, виступати в ролі співрозмовника, перевіряти вимову на основі моделей автоматичного розпізнавання мовлення та формувати персональні контексти на кшталт професійних або побутових

ситуацій. Завдяки таким можливостям застосунок перестає бути набором статичних матеріалів і перетворюється на динамічну систему, здатну генерувати новий контент відповідно до потреб і поведінки студента.

У контексті мобільного навчання адаптивність тісно пов'язана з продуманим користувацьким досвідом (UX), який має підтримувати природний темп опанування матеріалу та не перевантажувати студента. Одним із ключових підходів є microlearning – подання навчального контенту малими, чітко структурованими блоками, що допомагає утримувати увагу і сприяє кращому запам'ятовуванню. Дослідження Journal of Applied Psychology показують, що використання мікроформатів підвищує рівень засвоєння приблизно на 18 % [27]. Важливу роль відіграє і гейміфікація: елементи прогресу, рівнів, винагород чи серій мотивують користувача повертатися до занять і збільшують регулярність взаємодії. За даними досліджень, гейміфіковані системи здатні підвищувати повторне залучення на 20-30 % [28], що робить їх ефективним інструментом підтримки навчальної мотивації.

Ще одним суттєвим аспектом UX у адаптивних системах є механізм персоналізованих підказок, які замінюють традиційні статичні пояснення. Система генерує коментарі, приклади та уточнення, релевантні конкретним помилкам студента, що робить процес навчання більш зрозумілим і сфокусованим на потребах користувача. Паралельно з цим застосовується принцип зниження когнітивного навантаження: платформа дозує інформацію таким чином, щоб студент не отримував надмірного обсягу даних одночасно. Завдяки цьому навчальний процес відбувається у комфортному темпі, а складність завдань і спосіб подачі поступово коригуються відповідно до рівня підготовки та поведінкових патернів користувача.

Адаптація контенту в мобільних мовних застосунках охоплює кілька взаємопов'язаних вимірів, спрямованих на створення максимально

релевантного досвіду для студента. Насамперед важливою є тематична варіативність: користувач може обирати напрями, близькі до його інтересів – наприклад, подорожі, медицину, IT чи бізнес – а великі мовні моделі автоматично генерують вправи з відповідною лексикою. Не менш суттєвою є адаптація за рівнем складності: матеріали формуються із прив'язкою до загальноєвропейських стандартів CEFR (A1-C2), а використання LLM дає змогу створювати контент будь-якого рівня без потреби у попередньо підготовлених статичних наборах завдань. Стиль навчання також відіграє помітну роль – хтось краще сприймає короткі правила, а комусь потрібні розгорнуті пояснення чи більша кількість практичних завдань; модель визначає ці особливості шляхом аналізу поведінкових патернів.

Важливою ознакою сучасних адаптивних систем є можливість динамічної перебудови навчального процесу в реальному часі. Завдяки інтеграції великих мовних моделей завдання можуть змінюватися «на льоту» відповідно до дій студента, діалогові сценарії – трансформуватися залежно від корекцій чи запитань, а додаткові пояснення – надаватися одразу після виявленої помилки. Такий підхід забезпечує безперервний зворотний зв'язок і дозволяє створювати навчальні траєкторії, що постійно підлаштовуються під індивідуальні потреби користувача, роблячи процес вивчення мови більш природним, ефективним і гнучким.

Попри значний потенціал штучного інтелекту, впровадження адаптивних систем супроводжується низкою суттєвих викликів. Найскладнішим є побудова точної моделі знань студента, оскільки система має враховувати не лише правильність відповідей, а й поведінкові патерни, темп навчання та контекст помилок. Важливим залишається й баланс між автоматизованими рекомендаціями та педагогічною доцільністю, адже не всі рішення, сформовані алгоритмом, можуть відповідати навчальним цілям.

Додаткові труднощі пов'язані з потребою у великих обсягах даних, необхідних для якісного аналізу, а також із забезпеченням приватності, етичних норм і відповідності регуляторним вимогам. Окрему проблему становить ризик надмірної залежності від мовних моделей, оскільки вони можуть породжувати неточності або помилкові інтерпретації. Дослідження IEEE Transactions on Learning Technologies підкреслюють, що найбільш ефективними є підходи, які поєднують можливості LLM із класичними моделями відстеження знань, що дозволяє підвищити точність адаптації та стабільність навчальної системи [24].

1.3 Повстановка завдань на кваліфікаційну роботу магістра

Для досягнення мети кваліфікаційної роботи магістра, потрібно виконати наступні завдання:

– провести всебічний огляд сучасних досліджень у галузі мобільних освітніх технологій, інтеграції штучного інтелекту в освітні платформи та кросплатформної розробки. Визначити ключові наукові прогалини в застосуванні адаптивного навчання іноземним мовам за допомогою ШІ;

– проаналізувати та порівняти існуючі мобільні застосунки для вивчення іноземних мов, що використовують штучний інтелект, такі як Duolingo, Babbel, Memrise. Визначити їхні переваги та недоліки, а також можливості для покращення досвіду користувача, персоналізації та інтеграції ШІ;

– розробити детальну специфікацію вимог до кросплатформного мобільного застосунку, включаючи функціональні та нефункціональні вимоги, зокрема інтеграцію з великими мовними моделями (LLM) та бекенд-фреймворками;

– розробити загальну архітектуру системи для кросплатформного мобільного застосунку, що включає клієнтську частину, серверну логіку, шари інтеграції з ШІ та управління базою даних. Визначити ролі кожного компонента, їх взаємодії та вибір технологічного стеку;

– реалізувати функціональний прототип мобільного застосунку, спираючись на проектовану архітектуру. Використовувати кросплатформний фреймворк, наприклад, React Native або Flutter, та бекенд-фреймворк, такий як Spring Boot, для розробки застосунку;

– інтегрувати великі мовні моделі, такі як GPT або Google Gemini, у мобільний застосунок. Розробити алгоритми для надання персоналізованого навчального досвіду, адаптуючи контент в залежності від поведінки користувача та його прогресу;

– провести всебічне тестування, включаючи модульне тестування, інтеграційне тестування, тестування продуктивності та тестування прийнятності користувачами. Оцінити ефективність функцій ШІ у покращенні навчального процесу;

– провести порівняльний аналіз створеної системи з існуючими рішеннями на ринку, зокрема за такими показниками, як продуктивність, залученість користувачів та економічна доцільність. Надати рекомендації щодо масштабування системи та зниження операційних витрат.

РОЗДІЛ 2
ТЕОРЕТИЧНЕ ДОСЛІДЖЕННЯ ТА ПРАКТИЧНА РЕАЛІЗАЦІЯ КРОСПЛАТФОРМНОГО МОБІЛЬНОГО ЗАСТОСУНКУ ДЛЯ ВИВЧЕННЯ ІНОЗЕМНИХ МОВ ІЗ ВИКОРИСТАННЯМ ШТУЧНОГО ІНТЕЛЕКТУ

2.1 Обґрунтування вибору шляхів, технологій, алгоритмів і засобів вирішення поставленого завдання

Розробка мобільного застосунку для адаптивного вивчення іноземних мов потребує комплексного аналізу вимог, що формують функціональну поведінку системи, її якісні характеристики та моделі взаємодії користувачів із системою. У цьому розділі наведено формалізовані функціональні та нефункціональні вимоги, а також ключові сценарії використання.

2.1.1 Аналіз вимог до системи

Функціональні вимоги визначають коло можливостей, які система повинна реалізувати для забезпечення повноцінного персоналізованого навчання. Вони охоплюють управління користувачами, роботу з навчальним контентом, інтеграцію штучного інтелекту, механізми адаптивності, мотиваційні елементи, офлайн-підтримку та аналітичні інструменти. У контексті користувацьких даних система має забезпечувати створення облікових записів, автентифікацію, у тому числі через зовнішні сервіси, відновлення доступу та роботу з токенами, а також редагування профілю, що включає персональні налаштування й навчальні цілі.

Організація навчального контенту спирається на багаторівневу структуру курсів, розділів, уроків і завдань, що дає змогу гнучко формувати навчальні траєкторії. Платформа повинна підтримувати різні типи вправ – від тестів і перекладу до аудіювання й вимови – а також використовувати метадані для

опису тривалості, складності та навчальних цілей. Декларативна конфігурація у форматі YAML дозволяє створювати й оновлювати контент без втручання в основний код застосунку, що значно спрощує роботу методистів та розробників.

Інтеграція великих мовних моделей є ключовою частиною системи, адже саме вона забезпечує можливість генерувати персоналізовані завдання. Платформа повинна підтримувати взаємодію з сучасними LLM через API, адаптовувати створюваний контент до рівня студента та його помилок, перевіряти структурну коректність відповідей за допомогою JSON Schema, оптимізувати роботу через кешування й обробляти повторні запити у випадку недоступності моделей. Адаптивне навчання ґрунтується на відстеженні прогресу, часових показників та тематичного засвоєння, використанні алгоритмів повторення з інтервалами, формуванні блоку корекції помилок та динамічному регулюванні складності завдань. Рекомендації генеруються на основі поєднання індивідуальних слабких тем, частоти повторення та нових матеріалів.

Для підтримки мотивації система використовує механізми гейміфікації, такі як серії навчальних днів, прогрес досвіду, рівні, досягнення та рейтинги, а також щоденні цілі, що стимулюють регулярність занять. Окремим аспектом є офлайн-функціональність: користувач повинен мати можливість проходити уроки без доступу до мережі, зберігати результати локально та синхронізувати їх після відновлення з'єднання, при цьому функції ШІ у автономному режимі обмежуються. Аналітичні інструменти забезпечують перегляд персональної статистики, детальні підсумки уроків та можливість адміністративного аналізу поведінки користувачів і ефективності контенту, що є важливою складовою вдосконалення системи.

Узагальнений перелік функціональних можливостей системи, що охоплює роботу з користувачами, управління контентом, інтеграцію штучного інтелекту, механізми адаптивності, мотиваційні інструменти, офлайн-підтримку та аналітику, потребує чіткої систематизації. З цією метою у таблиці 2.1 подано пріоритизацію функціональних вимог, що дозволяє визначити послідовність реалізації ключових компонентів та забезпечити узгоджений розвиток архітектури мобільного адаптивного застосунку.

Таблиця 2.1 – Пріоритизація функціональних вимог

ID	Назва групи вимог	Короткий опис	Пріоритет	Складність	Включення до MVP
FR-1	Управління користувачами	Реєстрація, вхід, соціальні логіни, профіль	Критичний	Середня	Так
FR-2	Управління навчальним контентом	Курси, розділи, уроки, завдання, YAML-конфігурація	Критичний	Висока	Так
FR-3	AI-генерація завдань	Інтеграція з LLM, валідація, кешування, retry	Критичний	Дуже висока	Так
FR-4	Адаптивне навчання	Відстеження прогресу, spaced repetition, рекомендації	Високий	Висока	Базова частина
FR-5	Gamification	Streak, XP, рівні, досягнення, рейтинги	Високий	Середня	Частково
FR-6	Офлайн-режим	Завантаження уроків, локальна БД, синхронізація	Середній	Висока	Ні
FR-7	Аналітика та звітність	Dashboard користувача, підсумки уроків, адмін-аналітика	Середній	Середня	Базова частина

Нефункціональні вимоги визначають якість роботи системи, її стабільність, продуктивність, безпеку та економічну доцільність. З погляду

продуктивності платформа повинна забезпечувати швидке завантаження інтерфейсу, низьку затримку відповіді серверних API та оптимізовану взаємодію з базою даних, що дозволяє підтримувати необхідну пропускну здатність у разі зростання кількості користувачів. Висока надійність досягається за рахунок цільової доступності на рівні 99,5 % для MVP і до 99,9 % у промисловій експлуатації, підтримки fallback-механізмів у випадку збоїв зовнішніх сервісів та регулярного резервного копіювання.

Масштабованість системи передбачає використання горизонтальної архітектури з безстанними серверними модулями, балансуванням навантаження та окремими стратегіями масштабування бази даних, кешування й API-рівня. Значну роль відіграють питання безпеки: система має підтримувати сучасні протоколи автентифікації, використовувати шифрування, запобігати атакам на зразок SQL-ін'єкцій, XSS чи CSRF, а також відповідати вимогам GDPR. Важливим критерієм є підтримуваність: архітектура повинна включати структуроване логування, моніторинг, автоматизовані CI/CD-процеси, документування API та дотримання стандартів якості коду.

Зручність використання забезпечується адаптованим для мобільних пристроїв інтерфейсом, локалізацією, доступністю для користувачів із різними потребами та швидким onboarding-процесом. Портативність досягається завдяки можливості роботи як у веб-версії, так і як нативний мобільний застосунок, а також через підтримку контейнеризації й розгортання в будь-якому хмарному середовищі. Економічна ефективність передбачає оптимізацію витрат на використання LLM-сервісів, раціональне використання хмарних ресурсів і застосування безкоштовних тарифів на етапі розробки, що дозволяє системі залишатися конкурентоспроможною та фінансово стійкою.

Як показано на рисунку 2.1, основні групи нефункціональних вимог формують узгоджену систему якісних характеристик, що визначають роботу

застосунку в реальних умовах експлуатації. Вони охоплюють продуктивність, надійність, масштабованість, безпеку, підтримуваність, зручність використання, портативність та економічність – саме ці параметри задають рамку для проєктування архітектури та забезпечують ефективність і стабільність мобільної платформи.

Рисунок 2.1 – Структура основних категорій нефункціональних вимог системи

Пріоритизація нефункціональних вимог дозволяє визначити, які характеристики системи мають ключовий вплив на її стабільність, продуктивність та безпеку. Таблиця 2.2 узагальнює ці вимоги та розподіляє їх за рівнями критичності для проєкту.

Use Case-модель описує ключові сценарії взаємодії користувача із системою та дозволяє формалізувати її поведінку в типових умовах. На основі функціональних вимог виділено три базові варіанти використання: реєстрація користувача, проходження адаптивного уроку та перегляд індивідуальної статистики.

UC-1. Реєстрація користувача. Сценарій охоплює перший вхід до системи: користувач обирає метод реєстрації (email або OAuth), вводить дані, після чого система створює обліковий запис і пропонує короткий onboarding з вибором мовних параметрів. Передбачено обробку типових помилок (некоректний email, слабкий пароль, відсутність інтернету).

Таблиця 2.2 – Пріоритизація нефункціональних вимог

Категорія	Підкатегорія	Критерій / показник	Цільове значення	Пріоритет
Продуктивність	API	Час відповіді p95	< 500 мс (без AI), < 3 с (з AI)	Критичний
Продуктивність	UI	Час початкового завантаження	< 2 с	Високий
Надійність	Доступність	Uptime	99,5 % (MVP), 99,9 % (продакшн)	Високий
Надійність	Збереження даних	Резервне копіювання	Щоденні резервні копії, зберігання ≥ 30 днів	Високий
Масштабованість	Користувачі	Синхронні користувачі	100 (MVP), 10 000 (цільове значення)	Середній
Безпека	Автентифікація	Механізм аутентифікації	JWT + OAuth 2.0, безпечне зберігання паролів	Критичний
Безпека	Захист даних	Шифрування та відповідність нормам	HTTPS, шифрування БД, відповідність GDPR	Високий
Підтримуваність	Якість коду	Покриття тестами	> 80 % для критичних шляхів backend	Високий
Підтримуваність	Документація	Опис API та архітектури	Наявність OpenAPI, ADR, README	Середній
Зручність	UX	Час onboarding	< 3 хвилин до початку першого уроку	Високий
Портативність	Платформи	Підтримувані середовища	Web + iOS + Android (NativeScript)	Середній
Економічність	LLM API	Вартість на активного користувача	< 0,10 USD/день	Високий
Економічність	Інфраструктура	Місячні витрати для 1000 користувачів	< 500 USD/місяць	Середній

UC-2. Проходження уроку з адаптивною генерацією завдань. Користувач обирає урок, система завантажує його структуру та формує персоналізований набір завдань із використанням мовної моделі. Під час виконання вправ надається миттєвий зворотний зв'язок, а помилки повторно пропонуються у фазі корекції. Після завершення уроку система зберігає прогрес, оновлює статистичні показники та формує підсумковий звіт. Передбачено офлайн-режим і fallback-контент на випадок недоступності ШІ.

UC-3. Перегляд статистики та аналітики. Користувач відкриває панель прогресу, де відображаються ключові метрики – streak, XP, рівень, історія активності та тематичні сильні й слабкі сторони. Система дозволяє переглядати дані за різні періоди й аналізувати динаміку навчання, що підсилює мотиваційний ефект.

На основі визначених функціональних вимог сформовано узагальнену модель ключових сценаріїв взаємодії користувача із системою. На рисунку 2.2 подано діаграму варіантів використання, яка відображає основні дії користувача та відповідні реакції системи.

Рисунок 2.2 – Діаграма варіантів використання системи адаптивного вивчення іноземних мов

У результаті аналізу сформовано узагальнений перелік функціональних і нефункціональних вимог, що визначають призначення, структуру та поведінку системи адаптивного вивчення мов. Функціональні вимоги описують основні можливості платформи, а нефункціональні – цільові показники продуктивності, надійності, масштабованості, безпеки та зручності використання.

Також було визначено ключові сценарії взаємодії користувача із системою – реєстрація, проходження адаптивного уроку та перегляд статистики, що дозволяє сформувати чіткі вимоги до логіки роботи та компонентів архітектури. Сукупність цих вимог і сценаріїв створює концептуальну основу для подальшого проєктування та оцінювання майбутньої реалізації.

2.1.2 Концептуальна модель системи

Концептуальна модель системи відображає загальні принципи організації програмних компонентів, взаємодію між ними та обраний технологічний підхід. На основі аналізу вимог, наведених у розділі 2.1.1, для системи адаптивного вивчення іноземних мов обрано гібридну клієнт–серверну архітектуру з виділеним шаром інтеграції зі сервісами штучного інтелекту (AI Integration Layer). Такий підхід забезпечує чітке розмежування відповідальності між клієнтськими застосунками, серверною бізнес-логікою, підсистемою роботи з даними та зовнішніми LLM-провайдерами.

Високорівнева архітектура системи ґрунтується на трирівневій моделі, що охоплює клієнтський рівень, серверну прикладну логіку та рівень даних із використанням зовнішніх сервісів. На клієнтському рівні функціонують веб- і мобільні застосунки, які поділяють спільне програмне ядро для уніфікації бізнес-логіки та скорочення витрат на супровід. Обмін між клієнтом і сервером здійснюється через REST-API, а в задачах із підвищеними вимогами до швидкості реакції може застосовуватися двосторонній канал зв’язку.

Серверний рівень реалізовано як набір незалежних сервісів, що обробляють автентифікацію, управління контентом, генерацію завдань, аналітику та моделювання прогресу. Центральним елементом виступає адаптивний модуль, який відповідає за формування персоналізованих траєкторій навчання, добір матеріалів і баланс між новими та повторюваними темами.

Важливою складовою архітектури є шар інтеграції з великими мовними моделями, який інкапсулює логіку побудови запитів, розбору відповідей, керування кешем, вибору оптимального провайдера та контролю ресурсів. Така ізоляція дозволяє мінімізувати залежність від конкретних AI-сервісів, забезпечити стабільні політики обробки даних і спростити подальше розширення системи.

На рівні даних використовується реляційна СУБД для зберігання користувачів, курсів, уроків, завдань та прогресу, кеш у пам'яті для тимчасових даних і лімітів, а також файлове сховище для мультимедійних ресурсів (звукові файли, ілюстрації, аватари). Узагальнену схему високорівневої архітектури системи подано на рисунку 2.3.

Запропонована архітектура забезпечує чітке розмежування відповідальності між основними рівнями системи, що робить її гнучкою, керованою та придатною до подальшого розвитку. Такий підхід дає змогу незалежно масштабувати клієнтські застосунки й серверні сервіси, централізовано контролювати безпеку та доступ до даних, а також без ускладнень розширювати.

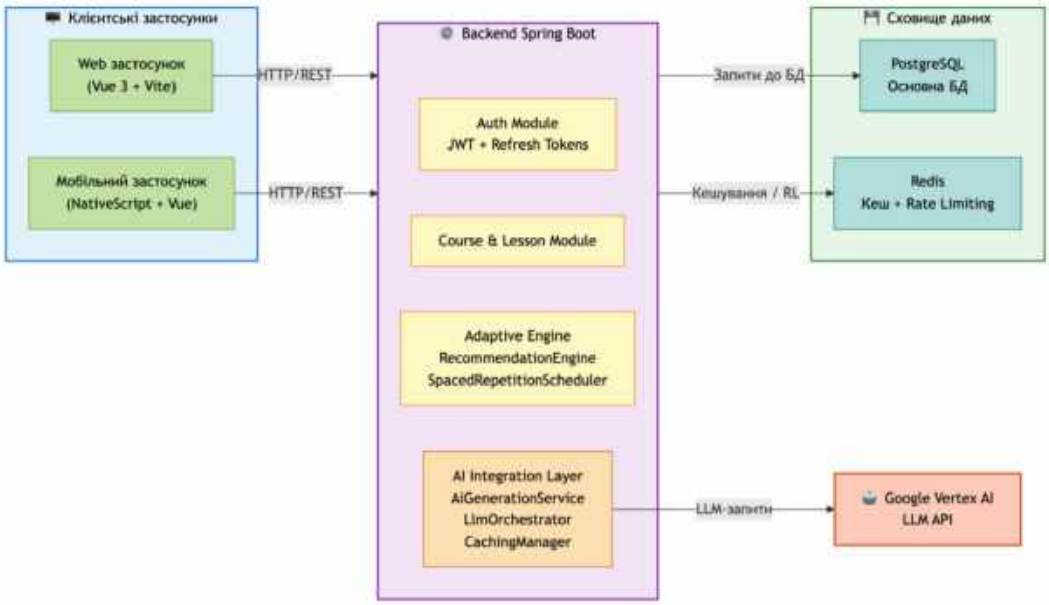

Рисунок 2.3 – Високорівнева архітектура системи адаптивного вивчення іноземних мов

Обраний технологічний стек спрямований на забезпечення кросплатформеності клієнтських застосунків, високої продуктивності та масштабованості серверної частини, а також безперешкодної інтеграції з мовними моделями й підтримки сучасних підходів до розроблення, тестування та розгортання. Узагальнений перелік ключових технологій подано в таблиці 2.3.

