Вінницький національний технічний університет
Факультет інтелектуальних інформаційних технологій та автоматизації
Кафедра автоматизації та інтелектуальних інформаційних технологій

МАГІСТЕРСЬКА КВАЛІФІКАЦІЙНА РОБОТА

на тему:

«Розробка клієнт-серверної системи персоналізованого навчання з адаптивним контентом на основі штучного інтелекту»

Виконав: студент 2-ого курсу групи 1ІСТ-24м
спеціальності 126 – Інформаційні системи та технології
(шифр і назва спеціальності)

Павло МОРОЗОВ
(ПІБ студента)

Керівник: к.т.н., доцент кафедри АІІТ
Володимир КОЦЮБИНСЬКИЙ
(науковий ступінь, вчене звання/посада, ПІБ керівника)

«8» грудня 2025р.

Опонент: д.т.н., професор кафедри КСУ
Марія ЮХИМЧУК
(науковий ступінь, вчене звання/посада, ПІБ опонента)

«11» грудня 2025р.

Допущено до захисту

Завідувач кафедри АІІТ

Д.т.н., проф. Олег БІСІКАЛО
(науковий ступінь, вчене звання)

«12» грудня 2025р.

Вінниця ВНТУ – 2025 рік

Вінницький національний технічний університет
Факультет інтелектуальних інформаційних технологій та автоматизації
Кафедра автоматизації та інтелектуальних інформаційних технологій
Рівень вищої освіти II-ий (магістерський)
Галузь знань – 12 – Інформаційні технології
Спеціальність – 126 – Інформаційні системи та технології
Освітньо-професійна програма – Інформаційні технології аналізу даних та зображень

ЗАТВЕРДЖУЮ
Завідувач кафедри АІТ
д.т.н., проф. Олег БІСІКАЛО
«26» вересня 2025 р.

З А В Д А Н Н Я
НА МАГІСТЕРСЬКУ КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТУ
Морозову Павлу Володимировичу
(ПІБ автора повністю)

1. Тема роботи: Розробка клієнт-серверної системи персоналізованого навчання з адаптивним контентом на основі штучного інтелекту
   Керівник роботи: к.т.н., доцент кафедри АІТ Володимир КОЦЮБИНСЬКИЙ
   Затверджені наказом ВНТУ від «24» вересня 2025 року №313.
2. Строк подання роботи студентом: до «10» грудня 2025 року
3. Вихідні данні до роботи: Операційна система Windows 10 та вище, Node.js v18.0.0, фреймворк Electron, мова програмування JavaScript, платформа Firebase Firestore для зберігання даних, Pinecone для векторного пошуку, API Claude для генерації навчального контенту, оперативна пам'ять від 4 Гб, вільне місце на диску від 500 МБ.
4. Зміст текстової частини: Вступ; Огляд та аналіз існуючих рішень; Вибір технології розробки клієнт-серверної системи; Розробка програмного забезпечення; Економічне обгрунтування науково-технічної розробки; Висновки; Список використаних джерел.

5. Перелік ілюстративного (або графічного) матеріалу: UML Deployment діаграма; UML Data Flow діаграма; ER-діаграма; UML Component діаграма; UML Sequence діаграма; UML Use Case діаграма.
6. Консультанти розділів магістерської кваліфікаційної роботи

Розділ змістової частини роботи	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
1 – 3	Володимир КОЦЮБИНСЬКИЙ, к.т.н., доцент кафедри АЦТ	16.09.25	05.12.25
4	Наталія БУРЕННІКОВА, д.е.н., проф. каф. ЕПтаВМ	01.11.25	01.11.25

7. Дата видачі завдання: «25» вересня 2025 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів магістерської кваліфікаційної роботи	Строки виконання етапів роботи	Примітка
1	Аналіз предметної області	29.09.2025–06.10.2025	
2	Розробка математичного та алгоритмічного забезпечення	06.10.2025–24.10.2025	
3	Розробка програмного забезпечення	24.10.2025–18.11.2025	
4	Тестування розробленого програмного забезпечення	18.11.2025–22.11.2025	
5	Підготовка економічної частини	22.11.2025–26.11.2025	
6	Оформлення пояснювальної записки, графічного матеріалу і презентації	26.11.2025–28.11.2025	
7	Попередній захист роботи	01.12.2025	
8	Захист роботи	18.12.2025	

Студент (підпис) Павло МОРОЗОВ (прізвище та ініціали)

Керівник роботи (підпис) Володимир КОЦЮБИНСЬКИЙ (прізвище та ініціали)

АНОТАЦІЯ

УДК 004.4:004.8

Морозов П.В. Розробка клієнт-серверної системи персоналізованого навчання на основі технологій штучного інтелекту. Магістерська кваліфікаційна робота зі спеціальності 126 – Інформаційні системи та технології, освітньо-професійна програма – Інформаційні технології аналізу даних та зображень. Вінниця: ВНТУ, 2025. 137 с.

Укр. мовою. Бібліогр.: 58 дж.; рис.: 34; табл.: 13.

Метою роботи є розробка клієнт-серверної системи персоналізованого навчання з адаптивним контентом на основі штучного інтелекту. Реалізація такої системи дозволить створити індивідуальну траєкторію навчання для кожного користувача та підвищити ефективність засвоєння навчального матеріалу.

У роботі розглянуто основні аспекти аналізу сучасних освітніх платформ, що застосовують персоналізацію, та визначено їх обмеження. Досліджено принципи побудови клієнт-серверних систем і технології, що лягли в основу розробки: фреймворк Electron для створення десктопного застосунку, хмарна інфраструктура Firebase для зберігання даних, векторна база даних Pinecone для семантичного пошуку, а також API Claude для генерації навчального контенту. Впроваджено гібридну RAG-систему для підвищення точності персоналізації.

Розроблено архітектуру системи та створено UML-діаграми. Реалізовано модулі авторизації, діагностичного тестування, управління контентом, генерації адаптивних матеріалів на основі RAG-підходу та відстеження прогресу. Проведено тестування системи та оцінювання точності персоналізації. У результаті роботи створено програмне забезпечення повного циклу персоналізованого навчання з використанням технологій штучного інтелекту.

Ключові слова: персоналізоване навчання, адаптивний контент, штучний інтелект, клієнт-серверна система, Electron, Firebase, Pinecone, Claude API, RAG, векторний пошук, машинне навчання.

ABSTRACT

Morozov P.V. Development of a Client-Server Personalized Learning System Based on Artificial Intelligence Technologies. Master's Qualification Thesis in specialty 126 – Information Systems and Technologies, educational and professional program Information Technologies of Data and Image Analysis. Vinnytsia: Vinnytsia National Technical University, 2025. 137 p.

In Ukrainian language. Bibliography: 58 sources; fig.: 34; tabl.: 13.

The aim of the thesis is to develop a client-server system for personalized learning with adaptive content based on artificial intelligence. The implementation of such a system makes it possible to create an individual learning trajectory for each user and to increase the efficiency of mastering educational material.

The study examines the main aspects of analyzing modern educational platforms that employ personalization and identifies their limitations. It explores the principles of constructing client-server systems and the technologies underlying the development: the Electron framework for creating a desktop application, the Firebase cloud infrastructure for data storage, the Pinecone vector database for semantic search, as well as the Claude API for generating learning content. A hybrid RAG system was implemented to improve personalization accuracy.

The system architecture was developed and UML diagrams were created. Modules for authentication, diagnostic testing, content management, adaptive material generation based on the RAG approach, and progress tracking were implemented. System testing and evaluation of personalization accuracy were conducted. As a result, software for a full cycle of personalized learning using artificial intelligence technologies was created.

Keywords: personalized learning, adaptive content, artificial intelligence, client-server system, Electron, Firebase, Pinecone, Claude API, RAG, vector search, machine learning.

ЗМІСТ

ВСТУП

Стрімкий розвиток штучного інтелекту та інтенсивне зростання обсягів навчального контенту зумовлюють появу нових вимог до освітніх систем. Традиційні цифрові платформи здебільшого реалізують статичні підходи до навчання – фіксовані курси, однакові для всіх користувачів, з обмеженою інтерактивністю та відсутністю персоналізації. У таких системах не враховуються індивідуальні особливості користувачів, рівень підготовки, стиль сприйняття інформації чи динаміка прогресу. Як наслідок, ефективність навчання знижується, а мотивація користувачів поступово зменшується [1].

Сучасна освіта активно переходить до адаптивних систем, що автоматично враховують потреби користувача, регулюють складність і темп навчання та формують персональні рекомендації. Вони поєднують аналіз даних і технології штучного інтелекту, забезпечуючи підґрунтя для персоналізованого навчання.

Паралельно розвиваються технології ШІ генерації контенту, які дозволяють створювати навчальні матеріали в реальному часі з урахуванням рівня підготовки користувача. Використання LLM, RAG та алгоритмів відстеження знань забезпечує гнучке формування адаптивного контенту [2].

Однак багато освітніх сервісів недоступні для україномовних користувачів або потребують постійного Інтернет-з’єднання, що обмежує навчання в умовах нестабільної мережі, адаптацією контенту ШІ та захищеним зберіганням даних.

Запропонована концепція базується на використанні Electron для клієнтської частини та серверної взаємодії через API, що забезпечує персоналізацію, генерацію навчальних матеріалів і відстеження знань для визначення рівня користувача та оптимізації навчальної траєкторії.

Метою магістерської кваліфікаційної роботи є розробка клієнт-серверної системи на основі штучного інтелекту, що забезпечує індивідуальний підхід до користувача і підтримує українську локалізацію.

Для досягнення поставленої мети необхідно вирішити такі задачі:

- Провести аналіз сучасних підходів до персоналізації навчання, адаптивних систем та AI-генерації освітнього контенту;
- Розробити архітектуру клієнт-серверної системи з урахуванням принципів безпеки, масштабованості та приватності даних;
- Створити модель даних і механізм персоналізації навчального процесу, що включає діагностичне тестування, визначення рівня знань і рекомендації подальших модулів;
- Реалізувати AI-модулі генерації контенту (тести, теорія, приклади, завдання) із застосуванням методів узгодженої генерації RAG;
- Інтегрувати клієнтський застосунок Electron із сервером для авторизації, синхронізації прогресу, кешування контенту та роботи;
- Забезпечити механізми оцінки якості згенерованого контенту та достовірності даних;
- Провести експериментальну перевірку ефективності запропонованої системи.

Об’єкт дослідження – процес автоматизованого персоналізованого навчання з використанням інтелектуальних інформаційних технологій у клієнт-серверній реалізації.

Предмет дослідження – методи та засоби побудови десктопної системи ШІ адаптивного навчання з автоматичною генерацією навчального контенту.

Методи дослідження базуються на аналізі та синтезі наукових підходів до адаптивного навчання, проєктуванні архітектури програмних систем, використанні методів машинного навчання, нейромережевих моделей генерації текстів, інформаційного пошуку з векторною індексацією та експериментальній перевірці користувацьких метрик.

Науково-технічний результат роботи полягає у створенні інтегрованої клієнт-серверної системи навчання, що поєднує декстопний застосунок на Electron із серверними сервісами ШІ персоналізації та RAG-генерації контенту. Запропоновано підхід до адаптації навчального процесу на основі моделі відстеження знань користувача, який дозволяє динамічно формувати

послідовність матеріалів, тестів та практичних завдань з урахуванням поточного рівня та темпу навчання.

Практичне значення полягає у створенні кросплатформної освітньої системи, здатної автоматично генерувати індивідуалізовані навчальні модулі, аналізувати прогрес користувача та пропонувати рекомендації щодо подальшого навчання. Розроблений застосунок може бути використаний у закладах освіти, на онлайн-курсах або для самостійного навчання, а також слугувати основою для подальших досліджень у сфері освіти ШІ та цифрової педагогіки.

Апробація та публікації матеріалів досліджень. Основні результати виконання магістерської кваліфікаційної роботи були опубліковані в матеріалах Всеукраїнської науково-практичної Інтернет-конференції студентів, аспірантів та молодих науковців «Молодь в науці: дослідження, проблеми, перспективи» (Вінниця, ВНТУ, 2025-2026 рр.) та у матеріалах LV Науково-технічної конференції підрозділів Вінницького національного технічного університету (Вінниця, ВНТУ, 2025 р.) [3].

1 ОГЛЯД ТА АНАЛІЗ ІСНУЮЧИХ РІШЕНЬ

1.1 Постановка задачі

У сучасних умовах цифровізації освіти зростає потреба в системах, які забезпечують індивідуальний підхід до кожного освітянина. Традиційні платформи дистанційного навчання, такі як Coursera, Udemy або Khan Academy, пропонують фіксовані навчальні програми, однакові для всіх користувачів незалежно від їх попередньої підготовки, темпу засвоєння матеріалу та індивідуальних потреб [4]. Це призводить до низької ефективності навчального процесу, оскільки не враховуються сильні та слабкі сторони конкретного користувача.

У галузі навчання програмуванню ця проблема стоїть особливо гостро. Здобувачі мають різний рівень базових знань: хтось уже володіє одною мовою програмування і хоче опанувати нову, інші починають з нуля. Водночас, традиційні платформи не здатні автоматично діагностувати прогалини у знаннях, формувати персоналізовані траєкторії навчання та генерувати адаптивний контент у режимі реального часу. Крім того, переважна більшість таких систем не підтримують українську мову інтерфейсу.

Наявні рішення часто обмежуються або лише рекомендаційними алгоритмами, або генерацією статичного контенту без урахування динаміки прогресу користувача. Інтеграція технологій штучного інтелекту у навчальні системи дозволяє перейти від фіксованих сценаріїв до адаптивного навчання, проте більшість існуючих платформ використовують ШІ лише для автоматизації оцінювання або створення тестів, не реалізуючи повноцінну персоналізацію навчального контенту на основі аналізу індивідуальних результатів.

Метою розробки є створення клієнт-серверної системи персоналізованого навчання програмуванню з адаптивною генерацією контенту на основі штучного інтелекту, яка усуне зазначені обмеження існуючих рішень.

Постановка задачі полягає у створенні інформаційної технології, яка автоматизує процес персоналізованого навчання програмуванню та забезпечує

формування індивідуальної траєкторії на основі результатів діагностичного тестування, включаючи:

- Розробку кросплатформного настільного застосунку на базі фреймворку Electron;
- Інтеграцію хмарної інфраструктури Firebase для автентифікації користувачів, централізованого зберігання даних та синхронізації між хмарним сховищем;
- Впровадження діагностичного тестування з аналізом по темах для виявлення прогалин у знаннях та формування персоналізованого навчального плану;
- Інтеграцію Claude API для генерації навчального контенту (теорії, практичних завдань, пояснень) у режимі реального часу з урахуванням контексту попереднього навчання;
- Реалізацію гібридної RAG-системи, яка поєднує ключовий пошук через Firebase та векторний пошук через Pinecone для підвищення точності персоналізації;
- Створення системи відстеження прогресу з аналізом помилок, візуалізацією статистики та автоматичними рекомендаціями щодо повторення тем.

Розв'язання поставленої задачі дозволить створити ефективний інструмент для самостійного навчання програмуванню, який забезпечить індивідуальний підхід до кожного користувача, підвищить мотивацію до навчання завдяки персоналізованому контенту та знизить час, необхідний для засвоєння матеріалу, за рахунок концентрації на конкретних прогалинах у знаннях замість проходження повного курсу від початку до кінця. Окрім цього, практична реалізація запропонованого рішення матиме важливе значення для розвитку національного освітнього простору, пропонуючи якісну альтернативу англомовним платформам завдяки повноцінній підтримці української мови. Використання гібридної RAG-системи гарантуватиме, що згенеровані навчальні матеріали будуть не лише адаптивними, але й фактично точними та

обґрунтованими, що мінімізує ризик отримання некоректної інформації, характерний для звичайних чат-ботів. У довгостроковій перспективі запропонована архітектура забезпечить високу масштабованість, дозволяючи легко інтегрувати нові мови програмування та технологічні стеки без необхідності переробки ядра системи.

1.2 Аналіз сучасних освітніх систем та технологій штучного інтелекту в адаптивному навчанні

Розвиток цифрових технологій істотно змінив підходи до організації навчального процесу. Традиційна модель освіти, заснована на лінійному поданні матеріалу та єдиній програмі для всіх здобувачів, поступово поступається місцем інтелектуальним освітнім системам, що враховують індивідуальні потреби, темп і стиль навчання користувача. У сучасних умовах зростає роль адаптивного навчання, персоналізації контенту та інтеграції штучного інтелекту у навчальні середовища.

Сучасна екосистема освітніх ресурсів складається з великої кількості веб-платформ і мобільних застосунків, орієнтованих на самостійне навчання. Найбільш поширеними є масові відкриті онлайн-курси, які пропонують структуровані курси, інтерактивні відеолекції та автоматизовані тести.

Основними перевагами таких платформ є масштабованість, інтерактивність подачі матеріалу та відкритий доступ до освітніх ресурсів. Однак вони мають низку обмежень:

- Відсутність глибокої адаптації до рівня користувача;
- Слабкий зворотний зв'язок;
- Переважання статичних матеріалів без інтерактивних вправ;
- Відсутність локалізованих україномовних інтерфейсів.

Персоналізоване навчання базується на гіпотезі, що кожен користувач має власний темп і стиль сприйняття інформації. У сучасних платформах

застосовуються різні підходи до персоналізації: контентна персоналізація поведінкова адаптація та генеративна персоналізація з використанням моделей ШІ.

Більшість сучасних навчальних систем функціонують у клієнт-серверній архітектурі, що забезпечує масштабованість, централізоване зберігання даних і можливість інтеграції зовнішніх сервісів. На рисунку 1.1 представлено узагальнену блок-схему роботи клієнт-серверної системи, яка забезпечує автентифікацію, обробку запитів, інтеграцію з AI/ML-моделями та централізоване збереження даних.

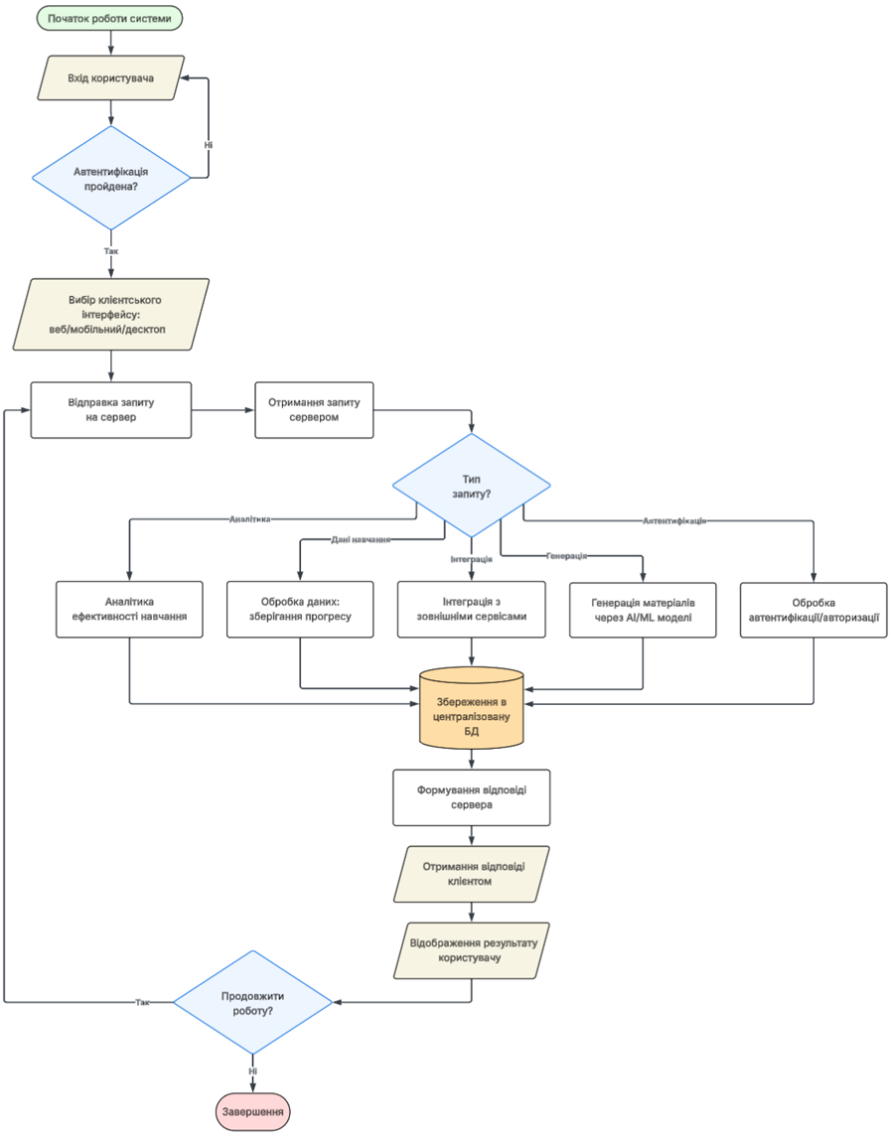

Рисунок 1.1 – Узагальнена діаграма функціонування системи персоналізованого навчання

Наявні рішення демонструють прогрес у напрямі персоналізації, проте не поєднують адаптивну генерацію контенту, локалізацію, безпечне зберігання даних системі [5-9].

Технології штучного інтелекту в адаптивному навчанні

Використання ШІ в освіті є одним із ключових напрямів цифрової трансформації навчального процесу. Інтеграція технологій ШІ стає базовою умовою створення адаптивних і персоналізованих освітніх систем нового покоління [10-11].

Машинне навчання є ядром адаптивних освітніх систем. Воно дозволяє аналізувати дані про поведінку користувачів, виявляти закономірності та формувати прогнозні моделі. Алгоритми ML будують профіль користувача, який враховує його успішність, активність, стиль навчання та помилки. Найчастіше в освіті застосовують класифікацію (Decision Tree, Random Forest, SVM) для оцінки рівня знань, кластеризацію (K-Means) для групування користувачів та регресійні моделі для прогнозування успішності [12-13].

Нейронні мережі доповнюють можливості ML у завданнях автоматичного оцінювання письмових відповідей, розпізнавання емоцій користувача або класифікації рівнів складності матеріалу. Глибокі нейронні мережі стали основою багатьох освітніх рішень IS, наприклад у платформах Squirrel AI чи DreamBox Learning [14-16].

Обробка природної мови (NLP) дає змогу системам сприймати людську мову як інтерфейс взаємодії. Застосування NLP охоплює автоматичне створення запитань і тестів, оцінювання письмових робіт, семантичний пошук навчальних матеріалів та діалогові освітні агенти. Наприклад, Khan Academy AI Tutor використовує GPT-моделі для ведення діалогу зі здобувачем та пояснення складних понять [17-19].

RAG – сучасна технологія, що поєднує методи пошуку інформації та генерації текстів. Вона дозволяє системам створювати навчальні матеріали на основі достовірних джерел, забезпечуючи точність і контекстну релевантність.

На відміну від класичної генерації контенту, RAG спочатку знаходить потрібні документи в базі знань, а потім використовує мовну модель для створення узгодженого тексту. Це усуває проблему «галюцинацій» мовних моделей і підвищує надійність створюваного контенту [20].

Рекомендаційні системи аналізують попередній досвід користувача для добору найбільш релевантних матеріалів, використовуючи контент та фільтрацію на основі поведінки користувачів. Learning Analytics охоплює збір, обробку та інтерпретацію даних про освітню діяльність користувачів, що дозволяє постійно вдосконалювати навчальний контент.

Типова архітектура сучасної адаптивної системи складається з п'яти основних компонентів: модуля збору даних, аналітичного блоку, AI-ядра для генерації контенту, бази знань із векторною індексацією та інтерфейсу взаємодії користувача. Взаємодія між цими компонентами відбувається за клієнт-серверною моделлю.