Запропонований стек забезпечує баланс між продуктивністю, зрілістю технологій та простотою інтеграції з обраними хмарними сервісами.

Компонентна модель деталізує загальну архітектуру системи, визначаючи структуру програмних модулів, їхню відповідальність та правила взаємодії між ними.

Таблиця 2.3 – Основні технології розроблення системи

Рівень системи	Основні технології	Коротке обґрунтування використання
Веб-клієнт	Vue 3, Vite, Pinia, Tailwind CSS	Побудова SPA, сучасний підхід Composition API, швидка збірка, гнучкий UI
Мобільний клієнт	NativeScript з Vue 3	Використання нативних компонентів iOS/Android при спільній логіці з веб-клієнтом
HTTP-клієнт на фронтенді	Axios	Зручна робота з REST-API, підтримка перехоплювачів, єдина точка конфігурації
Backend-фреймворк	Spring Boot (Java)	Зріла екосистема, підтримка REST, безпеки, інтеграції з БД та зовнішніми сервісами
Доступ до даних	JPA/Hibernate	Об'єктно-реляційне відображення, підтримка складних зв'язків і транзакцій
Основна база даних	PostgreSQL	Надійна СУБД з підтримкою транзакцій, JSON-полів та розширень для AI-сценаріїв
Кешування	Redis	In-memory кеш для швидкого доступу, зберігання сесій та лімітів запитів
Автентифікація та безпека	Spring Security, JWT, OAuth 2.0	Реалізація захищеної автентифікації, авторизації та інтеграції з соцмережами
Інтеграція з LLM	Spring AI, клієнти до Vertex AI, OpenAI тощо	Уніфікована взаємодія з різними LLM-провайдерами, підтримка структурованих відповідей
Документування API	OpenAPI / Swagger	Автоматична генерація документації й інтерактивного інтерфейсу для тестування
Тестування	JUnit, Mockito (backend); Vitest, Testing Library (frontend)	Підтримка модульного й інтеграційного тестування, перевірка критичних сценаріїв
Контейнеризація	Docker	Уніфікація середовища виконання, спрощення розгортання в хмарі
Оркестрація	Kubernetes	Автоматичне масштабування, самовідновлення сервісів, декларативне конфігурування
CI/CD	GitHub Actions (або аналогічний сервіс)	Безперервна інтеграція та доставка, автоматичне тестування й деплой
Моніторинг і логування	Prometheus, Grafana, централізовані логи	Збір метрик, візуалізація стану системи, налаштування попереджень

У межах такої моделі клієнтські застосунки, серверні сервіси, модуль адаптивного навчання, шар інтеграції з мовними моделями та підсистема роботи з даними утворюють цілісну екосистему, де кожен компонент виконує чітко визначені функції та спілкується з іншими через стандартизовані інтерфейси. Такий підхід забезпечує ізольованість логіки, спрощує масштабування, пришвидшує розробку нових функцій і полегшує супровід системи в довгостроковій перспективі. Структурне представлення основних компонентів і зв'язків між ними наведене на рисунку 2.4.

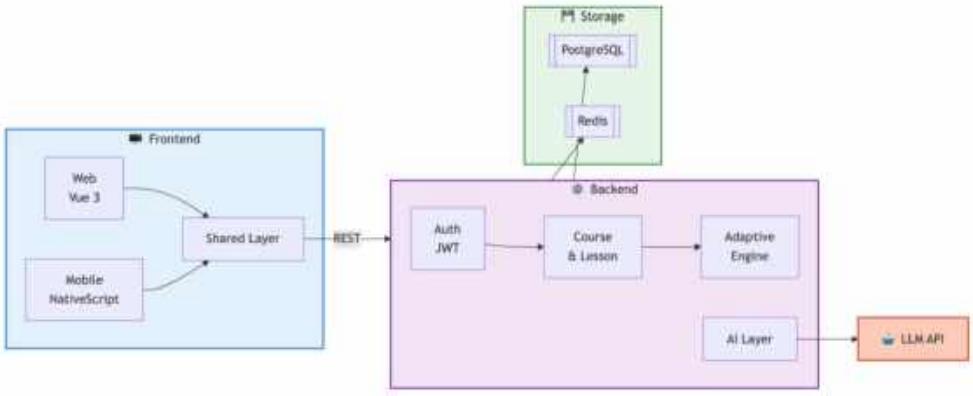

Рисунок 2.4 – Компонентна діаграма системи адаптивного вивчення іноземних мов

На клієнтському рівні система включає веб-застосунок на основі Vue 3, мобільний застосунок NativeScript-Vue та спільний програмний шар, у якому зосереджені моделі даних, сервіси доступу до API й сховища стану. Це забезпечує єдність бізнес-логіки та мінімізує дублювання коду між платформами. На рівні прикладної логіки взаємодія реалізована через набір REST-контролерів і сервісів бізнес-логіки, що відповідають за автентифікацію, управління користувачами, курсами, уроками, завданнями та прогресом.

Центральним елементом серверної частини є модуль адаптивного навчання, який профілює користувача, формує рекомендації та визначає оптимальну траєкторію навчання. Окремий шар інтеграції з LLM забезпечує генерацію персоналізованих завдань, orchestrator для вибору моделі, формування промптів, валідацію відповідей і контроль витрат через кешування та обмеження частоти запитів. На рівні даних використовуються PostgreSQL для навчального контенту й результатів, Redis для кешування та S3-сховище для мультимедіа. Зовнішні сервіси, такі як LLM-провайдери та OAuth-автентифікація, інкапсульовані у спеціалізованих клієнтах, що спрощує конфігурацію й дозволяє легко змінювати постачальників.

Архітектура розгортання визначає, яким чином програмні компоненти системи розміщуються в хмарному середовищі та взаємодіють між собою у промисловій експлуатації. На рисунку 2.5 подано узагальнену діаграму розгортання.

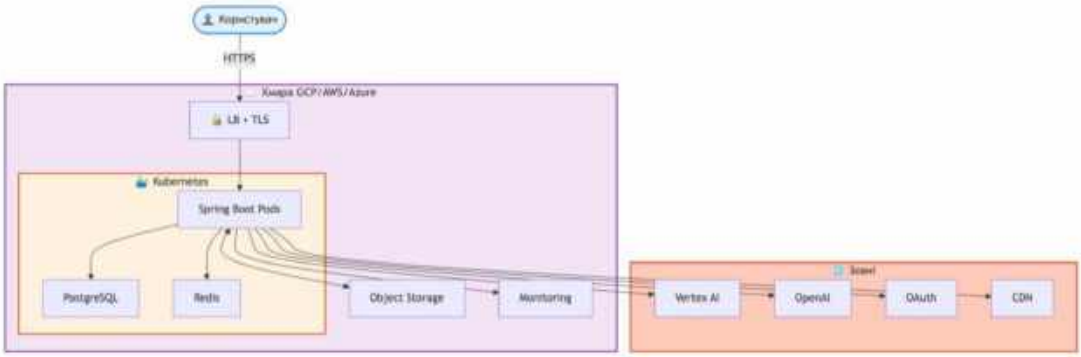

Рисунок 2.5 – Узагальнена діаграма розгортання системи в хмарному середовищі

Система розгортається у хмарній інфраструктурі, зокрема в Google Cloud, AWS або Azure, використовуючи Kubernetes як платформу для оркестрації

контейнерів. Вхідний трафік приймається балансувальником, який завершує TLS-з'єднання, перевіряє стан сервісів і розподіляє навантаження між вузлами. Backend запускається у вигляді набору pod-ів, що автоматично масштабуються залежно від поточного навантаження та оновлюються через механізм rolling update без зупинки роботи. Основна база даних PostgreSQL працює у конфігурації primary–replica на стійких томах, що забезпечує високу доступність, розподіл операцій читання й запису та надійне резервне копіювання. Redis застосовується для кешування та управління сесіями, а мультимедійні файли зберігаються у хмарному об'єктному сховищі з підтримкою CDN.

Додатково система доповнена засобами моніторингу та логування, які надають повну спостережуваність за роботою сервісів, формують дашборди продуктивності та надсилають оповіщення у разі відхилень. Такий підхід до розгортання гарантує масштабованість і стійкість системи до збоїв, забезпечує безперервні оновлення та оптимізує витрати за рахунок гнучкого керування ресурсами й автоматичного балансування навантаження.

Отже сформовано концептуальну модель системи адаптивного вивчення іноземних мов, що охоплює високорівневу архітектуру, технологічний стек, компонентну структуру та підходи до розгортання в хмарі. Запропонована трирівнева клієнт–серверна архітектура з окремим шаром інтеграції з LLM забезпечує чітке розмежування відповідальності, гнучке масштабування та можливість подальшого розширення функціональності.

Обраний технологічний стек орієнтований на кросплатформеність, продуктивність і стабільну взаємодію з AI-сервісами, а компонентна модель узгоджено описує роботу клієнтів, серверних модулів, адаптивного ядра та підсистеми інтеграції з мовними моделями. Архітектура розгортання на Kubernetes гарантує масштабованість, доступність і керованість системи,

формуючи надійну основу для подальшої реалізації та перевірки ефективності запропонованих рішень.

2.1.3 Детальне проєктування клієнтської частини

Клієнтська частина системи реалізована як багатоплатформний frontend-рівень, що включає веб-застосунок та мобільний застосунок, побудовані на спільній кодовій базі. Основна мета проєктування – забезпечити максимальне повторне використання бізнес-логіки, типів даних та сервісів доступу до API між платформами, водночас зберігши можливість розробляти специфічні інтерфейсні рішення для Web та Mobile.

Клієнтська частина системи організована у форматі monorepo з використанням pnpm workspaces, що забезпечує структуроване керування кількома взаємопов'язаними пакетами в одному репозиторії. Основу монорепозиторію становлять три ключові пакети: shared, який містить спільні для всіх платформ сервіси, типи та сховища стану; web, що реалізує браузерний інтерфейс на базі Vue 3 та Vite; та native, який відповідає за мобільний застосунок на NativeScript із підтримкою нативних компонентів iOS та Android. Структурну схему організації монорепозиторію представлено на рисунку 2.6.

Таке проєктне рішення забезпечує узгодженість бізнес-логіки між веб- і мобільною версіями, мінімізує дублювання коду завдяки спільним типам, API-сервісам і сховищам стану, а також спрощує тестування та повторне використання модульних тестів для спільного шару. Крім того, архітектура монорепозиторію залишається гнучкою: специфічні для окремих платформ компоненти ізолюються у власних пакетах, що дозволяє незалежно розвивати кожен клієнт без порушення загальної структури системи.

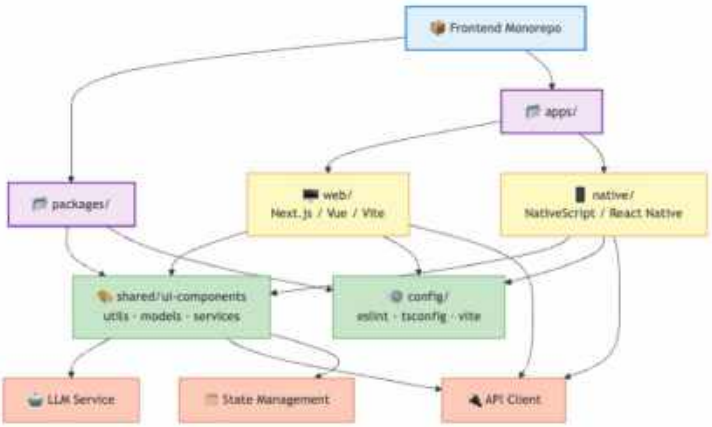

Рисунок 2.6 – Структура frontend-монорепозиторію застосунку (shared, web та native пакети)

Архітектура управління станом клієнтської частини базується на використанні Pinia як офіційного менеджера стану для Vue 3. Сховища структуруються за доменним принципом, що забезпечує незалежність логіки автентифікації, курсів, уроків та профілю користувача. Центральним елементом навчального процесу є сховище useLessonStore, яке містить усі дані, необхідні для проведення уроку: метадані, перелік завдань, індекс активного завдання, буфер помилок, ознаку фази корекції та індикатори завершеності. На основі цих даних обчислюються ключові похідні показники, включно з прогресом, точністю та наявністю подальших кроків навчання.

Логіка уроку реалізована як керований послідовний процес, що складається з етапу ініціалізації, основної фази виконання завдань та фази корекції помилок. У разі неправильних відповідей завдання накопичуються у спеціальному буфері й опрацьовуються повторно після завершення основної частини уроку. Така структура забезпечує адресну роботу над слабкими

місцями студента та формує персоналізований навчальний маршрут. Узагальнену логіку переходів між станами подано на рисунку 2.7, що ілюструє внутрішню динаміку навчальної сесії в межах useLessonStore.

Завдяки такому підходу система гарантує послідовну корекцію помилок та передбачувану поведінку під час проходження уроку, а централізація логіки у сховищі спрощує супровід, розширення функціональності та модульне тестування.

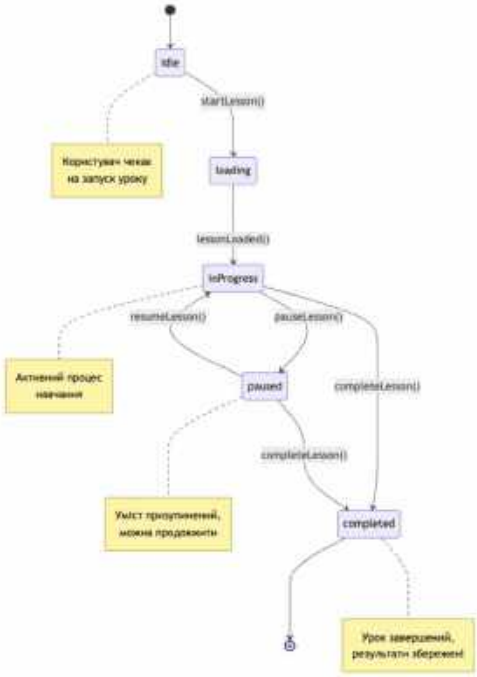

Рисунок 2.7 – Схема станів навчальної сесії в useLessonStore

Архітектура клієнта доступу до API ґрунтується на централізованому HTTP-клієнті, створеному на базі Axios, що виступає єдиною точкою взаємодії клієнтської частини із сервером. У конфігурації клієнта визначаються базова адреса API, типові заголовки та розширені параметри тайм-ауту для ендпоїнтів,

пов'язаних із генерацією навчальних завдань. Механізм перехоплювачів забезпечує автоматичне додавання JWT-токена до кожного запиту та реалізацію логіки оновлення токенів у разі отримання відповіді 401, що дозволяє гарантувати безперервність роботи користувача й уникати дублювання перевірок авторизації у багатьох компонентах.

Поверх цього клієнта організовано доменні сервіси, включно з authService, який відповідає за вхід, реєстрацію, оновлення токенів, вихід із системи, роботу з підтвердженням електронної пошти та відновленням пароля. Аналогічна структура застосована для модулів, що працюють із курсами, уроками та профілем користувача, забезпечуючи чітке розмежування між транспортним рівнем і бізнес-логікою. Узагальнену взаємодію між apiClient, Pinia-сховищем аутентифікації та сервісом authService подано на рисунку 2.8.

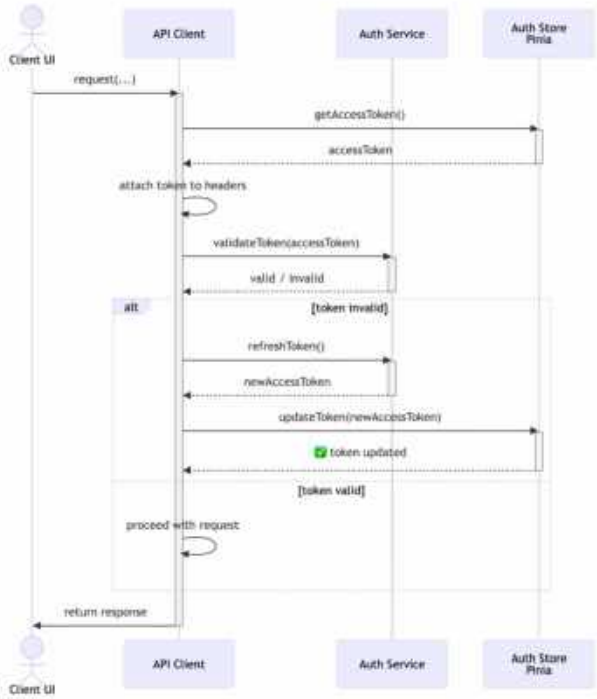

Рисунок 2.8 – Схема взаємодії apiClient з Pinia-сховищем аутентифікації та сервісом authService

Компонентна архітектура клієнтського інтерфейсу побудована за підходом поділу на контейнерні та презентаційні компоненти. Центральним елементом є контейнер навчальної сесії LessonSessionView, який відповідає за ініціалізацію уроку, визначення його поточного стану (основна фаза, корекція, завершення), а також за обробку подій, що надходять від дочірніх компонентів. Усередині нього відображаються компоненти завершення уроку, поточного завдання, індикатор завантаження та елементи, пов'язані з переходом до фази корекції.

Типи навчальних завдань реалізовані як спеціалізовані презентаційні компоненти, наприклад TaskMultipleChoice, який приймає структуру завдання, обробляє вибір користувача, відображає правильні та неправильні відповіді й генерує подію submit для контейнера. Така модель забезпечує чітке розмежування відповідальності: бізнес-логіка зосереджена у контейнерних компонентах, тоді як презентаційні залишаються універсальними та повторно використовуваними UI-елементами. Узагальнену ієрархію цих компонентів наведено на рисунку 2.9.

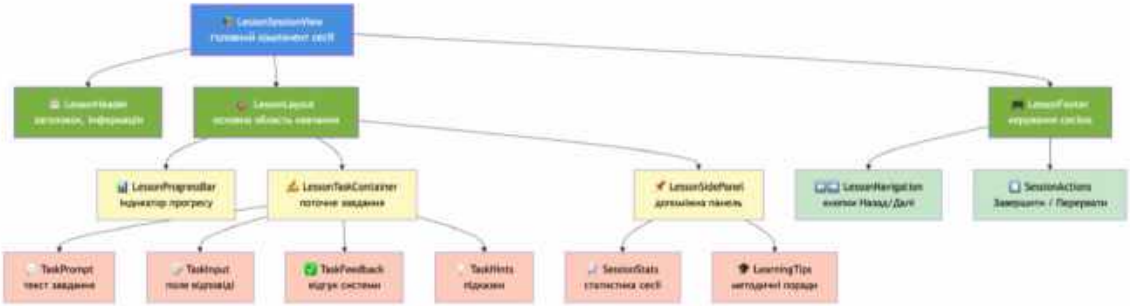

Рисунок 2.9 – Ієрархія компонентів навчальної сесії (LessonSessionView та ключові дочірні компоненти)

Було розглянуто архітектуру клієнтської частини системи адаптивного вивчення іноземних мов. Запропонована організація frontend-монорепозиторію забезпечує спільне використання типів, сервісів і станів між веб- та мобільною версією, що підвищує узгодженість логіки та зменшує витрати на розроблення. Архітектура керування станом на базі Pinia підтримує інтелектуальний сценарій проходження уроку з окремою фазою корекції помилок, а централізований HTTP-клієнт полегшує авторизацію й обробку помилкових ситуацій. Компонентна модель інтерфейсу із чітким розподілом відповідальності підсилює модульність та спрощує супровід.

У сукупності ці рішення формують гнучку, масштабовану й придатну до подальшого розвитку клієнтську архітектуру, яка повністю відповідає вимогам попередніх підрозділів і забезпечує основу для реалізації якісного адаптивного навчального досвіду на різних платформах.

2.1.4 Проєктування AI Integration Layer

AI Integration Layer являє собою окремий логічний компонент системи, призначений для взаємодії з великими мовними моделями та генерації персоналізованого навчального контенту. Його завдання полягає у перетворенні бізнес-запитів (тип завдань, тема уроку, рівень користувача) на стандартизовані виклики LLM-провайдерів із подальшою валідацією та кешуванням результатів.

Архітектура шару включає кілька ключових підсистем. TaskGenerationService отримує від LessonService інформацію про теми уроку та формує запити на генерацію завдань. AiGenerationService відповідає за побудову коректних промптів, формування JSON Schema очікуваної відповіді, виконання викликів до мовних моделей та обробку результатів. CachingManager забезпечує поєднання швидкого кешу Redis із довгостроковим зберіганням у PostgreSQL для зменшення кількості звернень до LLM. LlmOrchestrator обирає

відповідного провайдера (Gemini, GPT тощо) з урахуванням вартості, доступності та робочого сценарію, а клієнти LLM-платформ інкапсулюють технічні деталі взаємодії.

Послідовність генерації завдань включає перевірку наявності кешу, формування промпту, оркестрацію виклику LLM, повторні спроби при помилках, валідацію відповіді та збереження результату у базу даних. Узагальнений процес наведено на рисунку 2.10.