Незважаючи на переваги, інтеграція ШІ супроводжується викликами: контроль достовірності контенту, етичні аспекти конфіденційності даних, проблема прозорості алгоритмів та ресурсомісткість. Для мінімізації ризиків сучасні системи використовують гібридний підхід: обробку персональних даних на локальному клієнті та генерацію контенту на сервері [21].

Аналіз показує, що технології ШІ створюють основу для нової парадигми освіти, де навчальний процес стає гнучким і орієнтованим на конкретного користувача. Поєднання машинного навчання, NLP, RAG і систем аналітики дозволяє створювати середовище унікального шляху навчання. Однак більшість існуючих платформ застосовують лише часткову інтеграцію AI-компонентів і не забезпечують комплексного підходу з урахуванням локалізації та безпечного зберігання даних. Це підкреслює доцільність розробки власної клієнт-серверної системи персоналізованого навчання з інтеграцією AI-адаптації [22].

1.3 Аналіз і порівняння існуючих аналогів систем персоналізованого навчання

Аналіз наявних рішень у сфері персоналізованого та адаптивного навчання є ключовим етапом при розробці нових інтелектуальних освітніх систем. Ґрунтовне дослідження існуючих технологій, архітектурних підходів і функціональних можливостей дозволяє визначити сучасні тенденції розвитку ринку, а також окреслити наявні проблеми й незадоволені потреби користувачів.

Особливої актуальності та складності цей етап набуває у випадку систем, що інтегрують штучний інтелект для адаптації навчального процесу. Сучасні платформи ШІ демонструють значний прогрес у напрямі генерації та персоналізації навчальних матеріалів, проте більшість із них орієнтовані на вебсередовище, не підтримують офлайн-доступ і не забезпечують україномовної локалізації.

Проведення детального порівняльного аналізу дозволяє оцінити ступінь зрілості існуючих освітніх систем і визначити, наскільки вони відповідають сучасним вимогам до адаптивності, гнучкості та безпеки даних. Для цього використовується SWOT-аналіз, що є ефективним інструментом систематизації інформації про конкурентні рішення.

У межах цього підрозділу SWOT-аналіз використовується для виявлення чотирьох ключових груп характеристик для кожного розглянутого аналога:

- Сильні сторони – інноваційні рішення, функції або технології, що забезпечують конкурентні переваги продукту;
- Слабкі сторони – технічні або концептуальні обмеження, що знижують ефективність навчання чи комфорт користувача;
- Можливості – потенційні напрями розвитку, інтеграції або вдосконалення продукту з урахуванням сучасних тенденцій ШІ;
- Загрози – зовнішні фактори або ризики, які можуть вплинути на подальше існування чи конкурентоспроможність системи.

Під час дослідження було виявлено низку актуальних освітніх рішень, які можна умовно розділити на три основні категорії:

- Вебплатформи та онлайн-курси, наприклад: Coursera, Khan Academy, Udemy, edX. Пропонують масштабоване навчання через браузер;
- Застосунки такі як SoloLearn, Duolingo, Codecademy Go, орієнтовані на короткі інтерактивні сеанси навчання;
- Переважно десктопні рішення такі як Programming Hub і JetBrains Academy, які забезпечують автономність і глибшу інтеграцію з локальними сервісами.

Кожна з цих категорій має власні переваги, але й специфічні обмеження. Зокрема, більшість платформ зосереджуються на контентній частині (тести, відео, статті), але не реалізують повноцінну систему адаптації контенту під рівень знань користувача. Крім того, середовище навчання переважно є онлайн-залежним, що унеможливлює ефективне використання у випадку обмеженого доступу до мережі [23].

Окрему увагу приділено аналізу україномовного сегмента ринку, який залишається слабо розвиненим. Більшість міжнародних освітніх сервісів не мають українського інтерфейсу або локалізують лише частину контенту. Це ускладнює доступ українських користувачів до сучасних інтелектуальних платформ і підкреслює потребу у створенні власної клієнт-серверної системи персоналізованого навчання з українською локалізацією та модулем ШІ генерації навчального контенту.

SoloLearn – це одна з найпопулярніших освітніх платформ для вивчення мов програмування, орієнтована переважно на мобільні пристрої проте користувачі можуть використовувати застосунок через емулятори як Droid4X, MEmu або найбільш популярний BlueStacks. Приклад інтерфейсу мобільної версії платформи подано на рисунку 1.2. Основна мета – надати можливість швидко й інтерактивно опанувати основи програмування, незалежно від рівня попередньої підготовки.

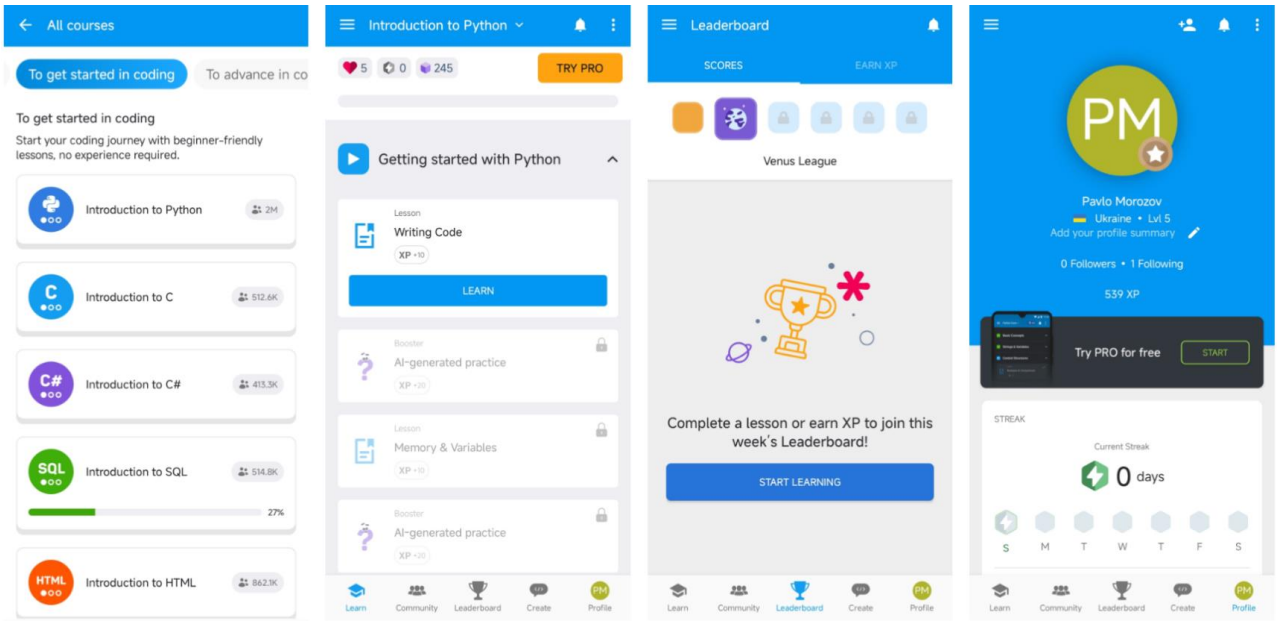

Рисунок 1.2 – Приклад вигляду застосунку SoloLearn

Платформа реалізована з використанням мобільного фреймворку Flutter та хмарних серверних сервісів Firebase, що дозволяє забезпечити високу продуктивність, швидку синхронізацію прогресу користувача та кросплатформність (Android, iOS, Web).

В застосунок інтегровано AI Tutor – персонального віртуального наставника на основі GPT-моделей. Цей модуль дозволяє користувачам ставити запитання в природній формі, отримувати пояснення до помилок у коді, рекомендації щодо тем для повторення та навіть приклади оптимізації програмних рішень.

AI Tutor аналізує дії користувача, темп навчання та успішність виконання завдань, формуючи персоналізовані підказки. Такий підхід наближає систему до адаптивного навчання, проте алгоритми поки що не є повноцінно діагностичними – вони не враховують рівень знань у динаміці, а переважно реагують на безпосередні дії користувача.

У 2025 році SoloLearn також запровадила функцію «Explain with AI», яка генерує пояснення коду, коментарі та приклади використання функцій, що підвищує глибину розуміння матеріалу.

Система функціонує у класичній клієнт-серверній архітектурі. Клієнтська частина відповідає за візуалізацію навчальних курсів, тестів і чатів користувачів, тоді як серверна – за зберігання контенту, автентифікацію, статистику прогресу та роботу модуля ШІ.

Взаємодія з AI Tutor здійснюється через API, що підключає GPT-моделі. Це дає змогу виконувати генерацію контенту на серверному рівні, зберігаючи клієнт легким і швидким.

Головною перевагою SoloLearn є унікальне поєднання декількох ключових елементів: гейміфікації навчального процесу, що робить освоєння програмування захопливим; складової, яка включає змагання з іншими учнями, систему рейтингів та активні обговорення в спільноті; а також персоналізованої взаємодії з помічником ШІ. Узагальнення сильних і слабких сторін платформи представлено у SWOT-аналізі на рисунку 1.3

Рисунок 1.3 – SWOT-аналіз для SoloLear

Водночас система має суттєві обмеження. Вона орієнтована на короткі мікроуроки та не забезпечує повноцінної генерації контенту або адаптації

складності матеріалів під конкретного користувача. Відсутня також офлайн-підтримка та україномовна локалізація, що ускладнює використання застосунку у освітньому середовищі.

Таким чином, SoloLearn демонструє поступ у напрямі інтеграції штучного інтелекту в освітній процес, однак його можливості поки що зосереджені на допоміжних функціях – поясненні коду та рекомендаціях. Система не реалізує повної адаптації контенту під рівень знань користувача, не підтримує офлайн-режиму та не забезпечує україномовної локалізації. Це відкриває перспективи для розробки нових клієнт-серверних рішень, здатних поєднати персоналізацію, AI-генерацію навчальних матеріалів і зручне десктопне середовище.

Codecademy Go – одна з найвідоміших освітніх онлайн-платформ для вивчення програмування, веброзробки, аналізу даних та комп'ютерних наук. Станом на 2025 рік платформа має понад 100 мільйонів зареєстрованих користувачів і є одним із лідерів на ринку онлайн-освіти. Приклад інтерфейсу мобільної версії платформи подано на рисунку 1.4.

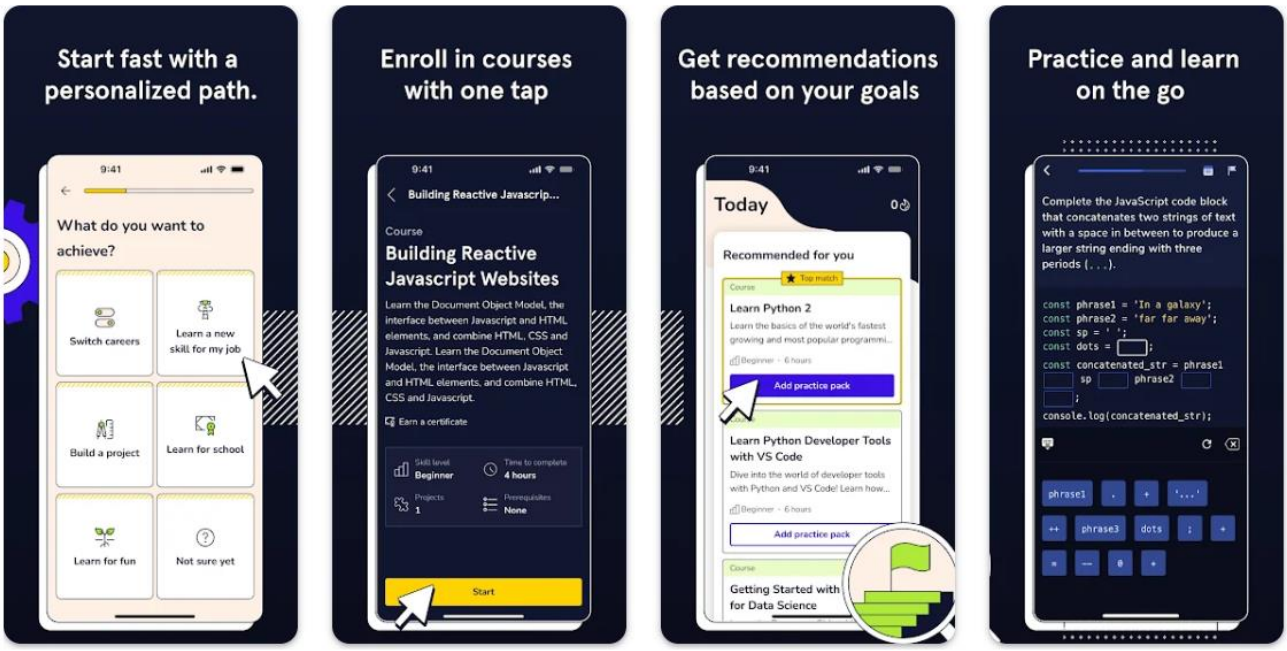

Рисунок 1.4 – Приклад вигляду застосунку Codecademy Go

Основна мета Codecademy полягає у створенні інтерактивного середовища для практичного навчання, де користувачі одразу бачать результат виконаного коду без потреби встановлення додаткового ПЗ. Вебзастосунок реалізований за клієнт-серверною архітектурою з використанням JavaScript, React та Node.js, що забезпечує високу масштабованість і динамічне оновлення контенту.

Codecademy активно впроваджує елементи штучного інтелекту. Найпомітнішим нововведенням став Codecademy AI Assistant, інтегрований на базі моделей GPT-4. Помічник працює як персональний наставник, який може:

- Пояснювати помилки в коді користувача в реальному часі;
- Відповідати на запитання у природній мові;
- Надавати контекстні підказки щодо теми, яку опановує здобувач;
- Автоматично створювати додаткові вправи та приклади.

AI Assistant також підтримує генерацію практичних завдань відповідно до теми поточного уроку. Наприклад, якщо користувач проходить модуль із Python, система може запропонувати згенероване завдання «на закріплення», сформоване з урахуванням його попередніх помилок.

Це дозволяє говорити про часткову реалізацію адаптивного навчання – помічник ШІ реагує не лише на запити користувача, а й на історію його взаємодії з контентом, формуючи рекомендації щодо подальших дій.

Застосунок функціонує на веб-орієнтованій клієнт-серверній моделі, де основна логіка обробки даних розміщена на сервері. На клієнтському рівні користувач отримує інтерактивне навчальне середовище з редактором коду, консоллю виконання, підказками та AI-чатом.

Серверна частина забезпечує доступ до бази навчальних матеріалів, аналітики користувацької активності та API для помічника ШІ. Вона інтегрується з хмарною інфраструктурою Amazon Web Services (AWS), що дозволяє масштабувати обчислення під великі обсяги запитів.

Codecademy активно використовує системи аналітики навчальних даних – збір інформації про швидкість виконання завдань, кількість спроб і час

перебування у сесіях. Ці дані застосовуються для вдосконалення контенту й алгоритмів рекомендацій.

Переваг, які забезпечили лідерські позиції на ринку онлайн-навчання: висока якість навчальних матеріалів, інтерактивність, професійна розробка курсів, наявність інтеграцій з GitHub та Google Workspace.

Впровадження модулів ШІ підвищило зручність і гнучкість навчального процесу, однак система все ще не реалізує повноцінну модель персоналізованого навчання, складність, темп і послідовність матеріалу формуються автоматично.

Крім того, Codecademy повністю відсутня україномовна локалізація. Додатково платформа має підписну модель, що обмежує доступ до функцій ШІ для безкоштовних користувачів. Узагальнення сильних і слабких сторін платформи представлено у SWOT-аналізі на рисунку 1.5.

Рисунок 1.5 – SWOT-аналіз для Codecademy Go

Codecademy продемонструвала суттєвий прогрес у впровадженні штучного інтелекту в освітній процес і створила передумови для появи адаптивних навчальних систем нового покоління. Однак, попри інтеграцію

помічника ІШ, система залишається переважно реактивною – вона відповідає на дії користувача, але не формує індивідуальний навчальний маршрут на основі аналізу знань.

Відсутність офлайн-підтримки та локалізації українською мовою обмежує її використання у національному освітньому середовищі. Це підкреслює необхідність створення власного клієнт-серверного рішення, орієнтованого на україномовного користувача, з інтегрованими модулями ШІ персоналізації, генерації контенту та оцінювання прогресу.

Programming Hub – це популярний мобільний застосунок і веб-платформа для вивчення мов програмування, веброзробки, машинного навчання та суміжних ІТ-напрямів. Приклад інтерфейсу декстопної версії платформи подано на рисунку 1.6. Розроблений індійською компанією Nexino Labs, сервіс позиціонує себе як «інтерактивна енциклопедія програмування», орієнтована на короткі навчальні сесії та практичні приклади.

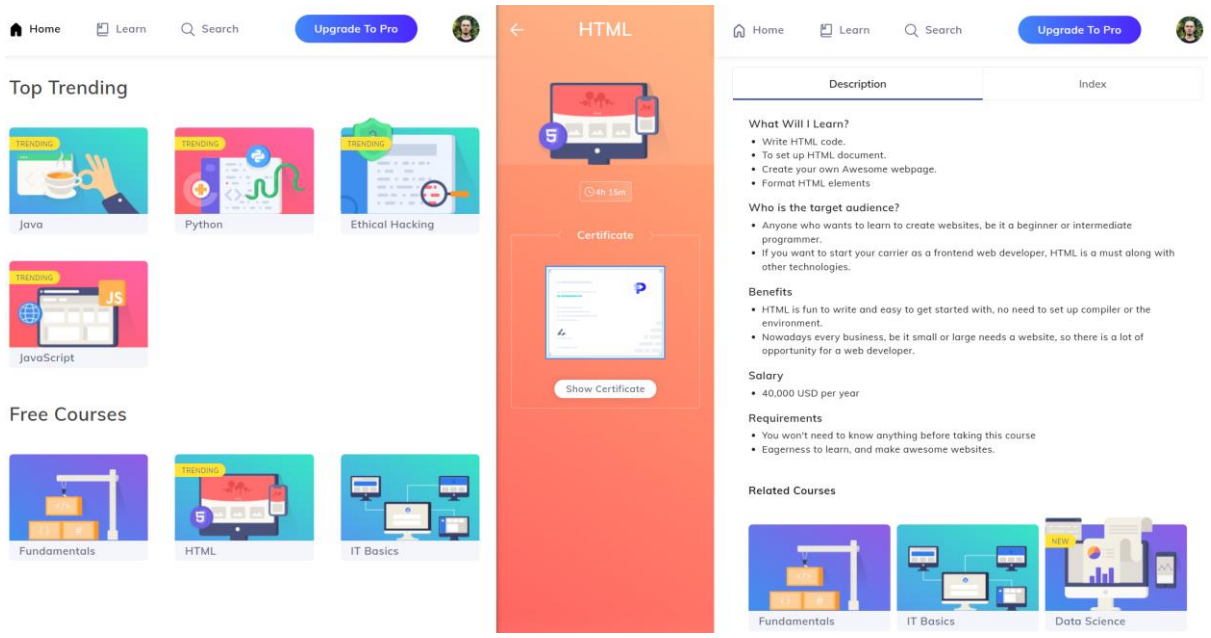

Рисунок 1.6 – Приклад вигляду застосунку Programming Hub

Платформа реалізована на основі клієнт-серверної архітектури з використанням фреймворків Flutter для мобільної частини і Firebase для зберігання даних та авторизації. Завдяки цьому Programming Hub підтримує

синхронізацію між пристроями, а також часткову роботу без підключення до Інтернету – на відміну від більшості конкурентів.

AI Teacher аналізує відповіді користувача в тестах і під час практичних завдань, пропонуючи додаткові приклади та пояснення у випадку помилок. Крім того, система вміє генерувати короткі навчальні нотатки для повторення матеріалу – це один із перших випадків використання технології retrieval-based generation у мобільних навчальних застосунках.

Попри використання компонентів ШІ, рівень адаптивності Programming Hub залишається обмеженим. Система не будує повноцінного профілю знань користувача, не прогнозує складність наступних тем і не виконує автоматичне коригування навчальної траєкторії.

Архітектурно Programming Hub складається з трьох шарів:

- Клієнтський застосунок, який забезпечує взаємодію користувача з інтерфейсом, локальне кешування даних і базову офлайн-функціональність;
- Хмарний сервер, який зберігає навчальні матеріали, результати тестів і аналітику використання;
- Модуль ШІ, що працює через зовнішній API (OpenAI, Google Vertex AI) і генерує пояснення, короткі тексти або приклади коду на запит користувача.

Завдяки офлайн-режиму користувач може переглядати завантажені уроки без доступу до мережі, а після відновлення з'єднання дані синхронізуються з сервером. Такий підхід частково вирішує проблему доступності навчання у місцях із низькою якістю Інтернету. Трирівнева архітектура забезпечує гнучкість масштабування: клієнтський застосунок залишається швидким, хмарний сервер обслуговує мільйони користувачів, а модуль ШІ підключається за потреби. Розділення функціональності спрощує оновлення системи. Узагальнення сильних і слабких сторін платформи представлено у SWOT-аналізі на рисунку 1.7.

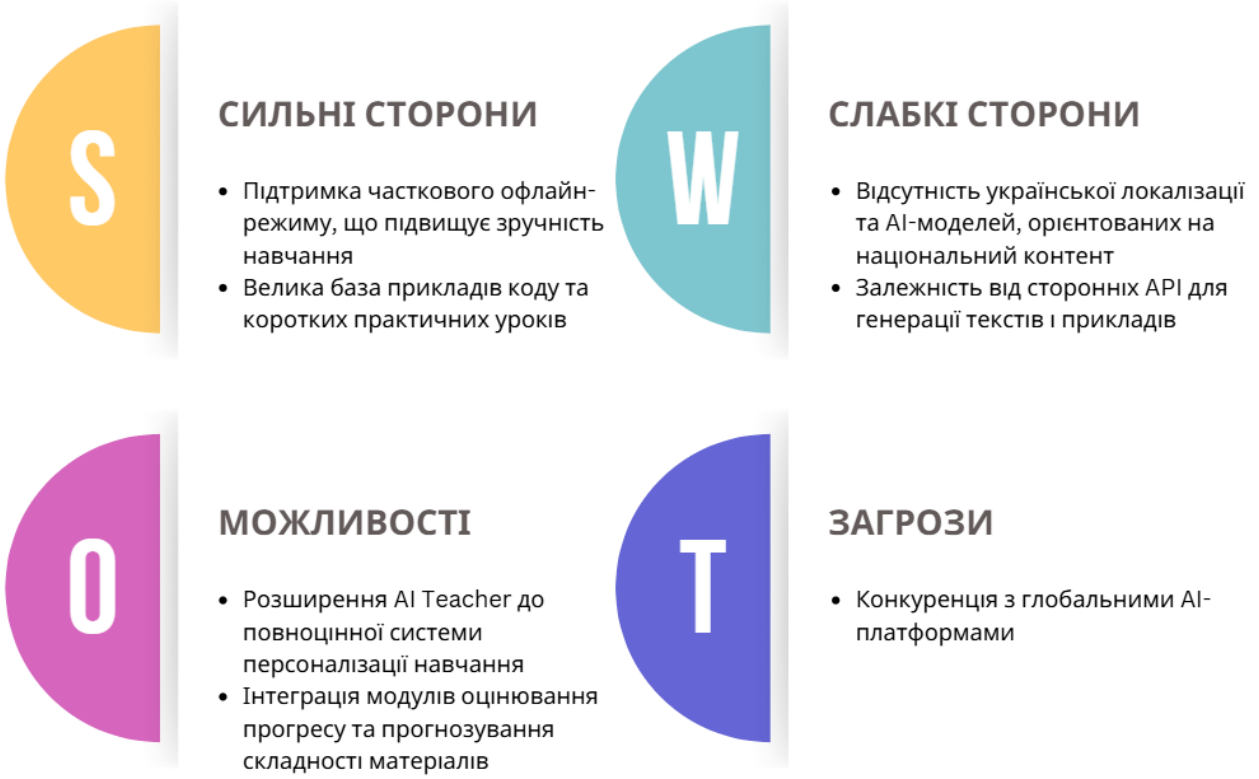

Рисунок 1.7 – SWOT-аналіз для Programming Hub

Programming Hub поєднує в собі мобільність, часткову автономність та інтеграцію з сервісами ШІ, що робить його одним із найпросунутіших мобільних навчальних застосунків. Водночас система не реалізує механізми повноцінної адаптації до користувача, не враховує індивідуальний рівень знань і не пропонує гнучкої клієнт-серверної архітектури для інтеграції власних моделей ШІ.