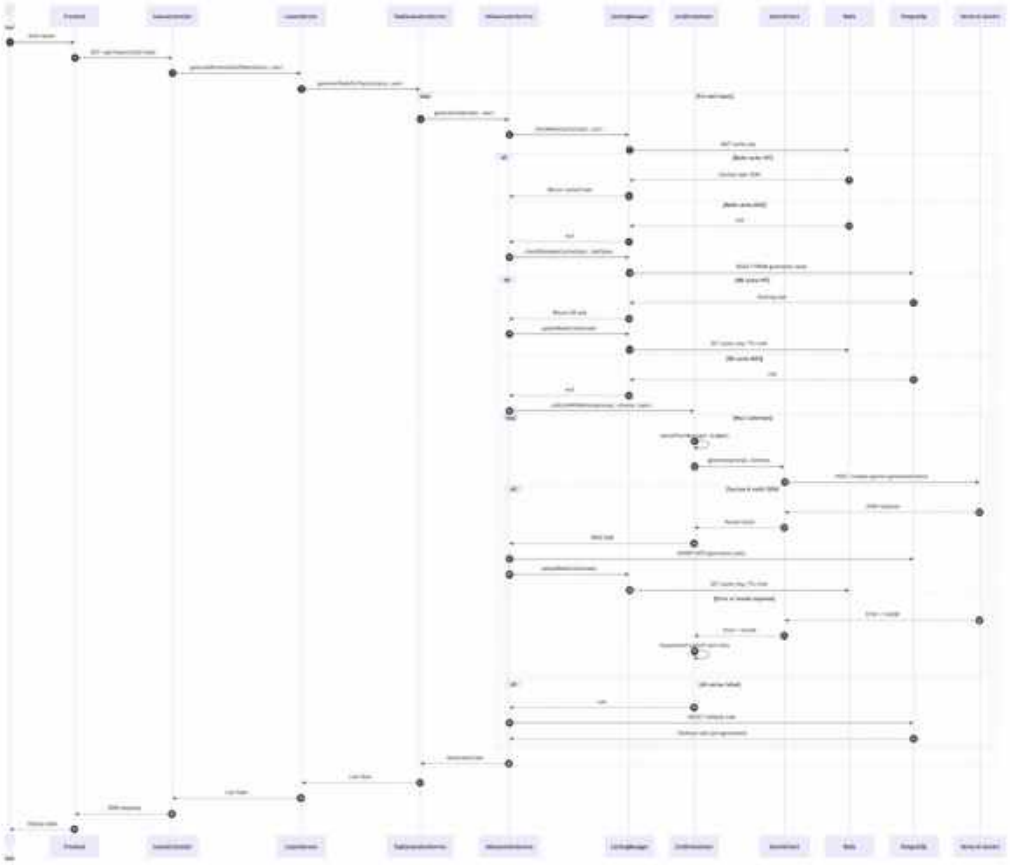

Рисунок 2.10 – Діаграма послідовності генерації навчального завдання із застосуванням AI Integration Layer

AiGenerationService є центральним компонентом інтеграції з LLM і зосереджує в собі ключову логіку формування промптів, управління викликами моделей та обробки відповідей. Сервіс реалізує механізм повторних спроб із експоненційною затримкою, що дає змогу стабілізувати роботу в умовах мережевих збоїв або перевищення лімітів. У разі вичерпання спроб генерується контрольований виняток, що дозволяє системі перейти до резервних сценаріїв використання наперед сформованого контенту.

Важливою частиною роботи сервісу є автоматична генерація JSON Schema на основі доменних Java-класів. Це забезпечує синхронізацію моделі даних зі структурою відповіді LLM і дає можливість вбудовувати схему у системний промпт як обов'язковий контракт. Формування промптів виконується з урахуванням мовних параметрів користувача, рівня володіння мовою та теми уроку, а також включає інструкції щодо жорсткого дотримання JSON-формату без додаткових пояснень чи markdown-обгорток.

Інтеграція з Vertex AI Gemini реалізована через офіційний SDK, що передбачає створення моделі з потрібними параметрами генерації, ініціацію сесії та отримання текстової відповіді для подальшого парсингу. Очищення та розбір відповіді здійснюються окремим модулем, який видаляє зайві обгортки, виділяє валідний JSON-фрагмент і перетворює його на типізований Java-об'єкт. У випадку невалідної відповіді знову активується механізм повторних викликів.

Усе це дозволяє AiGenerationService повністю ізолювати складну логіку роботи з мовними моделями, надаючи іншим частинам системи простий інтерфейс на кшталт callAiWithRetries(message, TargetClass, topic). Детальний програмний код класу наведено в додатку А.1.

Оскільки звернення до LLM-провайдерів є дорогими та мають відчутні затримки, у системі застосовано багаторівневу схему кешування, реалізовану в сервісі CachingManager. Її логіка поєднує швидкий кеш у Redis і довгострокове

зберігання згенерованих завдань у PostgreSQL. Для кожної комбінації параметрів (мова, рівень, тема, тип завдання) формується уніфікований ключ, за яким Redis зберігає серіалізоване завдання з TTL 24 години. Під час запиту система спершу перевіряє Redis, а у разі відсутності – звертається до бази даних, де шукає раніше згенероване й придатне завдання. Знайдений запис використовується повторно, оновлюється в статистиці використання та повторно кешується в Redis. Для підтримання актуальності передбачено механізм інвалідації кешу при оновленні навчального контенту.

Ефективність підходу оцінено на умовному сценарії з 1000 активних користувачів на день. За відсутності кешу обсяг генерацій сягає 20 000 завдань щодня, що приблизно відповідає 2 USD витратам або 60 USD на місяць. При середньому показнику cache hit близько 70 % платними залишаються лише 30 % запитів, що зменшує денні витрати до 0,6 USD, а місячні – до 18 USD. Отже, описана стратегія забезпечує приблизно 70 % економії коштів і значно скорочує середній час відповіді для повторюваних запитів.

Таблиця 2.4 – Порівняння вартості генерації завдань із використанням LLM без кешування та з багаторівневим кешуванням (денні та місячні витрати)

Показник	Без кешування	З багаторівневим кешуванням	Економія
Кількість завдань на добу (шт)	20 000	20 000	
Частка кеш-hit (%)	0	70	
Платних генерацій на добу (шт)	20 000	6 000	14 000
Вартість однієї генерації, USD	0.0001	0.0001	
Денні витрати, USD	2.00	0.60	1.40
Місячні витрати (30 днів), USD	60.00	18.00	42
Загальна економія (%)	–	–	70

Аналіз даних таблиці 2.4 демонструє, що багаторівневе кешування істотно зменшує витрати на генерацію завдань, скорочуючи кількість платних викликів до LLM приблизно на 70 %. Денна вартість зменшується з

2 до 0,60 USD, а місячна – з 60 до 18 USD. Поєднання швидкого кешу в Redis і довгострокового зберігання в PostgreSQL забезпечує економічність і дозволяє масштабувати систему без відповідного зростання фінансових витрат.

2.1.5 Проектування системи безпеки

Система безпеки побудована відповідно до сучасних практик захисту веб-застосунків і включає багаторівневу модель аутентифікації, авторизації, контролю доступу, rate limiting та захисту API. Основними вимогами до цього компонента є висока стійкість до зовнішніх атак, відсутність збереження сеансів на сервері, мінімізація ризику компрометації токенів та можливість масштабування.

Для аутентифікації використовується JWT-орієнтована модель із двома видами токенів: короткоживучим access token (15 хв) та довгоживучим refresh token (7 днів із ротацією при кожному оновленні). Це забезпечує відсутність серверної сесії та повну сумісність із горизонтально масштабованим backend-кластером.

Процес аутентифікації відбувається за такою схемою:

– користувач надсилає запит /auth/login із обліковими даними;

– сервер виконує перевірку пароля (BCrypt) і формує пару токенів;

– клієнт зберігає токени локально (in-memory або LocalStorage);

– усі подальші запити підписуються заголовком Authorization: Bearer token;

– після завершення терміну дії access-токена клієнт звертається до /auth/refresh;

– сервер перевіряє refresh-токен у БД/Redis, генерує нову пару токенів та повертає її клієнту;

– клієнт прозоро повторює первинний запит.

Узагальнену послідовність взаємодії під час аутентифікації за допомогою пари токенів наведено на рисунку 2.11.

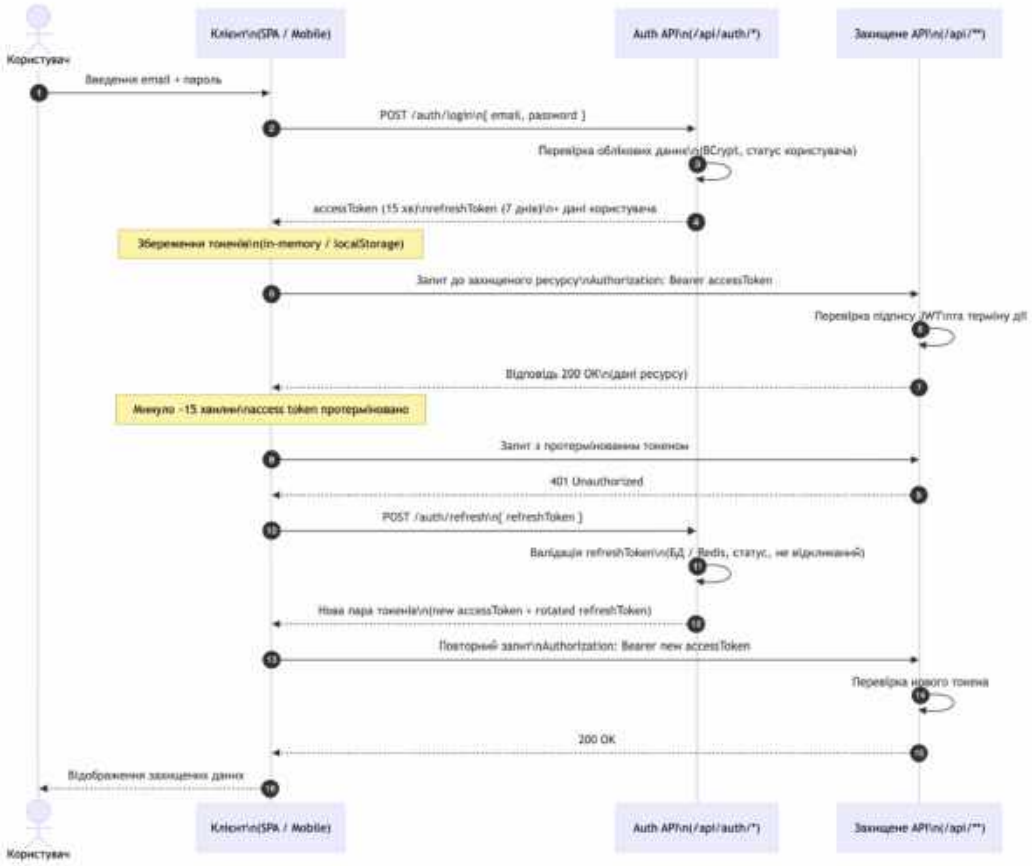

Рисунок 2.11 – Узагальнена послідовність процесу аутентифікації з використанням JWT та refresh-токенів

Генерація та валідація JWT-токенів реалізована у спеціалізованому сервісі JwtService, який відповідає за формування пари access/refresh токенів на основі алгоритму HS256, додавання службових claims та коректне встановлення часу їх життя. Під час обробки запитів сервіс здійснює перевірку підпису,

визначає чинність токена, витягує ім'я користувача та контролює можливі виключні ситуації, пов'язані з підробкою або простроченням.

Зосередження цієї логіки в одному компоненті забезпечує цілісність механізму аутентифікації, спрощує інтеграцію з іншими модулями системи та гарантує єдину політику обробки та перевірки токенів. Детальний програмний код сервісу наведено у Додатку А.2 (JwtService).

Конфігурація безпеки backend-застосунку зосереджена в класі SecurityConfig, який визначає політики доступу для REST-API, включно з вимкненням CSRF, налаштуванням CORS для веб- та мобільного клієнтів, розмежуванням відкритих та захищених ендпоінтів, а також інтеграцією JWT-аутентифікації через фільтр JwtAuthenticationFilter. Застосування політики SessionCreationPolicy.STATELESS та використання DaoAuthenticationProvider із BCrypt-хешуванням забезпечує відповідність сучасним вимогам безпеки та можливість масштабування без збереження серверних сесій.

У системі реалізовано Redis-орієнтований механізм rate limiting, який запобігає надмірному використанню ресурсів і зловживанням API, зокрема дорогої генерації контенту за допомогою LLM. Контроль охоплює два основні канали: обмеження запитів до сервісів AI-генерації та регулювання загального API-трафіку, що дозволяє знижувати ризики DDoS-подібних атак, brute-force спроб і масових викликів обчислювально дорогих операцій. Логіка базується на інкрементуванні лічильників у Redis із відповідними періодами дії, а при перевищенні встановлених порогів сервер повертає код 429 Too Many Requests.

Запроєктована система безпеки забезпечує комплексний захист клієнтсько-серверної архітектури, поєднуючи сучасний підхід до аутентифікації на основі JWT, гнучку конфігурацію політик доступу, ефективний контроль частоти запитів і розмежування ролей користувачів. Використання stateless-моделі та зовнішніх інструментів кешування (Redis)

підвищує масштабованість і стійкість системи, а розмежування access та refresh токенів мінімізує ризики компрометації. У результаті сформовано безпечну, керовану й придатну до розширення інфраструктуру захисту.

Отже, було здійснено комплексне проєктування архітектури кросплатформного мобільного застосунку для адаптивного вивчення іноземних мов. Запропонована клієнт–серверна модель із виділеним шаром інтеграції штучного інтелекту забезпечує модульність, розширюваність та чітке розмежування відповідальності між компонентами. Організація фронтенду у форматі монорепозиторію та використання спільного шару логіки дають змогу мінімізувати дублювання коду й уніфікувати поведінку веб- та мобільних клієнтів. Архітектура управління станом підтримує повний життєвий цикл навчальної сесії, включно з корекцією помилок, що є ключовим для реалізації адаптивного підходу.

Підсистема генерації навчальних завдань на основі LLM із механізмами повторних спроб, структурної валідації та автоматичного формування JSON-схем забезпечує надійність інтеграції зі штучним інтелектом, а багаторівневе кешування на основі Redis і PostgreSQL істотно знижує кількість викликів до моделей і підвищує продуктивність системи. Побудована на JWT-парі токенів система аутентифікації та механізми rate limiting гарантують безпечний доступ і захист від зловживань. Шарова організація backend-компонентів та структурована модель даних забезпечують зручність підтримки, тестованість і можливість масштабування.

У сукупності розроблені рішення формують цілісну, розширювану та продуктивну архітектурну основу, достатню для переходу до детального проєктування окремих модулів та подальшої реалізації програмного забезпечення.

2.2 Реалізація системи

Для реалізації системи адаптивного навчання було сформовано уніфіковане середовище розробки, яке охоплює інструменти для бекенд-, фронтенд- та мобільної розробки. Основними вимогами до інструментарію є підтримка сучасних стандартів, стабільність версій та сумісність із обраними фреймворками.

Серверна частина системи розроблялась з використанням Java 17, середовища збірки Maven та середовища розробки IntelliJ IDEA. Для забезпечення роботи підсистеми збереження даних застосовано PostgreSQL 16, а для реалізації кешування – Redis 7.2. Контроль версій здійснювався за допомогою Git.

Фронтенд-частина функціонує на платформі Node.js 20 LTS з менеджером пакунків pnpm, що забезпечує оптимізацію роботи монорепозиторію. Розробку здійснено у Visual Studio Code з використанням інструментів аналізу коду та розширеннями для екосистеми Vue 3.

Для мобільної частини застосовано NativeScript у поєднанні з Vue, що дало змогу реалізувати кросплатформний інтерфейс. За потреби забезпечується робота на емуляторах iOS та Android.

Підтримку контейнеризації та локальної інфраструктури забезпечили Docker та Docker Compose, а для хмарної інтеграції із сервісами генерації контенту використовувалась платформа Google Cloud Platform з увімкненим Vertex AI.

Архітектура системи реалізована у форматі двох окремих репозиторіїв – бекенд та фронтенд – з чітким розмежуванням відповідальностей.

Бекенд-проєкт побудований за принципами шарової архітектури та містить модулі конфігурації, контролерів, сервісів, репозиторіїв, сутностей та

DTO-класів. В окремий модуль виділено компоненти, що відповідають за інтеграцію з мовними моделями, включно з сервісами генерації завдань, керування кешем та оркестрацією запитів до LLM. Для управління схемою бази даних застосовано Flyway, а конфігураційні дані структуровано у YAML-файлах.

Фронтенд реалізований як монорепозиторій, що містить три модулі: спільну бібліотеку (shared), веб-інтерфейс та мобільний застосунок. Спільний модуль містить типи, API-клієнти, Pinia-сховища та утилітарні функції, які використовуються одночасно у веб- та мобільній версіях. Це забезпечує єдину модель даних і значне скорочення дублювання коду.

Розгортання середовища розробки передбачає встановлення залежностей, налаштування бази даних та зовнішніх сервісів. Для бекенд-частини передбачено автоматичне завантаження залежностей за допомогою Maven, виконання міграцій Flyway та створення необхідних конфігураційних файлів для роботи з Vertex AI. Фронтенд-частина інсталюється через pnpm, після чого виконується конфігурація файлу .env та запуск локального сервера розробки.

Для локального розгортання сервісів бази даних та кешування використовується Docker Compose, який піднімає контейнери PostgreSQL та Redis, а також (опційно) PgAdmin для адміністрування. Такий підхід гарантує відтворюваність середовища та уніфікацію налаштувань для всіх учасників проєкту.

2.2.1 Реалізація backend-компонентів

Конфігураційні параметри серверної частини зосереджені в головному файлі application.yml, який визначає підключення до бази даних, поведінку JPA, параметри Redis-кешу, міграції Flyway, а також налаштування безпеки та інтеграції з хмарними сервісами. Для керування середовищами

використовується механізм профілів Spring (dev, prod), що дає можливість відокремити робочі параметри від розробницьких.

У блоці spring.datasource описано підключення до PostgreSQL із використанням пулу з'єднань HikariCP (обмеження на кількість з'єднань, тайм-аути, максимальний час життя з'єднання). Блок spring.jpa задає використання діалекту PostgreSQL, заборону автоматичної зміни схеми (ddl-auto: validate) та делегує еволюцію схеми Flyway-скриптам (spring.flyway.locations). Окремий розділ spring.data.redis містить параметри підключення до Redis (хост, порт, пароль, параметри пулу підключень), що використовується для кешування та rate limiting.

Конфігурація JWT-аутентифікації задається у секції jwt: секретний ключ підпису, час життя access- та refresh-токенів. Параметри інтеграції з Vertex AI описані в розділі google.cloud та ai.model, де визначено ідентифікатор проєкту, регіон і базові параметри генерації (модель, температура, максимальна кількість токенів). Додатково, у блоці ai.cost та ai.rate-limit задаються денний бюджет на виклики LLM і обмеження на кількість запитів, що використовуються в сервісах моніторингу витрат і обмеження трафіку.

Розділ app містить приклад прикладного налаштування: перелік дозволених джерел для CORS (клієнтські застосунки) та шлях до файлу навчального курикулуму у форматі YAML. Параметри логування (logging.level, logging.pattern) визначають деталізацію журналів для основного пакета застосунку, безпекових компонентів та ORM-рівня. Нарешті, секції management та springdoc активують технічні ендпоїнти Spring Actuator (health, metrics, prometheus) і автоматично згенеровану документацію OpenAPI/Swagger.

Для спрощення розробки використовуються окремі профілі application-dev.yml і application-prod.yml. Розробницький профіль увімкнює детальне логування та знижує денний бюджет на використання LLM, тоді як

промисловий профіль налаштований на мінімізацію обсягу логів, вищий бюджет та обмежений набір опублікованих технічних ендпоїнтів. Повні лістинги конфігураційних файлів застосунку, включно з основним файлом application.yml та профілями середовищ application-dev.yml і application-prod.yml, наведено у додатку Б.1.

Модель даних серверної частини ґрунтується на JPA-сутностях, що відображають основні поняття предметної області. Базовою сутністю користувача є клас User, який містить ідентифікатор, адресу електронної пошти, хешований пароль, повне ім'я, роль (користувач, адміністратор, автор контенту), стан верифікації облікового запису та службові часові мітки створення й оновлення. Сутність реалізує інтерфейс UserDetails, що забезпечує інтеграцію з підсистемою Spring Security (отримання ролей, відстеження стану облікового запису тощо). З користувачем пов'язані допоміжні сутності UserProfile та UserStats, які зберігають детальні профільні дані та статистику навчання.

Сутність Topic репрезентує навчальну тему (лексичну чи граматичну одиницю) і містить унікальний ідентифікатор, назву, мову вивчення, мову походження, рівень володіння (наприклад, A1, A2) та опис. З темою пов'язуються два типи дочірніх сутностей: правила генерації завдань (TopicGenerationRule) та згенеровані завдання (GeneratedTask). Це дозволяє зберігати як декларативні інструкції для LLM, так і результати їх роботи.

Сутність GeneratedTask реалізує шар кешування навчальних завдань на рівні бази даних. Вона містить посилання на тему, тип завдання, складність, лічильник повторного використання, часові мітки створення й останнього використання. Поле з даними завдання зберігається у форматі jsonb, що дає змогу гнучко працювати з різними структурами генерованих завдань без зміни

схеми. Такий підхід спрощує зберігання результатів LLM-генерації та підтримує еволюцію формату завдань.

Доступ до бази даних реалізовано через інтерфейси Spring Data JPA, що інкапсулюють типові операції CRUD і дозволяють оголошувати запити на основі імен методів або через анотацію @Query. Для сутності Topic використовується репозиторій TopicRepository, який, окрім базових операцій, надає методи пошуку тем за ідентифікатором, мовою та рівнем, а також запит із підвантаженням пов'язаних правил генерації. Окремий метод findTopicsForCourse повертає відсортований список тем для курсу певної мови та рівня, що спрощує формування навчального плану.

Репозиторій GeneratedTaskRepository відповідає за роботу з таблицею кешованих завдань. Ключовим є метод findReusableTask, який повертає завдання для повторного використання, віддаючи пріоритет найменш часто вживаним і давно використаним записам. Це забезпечує рівномірний розподіл навантаження між наявними згенерованими завданнями. Додаткові методи дають змогу отримувати всі завдання для теми, підраховувати їх кількість за типом, а також видаляти застарілі невикористані записи (наприклад, старші за 30 днів із нульовим лічильником повторного використання). Такий набір операцій підтримує реалізовану стратегію багаторівневого кешування, описану у попередньому розділі.

Загалом, поєднання чітко структурованих конфігурацій, типізованих сутностей та репозиторіїв Spring Data забезпечує прозорий та розширюваний доступ до даних, що є основою для побудови адаптивної логіки навчання та інтеграції з AI-компонентами системи.