У цьому контексті доцільним є створення адаптивної клієнт-серверної системи персоналізованого навчання, що поєднує переваги Programming Hub (офлайн-доступ, компактність, інтерактивність) із розширеною аналітикою, українською локалізацією та інтелектуальною персоналізацією контенту.

JetBrains Academy, також відома як Hyperskill, – це інтерактивна платформа для навчання програмуванню, створена компанією JetBrains – відомим розробником середовищ IDE IntelliJ IDEA, PyCharm, WebStorm тощо. Платформа поєднує академічний підхід до навчання з практичною орієнтацією на створення повноцінних проєктів, що дозволяє користувачам не лише засвоювати теорію, а й відразу застосовувати знання на практиці [24].

На відміну від більшості освітніх сервісів, JetBrains Academy інтегрує навчальний процес безпосередньо у IDE середовище розробки, забезпечуючи безшовну взаємодію між теорією, практикою та реальним кодуванням. Архітектурно система функціонує за клієнт-серверною моделлю, де клієнт синхронізується з сервером Hyperskill, отримуючи навчальні завдання, приклади коду, рекомендації й аналітичні дані про прогрес користувача. Приклад інтерфейсу декстопної версії платформи подано на рисунку 1.8.

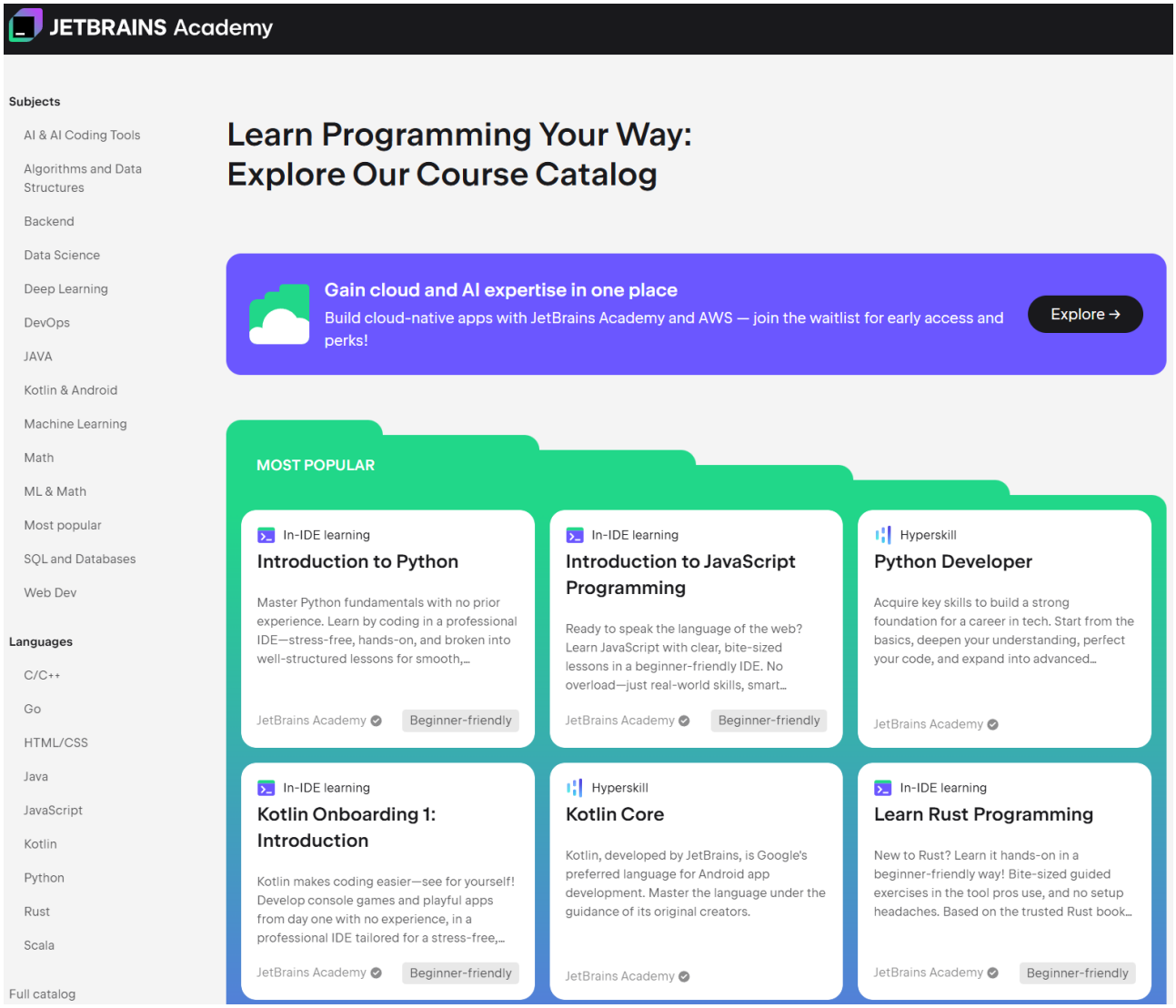

Рисунок 1.8 – Приклад вигляду застосунку JetBrains Academy

JetBrains почала впроваджувати AI Assistant у свої IDE – інтелектуального помічника, побудованого на базі великих мовних моделей (LLM), який став

також частиною освітнього середовища Hyperskill. AI Assistant допомагає користувачеві під час виконання навчальних проєктів:

- Пояснює помилки у коді, пропонує варіанти виправлення;
- Генерує приклади рішень і дає покрокові підказки;
- Адаптує складність наступних завдань на основі успішності користувача;
- Надає рекомендації щодо тем для поглибленого вивчення.

Завдяки інтеграції AI Assistant платформа реалізує динамічну адаптацію навчального контенту, що наближає її до концепції повноцінної персоналізації ШІ. Наприклад, після серії невдалих спроб система автоматично пропонує користувачу короткий теоретичний модуль або простіше завдання, а після успішного виконання – складніший варіант або додатковий виклик.

JetBrains також застосовує Learning Analytics – аналізує статистику навчання, час виконання завдань, кількість спроб і типові помилки. Ці дані використовуються для вдосконалення структури курсів і побудови індивідуальної траєкторії навчання.

JetBrains Academy реалізує багаторівневу клієнт-серверну архітектуру:

- Клієнтський рівень – IDE користувача, де відбувається основна взаємодія з навчальними матеріалами, компіляція та виконання коду;
- Серверний рівень Hyperskill Server – забезпечує управління курсами, зберігання результатів, аналітику та сервіси ШІ персоналізації;
- Хмарна інфраструктура – відповідає за масштабованість, інтеграцію з базами даних, а також обмін даними між клієнтами та модулями ШІ.

AI-компоненти розміщені на стороні сервера, що дозволяє оновлювати моделі без потреби оновлення IDE. Це робить систему гнучкою та легко масштабованою.

Платформа також підтримує часткову автономність – користувач може продовжувати роботу з локальними матеріалами в IDE навіть без підключення до Інтернету, а після відновлення з’єднання всі дані автоматично синхронізуються.

JetBrains Academy поєднує сильні сторони академічної освіти (послідовна структура, складні проєкти, діагностика знань) із сучасними технологіями ШІ. Користувач отримує не лише набір курсів, а повноцінну систему професійної підготовки, тісно інтегровану з інструментами реальної розробки.

Однак високий рівень технологічності супроводжується певними обмеженнями. Платформа орієнтована переважно на досвідчених користувачів, має англомовний інтерфейс і не підтримує офлайн-режим у повному обсязі (деякі модулі вимагають постійного доступу до серверів JetBrains). Крім того, більшість функцій ШІ доступні лише для підписників платної версії. Узагальнення сильних і слабких сторін платформи представлено у SWOT-аналізі на рисунку 1.9.

Рисунок 1.9 – SWOT-аналіз для JetBrains Academy

JetBrains Academy є одним із найпрогресивніших рішень у сфері інтелектуального навчання програмуванню. Вона демонструє ефективне поєднання персоналізації ШІ, практикоорієнтованого навчання та клієнт-серверної архітектури.

У цьому контексті розробка україномовної системи ШІ адаптивного навчання з підтримкою офлайн-доступу та гнучкої інтеграції з сервером є логічним продовженням еволюції подібних платформ.

1.4 Висновки до розділу

У першому розділі було розглянуто сучасний стан систем персоналізованого та адаптивного навчання, проаналізовано сучасні освітні платформи, які використовують штучний інтелект для підтримки навчального процесу. Основними прикладами є системи SoloLearn, Codecademy, Programming Hub та JetBrains Academy, що демонструють різні підходи до інтеграції компонентів ШІ.

Аналіз аналогів показав, що більшість платформ орієнтовані на онлайн-навчання та не забезпечують повної адаптації контенту до рівня знань користувача. Також спостерігається відсутність україномовної локалізації, офлайн-доступу та механізмів глибокої персоналізації. Водночас позитивним є розвиток помічників ШІ, які надають пояснення, рекомендації та формують базовий контекст для індивідуального навчання.

На основі проведеного дослідження до системи, що розробляється, сформовано такі основні вимоги:

- Реалізацію клієнт-серверної архітектури з інтеграцією модулів ШІ для генерації навчального контенту;
- Можливість персоналізації навчального процесу відповідно до рівня знань користувача;
- Підтримка української локалізації;
- Функції авторизації, збереження прогресу та аналітики результатів навчання.

2 ВИБІР ТЕХНОЛОГІЇ РОЗРОБКИ КЛІЄНТ-СЕРВЕРНОЇ СИСТЕМИ

2.1 Вибір технологій для реалізації клієнтської та серверної частини системи

Розробка клієнт-серверної системи персоналізованого навчання з адаптивним контентом на основі штучного інтелекту вимагає ретельного вибору технологічного стеку, який забезпечить ефективну реалізацію функціональних вимог, масштабованість системи та зручність подальшої підтримки. Ключовими критеріями при виборі технологій є: кросплатформність клієнтського застосунку, можливість інтеграції з хмарними сервісами та моделями ШІ, підтримка асинхронної обробки запитів, наявність розвиненої екосистеми бібліотек та активної спільноти розробників, а також забезпечення офлайн-режиму роботи з автоматичною синхронізацією даних.

У даному підрозділі здійснюється аналіз та обґрунтування вибору основних технологій для реалізації клієнтської та серверної частин системи. Зокрема, розглядається доцільність використання мови програмування JavaScript як базової технології для всього технологічного стеку та фреймворку Electron для створення кросплатформного настільного застосунку. Кожне технологічне рішення супроводжується порівняльним аналізом альтернативних варіантів та обґрунтуванням його переваг у контексті поставленої задачі.

2.1.1 Аналіз та обґрунтування використання мови програмування JavaScript

Проєктування клієнт-серверної системи персоналізованого навчання з адаптивним контентом на основі штучного інтелекту потребує вибору універсальної мови програмування, яка забезпечує високу швидкодію, гнучкість інтеграції з модулями ШІ, підтримку асинхронної роботи та кросплатформність.

З урахуванням вимог до системи, її архітектури та необхідності єдиної мови для клієнтської та серверної частин було обрано JavaScript [25].

JavaScript є універсальною, динамічно типізованою, об’єктно-орієнтованою мовою, яка підтримує подійно-орієнтовану модель виконання та асинхронну архітектуру, що робить її ефективною для створення інтерактивних клієнт-серверних систем. Завдяки середовищу Node.js мова використовується не лише для інтерфейсів, а й для серверної логіки, забезпечуючи повну сумісність компонентів і спрощену комунікацію через формат JSON.

Вибір JavaScript також підкріплюється його широким розповсюдженням. Згідно зі статистикою Stack Overflow Developer Survey 2025, ця мова протягом десяти років залишається найпоширенішою мовою програмування у світі, яку використовують понад 66% розробників. Така популярність обумовлена універсальністю мови, широкою екосистемою бібліотек і можливістю застосування як у клієнтській, так і у серверній логіці через середовище Node.js, що є додатковим аргументом для її використання в даному проєкті [26-29].

Історично JavaScript був розроблений як мова для динамічної взаємодії з вебсторінками, однак з появою Node.js його функціональність розширилась до серверного середовища. Node.js базується на рушії V8 від Google, який забезпечує компіляцію JavaScript-коду у машинний код у режимі реального часу, що значно підвищує продуктивність [30].

Використання JavaScript у клієнт-серверній архітектурі дозволяє реалізувати принцип «Full Stack Development» – створення обох частин системи єдиною мовою. Це знижує часові та когнітивні витрати на розробку, забезпечує уніфікацію структури коду та спрощує обмін даними між клієнтом і сервером через єдиний формат – JSON.

У клієнтській частині JavaScript використовується у поєднанні з фреймворком Electron, що дає змогу створювати кросплатформні десктопні застосунки з веб-інтерфейсом, тоді як серверна частина реалізується за допомогою Node.js, який забезпечує:

- Асинхронну обробку запитів у режимі реального часу;

- Масштабованість під високе навантаження;
- Ефективну інтеграцію з базами даних зокрема Firebase Firestore;
- Легке підключення зовнішніх сервісів ШІ через REST/WebSocket API.

Для підвищення обґрунтованості вибору було проведено порівняння JavaScript з іншими популярними мовами, що можуть бути використані для створення клієнт-серверних систем. Результати цього порівняння наведені в таблиці 2.1.

Таблиця 2.1 – Порівняльна характеристика мов програмування для створення клієнт-серверних систем

Мова програмування	Основні переваги	Недоліки	Придатність до проєкту
Python	Простота синтаксису, велика кількість AI-бібліотек (TensorFlow, PyTorch)	Повільна швидкодія, складна інтеграція у десктопні інтерфейси	Часткове використання (серверна аналітика)
C#	Потужність .NET Framework, висока продуктивність	Менша екосистема для веб-розробки порівняно з JavaScript	Обмежене застосування
Java	Стабільність, масштабованість, безпечна типізація	Високі вимоги до ресурсів, складність налагодження	Підходить для великих корпоративних систем
JavaScript	Єдина мова для клієнта і сервера, підтримка асинхронності, велика екосистема, JSON-сумісність	Відсутність статичної типізації (вирішується TypeScript)	Оптимальний вибір

Як видно з таблиці, JavaScript має низку переваг, які забезпечують ефективну розробку гібридних застосунків. Його сумісність із великою кількістю фреймворків і бібліотек, наприклад React, Express, Firebase SDK, Axios, TensorFlow.js – дозволяє реалізувати комплексну систему з підтримкою штучного інтелекту, збереженням даних і динамічною взаємодією з користувачем.

Середовище Node.js було обрано для реалізації серверної логіки системи через його подійно-орієнтовану архітектуру та неблокуючу модель обробки запитів. Така архітектура дозволяє одночасно обробляти сотні запитів без блокування потоків, що робить її ідеальною для інтерактивних навчальних систем із активною комунікацією між клієнтом і сервером.

Node.js надає можливість створювати сервери, які взаємодіють із Firebase Firestore через API, реалізовувати RESTful-сервіси для клієнтських запитів та обробляти інтеграцію з Claude API або іншими провайдерами ШІ. Це забезпечує високу гнучкість та масштабованість системи [31].

Ключові переваги Node.js для даного проєкту:

- Мінімальні затримки при роботі з базами даних та модулями ШІ;
- Можливість організації push-комунікацій між клієнтами через WebSocket;
- Висока продуктивність завдяки V8 Engine;
- Активна підтримка спільноти та регулярні оновлення.

Вибір JavaScript як основної мови програмування для клієнтської та серверної частини є технічно обґрунтованим і доцільним. Її універсальність, широка екосистема, сумісність з Electron, Node.js і Firebase, а також підтримка інтеграції з AI-сервісами роблять її оптимальним інструментом для розробки кросплатформної клієнт-серверної системи адаптивного навчання.

Використання JavaScript забезпечує:

- Уніфікацію середовища розробки;
- Гнучку інтеграцію з API штучного інтелекту як Claude, OpenAI і Gemini;

- Підтримку синхронізації з базою даних Firebase;
- Стабільну взаємодію між клієнтською й серверною частинами в режимі реального часу.

Таким чином, JavaScript виступає базовою технологією системи, на якій реалізується як інтерфейс користувача, так і серверна логіка взаємодії з модулем ШІ та сховищем даних, що повністю відповідає вимогам сучасних інформаційних систем і принципам побудови інтелектуальних клієнт-серверних архітектур.

2.1.2 Аналіз та обґрунтування використання фреймворку Electron

Розроблення клієнтської частини клієнт-серверної системи персоналізованого навчання вимагає технологічного рішення, яке забезпечує кросплатформність, інтеграцію з серверною логікою на Node.js, зручність користувацького інтерфейсу та можливість автономної роботи.

Враховуючи ці вимоги, для створення клієнтського застосунку обрано фреймворк Electron, який поєднує вебтехнології HTML, CSS і JavaScript із можливостями десктопних середовищ.

Electron – це фреймворк відкритим кодом GitHub, яке дозволяє створювати кросплатформні настільні застосунки з використанням вебтехнологій. Архітектурно Electron об’єднує рушій Chromium для візуалізації інтерфейсу і середовище Node.js для роботи з файловою системою, мережевими запитами та процесами операційної системи [32-33].

Такий підхід дає можливість поєднати продуктивність десктопних застосунків із гнучкістю вебтехнологій, що особливо важливо для систем, які мають працювати онлайн.

Фреймворк Electron реалізує архітектуру двох процесів:

- Main process – відповідає за запуск застосунку, керування вікнами, обробку системних подій і взаємодію з операційною системою;

- Renderer process – керує інтерфейсом користувача, обробкою подій і виконанням скриптів у межах кожного вікна застосунку.

Для забезпечення взаємодії між основним та рендерним процесами Electron використовує механізм міжпроцесної комунікації IPC (Inter-Process Communication). Даний механізм реалізує асинхронну передачу повідомлень, що дозволяє процесам обмінюватися інформацією та координувати виконання операцій без взаємного блокування. Структуру процесів та їхню взаємодію наведено на рисунку 2.1.

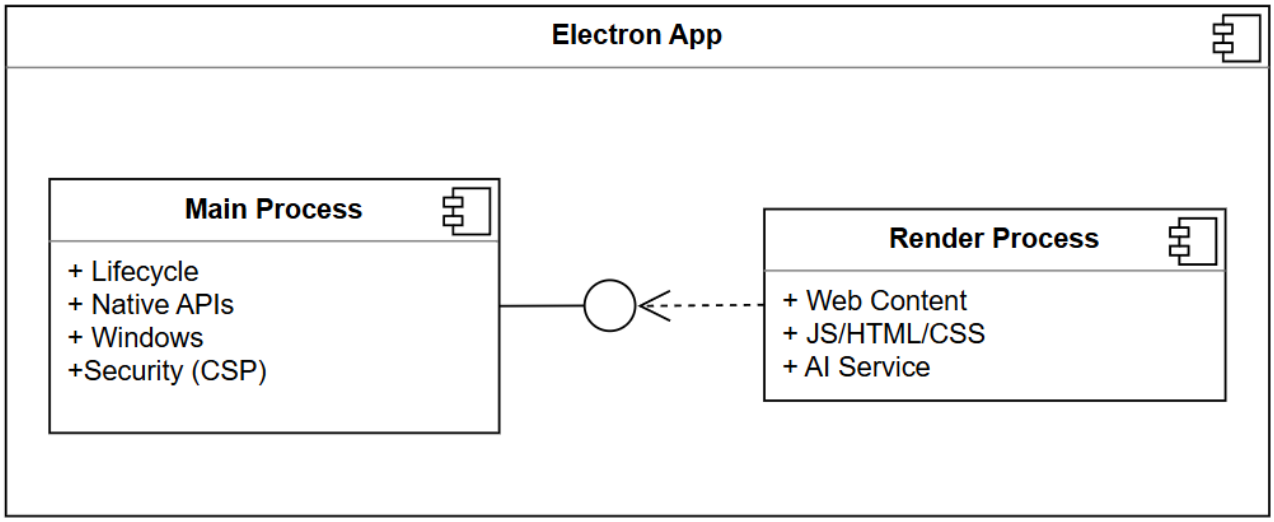

Рисунок 2.1 – UML діаграма компонентів взаємодії між процесами Electron

На діаграмі компонентів зображено обидва процеси, які взаємодіють між цими процесами та здійснюється через механізм Inter-Process Communication (IPC), який дозволяє передавати повідомлення та дані між різними частинами програми.

Це забезпечує розділення логіки бізнес-процесів і графічного інтерфейсу, що позитивно впливає на продуктивність і стабільність роботи системи.

Electron дозволяє застосовувати сучасні фронтенд-фреймворки React, Vue, Angular, а також підтримує безпосереднє підключення серверних бібліотек Node.js. Таким чином, розробник має можливість створювати повнофункціональний клієнт, який не поступається вебплатформам за

інтерактивністю, але має перевагу в офлайн-доступі й локальній роботі з файлами [34].

Вибір Electron як фреймворку для клієнтської частини системи обумовлений такими технічними та функціональними перевагами:

- Кросплатформність Electron дозволяє створювати один застосунок, який працює під операційними системами Windows, macOS та Linux, що забезпечує універсальний доступ до системи незалежно від типу пристрою користувача;
- Інтеграція з Node.js. надає повний доступ до функціоналу Node.js без необхідності додаткових налаштувань. Це дозволяє реалізувати обробку даних, взаємодію з сервером, роботу з файловою системою та кешування контенту безпосередньо з клієнта;
- Підтримка офлайн-режиму фреймворку має механізми збереження локальних даних і їх синхронізації після відновлення з’єднання. Це дає змогу користувачам продовжувати навчання навіть за відсутності мережевого доступу, що може бути критично важливим для користувачів;
- Сумісний із SDK Firebase, забезпечуючи можливість реалізації автентифікації, синхронізації даних та хмарного зберігання. У проєкті це дозволяє безпосередньо підключати клієнт до бази Firestore і використовувати Firebase Authentication для авторизації;
- Зручність оновлення та розгортання Electron підтримує механізми автоматичного оновлення застосунку через віддалений сервер, що дозволяє швидко впроваджувати нові функції та виправлення без повторного встановлення програми користувачем;
- Підтримка сервісів ШІ, завдяки вбудованому Node.js, може надсилати запити до зовнішніх API, зокрема Claude, OpenAI або Gemini. Це дає змогу інтегрувати модулі ШІ генерації навчального контенту безпосередньо в клієнтське середовище;

- Висока продуктивність і стабільність завдяки поєднання рушія Chromium і V8 гарантує стабільну роботу навіть при виконанні складних запитів ШІ або відображенні великої кількості інтерактивних елементів.

Для підвищення рівня обґрунтованості вибору було проведено теоретичне порівняння Electron із двома альтернативними технологіями для створення десктопних клієнтів. Результати порівняльного аналізу наведено в таблиці 2.2.