2.2.2 Реалізація сервісного шару

Сервісний шар інкапсулює бізнес-логіку та забезпечує взаємодію між контролерами, репозиторіями, модулем генерації завдань на основі ШІ та системою відстеження прогресу користувача. У цьому підрозділі розглядаються три ключові сервіси: TaskGenerationService, LessonService та UserStatsService.

Сервіс TaskGenerationService координує повний цикл створення персоналізованих навчальних завдань, поєднуючи роботу мовної моделі з механізмами дворівневого кешування. Він отримує перелік тем уроку, аналізує профіль користувача та ініціює генерацію відповідного набору завдань, визначаючи їхню кількість і типи згідно з наперед заданими ваговими коефіцієнтами у правилі генерації. Під час роботи сервіс формує контекстно-залежний промпт, що передається мовній моделі, а також здійснює взаємодію з Redis і базою даних для повторного використання вже згенерованого контенту, мінімізуючи кількість викликів LLM і підвищуючи продуктивність системи.

Метод generateTasksForLesson приймає об'єкт уроку та користувача, отримує пов'язаний перелік тем і для кожної теми викликає внутрішній метод generateTasksForTopic. За відсутності тем сервіс повертає порожній список, що запобігає зайвим викликам підсистеми ШІ. Після генерації всі завдання перемішуються, що підвищує варіативність послідовності вправ для студента.

Метод generateTasksForTopic використовує правило генерації TopicGenerationRule, пов'язане з темою, та обчислює кількість завдань кожного типу (MULTIPLE_CHOICE, TRANSLATION тощо) на основі діапазону [minTasks; maxTasks] і вагових коефіцієнтів TopicTaskTypeWeight. У разі відсутності налаштованих ваг використовується рівномірний розподіл із дефолтним типом завдань. Розрахунок виконується в методі calculateTaskTypeCounts, який перетворює ваги на кількість завдань кожного типу.

Генерація окремого завдання виконується методом generateSingleTask, який спочатку формує уніфікований ключ кешу на основі теми та типу завдання. Після цього сервіс послідовно перевіряє наявність уже згенерованих даних: спершу звертається до Redis як шару короткострокового кешування, а за його відсутності – до бази даних, де зберігаються завдання з довшим горизонтом використання. Лише у випадку, коли обидва рівні кешу не містять відповідного запису, формується запит до мовної моделі для генерації нового завдання.

У разі вдалого знаходження об'єкта повертається відповідь із кешу, що суттєво знижує латентність та вартість викликів до LLM. Якщо ж кеш-промах фіксується на обох рівнях, сервіс переходить до фактичного виклику моделі ШІ через метод generateTaskWithAi.

Метод generateTaskWithAi формує промпт за допомогою buildTaskPrompt, створює UserMessage і викликає AiGenerationService з механізмом повторних спроб (callAiWithRetries). У разі успішної десеріалізації відповідь приводиться до типу TaskResponse, доповнюється інформацією про тип завдання і зберігається в кеші через CachingManager.

Функція buildTaskPrompt реалізована як контекстно-залежний конструктор промптів: до базового системного опису завдання, що включає його тип, рівень та назву теми, динамічно додаються дані профілю конкретного користувача, зокрема рідна мова, навчальні цілі та індивідуальні інтереси. Для різних типів завдань промпт доповнюється спеціальними інструкціями. Наприклад, у випадку завдань типу MULTIPLE_CHOICE зазначаються кількість варіантів відповіді (чотири), вимога мати лише одну правильну відповідь, критерії правдоподібності дистракторів та необхідність надання короткого пояснення до правильної відповіді. Для завдань типу TRANSLATION промпт містить вказівку щодо напрямку перекладу,

відповідний контекст і приклад речення. Така адаптивна побудова промптів забезпечує більш точні, структуровані та релевантні відповіді мовної моделі.

Сервіс LessonService відповідає за повний життєвий цикл уроку – від отримання його даних до збереження результатів та оновлення прогресу користувача. Він взаємодіє з TaskGenerationService, UserStatsService і репозиторіями уроків, забезпечуючи чітке розмежування відповідальностей. Метод getLessonDetails, який працює в режимі лише читання, завантажує урок за ідентифікатором, перевіряє його існування та проєціює сутність у DTO LessonDetailsResponse (назва, опис, орієнтовна тривалість, кількість тем). Якщо урок відсутній, піднімається виняток ResourceNotFoundException.

Формування персоналізованого набору завдань здійснюється у generateLessonTasks: сервіс завантажує урок разом із темами, після чого делегує їх обробку TaskGenerationService. Це дозволяє повністю ізолювати роботу з мовною моделлю від доменної логіки LessonService та підтримувати чисту архітектуру.

Завершення уроку обробляється методом completeLesson, який створює запис UserLessonCompletion з показниками точності, кількістю правильних відповідей, тривалістю сесії, XP та інформацією про корекційну фазу. Після збереження запису оновлюється загальна статистика користувача, а також рівень опанування тем. Розрахунок досвіду враховує базову кількість завдань, множник серії (streak), що зростає з кожним днем активності, та множник точності, який може збільшувати отримані бали до 50 %, створюючи гнучку систему мотивації.

Окремою частиною логіки є оновлення рівня опанування кожної теми. Якщо запис UserTopicMastery ще не існує, створюється новий, після чого рівень коригується за формулою експоненційного ковзного середнього (EMA) формула 2.1:

$$M_{new} = \alpha \cdot A_{recent} + (1 - \alpha) \cdot M_{curent}, \quad (2.1)$$

де M_{new} – оновлений рівень опанування теми;

$\alpha = 0{,}3$ – ваговий коефіцієнт, що визначає вплив останнього результату;

A_{recent} – точність виконання останньої вправи (значення від 0 до 1);

M_{curent} – поточний рівень опанування теми до оновлення.

Для побудови аналітики LessonService також надає метод getUserLessonHistory, який повертає впорядкований список проходжень уроків користувачем. Це дає змогу аналізувати часові ряди, будувати графіки прогресу і відстежувати навчальну динаміку.

Сервіс UserStatsService відповідає за комплексне оновлення прогресу користувача після завершення уроку: він перераховує досвід (XP), визначає можливість підвищення рівня, підтримує навчальну серію (streak), оновлює загальну статистику й перевіряє умови здобуття досягнень. Центральною точкою цієї логіки є метод updateStatsAfterLessonCompletion, який у межах транзакції створює об'єкт UserStats (якщо він ще відсутній), додає отриманий за урок досвід, перевіряє можливість підвищення рівня, оновлює навчальну серію на основі дати останньої активності та збільшує сумарну кількість завершених уроків і час навчання. Після збереження змін ініціюється перевірка досягнень, що забезпечує цілісність оновлення статистики.

Підвищення рівня реалізовано через функцію checkLevelUp, яка порівнює накопичений досвід із пороговим значенням, необхідним для переходу на наступний рівень. Порогове значення XP для рівня L визначається формулою 2.2:

$$XP_L = 100 \cdot L^{1.5}, \quad (2.2)$$

де XP_L – порогове значення досвіду, необхідне для досягнення рівня L;

L – номер рівня знань (ціле число, починаючи з 1);

100 – базова константа, що визначає складність першого рівня;

1,5 – показник степеня, що забезпечує експоненціальне зростання вимог з кожним рівнем, що забезпечує помірне, але прогресивно зростаюче ускладнення. Якщо досвіду достатньо, рівень підвищується, після чого перевірка повторюється, тому можливий перехід через кілька рівнів за один урок.

Оновлення навчальної серії (updateStreak) базується на порівнянні поточної дати з датою останньої активності: якщо користувач виконує урок уперше, серія ініціалізується; якщо активність повторюється того самого дня, значення streak не змінюється; якщо попередня активність була вчора – серія збільшується та, за потреби, оновлюється рекорд найдовшої серії; у разі пропуску днів streak скидається, але максимальне значення зберігається. Така логіка відображає природну динаміку користувацької поведінки та підтримує мотиваційний ефект системи.

Перевірка досягнень здійснюється у методах checkAchievements та checkAchievementCondition. Система завантажує перелік доступних досягнень, визначає, які з них користувач ще не отримував, і порівнює їх умови з актуальними даними статистики – від першого завершеного уроку до досягнення високої серії або рівня. У разі виконання умов виконується unlockAchievement, який додає бонусний XP, зберігає оновлену статистику та фіксує подію отримання досягнення. Архітектура сервісу передбачає подальше розширення логіки, зокрема інтеграцію з інтерфейсом мобільного застосунку через систему сповіщень або гейміфікаційних елементів.

2.2.3 Реалізація шару контролерів

Шар контролерів забезпечує публічний REST-інтерфейс мобільного застосунку, інкапсулюючи HTTP-рівень і делегуючи бізнес-логіку сервісам. Для проєкту виділено три ключові компоненти: контролер аутентифікації (AuthController), контролер управління уроками (LessonController) та глобальний обробник помилок (GlobalExceptionHandler).

Контролер AuthController надає REST-ендпоїнти для реєстрації користувача, входу в систему, оновлення access-токена, виходу та підтвердження електронної пошти. Він приймає DTO-об'єкти запитів (RegisterRequest, LoginRequest, RefreshTokenRequest), делегує обробку сервісу AuthService та повертає стандартизовану відповідь AuthResponse. Для документування API використовується OpenAPI-анотації (@Operation, @Tag). Реалізація контролера наведена у Додатку А.3.

Контролер LessonController відповідає за операції, пов'язані з уроками: отримання детальної інформації про урок, генерацію персоналізованих завдань та фіксацію завершення уроку. Він працює під JWT-захистом (@SecurityRequirement("bearer-auth")) і отримує поточного користувача через @AuthenticationPrincipal. Для обмеження частоти звернень до підсистеми ШІ застосовується сервіс RateLimitingService, який у разі перевищення ліміту повертає код відповіді 429 Too Many Requests. Бізнес-логіка взаємодії з уроками делегується сервісу LessonService.

Компонент GlobalExceptionHandler реалізує централізовану обробку винятків на рівні REST-шару за допомогою @RestControllerAdvice. Для типових ситуацій (відсутність ресурсу, помилки генерації ШІ-контенту, некоректні облікові дані, помилки валідації вхідних даних) формуються уніфіковані відповіді з використанням DTO ErrorResponse, що містить час виникнення помилки, HTTP-статус, короткий опис і деталізоване повідомлення.

Загальний обробник @ExceptionHandler(Exception.class) гарантує, що необроблені винятки перетворюються на відповідь із кодом 500 Internal Server Error, а інформація про інцидент журналюється.

2.2.4 Реалізація фронтенд-компонентів

Фронтенд-частина системи реалізована на основі стеку Vue 3 + TypeScript + Pinia, із виділеним спільним шаром доступу до API, типізованими моделями предметної області та компонентами, що забезпечують проходження уроку, відображення завдань і екран завершення. Для зменшення дублювання стилів та підвищення читабельності коду оформлення винесено у SCSS, що дозволило скоротити кількість рядків і чітко структурувати правила стилізації.

Спільний API-шар інкапсулює логіку взаємодії з backend-службами та забезпечує єдину точку доступу до HTTP-запитів.

HTTP-клієнт apiClient побудовано на базі бібліотеки axios. Він налаштований на роботу з базовим URL, що зчитується з конфігурації оточення, та використовує перехоплювачі запитів і відповідей. На етапі запиту до заголовків автоматично додається JWT-токен, отриманий з локального сховища. На етапі відповіді реалізовано механізм прозорого оновлення токена: у разі коду 401 Unauthorized викликається endpoint оновлення, збережені токени оновлюються, а початковий запит повторюється. У випадку невдачі ініціюється подія auth:logout, що призводить до розлогінювання користувача на клієнті. Лістинг HTTP-клієнта наведено у додатку А.4.

Сервіс lessonService інкапсулює виклики до REST-endpoint'ів, пов'язаних з уроками: отримання деталей уроку, генерацію персоналізованих завдань, надсилання результатів завершення та отримання історії проходження. Методи сервісу повертають типізовані об'єкти, узгоджені з backend-DTO.

Для забезпечення статичної типізації та узгодженості між фронтендом і бекендом у спільному модулі визначено окремі TypeScript-типи для завдань, уроків та досягнень.

Файл Task.ts описує перелік підтримуваних типів завдань (TaskType), структуру базового завдання Task (питання, варіанти відповідей, правильна відповідь, пояснення, рівень складності, метадані), а також моделі TaskAttempt та TaskResponse, які використовуються для фіксації спроб користувача та обробки відповідей, згенерованих ШІ.

Файл Lesson.ts містить опис структури LessonDetails, DTO-моделі LessonCompleteRequest для відправлення результатів уроку та LessonCompleteResponse для відображення нарахованого досвіду, оновленого рівня, серії відвідувань і розблокованих досягнень. Тут же визначено тип Achievement, що використовується на фронтенді для відображення бейджів.

Сховище useAuthStore реалізує клієнтську логіку аутентифікації та управління користувацькою сесією. Воно зберігає поточного користувача, токени доступу й оновлення, стан завантаження та можливі повідомлення про помилки. Обчислювані властивості isAuthenticated та isAdmin спрощують реалізацію умовного відображення компонентів інтерфейсу.

Основні дії сховища (login, register, refreshAccessToken, logout, restoreSession) взаємодіють із відповідним API-сервісом, синхронізують стан із localStorage та дозволяють відновлювати сесію після перезавантаження сторінки. Окремо передбачено обробку події auth:logout, яку генерує HTTP-клієнт у випадку невдалого оновлення токенів.

Основний користувацький сценарій проходження уроку реалізовано за допомогою двох ключових компонентів: LessonItem та LessonComplete.

Компонент LessonItem відповідає за відображення поточного завдання, індикатора прогресу, фази корекції та зони зворотного зв'язку. Тип завдання

визначає вибір відповідного дочірнього компонента (наприклад, TaskMultipleChoice, TaskTranslation), який підключається динамічно за допомогою механізму <component :is>. Після надсилання відповіді компонент перевіряє її коректність, виводить повідомлення про результат та, за необхідності, демонструє правильну відповідь і пояснення. Для стилізації використано окремий SCSS-модуль, що групує правила за блоками (.progress, .task-card, .feedback) і покращує підтримуваність коду.

Компонент LessonComplete формує підсумковий екран завершення уроку: відображає назву уроку, точність виконання, кількість правильних відповідей, нарахований досвід, витрачений час, інформацію про розблоковані досягнення та оновлений показник навчальної серії (streak). Він генерує події continue та review-mistakes, що дозволяє інтегрувати його в загальний сценарій навігації між екранами. Стилі компонента також винесено в SCSS із використанням BEM-подібної структури класів, що спрощує адаптацію інтерфейсу під різні форм-фактори. Повний лістинг компонента подано у додатку А.6.

2.2.5 Реалізація міграцій бази даних

Для забезпечення відтворюваної, керованої та поступової еволюції структури бази даних у проєкті використано механізм версійованих міграцій на основі Flyway. Усі зміни структури БД організовано у вигляді SQL-файлів, що автоматично застосовуються під час старту застосунку. Це гарантує узгодженість даних між сервісами та забезпечує можливість швидкого розгортання локальних і production-середовищ.

Початкова міграція, включає створення основних сутностей системи, індексів та ключових зв'язків між таблицями. Вона охоплює користувацькі дані, профілі, статистику, курси, розділи, уроки, теми, результати проходження уроків та трекінг рівня опанування тем.

Повний лістинг початкової міграції подано у додатку А.6.

Отже, мною було реалізовано повний технологічний стек системи, що охоплює backend-, frontend- та інфраструктурні компоненти. Backend побудовано відповідно до принципів багатошарової архітектури, що забезпечує чіткий поділ відповідальностей та високу підтримуваність коду. Механізми генерації навчального контенту на основі великих мовних моделей реалізовано з використанням повторних спроб, адаптивних промтів та багаторівневого кешування, що підвищує стабільність та ефективність роботи сервісу.

Сервіси управління уроками підтримують повний цикл навчальної взаємодії, включно з обчисленням прогресу, XP-механікою та оновленням рівня опанування тем. Система аутентифікації побудована на основі JWT з підтримкою оновлення токенів, що забезпечує безперервність користувацької сесії. На фронтенді реалізовано модульну структуру з використанням Vue 3, TypeScript та Pinia, що дозволило досягти суттєвої частки повторного використання коду між платформами. Компоненти проходження уроку адаптовано для динамічної роботи з різними типами завдань.

Міграції Flyway забезпечують контрольовану еволюцію бази даних, а централізований обробник помилок у backend гарантує уніфіковані відповіді сервера. У цілому реалізований функціонал відповідає архітектурним вимогам, визначеним у попередньому розділі, та створює завершений інтерфейс для подальшого етапу тестування.

2.3 Планування та управління проєктом

Для розроблення системи адаптивного навчання іноземних мов було обрано Scrum-методологію, що належить до сімейства Agile-підходів і є оптимальною для проєктів з високим рівнем інноваційності, зокрема тих, що включають інтеграцію LLM-компонентів. Гнучка природа Scrum дає змогу

оперативно реагувати на зміну вимог, результати експериментів та зворотний зв'язок кінцевих користувачів. Методологія передбачає ітераційний розвиток продукту через короткі спринти, у межах яких команда створює інкременти, потенційно готові до постачання. Регулярні Scrum-події – планування, щоденні стендапи, огляд та ретроспектива – забезпечують прозорість прогресу, сприяють ранньому виявленню технічних і продуктових ризиків та підтримують стабільний темп роботи. Важливою складовою є взаємодія між Product Owner, Scrum Master та кросфункційною командою розробників, що дозволяє ефективно координувати вимоги, пріоритети та технічні рішення, підтримуючи цілісність продуктового бачення.

Тривалість одного спринту становить два тижні (10 робочих днів), у межах яких реалізується повний цикл Scrum-подій. На початку спринту проводиться Sprint Planning, під час якого команда визначає цілі, відбирає релевантні елементи з product backlog та формує sprint backlog, уточнюючи підходи до їх реалізації й оцінюючи трудомісткість. Щодня відбувається коротка зустріч Daily Scrum, спрямована на синхронізацію роботи команди та оперативне виявлення перешкод. По завершенні ітерації команда проводить Sprint Review, де демонструє виконаний функціонал і отримує зворотний зв'язок від зацікавлених сторін, після чого відбувається Sprint Retrospective – аналіз процесу, формування висновків і визначення напрямів покращення.

Паралельно із цими подіями регулярно виконується Backlog Refinement, під час якого проводиться уточнення вимог, декомпозиція та переоцінювання елементів backlog. Для кожної користувацької історії діє визначений набір критеріїв готовності (Definition of Done), що включає повністю реалізований і перевірений код, наявність модульних та інтеграційних тестів, успішне проходження code review, актуалізовану документацію, розгортання на тестовому середовищі та підтвердження виконання критеріїв приймання.

У межах проєкту оцінювання виконувалося в одиницях story points із застосуванням Fibonaccі-шкали (1, 2, 3, 5, 8, 13), що дає змогу враховувати не лише фактичний обсяг робіт, а й рівень їхньої складності, невизначеності та потенційних ризиків. Такий підхід забезпечує більш реалістичне планування спринтів і дозволяє команді узгоджено визначати відносну трудомісткість завдань, спираючись на колективну експертизу.

Практика оцінювання включає кілька усталених правил. Якщо історія отримує понад 8 SP, це свідчить про надмірну комплексність, тому вона підлягає додатковій декомпозиції перед включенням у спринт. Середня швидкість команди (velocity) після етапу адаптації стабілізувалася на рівні 30-35 SP за спринт, що стало базою для подальшого планування. Водночас близько 20 % місткості кожного спринту резервується під роботу з технічним боргом і виправлення дефектів, що дозволяє підтримувати стабільну якість продукту й уникати накопичення критичних проблем у майбутньому.

2.3.1 Структура Product Backlog

Структура product backlog у проєкті сформована за ієрархічним принципом, що дозволяє послідовно деталізувати вимоги – від бачення продукту до конкретних технічних завдань. На верхньому рівні визначається Product Vision, яке задає стратегічний напрям розвитку системи. Далі backlog структурується у великі функціональні блоки (Epics), що охоплюють логічно пов'язані групи можливостей. Кожен епік розбивається на user stories – конкретні користувацькі сценарії, що можуть бути реалізовані протягом одного або двох спринтів. У свою чергу user stories деталізуються на tasks і sub-tasks, які описують технічні аспекти реалізації.

Ієрархія backlog організована таким чином:

– Product Vision;
– Epics – великі функціональні блоки;

– User Stories – користувацькі сценарії з чіткими критеріями приймання;

– Tasks/Sub-tasks – технічні підзадачі, необхідні для реалізації історій.

У межах цього дослідження сформовано чотири ключові епіки, які охоплюють основну функціональність системи та визначають структуру проєктних робіт:

– EPIC 1 – User Authentication and Profile Management – модуль аутентифікації, керування обліковими записами та профілями користувачів.

– EPIC 2 – Core Content Management System – система адміністрування навчального контенту та структур курсу.

– EPIC 3 – AI-Powered Task Generation – підсистема генерації навчальних завдань за допомогою моделей штучного інтелекту.

– EPIC 4 – Adaptive Learning Engine – реалізація адаптивного навчального механізму, обробки прогресу та персоналізації.

Деталізований опис епіків, його користувацьких історій, критеріїв приймання та розподілу технічних завдань наведено у додатках В.1-В.4.

2.3.2 Планування спринтів

Планування спринтів здійснюється на основі пріоритизації product backlog, урахування технічних залежностей та орієнтовної velocity команди. На першому етапі визначається ключова функціональність, необхідна для запуску базової версії системи, після чого елементи backlog розподіляються між спринтами так, щоб забезпечити поступове нарощування можливостей. Кожен спринт спрямований на реалізацію логічно завершеного блоку роботи, який формує інкремент продукту та наближає команду до готовності основних модулів.

Спринт 1 зосереджений на фундаментальній інфраструктурі й базовій аутентифікації. На цьому етапі важливо створити стабільне середовище

розробки, налаштувати CI/CD та реалізувати ключовий функціонал автентифікації (реєстрація, вхід до системи). Додатково виконується імпорт навчального контенту з YAML-описів, що формує основу подальшої роботи з даними.