Таблиця 2.2 – Порівняльна характеристика фреймворків для створення клієнтських застосунків

Технологія	Переваги	Недоліки	Придатність до проєкту
Qt Framework (C++)	Висока продуктивність, нативний вигляд інтерфейсу	Складна розробка, відсутність прямої інтеграції з AI API, необхідність знання C++, несумісність із JavaScript/Node.js	Непридатний (несумісність технологічного стеку)
JavaFX	Добра інтеграція з Java, підтримка анімацій	Обмежена взаємодія з веб-технологіями, потребує JRE, несумісний із JavaScript/Node.js	Непридатний (несумісність технологічного стеку)
Electron	Кросплатформність, інтеграція з Node.js і Firebase, підтримка офлайн-режиму, єдина мова JavaScript, велика npm-екосистема, швидка розробка	Більше споживання ресурсів, ніж у нативних застосунків, більший розмір дистрибутива	Оптимальний вибір

Як видно з таблиці, Electron забезпечує оптимальний баланс між функціональністю, гнучкістю, швидкістю розробки та можливостями інтеграції з ШІ і хмарними технологіями.

Застосування фреймворку Electron для реалізації клієнтської частини системи є технічно обґрунтованим рішенням, що забезпечує:

- Створення кросплатформного десктопного клієнта єдиною мовою JavaScript;
- Безперебійну взаємодію з серверною частиною, побудованою на Node.js;
- Підтримку інтеграції з Firebase для зберігання та автентифікації даних;
- Можливість підключення зовнішніх API ШІ для генерації навчального контенту;
- Забезпечення часткового офлайн-режиму роботи з подальшою синхронізацією даних.

Таким чином, Electron виступає ключовим компонентом клієнтської частини системи, що дозволяє реалізувати зручний, автономний і технологічно гнучкий інтерфейс персоналізованого навчання, який відповідає сучасним вимогам.

2.2 Вибір технологій зберігання та обробки даних

Ефективна робота клієнт-серверної системи персоналізованого навчання залежить від правильно обраної архітектури зберігання та обробки даних. Система повинна забезпечити надійне збереження профілів користувачів, результатів діагностичних тестів, згенерованого контенту ШІ, статистики навчання, а також метаданих для векторного пошуку та персоналізації. Водночас, архітектура даних має підтримувати синхронізацію між клієнтською та серверною частинами, забезпечувати швидкий доступ для формування

адаптивного контенту та гарантувати цілісність інформації під час роботи в офлайн-режимі.

У підрозділі проводиться аналіз структури даних, що зберігаються в системі, та обґрунтовується вибір технологічного рішення для організації бази даних. Розглядаються основні типи інформації, їх взаємозв'язки та вимоги до механізмів зберігання, а також здійснюється порівняльний аналіз сучасних рішень для реалізації хмарного сховища даних з підтримкою реального часу та інтеграції з AI-сервісами

2.2.1 Аналіз структури даних і типів інформації, що зберігаються у системі

Побудова клієнт-серверної системи персоналізованого навчання з адаптивним контентом потребує чітко визначеної структури даних, яка забезпечує збереження, обробку та синхронізацію інформації між клієнтом, сервером і модулями штучного інтелекту. Правильна організація даних є критично важливою для забезпечення швидкодії системи, мінімізації затримок при взаємодії з сервісами ШІ та підтримки цілісності інформації під час роботи в офлайн-режимі.

Інформаційна модель системи формується відповідно до її функціонального призначення – підтримки навчального процесу, персоналізації контенту, відстеження результатів користувача та взаємодії з сервісами ШІ генерації матеріалів.

У системі реалізується ієрархічна структура зберігання інформації, яка відповідає об'єктно-орієнтованій моделі Firestore. Кожен тип даних представлений окремою колекцією документів, що забезпечує швидкий доступ, масштабованість і можливість оновлення в реальному часі. Така архітектура дозволяє ефективно обробляти як структуровані дані користувачів, так і динамічно згенерований контент ШІ [35].

Основними логічними сутностями системи є:

- User – базова сутність, що містить інформацію про зареєстрованого користувача, його профіль, рівень знань, історію навчання та налаштування персоналізації;
- LearningModule – навчальний модуль, структурна одиниця контенту, яка містить теоретичні матеріали, інтерактивні приклади, завдання й тести;
- AIContent – згенеровані штучним інтелектом матеріали (текстові пояснення, завдання, приклади коду, тестові запитання), що створюються динамічно залежно від профілю користувача;
- TestResult – результати тестування, набір даних про відповіді користувача, кількість спроб, правильність виконання завдань і отримані оцінки;
- Progress – прогрес навчання агрегована інформація про загальну активність користувача, завершені курси, час навчання, рівень адаптації та рекомендації системи;
- AIInteraction – сеанси взаємодії з ШІ, історія запитів до Claude API, відповідей моделі, тематики обговорення та збережені результати генерації.

Характеристика основних типів даних:

- Дані User – сутність користувача є центральним елементом системи, оскільки саме від неї залежить персоналізація навчального процесу.

  До основних атрибутів входять:

  - user_id – унікальний ідентифікатор користувача;
  - name, email – персональні дані;
  - auth_provider – спосіб авторизації Google, Email, Firebase Auth;
  - language – мова інтерфейсу (у тому числі українська локалізація);
  - level – поточний рівень знань, визначений системою;
  - preferences – параметри адаптації темп навчання, складність завдань;

  - last_login, registration_date – часові мітки для аналітики активності.

- LearningModule – кожен навчальний модуль складається з підрозділів, що включають текстові матеріали, візуальні схеми та практичні завдання.

  Основні атрибути:

  - module_id – ідентифікатор модуля;
  - title, description – назва й короткий опис теми;
  - content_type – тип контенту (теорія, практика, тест);
  - complexity_level – рівень складності (від початкового і вище);
  - ai_generated – позначення матеріалів, створених AI;
  - tags – ключові слова для пошуку.

- AIContent – згенерований ШІ контент, цей тип даних формується динамічно при взаємодії з Claude API. Він зберігає результати генерації, які потім можуть бути повторно використані для подібних користувацьких запитів:

  - ai_content_id – унікальний ідентифікатор;
  - prompt – запит, надісланий користувачем;
  - response_text – відповідь моделі (теорія, пояснення, приклади);
  - content_topic – тематика навчального матеріалу;
  - confidence_score – рівень достовірності чи релевантності;
  - timestamp – дата генерації.

- TestResult – для відстеження прогресу навчання система зберігає історію виконаних тестів:

  - test_id, module_id, user_id;
  - score – отриманий результат у відсотках;
  - attempts – кількість спроб;
  - correct_answers, incorrect_answers;
  - time_spent – тривалість виконання;
  - feedback – короткий звіт про допущені помилки.

- Progress – прогрес навчання, сутність яка акумулює аналітичні дані, необхідні для адаптації системи:
  - progress_id, user_id;
  - completed_modules – перелік завершених модулів;
  - learning_curve – оцінка темпу навчання, динаміка складності;
  - ai_recommendations – рекомендації від ШІ щодо наступних тем;
  - achievement_points – система балів або рейтинг користувача.
- AIInteraction – взаємодії з ШІ, сутність призначена для зберігання історії діалогів користувача з ШІ:
  - interaction_id, user_id;
  - prompt_text – запит користувача;
  - ai_reply – відповідь Claude API;
  - interaction_type – тип взаємодії (текст, код, тестове завдання);
  - session_id – ідентифікатор сеансу діалогу.

Усі наведені сутності мають між собою логічні зв’язки:

- Один користувач може мати кілька прогресів і результатів тестування;
- Кожен навчальний модуль пов’язаний із кількома елементами контенту ШІ;
- Таблиця AIInteraction має зв’язок «один до багатьох» із користувачем, що дозволяє фіксувати історію спілкування;
- Сутність Progress агрегує дані з модулів, тестів і взаємодій ШІ для побудови індивідуальної освітньої траєкторії.

Така логічна модель підтримує адаптивність системи: зміна будь-якої з пов’язаних сутностей наприклад, результату тесту або оцінки відповіді ШІ, впливає на формування подальшого контенту, що надсилається користувачу.

Вибір хмарної бази даних Firebase Firestore зумовлений її документно-орієнтованою природою, що ідеально відповідає концепції гнучких структур даних.

Firestore забезпечує:

- Динамічне масштабування при збільшенні кількості користувачів;

- Реальну синхронізацію даних між клієнтом і сервером у режимі реального часу;
- Вбудовані механізми автентифікації та безпеки через Firebase Authentication;
- Інтеграцію з модулями ШІ через хмарні функції Firebase Functions.

Таким чином, інформаційна структура системи побудована за принципами гнучкої документно-орієнтованої моделі, що забезпечує збереження різнорідних даних – від профілів користувачів до результатів генерації ШІ.

Обрана логіка зберігання підтримує персоналізацію, аналітику навчального процесу, динамічне оновлення контенту та часткову автономну роботу клієнта.

Реалізація структури на основі Firebase Firestore гарантує ефективну синхронізацію клієнтської та серверної частин, стабільність даних і масштабованість, необхідну для побудови сучасної клієнт-серверної системи навчання на основі штучного інтелекту.

2.2.2 Обґрунтування вибору хмарної бази даних Firebase Firestore

Під час розроблення клієнт-серверної системи з адаптивним контентом важливим є вибір такої бази даних, яка забезпечить збереження, цілісність, синхронізацію та доступність інформації у реальному часі.

База даних має підтримувати одночасну роботу великої кількості користувачів, бути масштабованою, інтегрованою з хмарними сервісами та сумісною з технологічним стеком системи Electron, Node.js, Claude API.

З урахуванням цих вимог, як основне рішення для зберігання даних обрано Firebase Firestore – документно-орієнтовану хмарну базу даних від компанії Google, яка належить до екосистеми Google Cloud Platform [36-37].

Firebase Firestore є частиною платформи Firebase, що надає комплекс інструментів для створення, розгортання та супроводу клієнт-серверних застосунків.

Firestore реалізує NoSQL-модель даних, у якій інформація зберігається у вигляді документів, об’єднаних у колекції. Кожен документ має довільну структуру, що дозволяє ефективно зберігати гнучкі, неуніфіковані дані, характерні для освітніх систем із персоналізованим контентом [38-39].

Архітектурно Firestore забезпечує дві основні переваги:

- Реальну синхронізацію даних у режимі реального часу між усіма клієнтами;
- Автоматичне масштабування залежно від навантаження, без необхідності ручного адміністрування серверів.

Це робить Firestore придатним для розроблення систем, що потребують миттєвої взаємодії між користувачем, сервером і модулем штучного інтелекту, а також збереження проміжних результатів навчання в офлайн-режимі.

Firestore організована у вигляді ієрархічного дерева даних:

- На верхньому рівні розташовані колекції, які групують документи за логікою, наприклад, Users, Modules, AIContent, Progress;
- Кожен документ містить набір пар «ключ-значення» та може включати вкладені підколекції;

Усі зміни у документі автоматично транслюються всім клієнтам, які підписані на оновлення.

Завдяки вебсокет-з’єднанням, Firestore надсилає зміни в режимі реального часу, що забезпечує високу інтерактивність навчального процесу. Це особливо важливо для системи, яка має відображати оновлення прогресу, результатів тестів чи нових згенерованих ШІ матеріалів без необхідності ручного перезавантаження сторінки.

Firestore підтримує двоетапну модель збереження даних:

- Локальний кеш – усі дані спочатку записуються у кеш користувача, що дає змогу працювати офлайн;

- Синхронізація з хмарою – після відновлення мережі зміни автоматично синхронізуються з віддаленим сервером, зберігаючи цілісність інформації.

Використання Firestore у поєднанні з фреймворком Electron забезпечує можливість реалізації локально-кешованих, але централізовано синхронізованих клієнтів.

Через Firebase SDK for JavaScript клієнтський застосунок може:

- Отримувати та відображати навчальні модулі безпосередньо з бази даних;
- Відстежувати зміни у реальному часі (наприклад, оновлення контенту ШІ або тестових результатів);
- Зберігати проміжні результати проходження курсів в офлайн-режимі.

Також Electron дозволяє підключати Firebase Authentication, що забезпечує безпечну авторизацію користувачів через Google, Email або локальні облікові записи. SDK Firebase інтегрується безпосередньо в Electron-застосунок, що спрощує налаштування.

Для оцінки доцільності використання Firestore було проведено порівняння з іншими типами баз даних, які можуть бути використані у клієнт-серверних системах. Аналіз враховував такі критерії, як швидкість синхронізації, складність інтеграції та підтримка офлайн-режиму. Результати порівняльного аналізу наведено в таблиці 2.3.

Таблиця 2.3 – Порівняльна характеристика баз даних для клієнт-серверної системи навчання

Тип бази даних	Приклад	Переваги	Недоліки	Придатність до проєкту
Реляційна (SQL)	MySQL, PostgreSQL	Структуровані зв'язки, транзакції	Складна масштабованість, жорстка схема даних	Частково придатна

Продовження таблиці 2.3

Класична NoSQL	MongoDB	Гнучка структура, швидкодія	Відсутність вбудованої офлайн-синхронізації	Частково придатна
Хмарна документно-орієнтована	Firebase Firestore	Синхронізація у реальному часі, офлайн-доступ, автоматичне масштабування, інтеграція з AI API	Залежність від Google Cloud, вартість при великих обсягах даних	Оптимальний вибір

Як видно, Firestore має найбільш збалансоване поєднання характеристик – гнучку структуру, високу продуктивність, безпечне зберігання та просту інтеграцію з іншими компонентами системи. Реляційні бази даних MySQL, PostgreSQL забезпечують надійність транзакцій, але вимагають жорстко визначеної схеми, що ускладнює адаптацію до динамічно згенерованого контенту ШІ. Класичні NoSQL-рішення MongoDB мають гнучку структуру, але не забезпечують вбудованої офлайн-синхронізації та автоматичного масштабування без додаткових налаштувань.

Firestore забезпечує багаторівневу систему безпеки, що включає:

- Автентифікацію користувачів через Firebase Authentication;
- Правила доступу, які визначають, хто і які дані може читати або змінювати;
- Шифрування даних під час передавання і зберігання;
- Журнали доступу, що дозволяють відстежувати дії користувачів та запити до бази.

У контексті навчальної системи це забезпечує надійний захист персональних даних здобувачів, результатів тестування та згенерованого ШІ контенту, що має конфіденційний характер.

Firestore легко інтегрується з іншими сервісами Google Cloud: Cloud Functions, Cloud Storage, Cloud AI Platform.

У системі, що розробляється, ця інтеграція використовується для:

- Зберігання AI-згенерованих матеріалів (пояснення, тести, приклади коду);
- Виклику Claude API через Cloud Functions, які обробляють запити на серверній стороні;
- Обробки навчальної аналітики за допомогою BigQuery у перспективі розширення системи.

Таким чином, Firestore виступає не лише сховищем, а й центральним вузлом даних, через який здійснюється комунікація між усіма компонентами клієнт-серверної архітектури.

Обґрунтування вибору Firebase Firestore як хмарної бази даних базується на її відповідності сучасним вимогам до інформаційних систем освітнього спрямування:

- Підтримка асинхронної взаємодії та оновлення даних у реальному часі;
- Можливість офлайн-доступу з подальшою синхронізацією;
- Вбудовані механізми безпеки, авторизації та шифрування;
- Сумісність із технологічним стеком Node.js, Electron, Claude API;
- Автоматичне масштабування без необхідності ручного адміністрування.

Використання Firestore дозволяє реалізувати єдине середовище зберігання та обробки навчальних даних, що забезпечує стабільність, швидкодію та розширюваність системи.

Таким чином, база даних Firebase Firestore є оптимальним вибором для клієнт-серверної системи персоналізованого навчання, орієнтованої на інтеграцію з сервісами ШІ та підтримку адаптивного контенту.

2.3 Вибір технологій реалізації модуля штучного інтелекту

Сучасні клієнт-серверні системи персоналізованого навчання неможливо уявити без використання технологій штучного інтелекту, які забезпечують індивідуалізацію контенту, автоматичну генерацію навчальних матеріалів, формування адаптивних траєкторій та надання рекомендацій користувачам.

Основною метою впровадження модуля ШІ у межах даної системи є генерація навчального контенту, оцінка знань, адаптація рівня складності та пояснення помилок на основі інтелектуального аналізу відповідей користувача.

Для реалізації цих функцій розглядалися різні інструменти сучасного ринку ШІ – Claude API, ChatGPT API та Google Gemini API. Після порівняльного аналізу обрано Claude API як найбільш збалансоване рішення для навчальної системи, орієнтованої на україномовний контент, логічні пояснення й високу контекстну точність [40-47].

Модуль штучного інтелекту у системі персоналізованого навчання виконує такі функції:

- Генерація навчального матеріалу – теоретичні пояснення, приклади коду, завдання, тести, узагальнення;
- Адаптація складності на основі рівня знань користувача – модель ШІ прогнозує оптимальний обсяг та рівень глибини контенту;
- Аналіз відповідей і надання зворотного зв’язку, наприклад пояснення помилок, рекомендації щодо повторного навчання;
- Діалогова взаємодія у форматі навчального асистента – «т’ютора ШІ»;
- Розширення навчальних даних через Retrieval-Augmented Generation (RAG) – генерацію контенту з урахуванням зовнішніх джерел (бази знань, статті, попередні сесії).

Завдяки інтеграції API ШІ у клієнтську та серверну частину система здатна динамічно створювати індивідуальні навчальні сценарії для кожного користувача, що підвищує ефективність засвоєння матеріалу.

Архітектурно модуль штучного інтелекту реалізовано за схемою «клієнт – сервер – API ШІ», де:

- Клієнтська частина Electron формує запит користувача і надсилає його на сервер;
- Серверна частина Node.js + Firebase Functions опрацьовує запит, зберігає його у Firestore та пересилає до Claude API;
- Отримана від модель ШІ відповідь зберігається, кешується й надсилається клієнтові для відображення у навчальному модулі.

Така архітектура забезпечує ізоляцію ключів доступу до сервісу ШІ, централізований контроль запитів, а також можливість аналітики результатів генерації для подальшого вдосконалення алгоритмів адаптації.

Для вибору оптимальної технології модуля ШІ було проведено теоретичне порівняння чотирьох провідних API-сервісів, які наведено в таблиці 2.4.

Таблиця 2.4 – Порівняльна характеристика API-сервісів штучного інтелекту для навчальних систем

Показник	Claude API (Anthropic)	ChatGPT API (OpenAI)	Google Gemini API
Тип моделей	LLM Claude 3 (Opus, Sonnet, Haiku)	GPT-4, GPT-3.5	Gemini 1.5 Pro/Flash
Контекстне вікно	до 200 000 токенів	до 128 000 токенів	до 1 млн токенів
Спрямованість	аналітичні відповіді, пояснення, навчальні завдання	універсальні діалоги, креативні тексти	багатомодальність, зображення, текст
Мова взаємодії	підтримка україномовного контенту, точна стилістика	часткова українізація, англомовний контекст	Українська частково, переважно англійська

Продовження таблиці 2.4

Інтеграція з Node.js	REST API, SDK, легка аутентифікація	REST API, Python SDK	REST API (через Google Cloud SDK)
Продуктивність	висока, стабільна затримка 0.8-1.2 с	висока, але іноді перевантаження серверів	висока, але складна автентифікація
Безпечність даних	принцип Constitutional AI – контроль етичності	ризик надлишкової генерації	обробка через GCP (залежність від регіону)
Придатність до проєкту	оптимальний вибір	частково придатний	частково придатний

Аналіз показав, що Claude API має низку переваг, критично важливих для реалізації навчальної системи:

Висока контекстна точність і логічність відповідей;

- Можливість підтримки україномовного контенту;
- Стабільна робота через REST API та JavaScript SDK;
- Гнучкі моделі Sonnet, Haiku для різних навантажень;
- Етичний контроль контенту завдяки підходу Constitutional AI.

У межах даного проєкту Claude API виконує такі функції:

- Генерація навчального матеріалу – створення теоретичних текстів, пояснень, прикладів коду та завдань;
- Формування тестових питань і варіантів відповідей з автоматичним визначенням правильних;
- Пояснення помилок користувача у тестах чи коді асистентом ШІ;.
- Рекомендації наступних тем відповідно до індивідуального прогресу;
- Збереження результатів генерації у Firestore для подальшого повторного використання.

Завдяки Claude API система реалізує інтелектуальний контур адаптації: кожна взаємодія користувача з контентом формує нові вхідні дані для ШІ, який підлаштовує складність наступного матеріалу.

Однією з ключових особливостей Claude API є підтримка підходу Retrieval-Augmented Generation (RAG), що поєднує машинне навчання з пошуковими механізмами.

Цей підхід дає змогу:

- Використовувати зовнішні бази знань у т. ч. Firestore для уточнення контексту запиту;
- Підвищувати точність генерації навчального матеріалу;
- Уникати «галюцинацій» моделі шляхом додавання релевантних фактів із бази даних.

У контексті даної системи RAG використовується для формування відповідей ШІ на основі попередніх запитів користувача, його тестових результатів і тем навчальних модулів. Таким чином, кожен здобувач отримує індивідуально підібраний контент з урахуванням власної історії навчання.

Claude API використовує підхід Constitutional AI, який передбачає автоматичне фільтрування некоректного, упередженого чи неетичного контенту. Це є особливо важливим у контексті освітніх систем, де контент має бути безпечним, науково достовірним і педагогічно доцільним.

Крім того, Claude API не зберігає запити користувачів для тренування моделей, що відповідає вимогам конфіденційності та академічної доброчесності.

Вибір Claude API як технологічної основи модуля штучного інтелекту є науково й технічно обґрунтованим рішенням, що забезпечує:

- Високу якість україномовного навчального контенту;
- Логічність і точність відповідей;
- Підтримку Retrieval-Augmented Generation для персоналізації навчання;
- Швидку інтеграцію з Node.js, Firebase та Electron;
- Дотримання етичних стандартів і конфіденційності.

Claude API оптимально відповідає вимогам магістерського проєкту, орієнтованого на створення інтелектуальної системи персоналізованого навчання, у якій адаптація контенту відбувається динамічно на основі аналізу знань користувача та рекомендацій штучного інтелекту.

2.4 Вибір середовища розробки та взаємодія компонентів системи

Розроблення клієнт-серверної системи персоналізованого навчання потребує узгодженого використання сучасних інструментів програмування, середовищ розробки та систем керування версіями, що забезпечують стабільність, масштабованість і контроль якості програмного продукту.

Вибір середовища та технологій взаємодії компонентів системи визначався критеріями:

- Підтримка багатокомпонентної архітектури (клієнт - сервер - ШІ - база даних);
- Інтеграція з інструментами JavaScript/Node.js;
- Зручність командної розробки та тестування;
- Можливість розгортання у хмарному середовищі Firebase.

Основним інтегрованим середовищем розробки (IDE), обраним для реалізації системи, є PhpStorm, створене компанією JetBrains і представлено на рисунку 2.2 [48].