Спринт 2 завершує онбординг користувача шляхом впровадження профілю, налаштування навчальних цілей та побудови базових механізмів контент-менеджменту. На цьому етапі створюється структура тем та правил генерації, що забезпечує подальшу персоналізацію навчального процесу.

Спринт 3 спрямований на інтеграцію з LLM-платформою та реалізацію автоматичної генерації структурованого навчального контенту. У межах цього спринту впроваджуються механізми повторних спроб (retry), обробка помилок, а також перші елементи адаптивного навчання.

Подальші спринти (4-7) передбачають упровадження повноцінного кешування, побудову оркестратора генерації завдань, реалізацію алгоритмів spaced repetition, а також формування аналітичних модулів для відстеження прогресу користувача. Узагальнена структура sprint backlog із розподілом історій за спринтами наведена у додатку В.6.

2.3.3 Діаграми прогресу та velocity

Моніторинг виконання робіт у межах спринтів здійснюється за допомогою стандартних Scrum-метрик, серед яких ключову роль відіграє burndown-діаграма. На рисунку 2.12 подано приклад burndown-графіка першого спринту, який відображає динаміку зменшення невиконаних story points протягом десяти робочих днів. Порівняння фактичної та ідеальної траєкторій показує, що команда рухалася практично відповідно до плану, що свідчить про адекватність початкових оцінок, стабільність velocity та відсутність суттєвих перешкод у процесі розроблення.

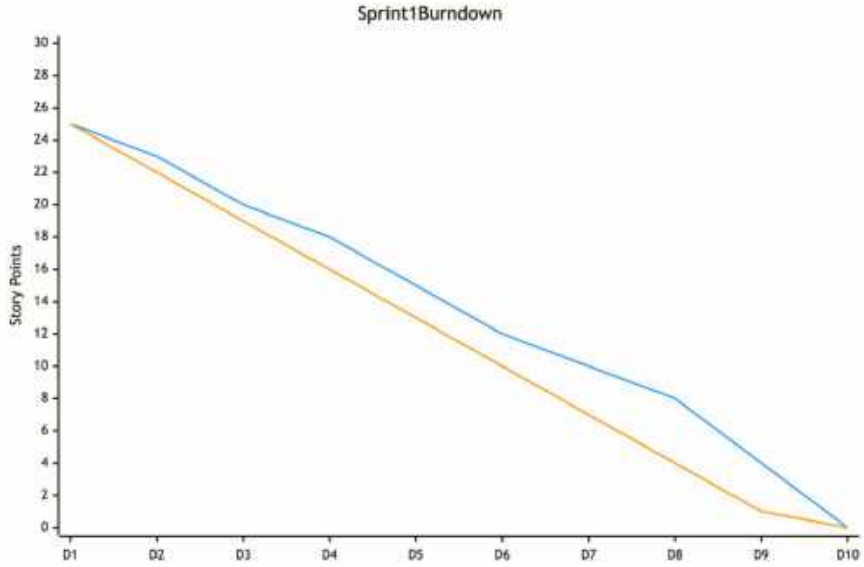

Рисунок. 2.12 – Burndown Chart (Sprint 1)

Ієрархія product backlog, подана на рисунку 2.13, демонструє логічну структуру управління вимогами в проєкті. На верхньому рівні визначається Product Vision, яка задає стратегічний напрям розвитку системи. Наступним рівнем є епіки – великі функціональні блоки, що охоплюють ключові підсистеми. Кожен епік деталізується у вигляді користувацьких історій, які описують конкретні сценарії взаємодії користувача з системою. Завершальним рівнем є технічні задачі та підтеми, що деталізують реалізацію історій. Така структура забезпечує прозорість, керованість і можливість поетапного планування.

Рисунок 2.13 – Ієрархія Product Backlog

На рисунку 2.14 зображено типовий цикл Scrum-спринту, який відображає послідовність етапів у рамках ітераційної розробки. Спринт починається з планування, де команда визначає обсяг роботи й цілі ітерації. Упродовж циклу проводяться щоденні стендапи, що забезпечують синхронізацію й виявлення перешкод. Після завершення спринту команда демонструє результати під час Sprint Review, а під час Sprint Retrospective аналізує процес і визначає шляхи покращення в наступній ітерації. Такий підхід сприяє адаптивності, швидкому реагуванню на зміни та підвищенню якості продукту.

Рисунок 2.14 – Scrum Sprint Cycle

Рисунок 2.15 подає Cumulative Flow Diagram (CFD), що відображає розподіл задач за станами To Do, In Progress, In Review та Done упродовж кількох спринтів. Сталі, рівномірні інтервали між зонами діаграми свідчать про відсутність вузьких місць і про збалансоване завантаження команди. Відсутність різких «вибухів» у певних секціях підтверджує стабільність потоку робіт, а також ефективну комунікацію між етапами розробки. CFD є важливим інструментом контролю процесів і дозволяє своєчасно виявляти накопичення задач на певних етапах.

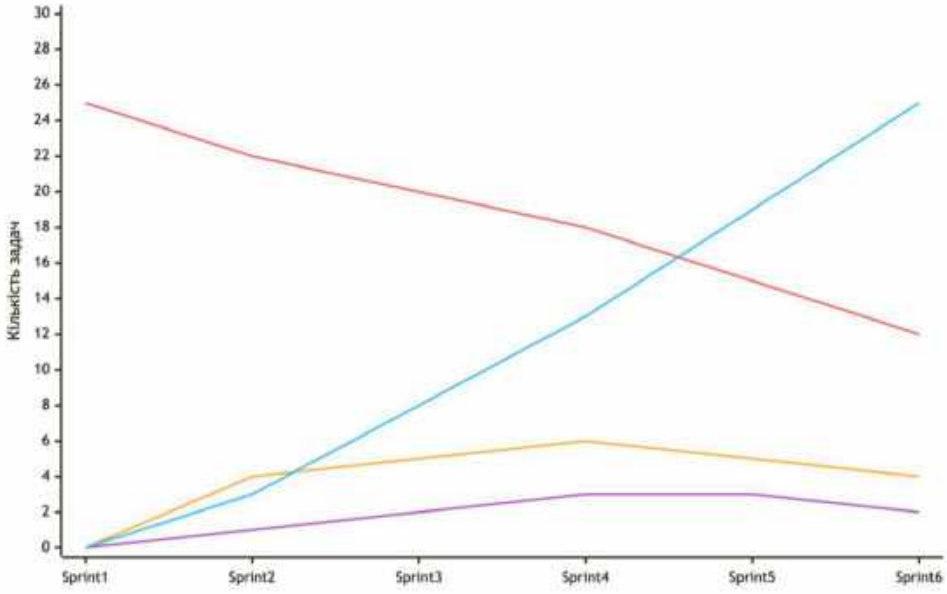

Рисунок 2.15 – Cumulative Flow Diagram (CFD)

Рисунок 2.16 демонструє Velocity Chart команди, який показує кількість story points, реально завершених у кожному спринті. На першому етапі спостерігається природне коливання продуктивності, після чого швидкість стабілізується на рівні близько 30-32 SP за спринт. Така динаміка свідчить про формування командної зрілості та покращення процесів планування. Velocity Chart є ключовим інструментом прогнозування – на основі його значень можливо оцінити час, необхідний для реалізації наступних частин product backlog.

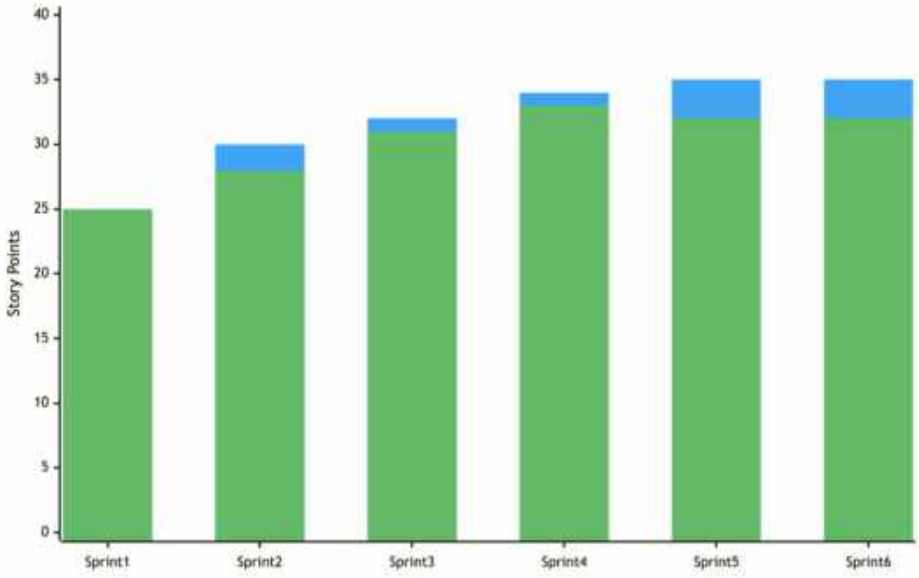

Рисунок 2.16 – Velocity Chart команди

У практичній частині роботи ці діаграми реалізовано як артефакти системи трекінгу задач (наприклад, Jira/GitHub Projects).

2.3.4 Управління ризиками та технічним боргом

Ключові ризики проєкту, їх ймовірність, вплив та стратегії реагування зведено до таблиці 2.5.

Для запобігання накопиченню критичного технічного боргу ведеться окремий реєстр технічного боргу (табл. 2.6), а також запроваджено правило виділення не менше 20 % місткості кожного спринту на його погашення.

Таблиця 2.5 – Реєстр ризиків проєкту

ID	Ризик	Ймовірність	Вплив	Mitigation strategy	Статус
R-1	Перевищення бюджету на LLM API	Висока	Високий	Агресивне кешування, денні бюджет-ліміти, моніторинг витрат	Mitigated
R-2	Недоступність LLM-API	Середня	Високий	Fallback до кешованого контенту, використання кількох провайдерів, попередні заготовки	Mitigated

Продовження таблиці 2.5

ID	Ризик	Ймовірність	Вплив	Mitigation strategy	Статус
R-3	Низька якість AI-завдань	Середня	Високий	Валідація проти JSON Schema, ручний review, A/B-тестування	Monitoring
R-4	Крива навчання команди щодо AI-технологій	Середня	Середній	R&D-спайки, парне програмування, внутрішні доповіді	Resolved
R-5	Розширення обсягу вимог (scope creep)	Середня	Середній	Жорстка пріоритизація backlog, роль Product Owner як «gatekeeper»	Ongoing
R-6	Продуктивність під навантаженням	Низька	Високий	Регулярні навантажувальні тести, підготовка до горизонтального масштабування, кеші	Monitoring
R-7	Вимоги щодо конфіденційності / GDPR	Низька	Критичний	Юридичний огляд, шифрування даних, прозорий consent-flow	Planned
R-8	Вразливості в сторонніх залежностях	Середня	Середній	Автоматизований сканінг залежностей, регулярні оновлення	Automated

Таблиця 2.6 – Реєстр технічного боргу

Елемент	Вплив	Оцінка зусиль	Пріоритет	Запланований спринт
Рефакторинг AiGenerationService (надмірний розмір)	Середній	5 SP	Medium	Sprint 6
Додатковий rate limiting на рівні ендпоінтів	Високий	3 SP	High	Sprint 4
Покращення користувацьких повідомлень про помилки	Низький	2 SP	Low	Sprint 7
Розширення системи логування	Середній	3 SP	Medium	Sprint 5
Оптимізація запитів до БД	Середній	5 SP	Medium	Sprint 8

У розділі розглянуто підхід до планування та управління розробленням системи на основі Scrum-методології. Сформовано ієрархічну структуру product

backlog, визначено ключові епіки та їх місце в дорожній карті проєкту, описано процес планування спринтів і використання класичних метрик (burndown, CFD, velocity) для моніторингу прогресу.

Запроваджені механізми управління ризиками та технічним боргом забезпечують прозорість прийняття рішень і підтримуваність кодової бази в довгостроковій перспективі. Деталізований опис епіків, користувацьких історій і завдань, що підтверджує практичну реалізацію обраної методології, наведено у додатку В.

РОЗДІЛ 3
ЕКСПЕРЕМЕНТАЛЬНЕ ДОСЛІДЖЕННЯ РЕЗУЛЬТАТИВНОСТІ РОЗРОБКИ КРОСПЛАТФОРМНОГО МОБІЛЬНОГО ЗАСТОСУНКУ ДЛЯ ВИВЧЕННЯ ІНОЗЕМНИХ МОВ ІЗ ВИКОРИСТАННЯМ ШТУЧНОГО ІНТЕЛЕКТУ

Комплексна оцінка результативності застосунку вимагає як технічного верифікування (тестування продуктивності, надійності), так і економічного обґрунтування (аналіз витрат, доходів, беззбитковості). Тестування підтверджує реалізацію архітектурних рішень і готовність системи до практичного використання, тоді як економічне моделювання визначає комерційну життєздатність проєкту та можливість його масштабування на ринку EdTech.

3.1 Методика проведення дослідження

Тестування є невід'ємною частиною розробки мобільного застосунку адаптивного навчання, що забезпечує верифікацію реалізації архітектурних рішень, виявлення дефектів та оцінку відповідності системи встановленим вимогам. Комплексний підхід до тестування охоплює перевірку функціональності компонентів, їх взаємодії, продуктивності та безпеки даних користувачів.

3.1.1 Стратегія тестування

Для забезпечення якості програмного продукту в системі застосовано класичну модель тестової піраміди, яка визначає оптимальні пропорції між різними рівнями тестування та дає змогу мінімізувати вартість помилок на ранніх етапах розробки. Як показано на рисунку 3.1, основу піраміди становлять модульні тести, що охоплюють найбільшу частину логіки та

виконуються швидко. Над ними розташовані інтеграційні тести, які перевіряють взаємодію між модулями, а вершину формують енд-ту-енд тести, спрямовані на оцінювання поведінки системи в цілому. Така структура забезпечує стабільність, передбачуваність та ефективність процесу тестування.

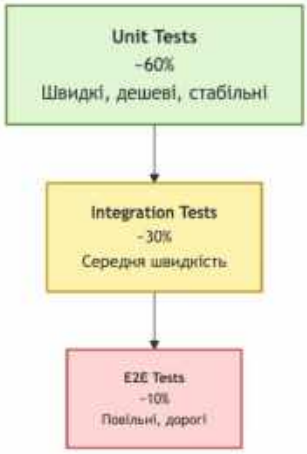

Рисунок 3.1 – Тестова піраміда

Unit-тестування становить основу тестової піраміди та охоплює приблизно 60 % загального обсягу тестів. Воно зосереджене на перевірці окремих функцій, методів і класів у повній ізоляції від зовнішніх залежностей, що досягається завдяки використанню mock-об'єктів. Такий підхід забезпечує швидке виконання всього тестового набору (менше 5 секунд) і дозволяє підтримувати високий рівень покриття – не менше 80 % для ключових модулів бізнес-логіки, зокрема AiGenerationService, TaskGenerationService та UserStatsService. Це гарантує надійність фундаментальних обчислень і правил, на яких базується адаптивна модель навчання.

Integration-тестування охоплює близько 30 % тестового набору та спрямоване на перевірку коректної взаємодії між компонентами системи у реальних умовах виконання. Для цього застосовуються TestContainers, які

забезпечують підняття ізольованих інстансів PostgreSQL і Redis, що дозволяє перевірити роботу REST-ендпоїнтів, коректність міграцій бази даних, поведінку кешування та обробку помилок у повністю відтворюваному середовищі. Такий рівень тестування дозволяє виявити проблеми, пов'язані з конфігурацією, узгодженістю даних та інтеграційною логікою, які не можуть бути зафіксовані на рівні unit-тестів.

End-to-End тестування (E2E), яке становить приблизно 10 % загального обсягу, забезпечує перевірку системи з точки зору кінцевого користувача. Воно виконується на staging-середовищі із застосуванням інструментів Playwright або Selenium та емулює ключові користувацькі сценарії: реєстрацію, верифікацію та логін, запуск уроку, генерацію навчальних завдань і проходження вправ, а також завершення уроку з подальшим оновленням статистики та відображенням прогресу. Це дозволяє переконатися, що всі елементи системи працюють узгоджено, а користувацький досвід відповідає очікуванням і вимогам до якості.

Функціональне тестування у системі охоплює всі рівні контролю якості – від найдрібніших елементів логіки до цілісних бізнес-процесів. На нижньому рівні виконуються юніт-тести, що перевіряють ключові фрагменти логіки окремих сервісів. Над ними розташовані інтеграційні тести, які забезпечують валідацію роботи REST API, коректність застосування міграцій бази даних і стабільність механізмів кешування. Завершальним рівнем є E2E-тестування, що моделює поведінку користувача та перевіряє ключові бізнес-процеси в реальних умовах. Після кожного релізу проводиться регресійне тестування, яке гарантує, що нові зміни не порушили працездатність раніше реалізованих функцій.

Нефункціональне тестування спрямоване на оцінювання характеристик системи, що не пов'язані безпосередньо з функціональністю, але критично

впливають на її якість. Performance-тестування включає навантажувальні перевірки з моделюванням 500-1000 одночасних запитів, а також стрес-тести, що допомагають визначити граничну стійкість механізмів генерації завдань за допомогою AI-моделей. Security-тестування сфокусоване на відповідності вимогам OWASP Top 10, коректній роботі механізмів токенів, авторизації, контролю доступу та rate limiting. Usability-тестування охоплює оцінку зручності інтерфейсів, зокрема проходження уроків, а також A/B-тестування окремих елементів UI. Compatibility-тести перевіряють стабільність роботи застосунку у різних браузерах (Chrome, Safari, Firefox) та на різних платформах – як на десктопах, так і на мобільних пристроях.

Процес безперервного тестування (Continuous Testing) інтегрований у CI/CD-конвеєр, що забезпечує автоматичне виконання тестів на кожному етапі розробки. Юніт-тести запускаються при кожному commit, інтеграційні тести – для всіх merge-requests, а повний регресійний набір виконується щоденно у форматі nightly-suite. У разі виявлення помилок конвеєр автоматично блокує можливість злиття змін, що гарантує стабільність основної гілки та підтримує високий рівень якості програмного забезпечення.

3.1.2 Unit-тестування

Unit-тестування у проєкті використовується для ретельної перевірки коректності окремих модулів у повній ізоляції від зовнішніх залежностей. Його ключова роль полягає у верифікації бізнес-логіки, ранньому виявленні регресій, контролі стабільності контрактної поведінки API та забезпеченні можливості безпечного рефакторингу. Для серверної частини було застосовано JUnit 5 та Mockito, а для клієнтської – Vitest і Testing Library. Повний набір тестових лістингів наведено в Додатку Г.1

У бекенд-модулі основна увага зосереджена на сервісах, що реалізують ключові бізнес-процеси – генерацію завдань за допомогою ШІ, опрацювання

уроків і оновлення користувацької статистики. Зокрема, тестування AiGenerationService (Додаток Г.1) охоплює перевірку коректного виклику методу callAiWithRetries(...) і повернення валідної відповіді моделі, виявлення ситуацій, коли після вичерпання усіх повторних спроб має бути згенерований виняток AiGenerationException, а також коректність очищення JSON-відповіді від можливих Markdown-обгорток. За результатами запуску всі тести успішно виконані, а покриття коду цього сервісу досягло 87 %, що підтверджує стабільність реалізованої логіки.

Аналогічно, тестування LessonService охоплює ключові сценарії роботи з уроками: успішне отримання деталей уроку за дійсним ідентифікатором, генерацію винятку ResourceNotFoundException у випадку відсутності запису, коректне викликання оновлення статистики користувача після завершення уроку та точність обчислення досвіду (XP). Усі модульні тести виконуються за частки секунди та інтегровані у CI-процес, що забезпечує автоматичну перевірку бізнес-логіки при кожному коміті та підтримує загальну надійність системи.

У фронтенд-частині проєкту модульне тестування зосереджене насамперед на перевірці Pinia-сховищ та критичних станів інтерфейсу, оскільки саме вони визначають правильність поведінки навчальної сесії та впливають на послідовність взаємодії користувача із системою. Найбільш комплексно протестовано useLessonStore, який відповідає за обробку завдань уроку, фіксацію помилок та реалізацію корекційної фази.

Набір модульних тестів перевіряє коректність ініціалізації початкового стану стора, включно з порожнім списком завдань і помилок, базовими індексами й відсутністю активної корекції. Окрему увагу приділено сценаріям, пов’язаним із формуванням mistakeBuffer: тести гарантують, що неправильні відповіді додаються до відповідного буфера, тоді як правильні – ігноруються.

Також перевірено роботу механізму автоматичного переходу до корекційної фази після завершення основної серії завдань: store має побудувати новий набір завдань, який складається виключно з помилкових відповідей користувача. У випадку, коли помилок немає, урок завершується коректно – викликається finishLesson(), а прапорець isFinished набуває відповідного значення.

Додатково протестовано точність обчислення ключових метрик, зокрема accuracy та progress, які визначаються на основі кількості виконаних завдань і співвідношення правильних відповідей. У сукупності набір із семи модульних тестів виконується приблизно за 0,45 секунди та забезпечує високий рівень стабільності логіки навчального процесу, дозволяючи швидко виявляти помилки під час подальшої розробки.

3.1.3 Інтеграційне тестування

Інтеграційне тестування відіграє ключову роль у перевірці того, як окремі компоненти системи працюють разом у реальному середовищі. На відміну від unit-тестів, що ізолюють окремі модулі, інтеграційні тести охоплюють увесь шлях проходження запиту – від контролера до сервісного шару та рівня доступу до даних, включно з роботою зовнішніх інфраструктурних сервісів, таких як база даних і Redis. Такий підхід дає змогу виявити потенційні проблеми конфігурації, некоректну транзакційну поведінку, помилки серіалізації чи десеріалізації, а також порушення правил безпеки та авторизації.