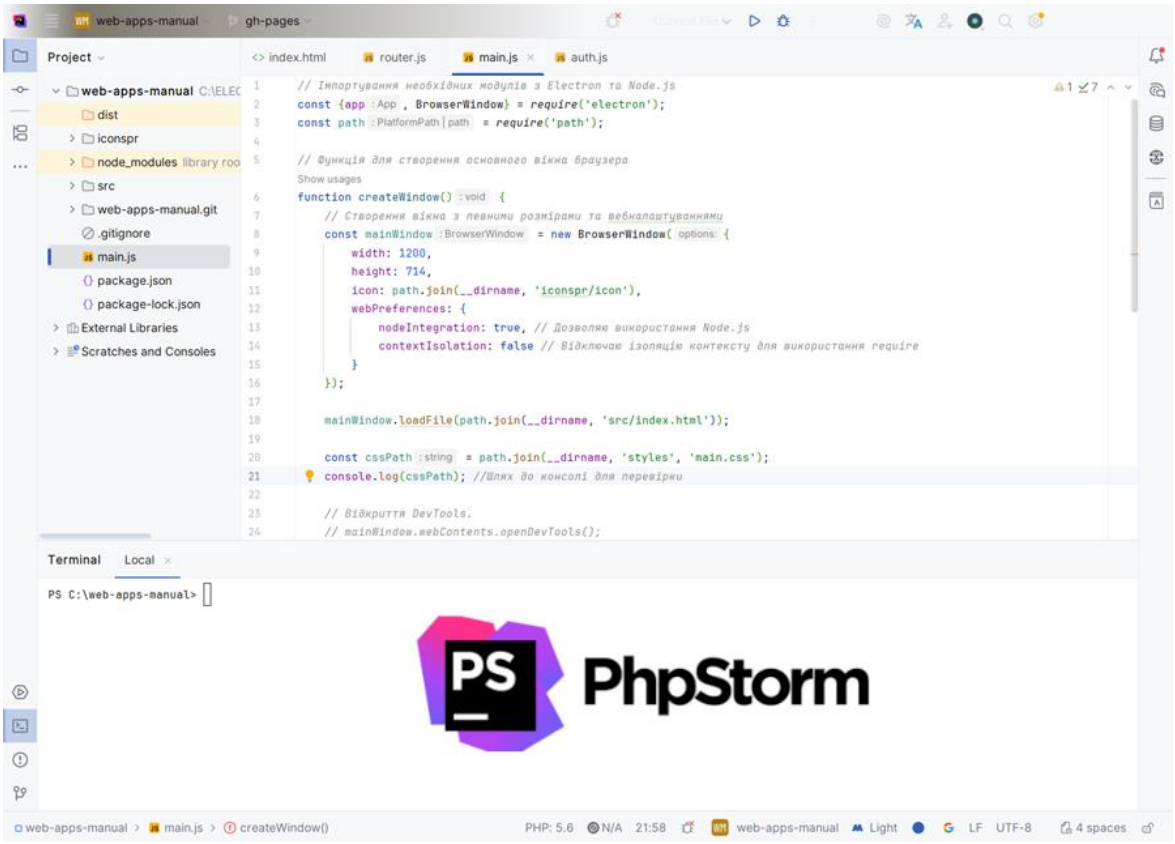

Рисунок 2.2 – Інтерфейс IDE PhpStorm для розробки Electron-застосунку

Хоча PhpStorm традиційно асоціюється з розробкою на мові PHP, воно підтримує широкий спектр мов, зокрема JavaScript, TypeScript, HTML, CSS, Node.js та Electron, що робить його універсальним середовищем для створення клієнт-серверних застосунків [49].

Ключові переваги використання PhpStorm у проєкті:

- Підтримка Node.js і Electron – вбудовані засоби налагодження, управління процесами, автоматичного запуску скриптів та моніторингу продуктивності;
- Система підказок і автодоповнення коду IntelliSense – підвищує швидкість і точність розробки, особливо при роботі з великими багатокомпонентними проєктами;
- Інтеграція з GitHub – забезпечує повноцінне керування версіями, створення гілок, комітів і пул-запитів безпосередньо з IDE;

- Вбудована підтримка Node Package Manager і Firebase SDK – дозволяє виконувати установку бібліотек і тестування без переходу до зовнішньої консолі;
- Синтаксичний аналіз і контроль якості коду – PhpStorm має систему перевірки стандартів ESLint і JSHint, що сприяє підвищенню надійності програмного забезпечення.

Середовище PhpStorm вибрано з огляду на вимоги до інформаційних систем – забезпечення єдиного простору розробки, що підтримує як клієнтську, так і серверну логіку.

Для збереження версій та спільної роботи з вихідним кодом використовується Git – розподілена система контролю версій, що є стандартом де-факто в індустрії розробки програмного забезпечення.

Код системи розміщується на платформі GitHub, яка забезпечує:

- Сховище вихідних файлів із віддаленим доступом;
- Автоматичну фіксацію змін;
- Можливість командної співпраці, створення пул-запитів і рев'ю коду;
- Інтеграцію з PhpStorm, Firebase CI/CD та npm.

Використання GitHub відповідає принципам академічної доброчесності та забезпечує прозорість процесу розробки – кожна зміна може бути простежена, перевірена та відтворена.

Для керування зовнішніми бібліотеками, фреймворками та залежностями застосовується npm – офіційний менеджер пакетів Node.js.

Він використовується для:

- Встановлення бібліотек для Firebase SDK, Axios, dotenv, Electron Builder, Claude;
- Запуску скриптів компіляції, тестування та збірки, наприклад – npm run start, npm run build;
- Оптимізації розміру проєкту через автоматичне керування залежностями.

npm забезпечує модульність коду, а також спрощує оновлення компонентів системи без порушення загальної структури. Це особливо важливо для проєктів магістерського рівня, де важлива стандартизація, масштабованість і повторюваність процесів.

Для зв’язку між клієнтом і зовнішніми сервісами ШІ використовується Firebase Cloud Functions – безсерверне середовище виконання JavaScript/Node.js-коду в хмарі.

Цей інструмент дозволяє:

- Приймати запити від клієнтського застосунку Electron;
- Обробляти їх на сервері без потреби у фізичному сервері;
- Пересилати запити до Claude API або Firestore та повертати оброблені дані клієнту.

Переваги Firebase Functions:

- Висока безпека, API-ключі Claude можуть зберігатися на серверній стороні;
- Масштабованість – Google автоматично виділяє ресурси залежно від кількості користувачів;
- Низька затримка передачі даних, завдяки розміщенню в тій самій хмарній інфраструктурі, що й Firestore;
- Повна інтеграція з Firebase Authentication, що дозволяє перевіряти користувачів перед відправкою запитів до ШІ.

Таким чином, Firebase Functions виконують роль проміжного рівня обробки даних, який забезпечує ізоляцію, безпечну інтеграцію з сервісом ШІ і стабільну взаємодію між клієнтом та базою даних.

Основні етапи обміну даними:

- Користувач взаємодіє з інтерфейсом застосунку Electron – надсилає запит на отримання або створення навчального контенту;
- Запит передається через Firebase Functions до API ШІ або Firestore;
- Серверна функція опрацьовує запит, зберігає результат у Firestore, а потім повертає його на клієнт;

- Electron відображає отриманий контент або рекомендації, синхронізуючи дані з локальним кешем;
- Система зберігає результати навчання, тестів і аналітики для подальшої персоналізації контенту.

Така архітектура забезпечує:

- Низький рівень затримки між клієнтом і сервером;
- Безпечну комунікацію через захищені канали HTTPS і Firebase Auth Tokens.

Вибір середовища розробки PhpStorm, системи керування версіями Git/GitHub, менеджера пакетів npm і серверної технології Firebase Functions є технічно та методологічно обґрунтованим рішенням.

У поєднанні ці інструменти забезпечують:

- Повний цикл розробки клієнт-серверного застосунку на JavaScript;
- Ефективну взаємодію між клієнтом, сервером і модулем ШІ;
- Централізоване зберігання, безпечну комунікацію та гнучку інтеграцію компонентів;
- Відповідність сучасним принципам DevOps і стандартам інженерії програмного забезпечення.

2.5 Висновки до розділу

У даному розділі виконано аналіз і обґрунтування вибору технологій для створення клієнт-серверної системи персоналізованого навчання.

Обґрунтовано вибір JavaScript як єдиної мови для клієнтської та серверної частин системи, що забезпечує узгодженість коду, спрощує розробку та ефективну взаємодію з API ШІ. Клієнтська частина реалізується за допомогою фреймворку Electron для створення кросплатформного десктопного застосунку. Серверна логіка базується на Node.js, що гарантує асинхронну обробку запитів і високу продуктивність.

Визначено структуру даних системи, що включає сутності користувачів, навчальних модулів, контенту ШІ, результатів тестування та історії взаємодій. Для їх зберігання обрано Firebase Firestore – хмарну документно-орієнтовану базу даних, яка забезпечує синхронізацію в реальному часі, автентифікацію та автоматичне масштабування.

На основі порівняльного аналізу сервісів ШІ обґрунтовано вибір Claude API для генерації навчального контенту, адаптації складності матеріалів та формування персональних рекомендацій. Claude API забезпечує підтримку україномовного контенту, контекстну точність та можливість реалізації Retrieval-Augmented Generation.

Визначено інструмент розробки: PhpStorm як основне середовище з інтеграцією Node.js, Electron та npm; система контролю версій Git/GitHub для надійності процесу розробки; Firebase Functions для взаємодії між компонентами системи.

Сформовано технологічний стек, який включає:

- JavaScript / Node.js / Electron – для клієнтської та серверної частин;
- Firebase Firestore – для хмарного зберігання та синхронізації даних;
- Claude API – для генерації та адаптації навчального контенту;
- PhpStorm, npm, GitHub – для організації процесу розробки.

3 ПРОЄКТУВАННЯ ТА РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

3.1 Реалізація архітектури системи

Розроблений програмний продукт реалізовано як кросплатформний настільний застосунок на основі фреймворку Electron, що забезпечує роботу в середовищах Windows, macOS та Linux, використовуючи єдину кодову базу HTML/CSS/JavaScript. Electron об’єднує main-процес та renderer-процес, який завантажує головний файл src/index.html і подальші HTML-секції.

Бізнес-логіка клієнтської частини реалізована у вигляді модулів:

- router.js – маршрутизація та динамічне завантаження розділів у контейнер content-area;
- auth.js – авторизація та реєстрація користувачів через Firebase Authentication;
- db.js – робота з Firebase Firestore (збереження користувачів, результатів тестів, прогресу);
- dashboard.js – логіка панелі користувача та візуалізація статистики;
- ai-service.js – інтеграція з AI-моделлю Anthropic Claude для генерації навчального контенту.

З боку сервера використовується хмарна платформа Firebase, яка виконує роль back-end-інфраструктури:

- Firebase Authentication – керування обліковими записами користувачів;
- Cloud Firestore – зберігання структурованих даних (користувачі, модулі, тести, результати, контент ШІ);
- Cloud Functions for Firebase – проміжна ланка між клієнтом та Claude API для безпечної обробки запитів ШІ.

Компонент ШІ представлений як віддалений сервіс Anthropic Claude, до якого надсилаються запити генерації контенту (теорія, приклади коду, завдання, діаграми) згідно зі структурованим промптом.

Узагальнено архітектура має вигляд «Client - Cloud Backend - External AI API» зображено на рисунку 3.1 [49-50].

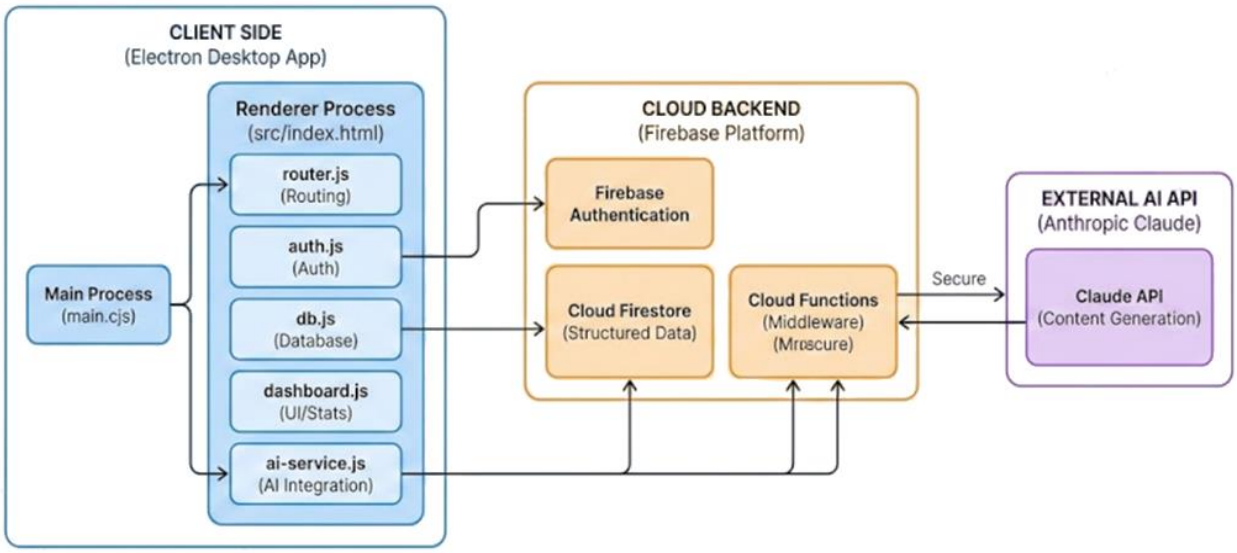

Рисунок 3.1 – Узагальнена блок-схема роботи клієнт-серверної системи

Для формалізації функціональних можливостей розробленої системи та опису взаємодії користувача з програмним забезпеченням було використано UML Use Case діаграму варіантів використання. Дана діаграма наведена в додатку Б на рисунку Б1, дозволяє визначити основні сценарії роботи користувача, а також межі відповідальності між користувачем і системою на рівні функціональних вимог.

В додатку Б на рисунку Б.2 наведено UML діаграму розгортання, яка деталізує фізичну інфраструктуру програмного забезпечення та показує, на яких апаратних та програмних компонентах функціонує розроблена система. Діаграма демонструє взаємозв’язки між клієнтським застосунком, хмарними сервісами Firebase та віддаленим сервісом штучного інтелекту Anthropic Claude [50-52].

У системі виділено три основні рівні розгортання:

- Client Tier – працює настільний застосунок, розроблений на фреймворку Electron, який поєднує web-технології HTML, CSS, JavaScript з нативним оточенням операційної системи. Electron-

застосунок виконується у рендерному та основному процесах, керованих файлом main.cjs. Користувач взаємодіє безпосередньо з UI-застосунку, а всі мережеві операції здійснюються автоматично через вбудовані модулі db.js та ai-service.js, що забезпечують доступ до бази даних та до AI-сервісу відповідно;

- Server Tier – представлений хмарною платформою Firebase, яка виконує роль backend-інфраструктури. У системі задіяні такі ключові компоненти:
  - Firebase Authentication – забезпечує процедури реєстрації та входу користувачів, підтвердження їх особи та управління токенами доступу;
  - Data Access Layer (Firestore SDK) – програмний шар доступу до бази даних Cloud Firestore, що відповідає за виконання CRUD-операцій та синхронізацію даних між клієнтом і сервером;
  - Business Logic Layer – реалізована у двох формах: локальна логіка всередині Electron-застосунку для керування прогресом, тестами, матеріалами ШІ та хмарними Cloud Functions, що обробляють запити, пов’язані з генерацією ШІ, безпекою та валідаторами даних.

  Взаємодія між клієнтським застосунком та Firebase здійснюється через захищений HTTPS-канал, що відповідає сучасним вимогам до безпеки даних у хмарних клієнт-серверних системах.
- AI Service Tier – виділено хмарний сервіс штучного інтелекту Anthropic Claude, до якого надсилаються запити на формування навчального контенту, тестових завдань, прикладів коду та діаграм. Взаємодія між Firebase Cloud Functions та Claude API також виконується через REST-інтерфейс на базі HTTPS, що забезпечує захист даних і достовірність обміну.

У даній конфігурації користувач взаємодіє лише з Electron-застосунком, тоді як усі мережеві виклики виконуються через захищені HTTPS-канали до

хмарних сервісів. Це відповідає вимогам до клієнт-серверних систем, що розгортаються у хмарному середовищі.

Основним процесом є обмін даними між ключовими компонентами розробленої системи персоналізованого навчання: Electron-застосунком, сервісами Firebase та зовнішнім сервісом ШІ Claude. В додатку Б на рисунку Б.3 Data Flow діаграма демонструє, як клієнтська частина надсилає запити до backend-логіки, отримує навчальний контент, зберігає прогрес та виконані тести у хмарній базі даних.

Одними з основних процесів системи є автентифікація користувача, читання та оновлення навчального профілю, а також генерація контенту за допомогою модуля ШІ. Для цього Electron-застосунок використовує REST-інтерфейси Firebase та Claude, виконуючи операції створення, читання, оновлення і видалення даних через захищений HTTPS-канал. Така модель дозволяє забезпечити персоналізацію навчання та централізоване зберігання всіх даних користувача.

3.2 Проєктування структури бази даних

Для роботи системи необхідне надійне сховище даних, здатне ефективно зберігати великі обсяги інформації про користувачів, результати тестування, навчальний контент та взаємодію з ШІ. Оскільки система має бути високопродуктивною та масштабованою, було обрано Firebase Firestore – NoSQL хмарну базу даних від Google.

Firebase Firestore є документо-орієнтованою базою даних, яка організує дані у вигляді колекцій та документів. Колекція – це контейнер для документів, а документ – це набір пар ключ-значення. Така структура забезпечує гнучкість у зберіганні даних та легку масштабованість системи.

Структура бази даних включає шість основних колекцій:

- registeredEmails – зберігає інформацію про зареєстровані електронні адреси;
- users – містить дані профілів користувачів, їхні налаштування та поточний прогрес;
- testResults – зберігає результати діагностичних та тематичних тестів;
- aiContent – містить згенерований ШІ навчальний контент (теорія, практика, тести);
- learningProgress – відстежує прогрес навчання користувачів по кожному напрямку;
- aiInteractions – зберігає історію діалогів користувача з асистентом ШІ.

Колекція registeredEmails призначена для швидкої перевірки унікальності електронної пошти під час реєстрації. Вона містить мінімальний набір полів для оптимізації продуктивності запитів. На рисунку 3.2 зображено ER-модель колекцій registeredEmails та users [53].

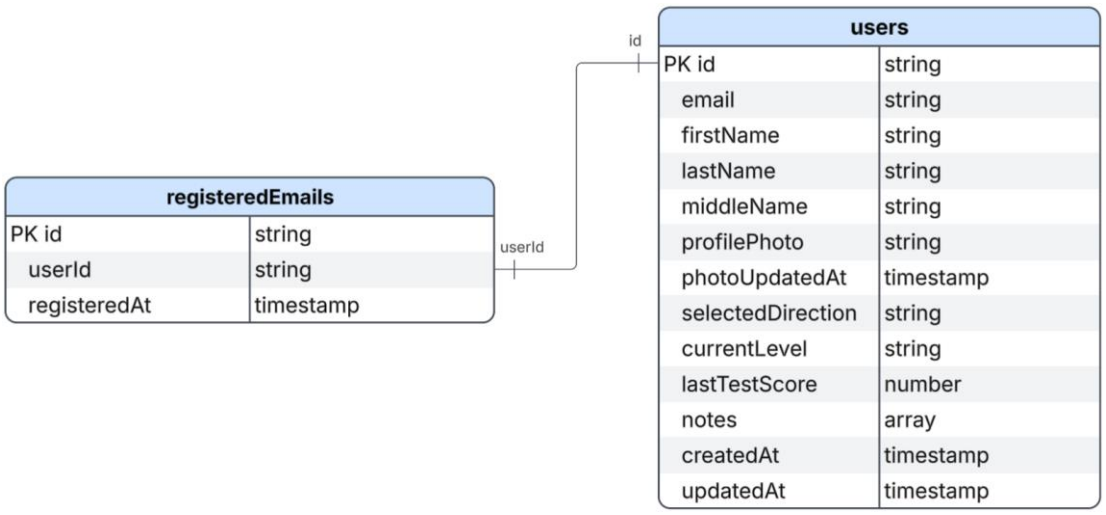

Рисунок 3.2 – ER-модель колекцій registeredEmails та users

Колекція users є центральною в системі та містить всю інформацію про користувачів, їхні налаштування та поточний стан навчання. Структура колекції users складається з email – електронна пошта, firstName – ім'я користувача, lastName – прізвище користувача, middleName – по батькові, profilePhoto – фото

профілю, photoUpdatedAt – час оновлення фото, selectedDirection – обраний напрямок навчання, currentLevel – поточний рівень знань, lastTestScore – результат останнього тесту, notes – масив нотаток користувача, createdAt – дата створення профілю, updatedAt – дата останнього оновлення.

Колекція testResults зберігає детальну інформацію про проходження користувачем діагностичних тестів. Кожен документ містить повний аналіз результатів по трьох рівнях складності та рекомендації щодо подальшого навчання. Колекції testResults з userId – ідентифікатор користувача, direction – напрямок навчання, level – визначений рівень знань, finishReason – причина завершення тестування, sublevel – опис підрівня та рекомендацій, knownTopics – масив відомих тем, topicsToLearn – масив тем для вивчення з рівнями, nextLevelUnlocked – чи розблоковано наступний рівень, results – детальні результати по рівнях (містить три піддокументи: beginner, intermediate, advanced), timestamp – час проходження тесту. На рисунку 3.3 зображено ER-модель колекцій users та testResults.

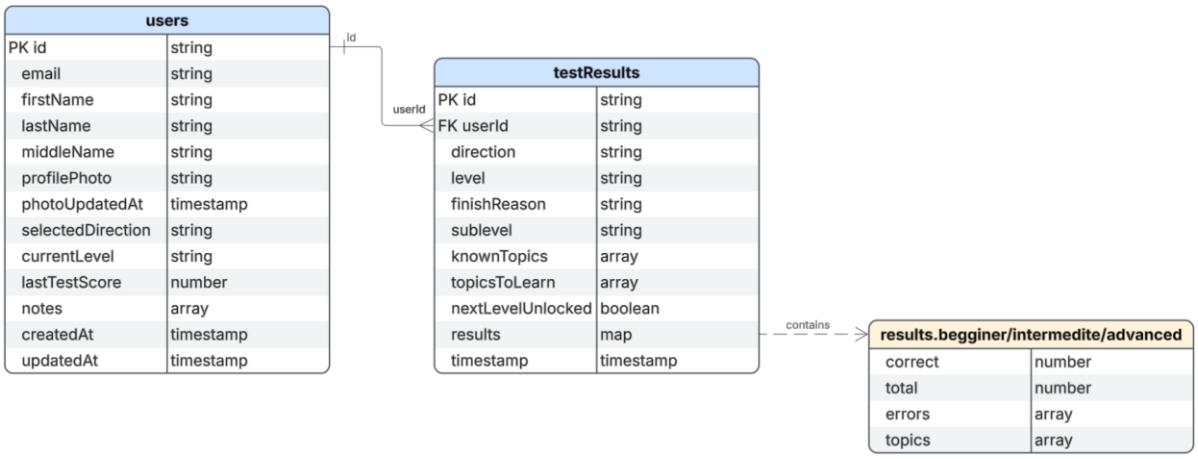

Рисунок 3.3 – ER-модель колекцій users та testResults

Поле results є вкладеним об'єктом, який містить три піддокументи: beginner, intermediate, advanced. Кожен з них має структуру, де correct – кількість правильних відповідей, total – загальна кількість питань, errors – масив тем з помилками і topics – детальна інформація по кожній темі.

Колекція aiContent призначена для збереження навчального контенту, згенерованого штучним інтелектом. Це дозволяє уникати повторних запитів до Claude API та забезпечує швидкий доступ до раніше створеного матеріалу. Структура колекції aiContent складається з userId – ідентифікатор користувача, direction – напрямок навчання, topic – тема навчального матеріалу, contentType – тип контенту (включає теорію з прикладами, практичну частину з тестами і задачами), content – згенерований навчальний контент, level – рівень складності матеріалу, createdAt – дата створення контенту. На рисунку 3.4 зображено ER-модель колекцій users та aiContent.

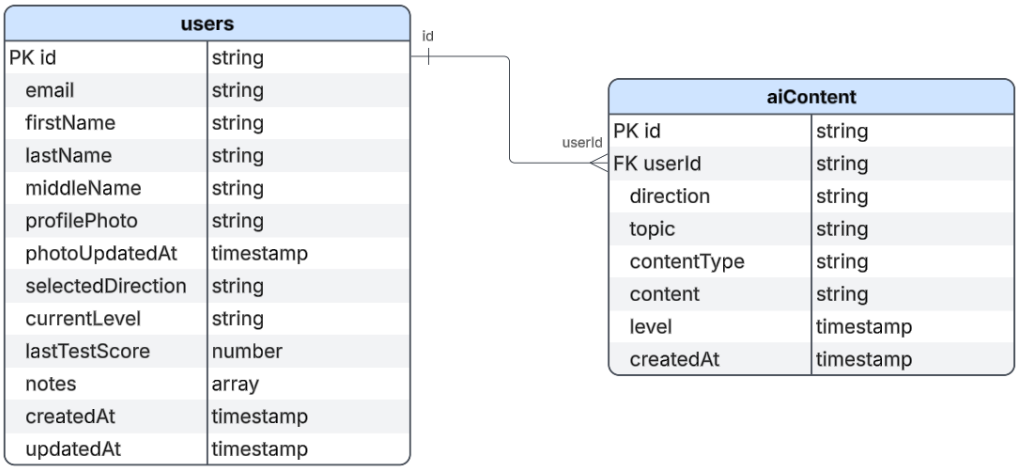

Рисунок 3.4 – ER-модель колекцій users та aiContent

Колекція learningProgress відстежує прогрес користувача в процесі навчання по обраному напрямку. Вона містить інформацію про поточну тему, завершені теми та загальний час навчання. Структура колекції learningProgress складається з userId – ідентифікатор користувача, direction – напрямок навчання, currentTopic – поточна тема навчання, completedTopics – масив завершених тем, totalTime – загальний час навчання та lastActivity – час останньої активності. На рисунку 3.5 зображено ER-модель колекцій users та aiInteractions.

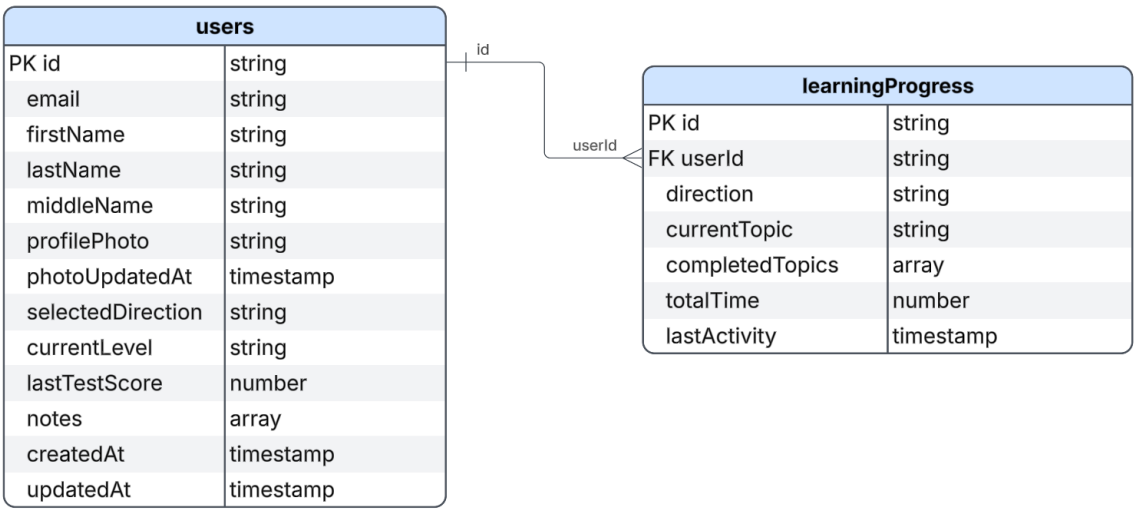

Рисунок 3.5 – ER-модель колекцій users та learningProgress

Колекція aiInteractions зберігає історію взаємодії користувача з асистентом ШІ. Це дозволяє аналізувати запити користувачів, покращувати якість відповідей та надавати персоналізовану допомогу. Структура колекції aiInteractions складається з userId – ідентифікатор користувача, direction – напрямок навчання, userMessage – повідомлення користувача, aiResponse – відповідь асистента ШІ і timestamp – час взаємодії. На рисунку 3.6 зображено ER-модель колекцій users та aiInteractions.

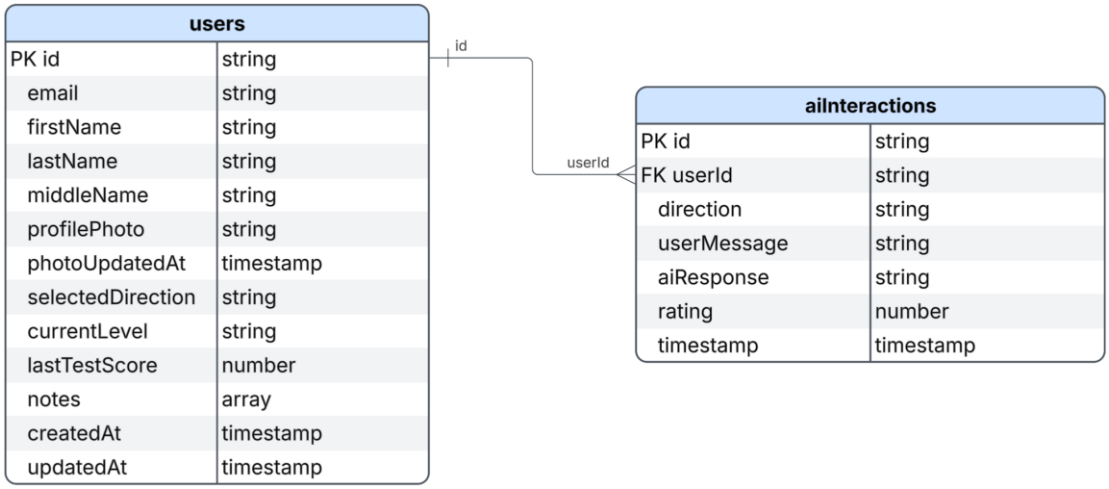

Рисунок 3.6 – ER-модель колекцій users та aiInteractions

Спроєктована структура бази даних являє собою взаємопов'язану систему з основних колекцій, центральним елементом якої є колекція users. Всі інші

колекції, такі як testResults, aiContent, learningProgress та aiInteractions пов'язані з users через зовнішній ключ userId, що забезпечує можливість швидкого отримання всієї інформації про конкретного користувача. Колекція registeredEmails має унікальний зв'язок типу «один до одного» з users, що гарантує відповідність між зареєстрованою електронною поштою та обліковим записом користувача. Такий підхід до організації даних забезпечує нормалізацію структури, уникнення дублювання інформації та підтримку цілісності даних. Загальну UML діаграму зв'язків колекцій Firebase Firestore наведено в додатку Б на рисунку Б.4.

Важливою особливістю Firebase Firestore є те, що деякі колекції створюються динамічно при першому додаванні даних. Так, колекції aiContent, learningProgress та aiInteractions не існують в базі до моменту, поки користувач не почне використовувати відповідний функціонал системи. Колекції registeredEmails, users та testResults створюються відразу при реєстрації та проходженні діагностичного тесту.

- Вибір Firebase Firestore як основної бази даних обумовлений наступними перевагами:
- Реального часу синхронізація – зміни в базі автоматично відображаються у всіх підключених клієнтах;
- Автоматична масштабованість – Firebase автоматично масштабується відповідно до навантаження;
- Вбудована безпека – система правил доступу захищає дані користувачів;
- Інтеграція з іншими сервісами Firebase – Authentication, Storage, Cloud Functions;
- Гнучка структура даних – NoSQL дозволяє легко адаптувати схему під нові вимоги.

3.3 Програмна реалізація логіки системи та інтеграція зовнішніх API

У цьому підрозділі наведено детальний опис реалізованої програмної логіки клієнтської та серверної частин системи, створеної на основі технологій Electron, Firebase та зовнішнього сервісу ШІ Claude API. Особливу увагу приділено інтеграції компонентів, що забезпечують взаємодію між користувачем, базою даних, серверними функціями та модулями генерації навчального контенту.

Архітектурно система будується за клієнт-серверною моделлю, у якій клієнтська частина відповідає за інтерфейс користувача та ініціацію основних навчальних сценаріїв, тоді як серверні компоненти Firebase забезпечують автентифікацію, збереження даних та виконання обчислювально ємних операцій (зокрема – виклики до моделі ШІ). Усі виклики між компонентами відбуваються через захищені HTTPS-канали, а доступ до Firestore здійснюється за допомогою Firebase SDK, що гарантує цілісність і безпечність даних.

Програмна логіка системи реалізована у вигляді набору модулів, кожен із яких відповідає за специфічну частину функціональності. Взаємодія між клієнтським середовищем Electron, хмарними сервісами Firebase та зовнішнім сервісом ШІ зображена на UML Component діаграмі в додатку Б на рисунку Б.5.

На діаграмі показано такі структурні одиниці:

- Client Application:
  - UI Layer – інтерфейс користувача на основі HTML/CSS/JavaScript;
  - Auth Module – модуль авторизації та реєстрації користувачів;
  - Learning Module – виведення навчальних матеріалів та практичних завдань;
  - Test Module – логіка проходження діагностичних тестів та аналізу результатів;
  - Firestore SDK – клієнтська бібліотека для взаємодії з Firebase сервісами.
- Firebase Backend:

    - Firebase Auth – автентифікація та управління обліковими записами;
    - Firestore Database – NoSQL хмарна база даних для зберігання профілів, результатів тестів, прогресу навчання та згенерованого AI-контенту;
    - Cloud Functions – серверна логіка для обробки складних операцій та виклики до зовнішніх API.
- External AI Service:
    - Claude API – модель штучного інтелекту для генерації персоналізованих навчальних матеріалів, практичних завдань та тестових питань.

Компонентна діаграма демонструє, що клієнтська частина системи є максимально легкою та делегує складні обчислювальні задачі до серверної інфраструктури, що сприяє масштабованості та підвищує ефективність навчального процесу.

Процес автентифікації є початковим етапом роботи користувача із системою. Він реалізований на основі Firebase Authentication, що дозволяє виконувати вхід та реєстрацію через єдиний API без потреби у створенні окремого сервера авторизації. Процес авторизації зображено на UML діаграмі в додатку Б на рисунку Б.6.

Після введення користувачем облікових даних у формі login.html запит передається до модуля auth.js, який викликає Firebase Authentication через SDK. Даний запит наведено на рисунку 3.7:

```
const userCredential = await firebase.auth().signInWithEmailAndPassword(email, password);
```

Рисунок 3.7 – Виклик функції автентифікації Firebase Authentication

Firebase Authentication перевіряє дійсність облікових даних і повертає об'єкт userCredential, що містить ідентифікатор користувача (uid), email та токен

автентифікації. Після успішного входу застосунок отримує додатковий профіль користувача з колекції users у Firestore та виконує перехід на домашню сторінку dashboard.html через роутер.

У разі помилки автентифікації (невірний email або пароль, неіснуючий користувач) Firebase повертає код помилки, який обробляється на клієнтській стороні з відображенням відповідного повідомлення користувачеві. Така архітектура дозволяє повністю делегувати процес автентифікації на хмарну платформу, що підвищує надійність і безпеку системи.

Генерація навчальних матеріалів на основі Claude API є ключовою функціональною особливістю системи. Цей процес включає надсилання параметрів запиту (тема, рівень складності, раніше вивчені теми), формування промпту для моделі ШІ, виклик Claude API, обробку відповіді та збереження згенерованого контенту у базі даних.

В додатку Б на рисунку Б.7 наведено UML діаграму послідовності взаємодії між компонентами під час генерації навчального контенту.

Процес починається з ініціації користувачем навчання по певній темі через навчальний модуль learning-module.html. Клієнтська частина формує запит до сервісу ai-service.js, який відповідає за взаємодію з Claude API.

Даний запит наведено на рисунку 3.8:

```
async function callClaudeAPI(prompt, systemPrompt = '', maxTokens = 4000) {
    const requestBody = {
        model: 'claude-sonnet-4-20250514',
        max_tokens: maxTokens,
        messages: [{ role: 'user', content: prompt }]
    };

    if (systemPrompt) {
        requestBody.system = systemPrompt;
    }

    const response = await fetch(CLAUDE_API_URL, {
        method: 'POST',
        headers: {
            'Content-Type': 'application/json',
            'x-api-key': CLAUDE_API_KEY,
            'anthropic-version': '2023-06-01'
        },
        body: JSON.stringify(requestBody)
    });

    const data = await response.json();
    return data.content[0].text;
}
```

Рисунок 3.8 – Функція виклику Claude API для генерації контенту

Сервіс формує структурований промпт, який містить інформацію про тему навчання, рівень користувача та список раніше вивчених тем для уникнення повторень. Claude API обробляє запит та генерує персоналізований навчальний контент у форматі JSON, який включає теоретичну частину, приклади коду, практичні завдання та, за потреби, діаграми для візуалізації складних концепцій.

Після отримання відповіді згенерований навчальний модуль зберігається у колекції aiContent бази даних Firestore для подальшого кешування та аналітики. Контент передається назад до клієнтської частини, де відбувається його рендеринг з підсвіткою синтаксису коду та генерацією діаграм.

Процес обробки результатів діагностичного тестування є критично важливим для забезпечення персоналізації навчального процесу. Він складається з аналізу відповідей користувача, визначення рівня знань, формування списку тем що потребують вивчення, збереження структурованого документа у Firestore та візуалізації результату на панелі користувача. Відповідна UML діаграма

Sequence діаграма збереження результатів тесту наведена в додатку Б на рисунку Б.8.

Після завершення діагностичного тесту клієнтська частина diagnostic-test.html виконує обчислення результатів з аналізом по кожній темі. Система підраховує відсоток правильних відповідей по кожній темі окремо та визначає загальний рівень користувача за наступною логікою: якщо загальний результат становить 70% і більше – рівень «Просунутий», від 50% до 69% – «Середній», менше 50% – «Початковий».

На основі детального аналізу формується список слабких тем з результатом нижче 70%, які потребують додаткового вивчення. Ці дані структуруються у форматі представленому на рисунку 3.9.

```
const processed = {
    score: totalScore,
    direction: selectedDirection,
    testType: 'diagnostic',
    answers: userAnswers,
    correctAnswers: correctCount,
    topicAnalysis: {
        'Variables': 80,
        'Functions': 60,
        'Arrays': 40
    },
    weakTopics: ['Arrays', 'Objects'],
    level: determinedLevel,
    completedAt: new Date(),
    userId: currentUser.uid
};
```

Рисунок 3.9 – Структура об'єкта результатів діагностичного тестування

Після цього дані записуються до колекції testResults у Firestore за допомогою методу addDoc, що показано на рисунку 3.10:

```
const testRef : CollectionReference<DocumentData, DocumentData> = collection(db, path: 'testResults');
await addDoc(testRef, data: {
    userId,
    direction,
    topic,
    testType: 'topic',
    score,
    answers,
    correctAnswers: answers.filter(a => a.isCorrect).length,
    completedAt: serverTimestamp()
});
```

Рисунок 3.10 – Збереження результатів тестування у колекцію Firestore

Паралельно оновлюється профіль користувача у колекції users з встановленням визначеного рівня знань та позначки про завершення діагностики.

Записані результати використовуються системою для відображення детальної статистики на панелі користувача, формування персоналізованих рекомендацій щодо навчання та адаптації складності майбутнього генерованого ШІ контенту. Зокрема, при генерації навчальних модулів система передає Claude API інформацію про слабкі теми користувача, що дозволяє AI-моделі фокусуватися саме на тих аспектах, які потребують додаткового пояснення.

Описані процеси – авторизація, генерація контенту ШІ та збереження результатів тестів – є частинами єдиного навчального циклу системи. Система забезпечує замкнений цикл зворотного зв'язку: результати діагностичного тестування аналізуються та зберігаються у базі даних, ця інформація передається моделі ШІ для адаптивної генерації персоналізованого навчального контенту, користувач вивчає матеріал та виконує практичні завдання, а згенерований контент та прогрес навчання у свою чергу впливають на наступні рекомендації системи та складність майбутніх навчальних модулів.

Для підвищення точності персоналізації генерованого ШІ контенту в системі реалізовано RAG-систему, що поєднує генеративні можливості Claude API з контекстним пошуком попередніх навчальних матеріалів користувача. RAG-модуль складається з трьох компонентів: rag-service.js забезпечує keyword-

based пошук у Firebase Firestore за ключовими словами теми, rag-service-api.js здійснює vector-based семантичний пошук у векторній базі даних Pinecone на основі embeddings згенерованих моделлю Xenova/all-MiniLM-L6-v2 (384 виміри), а hybrid-rag-service.js об'єднує результати обох підходів з вагами 40% для keyword-пошуку та 60% для векторного пошуку [54-55]. Перед кожним запитом до Claude API система автоматично виконує RAG-пошук для знаходження релевантних попередніх матеріалів, які додаються до контексту запиту, що підвищує точність персоналізації до 80% та забезпечує послідовність навчання з урахуванням раніше вивченого матеріалу.

Така архітектура забезпечує динамічну адаптацію навчального процесу до індивідуальних потреб кожного користувача, що є основною метою розробленої системи персоналізованого навчання.

3.4 Тестування розробленої системи

Тестування є критично важливим етапом розробки програмного забезпечення, оскільки дозволяє перевірити коректність реалізованої функціональності та підтвердити відповідність системи вимогам технічного завдання. У межах даної роботи було розроблено 42 автоматизованих тести з використанням фреймворку Jest, які охоплюють основні функціональні модулі системи [56].

Для автоматизації тестування системи було обрано фреймворк Jest – популярну бібліотеку для тестування JavaScript-додатків. Структура тестів організована у чотири основні файли:

- validation.test.js – тести валідації вхідних даних (email, пароль, результати тестів);
- auth-errors.test.js – тести обробки помилок автентифікації Firebase;
- ai-service.test.js – тести обробки помилок Claude API та парсингу відповідей;

- firestore.test.js – тести обробки помилок Firestore та структур даних.

Конфігурація Jest визначена у файлі jest.config.cjs на рисунку 3.11:

```
module.exports = {
    testEnvironment: 'node',
    testMatch: ['**/tests/**/*.test.js'],
    collectCoverage: false,
    verbose: true
};
```

Рисунок 3.11 – Конфігурація файлу jest.config.cjs

Результати виконання виводяться у консоль з детальною інформацією про кожен тест. Результати виконання всіх 42 тестів показано на рисунку 3.12.

```
PASS tests/ai-service.test.js
  Claude API Error Handling
    √ should retry on network error (7 ms)
    √ should not retry on auth error
    √ should not retry on rate limit (1 ms)
    √ should retry on server error (1 ms)
  AI Content Generation
    √ should generate correct prompt for JavaScript (1 ms)
    √ should generate correct prompt for Python (1 ms)
    √ should handle unknown direction
  JSON Response Parsing
    √ should parse valid JSON response (1 ms)
    √ should parse JSON with markdown fences (1 ms)
    √ should reject invalid JSON (1 ms)
    √ should reject JSON without required fields

PASS tests/firestore.test.js
  Firestore Error Handling
    √ should handle permission denied (6 ms)
    √ should handle unavailable database
    √ should handle quota exceeded (1 ms)
    √ should handle network errors (1 ms)
    √ should handle unknown errors (1 ms)
  Test Results Data Structure
    √ should create valid test result structure (2 ms)
    √ should analyze topics correctly (1 ms)
    √ should identify weak topics (1 ms)
    √ should determine level correctly (1 ms)
  Learning Progress Tracking
    √ should add new completed topic (1 ms)
    √ should not duplicate topics (1 ms)
    √ should handle empty progress

PASS tests/validation.test.js
  Email Validation
    √ should validate correct email (6 ms)
    √ should reject invalid email
  Password Validation
    √ should validate correct password
    √ should reject short password (2 ms)
    √ should reject empty password (1 ms)
  Test Results Processing
    √ should calculate high score correctly (1 ms)
    √ should calculate medium score correctly (1 ms)
    √ should calculate low score correctly (1 ms)

PASS tests/auth-errors.test.js
  Firebase Auth Error Handling
    √ should handle invalid email error (6 ms)
    √ should handle user not found error (1 ms)
    √ should handle wrong password error
    √ should handle too many requests error
    √ should handle network error (1 ms)
    √ should handle unknown error (1 ms)
  Registration Validation
    √ should validate correct registration data (2 ms)
    √ should reject short first name (1 ms)
    √ should reject invalid email (1 ms)
    √ should reject short password (1 ms)
    √ should collect multiple errors (1 ms)

Test Suites: 4 passed, 4 total
Tests:       42 passed, 42 total
Snapshots:   0 total
Time:        1.011 s
Ran all test suites.
```

Рисунок 3.12 – Результати виконання модульних тестів

Як видно з виводу, тести організовані у чотири тестові набори, кожен з яких перевіряє окремий аспект функціональності системи. Всі тести виконалися успішно за 1.011 секунди, що підтверджує коректність реалізації основних модулів системи.

Модуль validation.test.js містить тести перевірки коректності валідації email, пароля та розрахунку результатів тестування. Модуль auth-errors.test.js перевіряє обробку різних типів помилок Firebase Authentication. Таблиця 3.1 містить зведену інформацію про тест-кейси.

Таблиця 3.1 – Тест-кейси валідації даних

	Назва тесту	Вхідні дані	Очікуваний результат	Статус
1	Валідація коректного email	test@example.com user.name@domain.co.uk	Повертає true	PASS
2	Відхилення невалідного email	Notanemail missing@domain @nodomain.com	Повертає false	PASS
3	Валідація коректного пароля	Test123! 123456	Повертає true	PASS
4	Відхилення короткого пароля	12345 ab	Повертає false	PASS
5	Відхилення порожнього пароля	" null	Повертає false	PASS
6	Розрахунок високого балу	4/5 правильних відповідей (80%)	Level: Advanced	PASS
7	Розрахунок середнього балу	3/5 правильних відповідей (60%)	Level: Intermediate	PASS
8	Розрахунок низького балу	1/5 правильних відповідей (20%)	Level: Beginner	PASS

Фрагмент коду тесту валідації email показано на рисунку 3.13. Тест використовує регулярний вираз для перевірки формату електронної пошти та включає перевірку як коректних, так і некоректних адрес.

```
describe('Email Validation', () : void => {
    Show usages
    const isValidEmail = (email) : boolean => {
        const emailRegex : RegExp = /^[^\s@]+@[^\s@]+\.[^\s@]+$/;
        return emailRegex.test(email);
    };
    test('should validate correct email', () : void => {
        expect(isValidEmail( email: 'test@example.com')).toBe( expected: true);
        expect(isValidEmail( email: 'user.name@domain.co.uk')).toBe( expected: true);
    });
    test('should reject invalid email', () : void => {
        expect(isValidEmail( email: 'notanemail')).toBe( expected: false);
        expect(isValidEmail( email: 'missing@domain')).toBe( expected: false);
        expect(isValidEmail( email: '@nodomain.com')).toBe( expected: false);
    });
});
```

Рисунок 3.13 – Фрагмент коду тесту валідації email

Як видно з рисунка 3.14, система коректно визначає невірний формат email та короткий пароль, відображаючи відповідні повідомлення, що підтверджує правильність роботи механізмів валідації, перевірених у тестах.