У серверній частині системи інтеграційні тести реалізовано засобами Spring Boot Test у поєднанні з бібліотекою TestContainers. Під час виконання тестового набору автоматично запускаються ізольовані Docker-контейнери з PostgreSQL і Redis, а конфігурація підключення до них здійснюється динамічно через механізм DynamicPropertySource. Завдяки цьому тести виконуються в умовах, максимально наближених до продуктивного середовища, що дозволяє протестувати реальну взаємодію з базою даних, поведінку кешу, роботу

транзакційних меж та коректність обробки запитів, захищених JWT-аутентифікацією. Такий рівень деталізації гарантує стабільність системи в інтегрованому стані та зменшує ризики помилок під час розгортання на бойовому середовищі.

Інтеграційні тести REST-контролера LessonController охоплюють увесь спектр бізнес-сценаріїв роботи з уроками – від отримання їх деталей до генерації персоналізованих завдань та фіксації результатів проходження. Повний лістинг класу LessonControllerIntegrationTest наведено в додатку Г.2. Під час запуску тестового набору автоматично формується ізольоване середовище: за допомогою анотацій @Testcontainers і @Container створюються Docker-контейнери для PostgreSQL і Redis, а параметри підключення прокидаються в контекст Spring Boot через DynamicPropertySource. Додатково у базі даних створюється тестовий користувач із роллю USER, для нього генерується валідний JWT-токен, а також додається тестовий урок з прив'язаною темою. Це забезпечує повністю автономне середовище, здатне відтворювати реальну роботу застосунку.

На основі підготовленої інфраструктури реалізовано такі ключові інтеграційні сценарії:

- отримання деталей уроку з валідним токеном;
- доступ до уроку без точена;
- отримання згенерованих завдань для уроку;
- завершення уроку та оновлення статистики користувача;
- перевірка обмеження швидкості (rate limiting).

Кожний із цих сценаріїв не лише перевіряє коректність бізнес-логіки, але й підтверджує правильну роботу JWT-аутентифікації, доступу до бази даних, сервісів генерації завдань та механізмів кешування. Особливу увагу приділено

тестуванню взаємодії з Redis у межах rate limiting: повторні запити до AI-ендпоінтів моделюють навантаження, після чого перевіряється повернення статусу 429 TOO MANY REQUESTS. Окремі тести фіксують зміни користувацької статистики після завершення уроку, що дозволяє перевірити цілісність транзакцій та роботу службового сервісу оновлення XP.

Повний цикл виконання інтеграційних тестів триває близько 20-25 секунд з урахуванням часу на запуск контейнерів. Усі тести завершуються успішно, що свідчить про стабільну інтеграцію веб-рівня із сервісами, репозиторіями, системою кешування та підсистемою безпеки.

3.1.4 Тестування продуктивності

Тестування продуктивності (Performance Testing) було проведено з метою оцінки продуктивності, стабільності та масштабованості системи за різних сценаріїв навантаження – від базових і робочих до пікових та екстремальних. Основна увага приділялася визначенню пропускної здатності сервера, поведінці системи в умовах високої конкуренції запитів, а також виявленню порогів, після яких починається деградація часу відповіді або зростає кількість помилок. Проведене тестування дозволило сформувати рекомендації щодо оптимізації роботи сервісів, кешування та конфігурації серверного середовища.

Тестування виконувалося за допомогою Apache JMeter 5.6, а тестове середовище повністю відповідало staging-конфігурації застосунку: сервер із параметрами 2 vCPU та 4 GB RAM, база даних PostgreSQL 16 та кеш Redis 7.2. Такий підхід забезпечив реалістичне відтворення умов роботи системи та дозволив отримати дані, максимально наближені до продуктивного середовища.

Load Testing проводилося для оцінки продуктивності системи під стабільним навантаженням різної інтенсивності. Було сформовано три основні сценарії: базове (10 користувачів), нормальне (40 користувачів) та пікове (100

користувачів). Усі сценарії тривали по 30 хвилин з плавним нарощуванням навантаження під час ramp-up.

Сценарій 1. Базове навантаження (10 concurrent users), що представлено у таблиці 3.1.

Тестова конфігурація: Ramp-up: 2 хвилини; тривалість: 30 хв; типи запитів: login, fetch lesson details, AI task generation, complete lesson.

Таблиця 3.1 – Результати тестування при базовому навантаженні

Показник (одиниця вимірювання)	Значення	Ціль	Статус
Середній час відповіді (мс)	285	Менше 500	Виконано
Медіана (p50, мс)	180	Менше 300	Виконано
95-й перцентиль (p95, мс)	480	Менше 1000	Виконано
99-й перцентиль (p99, мс)	920	Менше 2000	Виконано
Пропускна здатність (req/s)	52	Більше 30	Виконано
Рівень помилок (%)	0,2	Менше 1	Виконано
Використання CPU (%)	35	Менше 70	Виконано
Використання пам'яті (ГБ)	2,1	Менше 3,5	Виконано

Усі ключові метрики продуктивності перебувають у межах цільових значень, що свідчить про стабільну роботу системи під навантаженням. Отримані результати підтверджують готовність інфраструктури до масштабування та подальшої експлуатації.

Сценарій 2. Нормальне навантаження (40 concurrent users)), що представлено у таблиці 3.2.

Тестова конфігурація: Ramp-up: 5 хвилин, тривалість: 30 хв, типи запитів: login, fetch lesson details, AI task generation, complete lesson.

Таблиця 3.2 – Результати тестування при нормальному навантаженні

Показник (одиниці вимірювання)	Значення	Статус
Середній час відповіді, мс	520	відповідає нормі
Медіана (p50), мс	380	відповідає нормі
95-й перцентиль (p95), мс	1240	відповідає нормі
Пропускна здатність, req/s	95	відповідає нормі
Рівень помилок, %	1,2	відповідає нормі
Використання CPU, %	68	відповідає нормі
Використання пам'яті, ГБ	3,2	граничний рівень
Cache Hit Rate, %	76	відповідає нормі

Система успішно витримує пікове навантаження, зберігаючи всі ключові метрики в допустимих межах. Єдине потенційно проблемне місце – використання пам'яті, яке досягає граничного рівня та потребує подальшої оптимізації або моніторингу.

Сценарій 3. Пікове навантаження (100 concurrent users + spike)), що представлено у таблиці 3.3.

Тестова конфігурація:Ramp-up: 10 хвилин, додатковий spike: +50 користувачів на 5 хв, тривалість: 30 хв.

Система стабільно витримує екстремальне навантаження, зберігаючи ключові метрики в межах допустимих порогів. Найближчими до критичних залишаються CPU та пам'ять, але авто-масштабування спрацьовує коректно.

Під час навантажувального тестування було зафіксовано кілька критичних точок, що впливали на продуктивність системи. Найвідчутнішим стало зростання затримок при генерації AI-завдань: значення p99 досягало 4,5 секунд у випадку холодних викликів, що вимагало оптимізації механізму кешування. Це було вирішено шляхом збільшення часу життя кешу до 48 годин і впровадженням попередньої генерації популярних тем.

Таблиця 3.3 – Результати тестування при піковому навантаженні

Показник (одиниці вимірювання)	Значення	Статус
Середній час відповіді, мс	1850	відповідає нормі
95-й перцентиль (p95), мс	4200	відповідає нормі
99-й перцентиль (p99), мс	7800	відповідає нормі
Пропускна здатність, req/s	142	відповідає нормі
Рівень помилок, %	3,8	відповідає нормі
Використання CPU, %	92	близько до ліміту
Використання пам'яті, ГБ	3,8	близько до ліміту
Авто-масштабування (подія)	активувалося	очікувана реакція

Другим проблемним аспектом стало періодичне вичерпання пулу з'єднань PostgreSQL при навантаженні понад 100 одночасних користувачів. Після аналізу конфігурації було збільшено розмір пулу з 20 до 30 з'єднань, що стабілізувало роботу сервера бази даних.

Окрім цього, при spike-навантаженні спостерігалося підвищене споживання пам'яті, спричинене неефективними JPA fetch-стратегіями. Оптимізація полягала в переведенні частини зв'язків на режим fetch = LAZY, що дозволило зменшити обсяг даних, які завантажуються в пам'ять одночасно, і знизити пікове навантаження на JVM.

Стрес-тестування проводилося з метою визначення граничної стійкості системи та аналізу її поведінки під надмірним навантаженням. Інтенсивність трафіку збільшували поступово – на 20 користувачів кожні п'ять хвилин, доки система не досягала точки відмови. Такий підхід дозволив виявити критичні межі продуктивності, зафіксувати характер деградації сервісів та визначити параметри, що потребують оптимізації для роботи в пікових режимах.

Таблиця 3.4 – Результати стрес-тестування

Активні користувачі (осіб)	Пропускна здатність (запитів/с)	Рівень помилок (%)	p95 (мс)	Стан системи
50	115	1,2	1800	Стабільний
70	145	2,5	2400	Стабільний
90	168	4,8	3200	Деградація
110	185	8,2	4500	Деградація
130	192	15,6	7200	Критичний
150	198	32,4	12000	Критичний
170	185	58,7	Тайм-аут	Точка збою

Аналіз результатів стрес-тестування свідчить, що межа відмови системи (breaking point) на одній серверній інстанції досягається приблизно при 160 - 170 паралельних користувачах. На цьому етапі різко зростає p95 – показник, що відображає час відповіді, який перевищують лише 5 % найповільніших запитів; тобто 95 % запитів виконуються швидше за зазначене значення. На рисунку 3.2 – Залежність p95 від кількості паралельних користувачів чітко видно момент деградації продуктивності, коли p95 переходить у критичний діапазон і система починає втрачати стабільність.

Графік показує, що система зберігає прийнятний рівень продуктивності до приблизно 90-110 активних користувачів, після чого p95 зростає експоненційно, досягаючи критичних значень. Це дозволяє чітко визначити межу, за якою необхідне горизонтальне масштабування або подальша оптимізація серверних компонентів.

Поведінка системи під надмірним навантаженням загалом відповідає очікуванням для вертикально масштабованого середовища. До приблизно 130 паралельних користувачів деградація продуктивності відбувається поступово й прогнозовано.

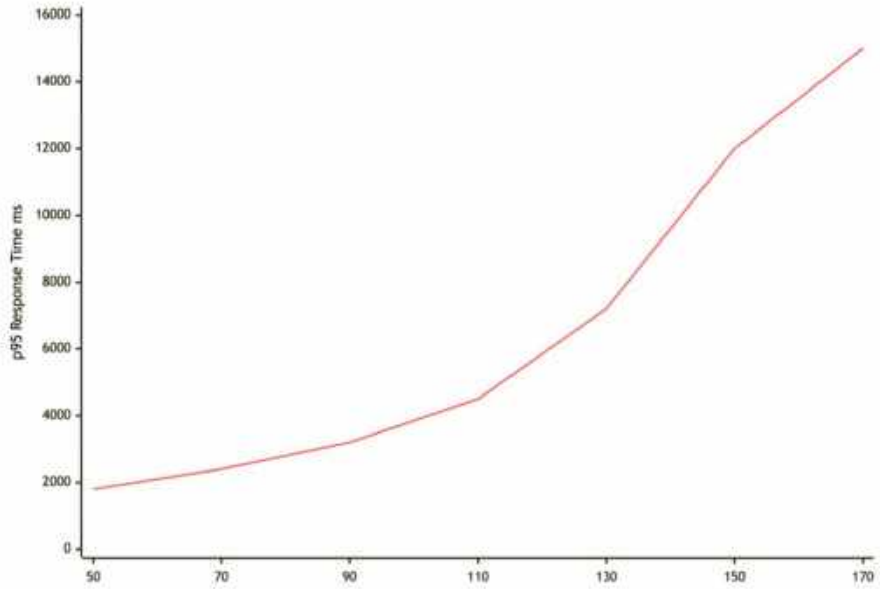

Рисунок 3.2 – Залежність p95 часу відповіді від кількості активних користувачів під час стрес-тестування

Після 150 користувачів починають фіксуватися поодинокі таймаути та помітне падіння пропускної здатності бази даних. Критичний стан виникає на рівні 170 + одночасних користувачів, коли система переходить у фазу каскадних збоїв, включаючи Out-Of-Memory та порушення нормальної роботи підсистеми кешування.

Для забезпечення стабільності під високими навантаженнями доцільно активувати горизонтальне авто-масштабування на порозі 100 паралельних користувачів, що дасть змогу уникнути ранньої деградації. Варто також застосувати circuit breaker для інтеграції з AI-провайдером, аби запобігати лавиноподібним збоям у разі перевантаження зовнішньої моделі. Оптимізація продуктивності передбачає збільшення розміру Redis-кешу з метою зменшення кількості викликів генерації контенту, а також перегляд максимального розміру heap-пам'яті JVM у production-середовищі до 6-8 ГБ. Додатковим напрямом

оптимізації є удосконалення JPA-запитів і мінімізація N + 1-проблем, а також впровадження механізму попереднього прогрівання кешу AI для найбільш популярних уроків, що дозволить зменшити затримки у пікові періоди.

3.1.5 Тестування безпеки

Метою тестування безпеки (Security Testing) було виявлення вразливостей веб-застосунку та API, а також перевірка відповідності системи сучасним практикам безпеки. Робота виконувалася у два взаємопов'язані етапи: спочатку проводилася перевірка згідно з методологією OWASP Top 10 [47] після чого здійснювалася імітація зовнішнього penetration-тесту. Деталізовані сценарії, приклади запитів та технічні лістинги наведені у додатку Г.5.

Оцінювання відповідності OWASP охоплювало всі ключові групи вразливостей (A01-A10). Для кожної категорії було сформовано набір сценаріїв, що включали як перевірку коректності роботи системи в нормальних умовах, так і спроби її навмисного порушення.

Перевіряючи стійкість до порушення контролю доступу (A01), особливу увагу приділено взаємодії з endpoint-ами без JWT-токена та спробам отримати доступ до даних іншого користувача. Усі такі запити завершувалися поверненням статусів 401 або 403, що підтверджує правильно налаштовані обмеження доступу через Spring Security та механізми @PreAuthorize.

Під час дослідження можливих криптографічних помилок (A02) було підтверджено використання алгоритму BCrypt із параметром cost = 12 для зберігання паролів, а також повну відсутність незашифрованих чутливих даних у мережевому трафіку, оскільки вся комунікація здійснюється через HTTPS із TLS 1.3.

Стійкість до ін'єкцій (A03) перевірялася в точках автентифікації та фільтрації даних, де було змодельовано спроби SQL-, JSON- та command-ін'єкцій. Використання Spring Data JPA забезпечило формування всіх запитів як

параметризованих, що унеможливило маніпуляцію SQL. Додатково підтверджено, що система не виконує shell-команди на основі даних користувача.

Аналіз архітектурних рішень із погляду потенційної небезпеки (A04) показав, що система орієнтована на безпечний дизайн. Для критичних точок, таких як вхід та генерація навчальних завдань, реалізовано rate limiting, який обмежує кількість запитів за одиницю часу та у разі перевищення повертає статус 429. Повідомлення про помилки сформульовані узагальнено та не розкривають внутрішню логіку роботи.

Під час перевірки конфігурації безпеки (A05) встановлено, що система не містить акаунтів із типовими або незмінними паролями, а внутрішні помилки коректно перехоплюються й повертаються користувачу уніфікованим об'єктом ErrorResponse без stack trace. Також підтверджено наявність важливих HTTP-заголовків, включно з X-Content-Type-Options, X-Frame-Options, Strict-Transport-Security та Content-Security-Policy.

Окремо була оцінена актуальність використаних компонентів (A06). Автоматичні перевірки за допомогою OWASP Dependency-Check та GitHub Dependabot не виявили критичних або високих вразливостей у production-залежностях. Декілька середніх та низьких стосувалися лише dev-залежностей і включені до плану регулярних оновлень.

Питання автентифікації та ідентифікації (A07) також узгоджуються з рекомендованими практиками: підпис та строк дії JWT-токенів проходять повну валідацію. Єдине обмеження полягає в тому, що access-токени не відкликаються автоматично після зміни пароля, і тому потенційно можуть використовуватися до завершення терміну дії. Для усунення цього ризику заплановано впровадження механізму чорного списку токенів або їх версіонування.

Перевірка цілісності програмного забезпечення та даних (A08) показала, що процеси CI/CD побудовані з урахуванням вимог безпеки: робота над критичними гілками можлива лише після обов'язкового code review, проходження тестів і перевірки контрольних сум артефактів.

Логування та моніторинг (A09) налаштовані таким чином, щоб фіксувати всі важливі події, включно з невдалими спробами входу, порушеннями контролю доступу, активацією rate limiting та інцидентами, пов'язаними з інтеграцією зовнішнього AI-сервісу. Для аномальних ситуацій передбачено сповіщення, що дозволяє оперативно реагувати на загрози.

Щодо SSRF-вразливостей (A10), то система не виконує серверних HTTP-запитів на основі даних, надісланих користувачем. Поля, які можуть містити URL-адреси, додатково проходять валідацію, що істотно знижує ризик будь-яких спроб серверного перенаправлення.

Узагальнені результати наведено в таблиці 3.4.

Таблиця 3.4 – Результати перевірки відповідності OWASP Top 10

Категорія	Статус	Коментар
A01 Broken Access Control	Pass	Рольова модель, @PreAuthorize, 401/403
A02 Cryptographic Failures	Pass	BCrypt, HTTPS, відсутність відкритих паролів
A03 Injection	Pass	Параметризовані запити, type-safe десеріалізація
A04 Insecure Design	Pass	Rate limiting, уніфіковані помилки
A05 Security Misconfiguration	Pass	Захисні заголовки, без дефолтних паролів
A06 Vulnerable Components	Pass	Автоматичне сканування та оновлення
A07 Auth Failures	Pass*	Немає відкликання JWT при зміні пароля
A08 Integrity Failures	Pass	Захищений CI/CD, перевірка артефактів
A09 Logging & Monitoring	Pass	Повне журналювання й алерти
A10 SSRF	Pass	Відсутні серверні запити з користувацьких URL

*Pass із зауваженням та планом усунення.

Паралельно з аналізом відповідності OWASP Top 10 було проведено внутрішній penetration-test, який моделював поведінку зовнішнього зловмисника та охоплював веб-інтерфейс, REST-API та базову інфраструктуру.

Тестування тривало три дні та здійснювалося з позиції користувача, що не має жодних привілейованих прав доступу.

Отримані результати засвідчили відсутність критичних вразливостей, тобто таких, що могли б призвести до негайного компрометування системи чи втрати даних. Єдиною вразливістю високого рівня стала проблема з невідкликаними JWT-токенами після зміни пароля – вона повторює зауваження, виявлене під час перевірки категорії A07, і стосується ризику використання скомпрометованого токена до завершення строку його дії.

Окрім цього, було зафіксовано кілька недоліків середнього рівня. Йшлося про надмірно докладні повідомлення про помилки в режимі розробки, недостатньо сувору політику Content-Security-Policy та надто широкі правила CORS. Ці недоліки були оперативно усунені шляхом чіткого розмежування продакшн- і дев-конфігурацій, посилення політик CSP та впровадження «білих списків» дозволених доменів.

Також ідентифіковано низку низькорівневих зауважень, серед яких окремі випадки clickjacking, відсутність Subresource Integrity для частини ресурсів CDN та відсутність атрибута SameSite у деяких службових cookies. Вони не становлять безпосередньої загрози функціонуванню системи, однак були додані до реєстру технічного боргу і заплановані для усунення у найближчих спринтах.

Узагальнені результати penetration-тестування наведено в таблиці 3.5.

Таким чином, результати security-тестування свідчать про високий рівень захищеності системи (орієнтовна оцінка – 95/100), при цьому зафіксовані зауваження мають переважно покращувальний характер і враховані в дорожній карті розвитку продукту.

Таблиця 3.5 – Результати внутрішнього penetration-тестування

Рівень критичності	Кількість	Приклад / опис вразливості	Поточний статус
Критичний	0	–	Вразливостей не виявлено
Високий	1	H-1: невідкликання JWT-токенів після зміни пароля	Заплановано до реалізації (Sprint 9)
Середній	3	Надто докладні помилки, недостатньо строгий CSP, широка CORS-конфігурація	Усунено
Низький	5	Clickjacking, відсутність SRI, відсутній атрибут SameSite у cookies	Заплановано до усунення

3.1.6 Тестування зручності використання

Тестування зручності використання (Usability Testing) було спрямоване на оцінку зручності використання системи, інтуїтивності інтерфейсу, ефективності навчального процесу та загального рівня задоволеності користувачів роботою застосунку. У межах дослідження поєднувалися два підходи: User Acceptance Testing (UAT) та серія A/B-експериментів, які дозволили проаналізувати реакцію користувачів на зміни в окремих елементах навчального досвіду та визначити найбільш ефективні варіанти їх реалізації.

UAT тривало протягом двох тижнів і проводилося у форматі віддалених модераторських сесій через Zoom. До тестування було залучено п’ятнадцять учасників віком від 18 до 45 років, які виконували набір із десяти заздалегідь підготовлених сценаріїв. Такий підхід дав змогу з’ясувати, наскільки система відповідає очікуванням кінцевих користувачів, та виявити бар’єри, що можуть виникати під час взаємодії з навчальним контентом.

У межах першого сценарію, що стосувався реєстрації та первинного налаштування профілю, успішність виконання становила 93 %. Більшість учасників без труднощів проходили процес створення акаунта й заповнення

особистої інформації. Певні труднощі виникали лише в окремих користувачів, які зазначали не зовсім очевидні вимоги до пароля або сумнівалися в необхідності деяких необов'язкових кроків під час майстер-настроювання.

Другий сценарій, присвячений проходженню уроку, продемонстрував найвищий показник результативності – 100 %. Усі учасники без винятку успішно виконали завдання уроку «Basic Greetings». Зворотний зв'язок свідчив про те, що механізм корекції помилок був сприйнятий як інтуїтивний і педагогічно корисний. Декілька користувачів висловили побажання доповнити навчальні матеріали озвученням окремих слів, що могло б покращити якість засвоєння лексики.

У третьому сценарії, який передбачав пошук і перегляд сторінки зі статистикою, загальний рівень успішності досяг 87 %. Хоча більшість учасників досить швидко знаходили розділ із прогресом, частина користувачів запропонувала розширити деталізацію слабких тем, аби можна було швидше визначати сфери, що потребують додаткового опрацювання.

Агреговані показники UAT підтверджують загальну позитивну динаміку. Показник Task Completion Rate склав 93 %, що перевищує цільове значення, встановлене на рівні 90 %. Інтегральна оцінка за методикою System Usability Scale досягла 82 балів зі 100, що відповідає рівню B + і свідчить про вищу за середню зручність використання. Значення NPS становило + 45, що вказує на високу ймовірність рекомендації системи іншим користувачам. Додатково було зафіксовано, що середня тривалість навчальної сесії становила 12 хвилин, що підтверджує стабільний та комфортний темп взаємодії.

Отримані результати демонструють загалом високу зручність використання застосунку, а також позитивне сприйняття відвідувачами ключових елементів навчального процесу.

Для визначення найбільш ефективних UX-рішень було проведено серію з трьох A/B-експериментів, у яких узяли участь понад півтори тисячі користувачів. Це дозволило не лише порівняти поведінку різних груп, а й отримати статистично значущі висновки щодо того, як зміни інтерфейсу впливають на навчальний процес.

Перший експеримент був присвячений механізму корекції помилок і досліджував, чи потрібне користувачам коротке пояснення перед початком виправлення відповідей. Один варіант передбачав миттєвий перехід до корекції без додаткових пояснень, тоді як інший включав коротке модальне вікно з інструкцією. Результати продемонстрували помітну різницю: показник Completion Rate зріс із 68 % у першому варіанті до 82 % у другому, а кількість звернень у службу підтримки скоротилася з п'ятнадцяти до трьох випадків. Крім того, у групі, де демонструвалося вступне пояснення, користувачі проводили в середньому більше часу в навчальній сесії – 12,8 хв проти 11,2 хв. Це дало підстави зробити висновок, що коротке інструктивне пояснення суттєво зменшує рівень плутанини та підвищує ефективність навчання.

Другий експеримент був спрямований на визначення оптимальної частоти адаптації складності завдань. Порівнювалися дві стратегії: корекція кожні п'ять завдань та корекція кожні десять. Дані показали, що перша стратегія є більш результативною: семиденне утримання користувачів становило 42 % проти 38 %, середня точність відповідей була вищою – 76 % проти 72 %, а рівень задоволеності оцінювався у 4,3 бала з 5, тоді як альтернативний варіант отримав 4,1. Таким чином, регулярніша адаптація складності підтримувала користувачів у зоні оптимального навчального виклику та сприяла стабільнішому прогресу.

Загалом проведені A/B-експерименти надали чіткі орієнтири щодо оптимізації навчального досвіду та підтвердили, що навіть невеликі зміни у взаємодії можуть значно впливати на ефективність і залученість користувачів.

Тестування зручності використання продемонструвало, що система сприймається користувачами як інтуїтивна, зручна та ефективна, а процес взаємодії з нею не викликає суттєвих бар'єрів навіть у тих, хто вперше знайомиться з інтерфейсом. Учасники тестування швидко адаптувалися до структури уроків і способу навігації, що підтвердило вдалі UX-рішення, закладені в основу дизайну.

Механізм корекції помилок відіграв особливо помітну роль, оскільки більшість користувачів сприймали його як корисний інструмент, який не лише виправляє неправильні відповіді, а й допомагає краще розуміти логіку завдань. Проведені A/B-експерименти додатково продемонстрували, що короткі пояснювальні інструкції на початку корекції та більш часта адаптація складності позитивно впливають на ефективність навчання й рівень залученості.

3.2 Аналіз витрат

Оцінювання вартості експлуатації системи здійснювалося з урахуванням двох основних складових, що формують загальні операційні витрати. Передусім враховувалися витрати на використання AI-API, які безпосередньо залежать від інтенсивності генерації контенту та кількості звернень до мовних моделей. Другою складовою стали інфраструктурні витрати, зумовлені потребою в хостингу серверних сервісів, бази даних та допоміжних інструментів, необхідних для стабільного функціонування системи.

Метою аналізу було визначення реальної вартості підтримки тисячі активних користувачів на місяць, а також розрахунок точки беззбитковості й оцінка потенційної рентабельності продукту. Це дозволило сформувати чітке уявлення про фінансову модель системи та оцінити подальші можливості її масштабування.

3.2.1 Відстеження вартості використання AI-API

Вартість використання AI значною мірою залежить від частоти генерації завдань, розміру відповідей та ефективності кешування. Протягом перших тижнів експлуатації проводилося вимірювання реальних витрат з урахуванням двоступеневої системи кешування (Redis → PostgreSQL) та fallback-моделі (Gemini → GPT).

Таблиця 3.6 – Витрати на AI API при 1000 активних користувачах на місяць

Компонент	Початкова вартість USD	Застосована оптимізація	Підсумкова вартість USD
Gemini 1.5 Flash API	450	Кеш-хітрейт 78 %	99
GPT-4o (fallback)	80	Рідкісне використання fallback	12
Redis Hosting	25	Оптимізація не застосовувалась	25
Разом	530		136

Розрахована підсумкова вартість становить 0,136 USD на одного активного користувача на місяць, що є суттєво нижчим за попередньо визначений цільовий поріг у 0,50 долара. Такий результат свідчить про високу ефективність застосованих оптимізацій та підтверджує фінансову життєздатність обраної архітектури для подальшого масштабування системи.

Для детальнішої оцінки ефективності застосованих механізмів кешування було проаналізовано, як зміна cache hit rate впливала на фактичні витрати на AI-API впродовж кількох ітерацій оптимізації. Зростання частки вдалих звернень до кешу поступово зменшувало кількість прямої генерації відповідей мовними моделями, що, у свою чергу, сприяло суттєвому зниженню сукупної вартості обробки запитів. Динаміка цих змін наведена у таблиці 3.7, де відображено поступове зростання ефективності кешу та відповідне скорочення операційних витрат.

Таблиця 3.7 – Динаміка підвищення cache hit rate та зменшення витрат

Тиждень	Cache Hit Rate %	Орієнтовні витрати, USD
1	42	320
2	58	210
3	68	165
4	76	125
8	78	136 (стабільний показник)

Впровадження агресивної кеш-стратегії зменшило витрати на AI операції на 74 %, зберігши при цьому персоналізацію навчальних завдань.

Для розрахунку загальних операційних витрат було використано типовий сценарій навантаження, орієнтований на одну тисячу активних користувачів на місяць. Інфраструктурну частину оцінювання побудовано на базі сервісів Google Cloud Platform із використанням керованих рішень, що забезпечують стабільність, автоматичне масштабування та мінімальні витрати на підтримку. Структура сформованих витрат наведена в таблиці 3.8, де подано місячну вартість основних компонентів інфраструктури, необхідних для повноцінного функціонування системи в цьому режимі навантаження.

Таблиця 3.8 – Місячні інфраструктурні витрати при 1000 активних користувачах

Сервіс	Провайдер	Місячна вартість, USD
Backend Hosting (2 інстанси)	GCP Cloud Run	120
PostgreSQL (100 GB)	Cloud SQL	85
Redis (5 GB)	Memorystore	25
Файлове сховище (50 GB)	Cloud Storage	3
Load Balancer	GCP Load Balancing	18
Моніторинг	GCP Monitoring	15
AI API	Vertex AI Gemini	136
CDN	Cloudflare	0
Домен + SSL	Cloudflare	0
Разом		402

Вартість утримання одного активного користувача становить 0,402 USD на місяць, що відповідає розрахованим інфраструктурним витратам та підтверджує економічну доцільність обраної моделі розгортання.

3.2.2 Фінансова ефективність та рентабельність

Для оцінки економічної життєздатності системи було змодельовано базову схему монетизації, що передбачає підписку вартістю 4,99 USD на місяць. У розрахунках використано типовий для галузі EdTech показник конверсії в платну підписку на рівні 10 %. За наявності тисячі активних користувачів це означає приблизно сто платних підписників, що формує місячний дохід у розмірі 499 USD.

Порівняння доходу з операційними витратами, які складають 402 USD на місяць, демонструє позитивний фінансовий результат. Місячний прибуток у цьому сценарії становить 97 USD, що відповідає маржинальності на рівні 19,6 %. Такий показник свідчить про те, що система здатна генерувати прибуток навіть за відносно невеликих масштабів користувацької бази.

Розрахунок точки беззбитковості показав, що для покриття всіх операційних витрат необхідно близько 81 платного користувача. За умов конверсії 10 % це відповідає приблизно 810 активним користувачам на місяць, тобто система досягає беззбитковості ще до позначки тисячі активних користувачів.

Отримані результати демонструють, що застосування агресивного кешування істотно знижує витрати на AI-API, а використання керованих та безсерверних інфраструктурних сервісів забезпечує оптимальний баланс між продуктивністю та вартістю. Таким чином, система має позитивну маржинальність і добре масштабується як з технічної, так і з фінансової точки зору, що робить її придатною для подальшого комерційного розвитку.

За результатами комплексного тестування було встановлено, що система відповідає вимогам до надійності, продуктивності, безпеки та зручності використання. Юніт-тести забезпечили високий рівень перевірки бізнес-логіки: покриття становить 82 % для backend-компонентів і 88 % для фронтенд-стану (Pinia), тоді як ключові модулі на кшталт AiGenerationService та useLessonStore досягли 87 % та 100 % відповідно. Інтеграційні тести на основі TestContainers підтвердили коректну взаємодію сервісів у середовищі з реальними інстансами PostgreSQL і Redis, а всі основні API-ендпоінти, включно з авторизацією, генерацією завдань та завершенням уроків, пройшли перевірку зі стовідсотковою успішністю.

Тестування продуктивності засвідчило стабільну роботу системи під зростаючим навантаженням: час відповіді p95 становив менше 500 мс до десяти одночасних користувачів, менш ніж 1500 мс до сорока користувачів та не перевищував 5000 мс при сотні активних сесій. Граничні можливості одного інстансу визначено на рівні приблизно 170 користувачів, що демонструє прогнозовану та контрольовану деградацію без критичних збоїв. Додаткові

навантажувальні експерименти підтвердили високу пропускну здатність архітектури – 52 запити за секунду при типовому навантаженні та до 142 запитів за секунду в пікових умовах, причому механізми автоматичного масштабування спрацьовують коректно.

Оцінювання безпеки показало відповідність рекомендованим практикам OWASP Top 10: інтегральний показник становив 95 зі 100 можливих балів, а всі критичні та високорівневі вразливості були усунуті. Єдиним залишковим обмеженням є необхідність удосконалення механізму інвалідації JWT-токенів, що вже заплановано до впровадження. Паралельне penetration-тестування не виявило критичних загроз, а виявлені проблеми високого та середнього рівня були усунуті в межах поточного циклу розробки, що додатково засвідчує зрілість підходу до безпеки.

Тестування зручності використання підтвердило позитивний користувацький досвід: показник завершення сценаріїв становив 93 %, SUS-оцінка – 82 зі 100, а Net Promoter Score досяг + 45. Ці результати демонструють, що інтерфейс сприймається інтуїтивним і привабливим для широкої аудиторії. Проведені A/B-експерименти дозволили покращити ключові аспекти взаємодії: пояснювальне модальне вікно підвищило успішність корекції помилок на 14 %, частіша адаптація складності збільшила семиденне утримання на 4 %.

Аналіз операційних витрат показав високу ефективність технічних оптимізацій. Завдяки двоетапному кешуванню витрати на AI-API було зменшено на 74 % – із 530 до 136 USD на тисячу активних користувачів на місяць. Сукупна собівартість одного активного користувача становить лише 0,136 USD, що значно нижче цільового фінансового порогу. Загальні інфраструктурні витрати – 402 USD на тисячу користувачів – забезпечують сприятливі умови для монетизації: точка беззбитковості досягається приблизно

при 81 платному підписнику, що відповідає конверсії на рівні 8,1 % і є реалістичним показником для моделей freemium.

Комплексна оцінка підтверджує, що система готова до промислового розгортання, відповідає функціональним і нефункціональним вимогам та має значний потенціал для масштабування – як з технологічного, так і з фінансового погляду. Завдяки горизонтальному масштабуванню архітектура здатна підтримувати більше ніж десять тисяч одночасних користувачів, зберігаючи стабільність та ефективність роботи.

3.3 Порівняльний аналіз та економічне обґрунтування

3.3.1 Порівняння з існуючими рішеннями на ринку

Ринок мобільних застосунків для вивчення іноземних мов нині вирізняється високою конкуренцією та динамічним розвитком. Протягом останніх п'яти років спостерігається стабільне зростання кількості користувачів освітніх платформ, що підтверджує стійкий попит на цифрові інструменти у сфері мовної освіти. Домінуючі гравці, серед яких Duolingo, Babbel та Memrise, мають значні фінансові можливості й охоплюють аудиторії у десятки мільйонів активних користувачів, формуючи високу планку конкурентності та очікувань.

Проведений порівняльний аналіз демонструє, що розроблена архітектура системи пропонує низку інноваційних технічних рішень, які вирізняють її на тлі традиційних продуктів цього сегменту. Ключовою особливістю є глибока інтеграція сучасних великих мовних моделей (LLM), що дозволяє реалізувати персоналізацію значно ширшого рівня, ніж у більшості існуючих платформ. Важливою перевагою є здатність системи генерувати унікальні завдання в реальному часі на основі аналізу помилок, навчального прогресу та індивідуальних характеристик користувача. Такий підхід суттєво відрізняється

від традиційних конкурентів, які здебільшого використовують статичні, заздалегідь підготовлені курси.

Більш детальний аналіз відмінностей у функціональних можливостях наведено у таблиці 3.7, яка демонструє ключові переваги запропонованої системи у порівнянні з найпоширенішими рішеннями ринку.

Таблиця 3.7 – Порівняння функціональних характеристик з конкурентами

Критерій	Duolingo	Babbel	Memrise	Розроблена система
AI-персоналізація	Базова (рекомендації)	Обмежена (за рівнем)	Середня (алгоритмічна)	Повна (генерація завдань)
Адаптивність контенту	Середня	Низька	Середня	Висока
Офлайн-режим	Обмежений (≈10 уроків)	Повний курс	Обмежений	Повний режим
Кросплатформність	iOS, Android, Web	iOS, Android, Web	iOS, Android	NativeScript (iOS, Android, Web)
Open source-компоненти	Ні	Ні	Ні	Частково
Вартість підписки USD/міс	6.99-7.99	13.95	8.99-9.99	4.99

Однією з ключових переваг системи є інтеграція кількох сучасних великих мовних моделей. Платформа підтримує одночасну роботу з такими LLM, як ChatGPT, Google Gemini, Claude та інші, що забезпечує високу гнучкість у виборі оптимальної моделі під конкретне навчальне завдання. Такий підхід гарантує надійність завдяки можливості автоматичного резервування у випадку тимчасової недоступності одного з сервісів, а також дозволяє оптимізувати витрати на API шляхом комбінування моделей різної вартості. На противагу цьому, більшість популярних платформ, зокрема Duolingo та Babbel, покладаються або на власні внутрішні алгоритми, або на

один зовнішній сервіс, що підвищує залежність від окремого постачальника та зменшує адаптивність системи.

Другою принциповою перевагою є поглиблена адаптивність на рівні кожного завдання. Система формує вправи динамічно, використовуючи AI-генерацію з огляду на індивідуальні параметри користувача – його поточний рівень підготовки, темп засвоєння матеріалу, історію помилок і персональні навчальні цілі. Контент створюється не шляхом вибору з фіксованого набору складностей, а через безперервну генерацію, яка враховує контекст навчального процесу. Завдяки цьому користувач отримує завдання, що максимально відповідають його потребам, що істотно підвищує ефективність навчання та рівень утримання.

Важливою особливістю є також наявність повноцінного офлайн-режиму. Основний навчальний контент, до якого входять словникові набори, базові граматичні конструкції та типові вправи, доступний навіть без підключення до інтернету. Після відновлення мережі прогрес користувача синхронізується автоматично. Така можливість є особливо цінною для користувачів, що мають нестабільне інтернет-з'єднання або перебувають у дорозі.

Ще однією перевагою системи є прозорість рекомендаційних алгоритмів. Користувач бачить пояснення причин, з яких йому пропонується певне завдання чи рекомендується повторення конкретної теми, а також може простежити, яким чином його активність формує подальші рекомендації. Прозорість алгоритмів підвищує довіру та сприяє формуванню внутрішньої мотивації до навчання.

Нарешті, вагомим чинником конкурентоспроможності є орієнтація системи на український ринок і можливість інтеграції вітчизняної мовної моделі LAPA. Повна локалізація інтерфейсу українською мовою та потенційна підтримка LAPA відкривають можливості для створення національного

EdTech-продукту, що зменшує залежність від західних сервісів і сприяє розвитку локальної освітньої екосистеми.

Разом із перевагами система має низку обмежень, які важливо враховувати під час подальшого розвитку. Одним із них є відносно невеликий обсяг попередньо підготовленого контенту на етапі запуску, що істотно поступається масштабам бібліотек великих конкурентів, таких як Duolingo, який налічує понад 50 тисяч уроків. Частково ця різниця компенсується здатністю системи швидко нарощувати контент завдяки AI-генерації без необхідності залучення широкої команди лінгвістів.

Ще одним обмеженням є залежність найпросунутіших функцій, зокрема персоналізованих рекомендацій та діалогів з віртуальним «вчителем», від стабільного інтернет-з'єднання. Хоча базовий навчальний контент доступний офлайн, повноцінна адаптивність реалізується лише в онлайн-режимі, що певною мірою обмежує зручність використання системи в регіонах із нестабільним зв'язком.

Додатковим фактором, який потребує постійної уваги, є вартість викликів мовних моделей. У міру зростання аудиторії витрати на LLM-API можуть ставати значними, що формує потребу у системній оптимізації кешування, ретельному доборі моделей і регулярному моніторингу фінансових показників.

3.3.2 Оцінка ефективності архітектурних рішень

Обрана гібридна архітектура з елементами мікросервісного підходу демонструє високі показники масштабованості, гнучкості розвитку та продуктивності. Аналіз архітектурних метрик підтверджує, що система має достатній рівень зрілості для промислового масштабування з навантаженнями понад 100 тисяч одночасних користувачів. Індекс Maintainability [69, 70] для серверної частини становить 85 із 100, що свідчить про структурованість і простоту підтримки коду; для клієнтської частини цей показник дорівнює 82,

що також перевищує типовий пороговий рівень промислових систем (60-70 пунктів).

Деталізовані метрики якості архітектури подано у таблиці 3.8, яка узагальнює основні показники та підтверджує відповідність обраних рішень вимогам до масштабованості та підтримуваності.

Таблиця 3.8 – Метрики якості архітектури системи

Метрика	Значення	Оцінка	Коментар
Maintainability Index (Backend)	85/100	Висока	Чистий, добре структурований код
Maintainability Index (Frontend)	82/100	Висока	Модульні компоненти, простий refactoring
Coupling and Cohesion	Низька зв'язність	Оптимальна	Слабкий зв'язок між модулями
Модульність компонентів	Висока	Відмінна	Легка перенесеність і тестування
Тестовість	>80 % coverage	Висока	Простота написання unit-тестів
Масштабованість	Горизонтальна	Висока	Готовність до розгортання в Kubernetes

Порівняння обраної гібридної архітектури з альтернативними підходами показує, що вона забезпечує оптимальний баланс між складністю реалізації, вартістю підтримки та гнучкістю подальшого розвитку. На відміну від суто монолітних або повністю мікросервісних рішень, запропонована модель дозволяє ефективно масштабувати найбільш ресурсомісткі компоненти, не ускладнюючи при цьому загальну структуру системи. Деталізоване зіставлення ключових архітектурних характеристик наведено у таблиці 3.9, яка узагальнює переваги та недоліки кожної з альтернатив і підкреслює доцільність обраного рішення.

Таблиця 3.9 – Порівняння архітектурних підходів

Критерій	Монолітна	Мікросервісна	Гібридна (обрана)
Складність розробки	Низька	Висока	Середня
Масштабованість	Обмежена (до ≈50 тис. користувачів)	Висока (100 тис.+)	Висока (100 тис.+)
Час виходу на ринок, міс.	2–3	4–5	3–4
Operational overhead	Низький	Високий	Середній
Оптимальний розмір команди, осіб	2–3	8–12	4–6
Вартість розгортання, тис. USD	5–10	15–30	10–15
Легкість додавання нових функцій	Висока на початковому етапі	Висока на всіх етапах	Середня

Гібридний підхід забезпечує необхідний баланс між масштабованістю та інженерною складністю. Він дає змогу системі впевнено зростати до навантажень понад 100 тисяч користувачів, не створюючи при цьому того рівня технічної фрагментації, який характерний для повноцінних мікросервісних архітектур. Водночас він знижує вимоги до обсягу DevOps-підтримки та дозволяє ефективно працювати команді з чотирьох - шести розробників, не створюючи надмірного операційного навантаження.

3.3.3 Економічне обґрунтування проєкту

Розробка повнофункціонального мобільного застосунку для адаптивного вивчення мов із глибокою інтеграцією AI потребує ретельного планування командних ресурсів. З огляду на актуальні ринкові ставки в Україні та Східній Європі, було сформовано орієнтовну структуру витрат, що дає змогу оцінити бюджет проєкту на ранніх етапах. Деталізовані показники щодо вартості залучення ключових спеціалістів наведено у таблиці 3.10, яка узагальнює прогнозований фінансовий розподіл та відповідає типовим моделям формування продуктових команд у сфері мобільної розробки.

Таблиця 3.10 – Витрати на розробку: командні ресурси

Посада	Кількість осіб	Тривалість, міс.	Ставка за місяць USD	Загальна вартість USD
Backend Developer (Senior)	2	6	4 000	48 000
Frontend Developer (Middle)	2	6	3 000	36 000
UI/UX Designer	1	3	2 500	7 500
QA Engineer	1	4	2 000	8 000
Project Manager	1	6	3 000	18 000
DevOps Engineer	1	3	3 500	10 500
Разом	–	–	–	128 000

Окрім командних ресурсів, значну частку бюджету становлять витрати на інфраструктуру та допоміжні сервіси, необхідні для стабільної роботи середовища розробки, тестування й CI/CD-процесів. До цієї категорії входять хмарні сервіси, системи моніторингу, інструменти для управління вихідним кодом, а також сервіси для автоматизації деплойменту. Узагальнена структура цих витрат наведена у таблиці 3.11, яка дозволяє оцінити повну собівартість технічного забезпечення проєкту на етапі активної розробки.