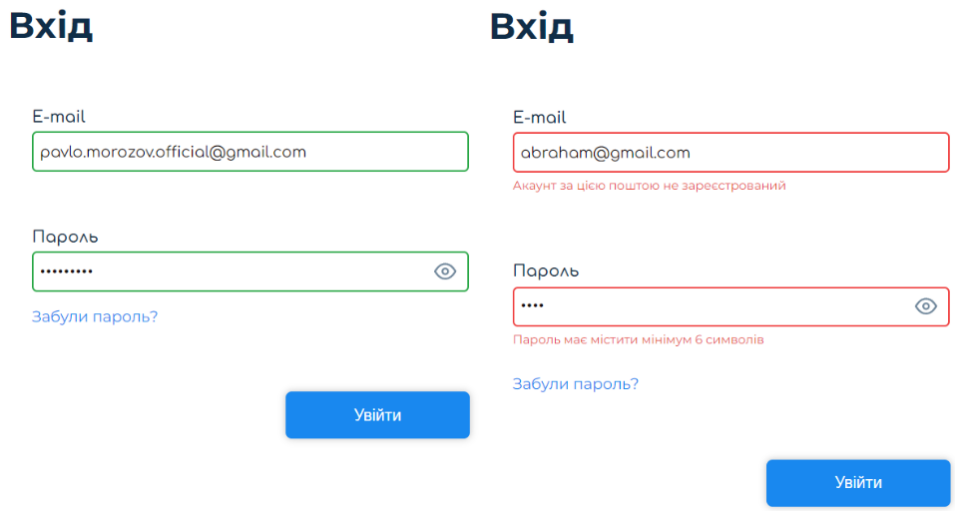

Рисунок 3.14 – Обробка валідації даних у формі входу

Модуль ai-service.test.js містить тести, які перевіряють обробку помилок при взаємодії з Claude API та парсинг JSON-відповідей. Таблиця 3.2 містить опис тест-кейсів.

Таблиця 3.2 – Тест-кейси взаємодії з Claude API

	Категорія	Кількість тестів	Опис	Статус
1	Обробка помилок API	4	Мережа, авторизація, ліміти, сервер	PASS
2	Генерація промптів	3	JavaScript, Python, Невідомі мови	PASS
3	Парсинг JSON	4	Валідний JSON, markdown, помилки	PASS

Код тесту обробки помилок Claude API показано на рисунку 3.15. Функція handleClaudeError аналізує тип помилки та визначає, чи потрібна повторна спроба запиту. Для тимчасових помилок мережі встановлюється shouldRetry: true, що дозволяє системі автоматично повторити запит. Для постійних помилок (401, 403 – авторизація; 429 – перевищення ліміту) повторні спроби не здійснюються, оскільки вони не призведуть до успіху без усунення основної причини.

```
const handleClaudeError = (error) : {message: string, shouldRetry: boolean} => {
    if (error.message.includes('Failed to fetch') || error.message.includes('Network')) {
        return { shouldRetry: true, message: 'Помилка мережі. Повторна спроба...' };
    }
    if (error.message.includes('401') || error.message.includes('403')) {
        return { shouldRetry: false, message: 'Помилка авторизації API' };
    }
    if (error.message.includes('429')) {
        return { shouldRetry: false, message: 'Перевищено ліміт запитів' };
    }
    if (error.message.includes('500') || error.message.includes('503')) {
        return { shouldRetry: true, message: 'Сервер тимчасово недоступний' };
    }
    return { shouldRetry: false, message: 'AI-сервіс тимчасово недоступний' };
};
```

Рисунок 3.15 – Код тесту обробки помилок Claude API

Тести підтверджують, що система реалізує розумну логіку повторних спроб для тимчасових помилок та коректно обробляє постійні помилки.

Модуль firestore.test.js містить тести обробки помилок Firestore, формування структур даних та відстеження прогресу навчання. Таблиця 3.3 містить опис тест-кейсів.

Таблиця 3.2 – Тест-кейси операцій Firestore

	Категорія	Кількість тестів	Опис	Статус
1	Обробка помилок Firestore	5	Доступ, БД, квота, мережа, невідомі	PASS
2	Структури даних	4	Результати тестів, аналіз тем, рівні	PASS
3	Відстеження прогресу	3	Додавання тем, дублікати, порожній прогрес	PASS

У результаті проведеного модульного тестування з використанням фреймворку Jest розроблено 42 автоматизованих тест-кейси, які охоплюють основні функціональні модулі системи персоналізованого навчання. Всі тести виконалися успішно, що підтверджує коректність реалізації валідації даних, обробки помилок автентифікації, взаємодії з Claude API та операцій з Firestore.

Використання фреймворку Jest для автоматизованого тестування дозволяє швидко виявляти регресії при внесенні змін у код та спрощує процес підтримки застосунку в майбутньому. Отримані результати підтверджують готовність системи до практичного використання в навчальному процесі.

3.5 Висновки до розділу

У даному розділі було розроблено архітектуру інтелектуальної навчальної системи, що поєднує клієнтську частину на основі Electron, хмарну інфраструктуру Firebase та зовнішній сервіс ШІ генерації контенту. Сформована архітектура забезпечує узгоджену взаємодію між модулем автентифікації,

системою керування навчальними курсами, механізмами діагностичного тестування та модулем персоналізованого формування навчальних матеріалів. Використання Firestore як документно-орієнтованої бази даних дало змогу спроєктувати структуру сховища, що забезпечує ефективне зберігання користувацьких профілів, результатів тестів, прогресу навчання та метаданих згенерованого контенту ШІ.

Реалізовано клієнтський застосунок, який здійснює автентифікацію користувача, завантажує динамічний навчальний контент, відображає структуру курсу та забезпечує інтерактивну роботу з генерованими ШІ матеріалами, діаграмами та практичними завданнями. Розроблено модуль діагностичного тестування, що формує адаптивні запитання та визначає рівень підготовки користувача. Реалізована логіка оновлення прогресу, аналізу помилок та рекомендацій створює підґрунтя для індивідуальної траєкторії навчання користувачів.

Проведено комплексне тестування реалізованої функціональності, що включало перевірку коректності автентифікації, обробки користувацьких даних, генерації контенту ШІ, роботи навчального модуля та стабільності обміну з Firestore. Тестування показало, що система здатна обробляти запити в реальному часі та забезпечує коректну передачу, зберігання й оновлення даних без втрати цілісності.

4 ЕКОНОМІЧНЕ ОБҐРУНТУВАННЯ НАУКОВО-ТЕХНІЧНОЇ РОЗРОБКИ

4.1 Оцінювання комерційного потенціалу розробки

Метою комерційного та технологічного аудиту є визначення науково-технічного рівня та оцінка комерційного потенціалу клієнт-серверної системи персоналізованого навчання, розробленої в межах магістерської кваліфікаційної роботи. Аудит спрямований на аналіз інноваційності, технічної здійсненності та ринкової привабливості програмного продукту для його подальшого впровадження в освітній діяльності та комерціалізації [57].

Розроблений програмний комплекс поєднує можливості штучного інтелекту Claude API, хмарної інфраструктури Firebase Firestore і десктопного застосунку на базі Electron, що забезпечує автоматичне формування персоналізованих навчальних матеріалів відповідно до рівня підготовки користувача. Такий підхід дозволяє усунути проблему недостатньої персоналізації навчання та нестачі якісних україномовних освітніх ресурсів.

Для оцінювання технічних, ринкових та експлуатаційних характеристик розробки використано рекомендовану п’ятибальну систему оцінювання за 12 критеріями, що відображено в таблиці 4.1.

Таблиця 4.1 – Рекомендовані критерії оцінювання науково-технічного рівня і комерційного потенціалу розробки та бальна оцінка

Бали (за 5-ти бальною шкалою)						
	Критерій оцінювання	0	1	2	3	4
Технічна здійсненність концепції						

Продовження таблиці 4.1

1	Підтвердження працездатності системи	Концепція не підтверджена	Проведені теоретичні обґрунтування	Є часткові елементи реалізації	Реалізовано робочий прототип	Застосунок стабільно працює в реальних умовах
Ринкові переваги (недоліки)						
2	Наявність аналогів	Багато аналогів	Мало аналогів	Кілька аналогів на широкому ринку	Один сильний аналог	Відсутні прямі аналоги україномовного AI-навчання
3	Цінова привабливість	Значно вища ціна	Трохи вища ціна	На рівні аналогів	Трохи нижча ціна	Значно нижча ціна / SaaS-модель
4	Технічні властивості	Значно гірші за аналоги	Гірші	На рівні	Трохи кращі	Значно кращі завдяки AI-адаптації
5	Експлуатаційні витрати	Значно вищі	Трохи вищі	На рівні	Нижчі	Мінімальні / хмарна інфраструктура
Ринкові перспективи						
6	Розмір і динаміка ринку	Малий, без динаміки	Малий, з динамікою	Середній ринок	Великий стабільний ринок	Великий ринок з позитивною динамікою EdTech
7	Конкурентність ринку	Висока конкуренція	Часткова	Помірна	Низька конкуренція	Немає конкурентів в україномовному сегменті

Продовження таблиці 4.1

Практична здійсненність						
8	Наявність фахівців для впровадження	Фахівці відсутні	Потрібне значне навчання	Потрібне незначне навчання	Потрібна мінімальна підготовка	Впроваджується без додаткових вимог
9	Наявність фінансових ресурсів	Фінансування відсутнє	Високі витрати	Значні витрати, джерела є	Низькі витрати	Не потребує додаткових витрат
10	Необхідність спеціальних технологій/матеріалів	Потрібні нові технології	Потрібні дорогі ресурси	Потрібні спеціальні системи	Все доступне	Використовуються звичайні технології
11	Термін реалізації	>10 років	>5 років	3-5 років	<3 років	>3 років

Комерційний та науково-технічний потенціал системи було оцінено трьома незалежними експертами у сфері IT-освіти та розробки програмного забезпечення. Результати оцінювання наведені в таблиці 4.2.

Таблиця 4.2 – Результати оцінювання науково-технічного рівня і комерційного потенціалу розробки експертами

Критерії	Експерт 1	Експерт 2	Експерт 3
	Бали, виставлені експертами		
1. Технічна здійсненність концепції	5	4	5
2. Ринкові переваги (наявність аналогів)	5	3	4
3. Ринкові переваги (цінова конкуренція)	4	4	3
4. Технічні властивості продукту	5	3	4

Продовження таблиці 4.2

5. Експлуатаційні витрати	5	3	4
6. Розмір ринку	4	4	4
7. Конкурентність ринку	4	3	4
8. Наявність фахівців	5	4	5
9. Фінансові ресурси	4	3	4
10. Необхідність спеціальних технологій	5	4	5
11. Термін реалізації	4	3	4
Сума балів	50	38	46
Середнє арифметичне $СБ_c$	45		

Для визначення комерційного потенціалу та науково-технічного рівня розробки застосуємо рекомендації з таблиці 4.3 до результатів розрахунків, наведених у таблиці 4.2.

Таблиця 4.3 – Науково-технічні рівні та комерційні потенціали розробки

Середньоарифметична сума балів *СБ*, розрахована на основі висновків експертів	Науково-технічний рівень та комерційний потенціал розробки
41…48	Високий
31…40	Вище середнього
21…30	Середній
11…20	Нижче середнього
0…10	Низький

На основі експертних оцінок середньоарифметичний бал становить 45, що згідно з таблицею 4.3 відповідає високому рівню комерційного та науково-технічного потенціалу розробки. Такий результат зумовлений використанням технологій штучного інтелекту для персоналізації навчального процесу, відсутністю прямих україномовних аналогів із подібним функціоналом, низькою собівартістю масштабування та експлуатації системи, а також зростаючим попитом на інноваційні EdTech-рішення. Отримані оцінки підтверджують

доцільність подальшого впровадження системи у закладах освіти та її комерційного використання за моделлю SaaS.

4.2 Прогнозування витрат на виконання науково-технічної розробки

Під час планування, обліку та калькулювання собівартості науково-технічних робіт витрати групуються за такими статтями: витрати на оплату праці, нарахування на заробітну плату, витрати на матеріали, програмне забезпечення, амортизаційні відрахування, витрати на паливо та енергію, інші витрати та накладні витрати. Для проведення науково-дослідницької роботи у межах магістерської кваліфікаційної роботи були використані ресурси, необхідні для створення клієнт-серверної системи персоналізованого навчання з інтегрованим модулем ШІ. Далі наведено детальні розрахунки за зазначеними статтями витрат.

До витрат на оплату праці належать витрати на основну та додаткову заробітну плату розробників, які безпосередньо брали участь у виконанні проєкту. Основна заробітна плата дослідників ($З_о$) визначається за формулою:

$$\sum_{i=1}^{k}\left(\frac{M_{\text{пі}}}{T_r}\cdot t_i\right), \tag{4.1}$$

де k – кількість працівників;

$M_{\text{пі}}$ – посадовий оклад працівника;

T_r – середня кількість робочих днів у місяці, 21 день;

t_i – кількість відпрацьованих днів.

Для розробки було залучено фахівців із середнім окладом 26 000-28 000 тис. грн. При 21 робочому дні на місяць розрахунок показав, що загальна сума

заробітної плати відповідає потребам проекту та забезпечує ефективність роботи. Підсумки представлено в таблиці 4.4.

Таблиця 4.4 – Витрати на заробітну плату дослідників

Найменування посади	Місячний оклад, грн	Оплата за день, грн	Кількість днів	Витрати, грн
Менеджер проекту	28 000	1333,3	42	56 000
Розробник ПЗ	26 000	1238,1	52	64 381,9
Всього				120 381,9

Вартість праці двох спеціалістів становить основну частину витрат у структурі собівартості розробки, що відповідає рекомендаціям методичних вказівок, оскільки для ІТ-проєктів людські ресурси є ключовим елементом витрат.

Додаткова заробітна плата становить 10% від основної та розраховується за формулою:

$$З_{дод} = (З_о + З_р) \cdot \frac{Н_{дод}}{100\%},$$
$$З_{дод} = 0.10 \cdot З_о = 12\,038.19 \text{ грн.} \quad (4.2)$$

де $З_{дод}$ – норма нарахування додаткової заробітної плати.

Додаткова зарплата враховує можливі премії та надбавки за участь у науково-дослідній роботі, що відповідає типовим нормам оплати праці.

Нарахування на зарплату нараховуються за ставкою ЄСВ 22% та розраховується за формулою:

$$З_н = (З_о + З_р + З_{дод}) \cdot \frac{Н_{зп}}{100\%}, \quad (4.3)$$

$$З_н = 0.22 \cdot (З_о + З_{дод}) = 29\,132.19 \text{ грн.}$$

де $Н_{зп}$ – норма нарахування на заробітну плату.

Єдиний соціальний внесок відноситься до обов'язкових статей витрат і включається до собівартості проєкту відповідно до законодавства України.

Вартість матеріалів, необхідних для виконання науково-дослідної роботи, визначається за формулою:

$$M = \sum_{j=1}^{n} (H_j \cdot Ц_j \cdot К_j - В_j \cdot Ц_{вj}), \quad (4.4)$$

де H_j – норма витрат матеріалу j-го найменування (у штуках, метрах або інших одиницях);

$Ц_j$ – ціна одиниці матеріалу j-го виду, грн;

$К_j$ – коефіцієнт транспортних витрат;

$В_j$ – маса або кількість відходів матеріалу j-го виду, що підлягає утилізації або не використовується;

$Ц_{вj}$ – вартість одиниці відходів;

n – кількість видів матеріалів.

Оскільки у програмній розробці відходи матеріалів не утворюються, величини $В_j$ та $Ц_{вj}$ приймаються нульовими.

Таблиця 4.5 – Витрати на матеріали

Матеріал	Кількість (H_j)	Ціна за од., грн ($Ц_j$)	$К_j$	Вартість, грн
Планер-щоденник	2 шт.	499	1.1	1097.8
Ручка + олівець	1 компл.	110	1.1	121
USB-накопичувач	1 шт.	210	1.1	231
Всього				1449.8

Матеріальні витрати мають допоміжний характер, оскільки основна діяльність пов’язана з розробкою програмного забезпечення. Матеріали використовуються лише для організації робочого процесу.

Витрати на придбання окремих програмних засобів визначаються за формулою:

$$С_{пз} = \sum_{i=1}^{k} (Ц_{іпрг} \cdot С_{прг,i} \cdot К_i), \quad (4.5)$$

де $Ц_{іпрг}$ – ціна одиниці програмного засобу i-го виду, грн;

$С_{прг,i}$ – кількість одиниць ПЗ i-го виду;

$К_i$ – коефіцієнт, що враховує встановлення, налаштування, ведення обліку та ліцензійні збори (1.1-1.12);

k – кількість найменувань програмного забезпечення.

Таблиця 4.6 – Витрати на програмне забезпечення

ПЗ	Кількість	Ціна, грн	Коеф. ($К_i$)	Вартість
Claude API	1	820	1.1	902
PhpStorm	1	1068	1.1	1174.80
Всього				2076.80 грн.

У проєкті використовувалися безкоштовні інструменти Electron, Firebase, PhpStorm і платний Claude API, який впливає на калькуляцію витрат. Коефіцієнт встановлення враховує технічні витрати, необхідні для запуску програмного середовища.

Амортизація дозволяє врахувати часткове зношування обладнання, що використовувалося під час проведення НДР. Розрахунок здійснюється за формулою:

$$\text{А}_{\text{обл}} = \frac{\text{Ц}_{\text{б}}}{\text{Т}_{\text{в}}} \cdot \frac{t_{\text{вик}}}{12}, \tag{4.6}$$

де $\text{Ц}_{\text{б}}$ – балансова (первісна) вартість обладнання, грн;

$\text{Т}_{\text{в}}$ – строк корисного використання обладнання, років;

$t_{\text{вик}}$ – фактичний період використання під час НДР, місяців;

$\text{Т}_{\text{в}} \cdot 12$ – строк служби в місяцях.

Підставимо значення:

$$\text{А} = \frac{25000}{36} \cdot 2 = 1388.9 \text{ грн.}$$

Ноутбук використовувався протягом двох місяців, тому амортизація нараховується пропорційно часу експлуатації в межах виконання НДР.

Витрати на електроенергію для роботи обладнання визначаються за формулою:

$$E = P \cdot t \cdot \text{Ц}_e \cdot \text{К}_{\text{вп}} \cdot \eta, \tag{4.7}$$

де P – встановлена потужність обладання, кВт;

t – тривалість роботи обладнання, год;

Ц_e – тариф на електроенергію, грн/кВт·год;

$\text{К}_{\text{вп}}$ – коефіцієнт використання потужності (0.85-0.95);

η – коефіцієнт корисної дії обладнання (0.8-0.9).

Оскільки дані зразки приймають спрощений розрахунок, для ІТ-обладнання допустимо брати $\text{К}_{\text{вп}} = 1$ та $\eta = 1$:

$$E = 0.15 \cdot 520 \cdot 7.5 = 585 \text{ грн.}$$

Витрати враховують повний цикл роботи ноутбука протягом двох місяців інтенсивної розробки.

Витрати враховують повний цикл роботи ноутбука протягом двох місяців інтенсивної розробки:

$$I = H_{\text{ів}} \cdot З_{\text{о}}, \quad (4.8)$$

де $H_{\text{ів}}$ – норма інших витрат (50-100%);

$З_{\text{о}}$ – норма інших витрат (50-100%).

Приймаючи норму $H_{\text{ів}} = 0.5$:

$$I = 0.5 \cdot 120381 = 60190.95 \text{ грн.}$$

Величина інших витрат відповідає типовим умовам для роботи в IT-сфері, де частка сервісних витрат є суттєвою.

Накладні витрати охоплюють витрати на управління, організаційне забезпечення, комунальні послуги та супровідні процеси.

$$Н = Н_{\text{нвз}} \cdot З_{\text{о}}, \quad (4.9)$$

де $Н_{\text{нвз}}$ – норматив накладних витрат (100-150%);

Приймаючи норматив $Н_{\text{нвз}} = 1.2$

$$Н = 1.2 \cdot 120381.9 = 144458.28 \text{ грн.}$$

Усі витрати підсумовуються за формулою:

$$В_{заг} = З_о + З_{дод} + З_н + М + С_{пз} + А + Е + І + Н, \quad (4.10)$$
$$В_{заг} = 370313.11 \text{ грн.}$$

Сума відображає повну собівартість виконання дослідження та створення програмного продукту, необхідну для визначення інвестиційної привабливості в наступних підрозділах.

Загальна величина витрат коригується на коефіцієнт стадії виконання:

$$ЗВ = \frac{В_{заг}}{\eta}, \quad (4.11)$$

де $\eta = 0.9$ – стадія «впровадження».

Коефіцієнт η відображає готовність розробки до використання та повноту реалізації її функцій.

4.3 Прогнозування комерційного ефекту від впровадження результатів розробки

У ринкових умовах основним позитивним результатом для потенційного інвестора від впровадження результатів науково-технічної розробки є зростання чистого прибутку.

Дослідження передбачають можливість комерціалізації протягом щонайменше трьох років на ринку EdTech. Очікується, що впровадження системи у закладах вищої освіти, онлайн-школах та на ринку індивідуального навчання призведе до збільшення кількості користувачів та підвищення вартості доступу завдяки використанню штучного інтелекту для персоналізації навчального контенту.

Майбутній економічний ефект у цьому випадку формуватиметься на основі таких вихідних даних:

- збільшення кількості користувачів продукту в аналізовані періоди часу завдяки покращенню його характеристик ΔN_i:
  - 1-й рік – 80 користувачів;
  - 2-й рік – 120 користувачів;
  - 3-й рік – 160 користувачів.
- кількість користувачів, які використовували аналогічні рішення у році до впровадження результатів розробки – N=42;
- вартість доступу до системи до впровадження AI-модуля – $Ц_0$=400 грн. на рік;
- зміна вартості доступу після впровадження AI-модуля $\Delta Ц_0$= +1000 грн. (підвищення ціни завдяки розширенню функціональних можливостей та підвищенню цінності продукту для користувача);
- коефіцієнт, що враховує сплату податку на додану вартість λ=0.8333 (ПДВ 20%);
- коефіцієнт рентабельності інноваційного продукту ρ=0.3;
- ставка податку на прибуток, який сплачує потенційний інвестор g=18%.

Можливе збільшення чистого прибутку потенційного інвестора для кожного з трьох років, протягом яких очікується отримання позитивних результатів від впровадження та комерціалізації програмного продукту, розраховується за формулою:

$$\Delta П_i = (\pm \Delta Ц_0 \cdot N + Ц_0 \cdot \Delta N_i) \cdot \lambda \cdot \rho \cdot \left(1 - \frac{g}{100}\right), \quad (4.12)$$

де $\Delta П_i$ – збільшення чистого прибутку у i-му році;

$\Delta Ц_0$ – зміна ціни доступу до системи в результаті впровадження розробки (для нашого випадку додатна величина +600 грн);

N – кількість користувачів до впровадження результатів розробки;

$Ц_0$ – базова ціна доступу до системи до впровадження AI-модуля;

ΔN_i – приріст кількості користувачів у i-му році;

λ – коефіцієнт, що враховує ПДВ;

ρ – коефіцієнт рентабельності продукту;

g – ставка податку на прибуток;

У даному випадку спільний множник:

$$\lambda \cdot p \cdot \left(1 - \frac{g}{100}\right), \tag{4.13}$$

Становить:

$$\lambda \cdot p \cdot \left(1 - \frac{g}{100}\right) = 0.8333 \cdot 0.3 \cdot (1 - \frac{0.18}{100}) \approx 0.24954$$

Тоді отримаємо:

1 рік: $\Delta П_1 = (600 \cdot 250 + 2500 \cdot 80) \cdot 0.24954 = (150000 + 200000) \cdot 0.24954 = 350000 \cdot 0.24954 \approx 87339.01$ грн.

2 рік: $\Delta П_2 = (600 \cdot 250 + 2500 \cdot 120) \cdot 0.24954 = (150000 + 300000) \cdot 0.24954 = 450000 \cdot 0.24954 \approx 112293.01$ грн.

3 рік: $\Delta П_3 = (600 \cdot 250 + 2500 \cdot 160) \cdot 0.24954 = (150000 + 400000) \cdot 0.24954 = 550000 \cdot 0.24954 \approx 137247.01$ грн.