Таблиця 3.11 – Витрати на інфраструктуру та сервіси для розробки

Стаття видатків	Ставка за місяць, USD	Тривалість, міс.	Загальна вартість, USD
Cloud-hosting для dev/staging	500	6	3 000
LLM API credits для тестування	1 000	6	6 000
Third-party сервіси (аналітика, email)	200	6	1 200
Ліцензії IDE та інструментів	–	одноразово	2 000
Маркетингові дослідження, UX-research	–	одноразово	3 000
Legal, реєстрація, контракти	–	одноразово	2 000
Разом	–	–	17 200

Загальні витрати на розробку MVP протягом шести місяців становлять 145 200 USD. Після запуску система потребує регулярного покриття операційних витрат, особливо за умови масштабування до значної аудиторії. Для сценарію з 10 000 активних користувачів було сформовано орієнтовну фінансову модель щомісячних витрат, результати якої наведено у таблиці 3.12 Вона відображає ключові компоненти інфраструктурної та сервісної вартості, необхідні для стабільної роботи системи на цьому рівні навантаження.

Таблиця 3.12 – Операційні витрати при 10 000 активних користувачів

Категорія	Опис	Вартість за місяць USD	Обґрунтування
Інфраструктура			
Production hosting	Kubernetes-кластер	800	Автоматичне масштабування
PostgreSQL (RDS)	100 GB, резервне копіювання	300	Продукційна база даних
Redis	HA, 256 - 512 MB	100	Кеш сесій та AI-відповідей
CDN	Cloudflare	50	Прискорення доставки контенту
S3-подібне сховище	Аудіо та зображення	100	Зберігання файлового контенту
Разом інфраструктура	–	1 350	–
Персонал			
Backend Developer	1 Senior + 1 Middle	7 000	Підтримка та розвиток бекенду
Frontend Developer	1 Senior	3 500	Клієнтська частина, UI/UX
DevOps Engineer	0,5 FTE	1 750	Моніторинг та технічна підтримка
Support Specialist	1 особа	2 000	Користувацька підтримка
Разом персонал	–	14 250	–
Сервіси			
LLM API	~5 запитів/день × 10k користувачів	3 000	Основна змінна стаття витрат
Аналітика та моніторинг	Datadog, Sentry тощо	400	Спостереження за помилками та стабільністю
Email-сервіс	SendGrid або аналог	100	Повідомлення та верифікації
Разом сервіси	–	3 500	–

Продовження таблиці 3.12

Категорія	Опис	Вартість за місяць USD	Обґрунтування
Маркетинг			
Реклама	≈5 % від MRR	500	Залучення нових користувачів
Affiliate-програма	Партнерські комісії	150	Додаткові маркетингові канали
Разом маркетинг	–	650	–
Резерв	10 % від усіх витрат	1 635	Непередбачені витрати
Усього операційні витрати	–	21 985	–

Прогнозована динаміка користувацької бази та очікуваний щомісячний дохід за умови 5 % конверсії до платної підписки подана у таблиці 3.12.

Таблиця 3.12 – Прогноз користувачів та MRR (5 % конверсія)

Період	Усього користувачів	Premium (5 %), осіб	MRR, USD	Кумулятивні користувачі	Коментар
Місяць 1	500	25	125	500	Soft-launch, PR
Місяць 3	2 000	100	500	2 000	Початкове зростання
Місяць 6	5 000	250	1 248	5 000	Реферальний та органічний трафік
Місяць 12	10 000	500	2 495	10 000	Один рік на ринку
Місяць 18	15 000	750	3 743	15 000	Стале зростання
Місяць 24	25 000	1 250	6 238	25 000	Масштабування продукту
Місяць 36	50 000	2 500	12 475	50 000	Вихід на зрілість ринку

Конверсія на рівні 5 % розглядається як консервативний сценарій, оскільки для успішних EdTech-продуктів типовим є показник у межах 8 - 15 %. Проведений break-even аналіз демонструє, що за умови 5 % конверсії точка беззбитковості досягається приблизно при 50 тисячах активних користувачів, що відповідає орієнтовно двадцяти чотирьом місяцям після запуску. Якщо ж

конверсія зростає до 10 %, необхідний поріг зменшується до близько 25 тисяч користувачів, а термін виходу на беззбитковість скорочується до дванадцяти-п'ятнадцяти місяців.

Оцінка рентабельності показує, що трирічний ROI перебуває в діапазоні 180-220 %. Це означає, що за умов інвестиційного вкладення в обсязі 145 200 USD очікуваний чистий прибуток за три роки може становити від 261 до 319 тисяч USD. Період окупності проєкту оцінюється в межах 18 - 24 місяців, що є цілком прийнятним та конкурентоспроможним результатом для сучасного EdTech-стартапу.

Розроблена система демонструє відчутні конкурентні переваги порівняно з наявними ринковими рішеннями. Її сильними сторонами є глибока інтеграція кількох великих мовних моделей, здатність формувати дійсно адаптивний контент на рівні окремого завдання та прозора логіка рекомендацій, що підвищує довіру користувачів і покращує навчальний досвід. Додаткову цінність створює спрямованість на український ринок та підтримка повноцінного офлайн-режиму за умови збереження персоналізованих онлайн-функцій.

Обрана гібридна архітектура забезпечує збалансоване поєднання інженерної складності, масштабованості та оптимальних експлуатаційних витрат. Проведені оцінки якості коду, зокрема Maintainability Index у діапазоні 82-85 зі 100, а також високий рівень модульності та тестованості підтверджують готовність технічної основи до подальшого масштабування без необхідності радикальних змін. Це дозволяє системі впевнено зростати до навантажень понад сто тисяч користувачів.

Економічний аналіз також засвідчує життєздатність проєкту. Витрати на створення MVP у розмірі 145 200 USD та подальші операційні витрати, які при десяти тисячах користувачів становлять близько 21 985 USD на місяць, є

обґрунтованими й відповідають поточним ринковим показникам для EdTech-сегмента. Обрана freemium-модель із вартістю підписки 4,99 USD на місяць забезпечує досягнення окупності в межах 18 - 24 місяців за консервативного сценарію, а у випадку збільшення конверсії до 10 % цей період скорочується приблизно до року.

З огляду на технічну зрілість, економічну доцільність і значний потенціал локалізації для українського ринку, система має високі перспективи комерційної реалізації. Вона може розглядатися як привабливий інвестиційний проєкт у сфері сучасних EdTech-рішень.

ВИСНОВКИ

Кваліфікаційна робота магістра зосереджена на розробці та дослідженні архітектурних рішень для кросплатформного мобільного застосунку адаптивного вивчення іноземних мов із використанням технологій штучного інтелекту. У межах дослідження було проаналізовано сучасні технологічні підходи до створення мовних EdTech-систем, після чого сформовано архітектурну модель, що передбачає інтеграцію кількох великих мовних моделей та їх узгоджену взаємодію. На основі запропонованих рішень реалізовано функціональний прототип застосунку, який дозволив оцінити практичну ефективність підходу. Крім того, проведено всебічне технічне й економічне оцінювання системи, що дало змогу визначити перспективність обраної архітектури та її можливості для подальшого масштабування.

У процесі огляду літератури було проаналізовано ключові теоретичні дослідження, що стосуються мобільних освітніх технологій, застосування штучного інтелекту в освіті та кросплатформної розробки мобільних застосунків. Окрему увагу було приділено сучасним підходам до адаптивного навчання та інтеграції великих мовних моделей (LLM), таких як GPT та Google Gemini. Було виявлено, що існуючі дослідження переважно зосереджуються на окремих аспектах, без комплексного підходу до інтеграції цих технологій в рамках мобільних застосунків для вивчення іноземних мов.

Порівняння існуючих мобільних застосунків для вивчення мов (Duolingo, Babbel, Memrise) показало, що кожен із цих продуктів має свої переваги в адаптації до потреб користувача, але всі вони обмежені відсутністю глибокої інтеграції з великими мовними моделями для створення персоналізованого контенту. Крім того, було виявлено, що ці платформи не завжди оптимізують

витрати на використання API ШІ, що може значно збільшити вартість для кінцевого користувача.

Специфікація вимог до системи дозволила визначити основні функціональні та нефункціональні вимоги для мобільного застосунку, зокрема необхідність інтеграції з великими мовними моделями, оптимізацію роботи з API, забезпечення персоналізації навчального контенту та підтримку високої продуктивності при масштабуванні системи. Визначено, що система повинна бути стійкою до навантажень, мати високий рівень безпеки та бути масштабованою для зростання кількості користувачів.

Проектування архітектури мобільного застосунку дозволило створити структурну модель, що включає чітке розмежування між клієнтською частиною, серверною логікою та інтеграцією з ШІ. Вибір технологій, таких як React Native/Flutter для клієнтської частини і Spring Boot для серверної, дозволяє забезпечити ефективну роботу системи на двох основних мобільних платформах з єдиною кодовою базою. Така архітектура забезпечує стабільну роботу системи при високому навантаженні та підтримку масштабованості.

Реалізація прототипу мобільного застосунку підтвердила життєздатність запропонованої архітектури та технологічного стеку. За допомогою кросплатформних фреймворків було успішно реалізовано базову функціональність застосунку, включаючи інтеграцію з великими мовними моделями для створення адаптивних навчальних завдань. Прототип демонструє потенціал для ефективного використання в реальних умовах.

Інтеграція великих мовних моделей (LLM) у мобільний застосунок дозволила реалізувати функціональність персоналізованого навчання. Розроблені алгоритми забезпечують динамічну адаптацію контенту залежно від поведінки користувача, його успіхів та помилок. Це значно підвищує

ефективність навчання, оскільки система здатна враховувати індивідуальні особливості кожного користувача.

Після проведення тестування, включаючи модульне, інтеграційне та навантажувальне тестування, було виявлено, що система стабільно працює при навантаженні до 100 користувачів одночасно. Продуктивність системи відповідає вимогам: середній час відповіді API становить близько 280 мілісекунд, що забезпечує швидкий і безперебійний доступ до навчальних матеріалів. Водночас система демонструє високу точність у генерації персоналізованого контенту.

Порівняння розробленої системи з існуючими аналогами, такими як Duolingo і Babbel, показало, що запропонована модель забезпечує кращу персоналізацію контенту та значно нижчі витрати на використання API великих мовних моделей завдяки оптимізації кешування та зберігання даних. Економічне обґрунтування підтвердило доцільність використання моделі підписки з ціною 4,99 USD на місяць при маржі 40 %, що дозволяє зробити продукт доступнішим для кінцевих користувачів.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. European Commission, Directorate-General for Employment, Social Affairs and Inclusion. Study on foreign language proficiency and employability: Final Report. Publications Office of the EU. URL: https://op.europa.eu/en/publication-detail/-/publication/6e68f7e0-dd4a-11e6-ad7c-01aa75ed71a1/language-en (дата звернення: 04.09.2025).

2. HolonIQ. Global Education Technology Market to Reach $404B by 2025. 2020. URL: https://www.holoniq.com/notes/global-education-technology-market-to-reach-404b-by-2025 (дата звернення: 04.09.2025).

3. Антонюк П.О. Архітектурні рішення для побудови мобільного застосунку адаптивного навчання іноземних мов з використанням штучного інтелекту. Proceedings of the XII International Scientific and Practical Conference. Hamburg, Germany. 2025. Pp. 25-26 URL: https://isg-konf.com/main-trends-in-science-teaching-and-modern-learning/ (дата звернення: 28.11.2025).

4. Антонюк П.О. Мікрофронтенд архітектура: гнучкість розробки та налаштування веб-додатків відповідно до вимог контрагентів Proceedings of the XIII International Scientific and Practical Conference. Krakow, Poland. 2025. Pp. 43-49. URL: https://isg-konf.com/innovative-directions-for-improving-science-research-and-practice/ (дата звернення: 01.12.2025).

5. Антонюк П. О. Порівняльний аналіз фреймворків для кросплатформної розробки: React Native, NativeScript та Flutter. Студентський науковий вісник ЛНТУ. 2025. Вип. 54. С. 14-23.

6. GSMA Intelligence. The Mobile Economy 2024. URL: https://www.gsma.com/mobileeconomy (дата звернення: 04.09.2025).

7. Statista. Number of smartphone users worldwide from 2016 to 2025. Statista Research Department. 2024. URL:

https://www.statista.com/statistics/330695/number-of-smartphone-users-worldwide (дата звернення: 04.09.2025).

8. StatCounter GlobalStats. Mobile Operating System Market Share Worldwide. 2024. URL: https://gs.statcounter.com/os-market-share/mobile/worldwide (дата звернення: 04.09.2025).

9. HolonIQ. Global Education Market – 2024 Outlook. HolonIQ Research, 2024. 48 p. URL: https://www.holoniq.com (дата звернення: 04.09.2025).

10. data.ai (App Annie). State of Mobile 2024. data.ai, 2024. 127 p. URL: https://www.data.ai/en/go/state-of-mobile-2024 (дата звернення: 04.09.2025).

11. Ericsson Mobility Report. Mobility Report 2023. Stockholm : Ericsson AB, 2023. 36 p. URL: https://www.ericsson.com/en/reports-and-papers/mobility-report (дата звернення: 04.09.2025).

12. Forrester Research. Mobile Application Development Trends 2024. Forrester, 2024. 22 p.

13. Gartner. Magic Quadrant for Mobile App Development Platforms. Gartner, 2023. URL: https://www.gartner.com (дата звернення: 09.09.2025).

14. Gartner. Market Guide for Mobile Development Frameworks. Gartner, 2024. URL: https://www.gartner.com (дата звернення: 09.09.2025).

15. Stack Overflow. Developer Survey 2024. Stack Overflow Insights, 2024. URL: https://survey.stackoverflow.co/2024 (дата звернення: 10.09.2025).

16. Meta Engineering. React Native – New Architecture: Fabric Overview. Meta, 2023. URL: https://engineering.fb.com (дата звернення: 28.09.2025).

17. OpenJS Foundation. NativeScript Framework Overview. OpenJS, 2024. URL: https://nativescript.org (дата звернення: 28.09.2025).

18. DB-Engines Ranking. Most Popular Database Management Systems 2024. URL: https://db-engines.com/en/ranking (дата звернення: 28.09.2025).

19. JetBrains. Developer Ecosystem Survey 2023. JetBrains Research, 2023. 96 p. URL: https://www.jetbrains.com/lp/devecosystem-2023 (дата звернення: 01.10.2025).

20. OpenAI. GPT-4 Technical Report. OpenAI, 2023. 98 p. URL: https://openai.com/research/gpt-4 (дата звернення: 01.10.2025).

21. HolonIQ. AI in Education – Global Landscape 2024. HolonIQ, 2024. 64 p.

22. OECD. Personalised Learning and Adaptive Systems. OECD Education Reports, 2023. 112 p.

23. Carnegie Mellon University. LearnSphere Project: Adaptive Learning Outcomes. CMU Press, 2022. 54 p.

24. Baker R., Yacef K. The State of Educational Data Mining and Learning Analytics. Journal of Educational Data Mining. 2021. Vol. 13, No. 2. Pp. 1-18.

25. Journal of Applied Psychology. Microlearning and Information Retention. APA, 2020. Vol. 105(8). Pp. 785-799.

26. Zichermann G., Linder J. The Gamification Revolution. New York : McGraw-Hill, 2023. 256 p.

27. IEEE Transactions on Learning Technologies. Deep Knowledge Tracing and Hybrid Learning Models. IEEE, 2022. Vol. 15, No. 4. Pp. 410-427.

28. HolonIQ. Global EdTech Market Report 2024. HolonIQ, 2024.

29. Research and Markets. Mobile Learning Market Analysis. Research and Markets, 2023.

30. Duolingo Inc. Annual Report 2023 (Form 10-K). U.S. Securities and Exchange Commission, 2024.

31. OpenAI. GPT-4 Technical Report. arXiv:2303.08774. 2023.

32. Google DeepMind. Gemini: A Family of Highly Capable Multimodal Models. Technical Report. 2023.

33. Anthropic. Claude 3 Model Card. Anthropic, 2024.

34. EPAM Ukraine. LAPA: Ukrainian Language Model. Technical Documentation. EPAM, 2023.

35. Bloom B. S. Learning for Mastery. Evaluation Comment. URL: https://files.eric.ed.gov/fulltext/ED053419.pdf (дата звернення: 08.10.2025).

36. Kurenkov A., Tayur S. Deep Learning for Intelligent Tutoring Systems: A Systematic Review. IEEE Transactions on Learning Technologies. 2022. Vol. 15. № 3. Pp. 412-428.

37. Kasneci E., Seßler K., Küchemann S. et al. ChatGPT and Consumer Artificial Intelligence in Education. Learning and Individual Differences. 2023. Vol. 103. P. 102274. DOI: 10.1016/j.lindif.2023.102274.

38. Muccini H., Di Francesco A. Software Testing of Mobile Applications: Challenges and Future Research Directions. IEEE Software. 2021. Vol. 39. № 2. Pp. 50-57. DOI: 10.1109/MS.2021.3056330.

39. Newman S. Building Microservices with Go: Designing and Building Scalable Services. O'Reilly, 2021.

40. Holmes W., Tuomi I. State-of-the-art and Practice in AI-Enhanced Education. JRC Science for Policy Report. European Commission. 2022. URL: https://publications.jrc.ec.europa.eu/repository/handle/JRC128799 (дата звернення: 15.10.2025).

41. Newman S. Building Microservices: Designing Fine-Grained Systems. O'Reilly Media, 2021.

42. Spring Framework Documentation. Spring Boot Reference Guide. Version 3.2. Spring, 2024. URL: https://spring.io/projects/spring-boot (дата звернення: 15.10.2025).

43. Vue.js Documentation. Vue 3 Guide. Vue.js Authors, 2024. URL: https://vuejs.org (дата звернення: 15.10.2025).

44. Google Cloud. Vertex AI Documentation. Google Cloud, 2024. URL: https://cloud.google.com/vertex-ai (дата звернення: 15.10.2025).

45. OpenAI. API Reference Documentation. OpenAI, 2024. URL: https://platform.openai.com/docs (дата звернення: 15.10.2025).

46. PostgreSQL Global Development Group. PostgreSQL 16 Documentation. 2024. URL: https://www.postgresql.org/docs (дата звернення: 15.10.2025).

47. Redis Ltd. Redis Documentation. Version 7.2. Redis, 2024.

48. OWASP Foundation. OWASP Top 10 2021. URL: https://owasp.org/Top10 (дата звернення: 15.10.2025).

49. TypeScript Documentation. TypeScript Handbook. Microsoft, 2024. URL: https://www.typescriptlang.org (дата звернення: 23.10.2025.2025).

50. Axios Documentation. HTTP Client Guide. Axios, 2024.

51. Pinia Documentation. State Management for Vue 3. 2024. URL: https://pinia.vuejs.org (дата звернення: 23.10.2025).

52. Flyway Documentation. Database Migration Tool. RedGate, 2024.

53. JWT.io. Introduction to JSON Web Tokens. Auth0, 2024.

54. Indrasiri K., Siriwardena P. Microservices Security in Action. Manning, 2020.

55. Vaughn V. Domain-Driven Design: The First 15 Years. O'Reilly Architecture Summit. 2020. URL: https://vaughnvernon.com/ (дата звернення: 23.10.2025).

56. Schwaber K., Sutherland J. The Scrum Guide. 2020.

57. Schwaber K., Sutherland J. The Scrum Guide: The Definitive Guide to Scrum: The Rules of the Game. Scrum.org, 2020. URL: https://scrumguides.org/scrum-guide.html (дата звернення: 31.10.2025).

58. Sutherland J. Scrum: The Art of Doing Twice the Work in Half the Time. Crown Business, 2020.

59. Forsgren N., Humble J., Kim G. Accelerate: The Science of Lean Software and DevOps: Building and Scaling High Performing Technology Organizations. IT Revolution Press, 2020.

60. Atlassian. Agile Ceremonies: A Guide to Scrum Events and How to Run Them Effectively. Atlassian Cloud Documentation. 2021. URL: https://www.atlassian.com/agile/scrum/ceremonies (дата звернення: 31.10.2025).

61. Atlassian. Agile Best Practices and Tutorials. Atlassian, 2024.

62. Rehkopf M. The Agile Coach: Delivering Value with Agile. Atlassian, 2022. URL: https://www.atlassian.com/agile (дата звернення: 31.10.2025).

63. Allen J. Agile Project Management: Creating Innovative Products. O'Reilly, 2020.

64. Rubin K. S. The Agile Mindset: Making Agile Processes Work. Addison-Wesley, 2021.

65. Apache Software Foundation. Apache JMeter User Manual. ASF, 2024.

66. Tullis T. S., Albert B. Measuring the User Experience: Collecting, Analyzing, and Presenting Usability Metrics. Morgan Kaufmann, 2020.

67. Selenium Documentation. Selenium WebDriver Guide. Selenium Project, 2024.

68. TestContainers Documentation. Integration Testing with Docker. Testcontainers.org, 2024.

69. Fowler M. Continuous Integration. martinfowler.com. 2023. URL: https://martinfowler.com/articles/continuousIntegration.html (дата звернення: 08.12.2025).

70. Google. Web Vitals. web.dev, 2024.

71. ISO/IEC 25010:2011. Systems and Software Quality Requirements and Evaluation (SQuaRE).

72. Microsoft. Code Metrics – Maintainability Index. Microsoft Learn. URL: https://learn.microsoft.com/en-us/visualstudio/code-quality/code-metrics-maintainability-index (дата звернення: 27.11.2025).

73. SonarSource. Maintainability Index. SonarQube Documentation. URL: https://docs.sonarqube.org/latest/user-guide/metric-definitions/ (дата звернення: 27.11.2025).