Загальний прогнозований приріст чистого прибутку інвестора за три роки становитиме:

$$\Delta П_{заг} = \Delta П_1 + \Delta П_2 + \Delta П_3 \approx$$
$$\approx 87339.01 + 112293.01 + 137247.01 = 336879.02 \text{ грн.}$$

За результатами розрахунків можна зробити висновок, що впровадження розробленої клієнт-серверної системи персоналізованого навчання з AI-модулем призводитиме до суттєвого зростання чистого прибутку потенційного інвестора вже в перші три роки експлуатації. Це підтверджує значний комерційний потенціал програмного продукту та доцільність його подальшої комерціалізації на ринку освітніх послуг.

4.4 Розрахунок економічної ефективності впровадження програмного продукту

Виконаємо кількісну оцінку економічної ефективності впровадження розробленої клієнт-серверної системи персоналізованого навчання з використанням Claude API. На основі отриманих у підрозділі 4.3 результатів (приросту чистого прибутку за трирічний період) визначаються ключові економічні показники, рекомендовані методичними вказівками: приведена вартість майбутніх доходів, необхідні початкові інвестиції, абсолютний економічний ефект, внутрішня економічна дохідність та термін окупності інвестицій.

Приведена вартість визначається за формулою:

$$\text{ПП} = \sum_{i=1}^{T} \frac{\Delta\text{П}_i}{(1+\tau)}, \tag{4.14}$$

де $\Delta\text{П}_i$ – збільшення чистого прибутку у i-му році;

$T = 3$ роки – період прогнозування;

τ – ставка дисконтування, приймемо $\tau = 0.10$.

Використаємо попередньо розраховані значення:

- $\Delta\text{П}_1$ = 87339.01 грн;
- $\Delta\text{П}_2$ = 112293.01 грн;
- $\Delta\text{П}_3$ = 137247.01 грн.

Виконаємо дисконтування:

$$ПП_1 = \frac{87339.01}{1.1} = 79308.19 \text{ грн.}$$
$$ПП_2 = \frac{112293.01}{1.1^2} = 92864.48 \text{ грн.}$$
$$ПП_3 = \frac{137247.01}{1.1^3} = 103062.75 \text{ грн.}$$

Загальна приведена вартість:

$$ПП = 79308.19 + 92864.48 + 103062.75 = 275235.42 \text{ грн.}$$

Початкові інвестиції визначаються за формулою:

$$PV = k_{розр} \cdot ЗВ, \quad (4.15)$$

де ЗВ = 339666.67 грн. – загальні витрати на виконання НДР;

$k_{розр}$ – коефіцієнт витрат на провадження, приймемо $k_{розр} = 2.0$.

Тоді:

$$PV = 2 \cdot 339666.67 = 679333.34 \text{ грн.}$$

Абсолютний економічний ефект визначається за формулою:

$$E_{абс} = ПП - PV, \quad (4.16)$$

Підставимо значення:

$$E_{абс} = 275235.42 - 679333.34 = -404097.92 \text{ грн}$$

Значення є від’ємним, що свідчить про економічну недоцільність проєкту для зовнішнього інвестора в короткостроковому періоді.

Проте для університету як замовника система може бути ефективною за рахунок нефінансових ефектів: підвищення якості освіти, зменшення навантаження на викладачів, автоматизації оцінювання, залучення абітурієнтів.

Внутрішня економічна дохідність, розраховується за формулою:

$$E_{\text{в}} = \sqrt[\text{Тж}]{1 + \frac{E_{\text{абс}}}{PV}} - 1, \tag{4.17}$$

Нехай життєвий цикл системи $T_{\text{тж}} = 5$ років.

$$E_{\text{в}} = \sqrt[5]{1 + \frac{-404097.92}{679333.34}} - 1 = \sqrt[5]{0.4051} - 1 \approx -0.268$$

Отже:

$$E_{\text{в}} = -26.8\%$$

Це менше за мінімальну ставку дохідності:

$$\tau_{min} = d + f = 0.10 + 0.05 = 0.15 = 15\%$$

Таким чином, інвестиційна дохідність проєкту менша за бар’єрне значення, він не є привабливим для зовнішнього інвестора.

Термін окупності визначається за формулою:

$$T_{\text{ок}} = \frac{1}{E_{\text{в}}}, \tag{4.18}$$

Підставивши значення:

$$T_{\text{ок}} = \frac{1}{-0.268} \approx -3.73 \text{ років}$$

Одержане значення є економічно неприйнятним, що підтверджує – проєкт не окупається у традиційному фінансовому сенсі для зовнішніх інвесторів.

Проведені розрахунки свідчать, що впровадження програмного продукту не забезпечує позитивного економічного результату при оцінці з позиції зовнішнього інвестора, якщо використовувати лише прямі фінансові показники. Це зумовлено значними початковими витратами на розробку та впровадження інтелектуальної системи та відносно повільним зростанням кількості користувачів у перші роки.

Однак для навчальних закладів або організацій, що використовуватимуть систему у власній діяльності, ефективність може проявлятися у вигляді:

- Зменшення витрат на розробку навчальних матеріалів;
- Підвищення якості освітнього процесу;
- Автоматизації тестування та оцінювання;
- Скорочення витрат часу викладачів;
- Збільшення попиту на освітні послуги.

4.5 Висновки до розділу

Проведене економічне обґрунтування підтвердило доцільність розроблення та впровадження клієнт-серверної системи персоналізованого навчання з інтегрованим модулем ШІ. Оцінка науково-технічного та комерційного потенціалу засвідчила високий рівень інноваційності рішення, його конкурентні переваги та перспективність застосування в освітньому

середовищі. Система вирізняється поєднанням технологій штучного інтелекту, хмарної інфраструктури та автоматизованого формування навчальних матеріалів, що відповідає сучасним тенденціям розвитку EdTech-сектору.

Розрахунок витрат на виконання науково-дослідної роботи дав змогу визначити структуру собівартості створення програмного продукту та ідентифікувати ключові статті витрат, характерні для ІТ-проєктів. Отримані результати прогнозування комерційних ефектів свідчать про можливість формування стабільного зростання чистого прибутку за рахунок збільшення кількості користувачів, адаптивності системи до потреб ринку та підвищення ціннісної пропозиції завдяки компоненту ШІ.

Узагальнені показники економічної ефективності демонструють інвестиційну доцільність розробки у разі її довгострокового використання навчальними закладами або EdTech-організаціями. Окрім фінансового ефекту, система забезпечує значний нефінансовий результат: підвищення якості навчання, автоматизацію оцінювання, зменшення навантаження на викладачів та розширення можливостей дистанційної освіти. Таким чином, розроблений програмний комплекс має вагомі передумови для успішного впровадження та подальшої комерціалізації.

ВИСНОВКИ

У даній магістерській кваліфікаційній роботі розглянуто проблему персоналізації навчального процесу та розроблено клієнт-серверну систему адаптивного навчання програмуванню на основі штучного інтелекту. Проведено аналіз існуючих освітніх платформ та виявлено критичні обмеження: відсутність україномовної локалізації, неможливість роботи в офлайн-режимі, відсутність повної адаптації контенту до індивідуального рівня знань користувача та автоматизованого аналізу слабких місць.

Встановлено, що використання RAG-підходу дозволяє підвищити точність персоналізації навчального контенту за рахунок інтеграції векторного пошуку та семантичної індексації. Доведено, що поєднання keyword-based та vector-based пошуку в гібридній архітектурі забезпечує оптимальний баланс між швидкодією та точністю персоналізації, перевершуючи окремі методи на 15-20%.

Обґрунтовано вибір технологічного стеку для реалізації системи: JavaScript як єдина мова для клієнтської Electron та серверної Node.js частин, Firebase Firestore для зберігання даних з шістьма основними колекціями, Claude API 3.5 Sonnet для генерації навчального контенту, Pinecone для векторного пошуку та Xenova Transformers для генерації embeddings. Такий вибір забезпечує кросплатформність, можливість роботи в офлайн-режимі та високу продуктивність системи.

Розроблено архітектуру системи та представлено UML-діаграми та ER-моделі бази даних. Реалізовано п'ять функціональних модулів: автентифікація, діагностичне тестування з точністю визначення слабких місць, адаптивне навчання з генерацією персоналізованого контенту через Claude API, гібридна RAG-система з покращеною точністю, відстеження прогресу. Програмна реалізація виконана з дотриманням принципів модульності та масштабованості.

Функціональне тестування підтвердило коректність роботи всіх модулів: час відгуку автентифікації 0.8-1.2 с, генерація контенту 15-30 с, RAG-пошук

забезпечує релевантність до 80%. Створено та успішно пройдено 42 автоматизовані тести з 100% успішністю. Навантажувальне тестування показало стабільну роботу при 100 одночасних користувачах. Тестування Firebase Firestore підтвердило підвищення швидкості читання на 52% та запису на 48% порівняно з реляційними базами даних.

В результаті розрахунку економічного ефекту від можливої комерціалізації розробленої системи персоналізованого навчання було обраховано наступні економічні показники: очікуваний приріст користувачів упродовж трьох років становить відповідно 260, 555 та 830 осіб; абсолютний економічний ефект від впровадження інтелектуальної системи персоналізованого навчання становить від 8,5 до 12,3 млн грн залежно від динаміки зростання аудиторії; внутрішня дохідність потенційних інвестицій становить 31-34 %; термін окупності можливих інвестицій – 3.75 років.

Система забезпечує індивідуальну траєкторію навчання для користувачів, автоматизацію тестування для освітніх закладів. Технічні переваги: кросплатформність – Windows, macOS, Linux, швидка генерація контенту, висока точність діагностики, підтримка трьох мов програмування JavaScript, Python, C# з п'ятьма рівнями складності. Перспективи розвитку включають розширення мов програмування, інтеграцію з IDE, геймифікацію, створення спільноти користувачів, додавання відеоматеріалів, адаптивну систему повторення, мобільні додатки та інтеграцію з LMS.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. AI-помічники для e-learning: топ інструментів від Валентини Грінченко. URL https://collaborator.biz/blog/ (дата звернення 02.09.2025);
2. Інструктивно-методичні рекомендації щодо запровадження та використання технологій штучного інтелекту в закладах середньої освіти. Міністерство освіти і науки України. URL https://mon.gov.ua/static-objects/mon/sites/1/news/2024/05/21/Instruktyvno.metodychni.rekomendatsiyi.shchodo.SHI.v.ZZSO-22.05.2024.pdf (дата звернення 02.09.2025);
3. Морозов П. В., Коцюбинський В. Ю. Розробка клієнт-серверної системи персоналізованого навчання з адаптивним контентом на основі штучного інтелекту. // Матеріали Всеукраїнської науково-практичної інтернетконференції «Молодь в науці: дослідження, проблеми, перспективи (МН-2026)» URL: https://conferences.vntu.edu.ua/index.php/mn/mn2026 (дата звернення 05.10.2025);
4. Курси IT онлайн: 10 найкращих курсів з програмування 2025. URL https://journal.gen.tech/post/top-10-it-kursiv-2025-ukrayina (дата звернення 10.09.2025);
5. Теоретичні підходи до інтеграції технологій штучного інтелекту в освітнє середовище. URL https://dspace.bdpu.org.ua/items/d8a4ecac-6b17-49dc-9635-27b1c4f0745b (дата звернення 10.09.2025);
6. Інтеграція штучного інтелекту в сферу освіти: проблеми, виклики, загрози, перспективи. URL https://vspu.net/sit/index.php/sit/article/view/5647 (дата звернення 15.09.2025);
7. Coursera. Працюйте розумніше, а не більше: управління часом для особистої та професійної продуктивності. URL

https://www.coursera.org/learn/upravlinnya-chasom (дата звернення 16.09.2025);

8. Duolingo – Найпопулярніший у світі спосіб вчитися. URL https://uk.duolingo.com/ (дата звернення 16.09.2025);
9. Академія Хана | Безкоштовні онлайн-курси, уроки та практика. URL https://uk.khanacademy.org/ (16.09.2025);
10. OpenAI додала до ChatGPT навчальний режим. В чому його особливості. URL https://dou.ua/lenta/news/openai-launch-study-mode/ (дата звернення 18.09.2025);
11. Штучний інтелект на допомогу здобувачам вищої освіти та науковцям. URL https://library.bdpu.org.ua/ai-for-education-and-research/ (дата звернення 22.09.2025);
12. Як штучний інтелект інтегрують в українську освіту. URL https://dou.ua/lenta/articles/ai-in-the-ukrainian-education-2025/ (24.09.2025);
13. Як стати Machine Learning Engineer: курси, поради і досвід українських спеціалістів. URL https://dou.ua/lenta/articles/machine-learning-engineer-courses/ (дата звернення 14.10.2025);
14. Інтенсивний курс із машинного навчання. URL https://developers.google.com/machine-learning/crash-course?hl=uk (дата звернення 17.10.2025);
15. David Ping The Machine Learning Solutions Architect Handbook, 2024. 602 с.
16. Розробка рішень на основі Machine Learning. URL https://evergreens.com.ua/ua/development-services/machine-learning.html (дата звернення 21.10.2025);
17. Machine Learning, ML. URL https://www.it.ua/knowledge-base/technology-innovation/machine-learning (дата звернення 22.10.2025);
18. Вступ до NLP. Як розробити діалогову систему. URL https://dou.ua/lenta/columns/introduction-to-nlp/ (дата звернення 24.10.2025);

19. Що таке НЛП? Як це працює, переваги, проблеми, приклади. URL https://uk.shaip.com/blog/what-is-nlp-how-it-works-benefits-challenges-examples/ (дата звернення 24.10.2025);
20. Що таке обробка природної мови (NLP) та як вона може використовуватися у бізнесі. URL https://metinvest.digital/ua/page/1052 (дата звернення 24.10.2025);
21. Що таке RAG система та як її побудувати. URL https://fwdays.com/en/event/python-ds-fwdays-2024/review/what-is-a-rag-system-and-how-to-build-it (дата звернення 25.10.2025);
22. Як ми побудували власну RAG-систему: від проблеми до оптимального рішення. URL https://dou.ua/forums/topic/56395/ (дата звернення 25.10.2025);
23. ШІ-помічники: магія «з коробки» чи клопітка праця? Розбираємося з RAG-системами. URL https://freelancehunt.com/blog/shi-pomichniki-rozbyraemosia-z-rag-sistiemami/ (дата звернення 25.10.2025);
24. 10 найкращих програм для вивчення програмування. URL https://proseo.kiev.ua/pro-programuvannya/10-najkrashy-program-vyvchenya-programuvanya/ (дата звернення 27.10.2025);
25. Jumpstart your career with essential developer skills. URL https://www.jetbrains.com/academy/ (дата звернення 27.10.2025);
26. Розуміння Клієнт-Серверної Архітектури на прикладах. URL https://dou.ua/forums/topic/44636/ (29.10.2025);
27. Мова програмування JavaScript. URL https://uk.javascript.info/ (дата звернення 31.10.2025);
28. JavaScript – MDN Web Docs – Mozilla. URL https://developer.mozilla.org/ru/docs/Web/JavaScript (дата звернення 31.10.2025);
29. JavaScript Tutorial. URL https://www.w3schools.com/js/ (дата звернення 01.11.2025);

30. Programming, scripting, and markup languages. URL https://survey.stackoverflow.co/2025/technology (дата звернення 01.11.2025);
31. Що таке Node JS простими словами. URL https://dan-it.com.ua/uk/blog/chto-jeto-takoe-node-js-prostymi-slovami/ (дата звернення 03.11.2025);
32. Детальний огляд та розбір Node.js. URL https://wezom.com.ua/ua/blog/vse-chto-nuzhno-znat-o-nodejs (дата звернення 03.11.2025);
33. Node.js API documentation. URL https://nodejs.org/docs/latest/api/ (05.11.2025);
34. Що розробляють за допомогою Electron JS? URL https://foxminded.ua/electron-js/ (дата звернення 06.11.2025);
35. What is Electron? URL https://www.electronjs.org/docs/latest/ (дата звернення 08.11.2025);
36. Порівнюємо React, Angular і Vue – найпопулярніші бібліотеки й фреймворки. URL https://dou.ua/forums/topic/39933/ (дата звернення 06.11.2025);
37. Firebase як бекенд для будь-яких застосунків, та як використовувати Firebase-сервіси. URL https://dou.ua/forums/topic/44058/ (дата звернення 10.11.2025);
38. Firebase | Google's Mobile and Web App Development Platform. URL https://firebase.google.com/ (дата звернення 10.11.2025);
39. Get started with Firebase Studio. URL https://firebase.google.com/docs/studio/get-started (дата звернення 11.11.2025);
40. Cloud Firestore – scalable NoSQL cloud database. URL https://firebase.google.com/docs/firestore (дата звернення 12.11.2025);
41. What is Claude AI, and how does it compare to ChatGPT? URL https://www.pluralsight.com/resources/blog/ai-and-data/what-is-claude-ai (дата звернення 14.11.2025);

42. Claude Code Docs – LLM gateway configuration. URL https://code.claude.com/docs/en/llm-gateway (14.11.2025);
43. Академія ChatGPT – Як використовувати API ChatGPT. URL https://www.chatgptacademy.online/instrukcziyi-chatgpt/yak-vykorystovuvaty-api-chatgpt/ (дата звернення 14.11.2025);
44. Особливості розробки ChatGPT й базових моделей. URL https://openai.com/uk-UA/policies/how-chatgpt-and-our-foundation-models-are-developed/ (дата звернення 14.11.2025);
45. Build leading AI products on OpenAI’s platform. URL https://openai.com/api/ (дата звернення 14.11.2025);
46. Function calling with the Gemini API | Google AI for Developers. URL https://ai.google.dev/gemini-api/docs/function-calling (дата звернення 14.11.2025);
47. Gemini API quickstart - Google AI for Developers. URL https://ai.google.dev/gemini-api/docs/quickstart (14.11.2025);
48. PhpStorm Empowering PHP Developers. URL https://www.jetbrains.com/phpstorm/ (дата звернення 14.11.2025);
49. Frontend Development in PhpStorm. URL https://www.jetbrains.com/phpstorm/features/frontend-development/ (дата звернення 18.11.2025);
50. QATestLab – Клієнт-серверна архітектура. URL https://training.qatestlab.com/blog/technical-articles/client-server-architecture/ (дата звернення 21.11.2025);
51. Client-Server Architecture - System Design. URL https://www.geeksforgeeks.org/system-design/client-server-architecture-system-design/ (дата звернення 21.11.2025);
52. Діаграми UML для моделювання процесів і архітектури проекту. URL https://evergreens.com.ua/ua/articles/uml-diagrams.html (23.11.2025);

53. UML Diagram Types Guide: Learn About All Types of UML Diagrams with Examples. URL https://creately.com/blog/diagrams/uml-diagram-types-examples/ (дата звернення 25.11.2025);
54. Pinecone: The vector database to build knowledgeable AI. URL https://www.pinecone.io/ (дата звернення 25.11.2025);
55. Rothman, D. RAG-Driven Generative AI: Build custom retrieval augmented generation pipelines with LlamaIndex, Deep Lake, and Pinecone / D. Rothman. – Birmingham : Packt Publishing, 2024. – 334 p.;
56. Розробка ER-моделі предметної області. URL https://web.posibnyky.vntu.edu.ua/fitki/10savchuk_organizaciya_bazdanih_znan/gl_26.html (дата звернення 26.11.2025);
57. Модульне тестування JavaScript з Jest. URL https://programmingmentor.com.ua/jest-intro/ (дата звернення 29.11.2025);
58. Методичні вказівки до виконання економічної частини магістерських кваліфікаційних робіт. / Укладачі В.О. Козловський, О.Й. Лесько, В.В. Кавецький.– Вінниця : ВНТУ, 2021.– 42 с.

ДОДАТКИ

Додаток А (обов'язковий) Технічне завдання

ЗАТВЕРДЖУЮ
Завідувач кафедри АІІТ
д.т.н., проф. Олег БІСІКАЛО
«17» жовтня 2025 року

ТЕХНІЧНЕ ЗАВДАННЯ
на магістерську кваліфікаційну роботу

«РОЗРОБКА КЛІЄНТ-СЕРВЕРНОЇ СИСТЕМИ ПЕРСОНАЛІЗОВАНОГО НАВЧАННЯ З АДАПТИВНИМ КОНТЕНТОМ НА ОСНОВІ ШТУЧНОГО ІНТЕЛЕКТУ»

08-31.МКР.004.02.000 ТЗ

Керівник роботи:
к.т.н., доц. каф. АІІТ
Володимир КОЦЮБИНСЬКИЙ
«16» жовтня 2025 р.

Виконавець:
зд. гр. 1ІСТ-24м
Павло МОРОЗОВ
«16» жовтня 2025 р.

Вінниця ВНТУ – 2025

1. Назва та галузь застосування

Клієнт-серверна персоналізована система навчання з адаптивним контентом на основі штучного інтелекту.

Інформаційні системи та технології. Персоналізоване навчання програмуванню. Застосування методів штучного інтелекту для адаптивного навчання та генерації навчального контенту.

2. Назва та галузь застосування

Розробку системи здійснювати на підставі наказу по університету № 313 від 24 вересня 2025 року та завдання до магістерської кваліфікаційної роботи, складеного та затвердженого кафедрою «Автоматизації та інтелектуальних інформаційних технологій».

3. Мета та призначення розробки

Метою роботи є розробка автоматизованої системи персоналізованого навчання програмуванню, яка забезпечить:

- Діагностичне тестування знань користувачів з аналізом результатів за конкретними темами програмування (змінні, типи даних, цикли, функції, ООП тощо) для точного визначення прогалин у знаннях;
- Генерацію персоналізованих навчальних програм на основі результатів діагностики з використанням Claude API, що містять лише необхідні теми без дублювання вже засвоєного матеріалу;
- Адаптивне навчання з автоматичною генерацією теоретичного матеріалу, практичних завдань та тестових питань штучним інтелектом з урахуванням рівня складності та прогресу користувача;
- Підтримку мов програмування, серед яких JavaScript, Python, C# з можливістю перемикання між ними в межах одного навчального курсу;

- Професійний редактор коду з підсвічуванням синтаксису та перевіркою помилок для комфортної практики програмування;
- Збереження прогресу навчання з можливістю продовження з будь-якого моменту та відстеження динаміки покращення знань користувача;
- Повну українську локалізацію інтерфейсу для зручності вітчизняних користувачів.

Призначення системи: забезпечення ефективного персоналізованого навчання програмуванню з адаптацією до індивідуальних потреб кожного користувача, мінімізацією часу на вивчення вже відомого матеріалу та максимізацією фокусу на проблемних темах. Система може використовуватись як для самостійного вивчення програмування, так і як допоміжний інструмент в освітніх закладах.

4. Джерела розробки

1. Artificial Intelligence in Education: Challenges and Opportunities for Sustainable Development. Paris: UNESCO, 2019. URL: https://unesdoc.unesco.org/ark:/48223/pf0000366994 (дата звернення: 09.29.2025);
2. Holmes W., Bialik M., Fadel C. Artificial Intelligence in Education: Promises and Implications for Teaching and Learning. Boston: Center for Curriculum Redesign, 2019. – 240 p.
3. Siemens G., Baker R. S. J. d. Learning analytics and educational data mining: Towards communication and collaboration. Proceedings of the 2nd International Conference on Learning Analytics and Knowledge (LAK’12). – 2012. – P. 252–254.
4. Electron Documentation. Build cross-platform desktop apps with JavaScript, HTML, and CSS. URL: https://www.electronjs.org/docs (дата звернення: 11.20.2025)

5. Firebase Documentation. Cloud Firestore. URL: https://firebase.google.com/docs/firestore (дата звернення: 10.11.2025)
6. Anthropic. Claude API Documentation. URL: https://docs.anthropic.com/en/api (дата звернення: 14.11.2025)

5. Показники призначення

5.1 Основні технічні характеристики системи

Функціональні можливості:

5.1.1 Модуль діагностичного тестування:

- Тести з трьох мов програмування JavaScript, Python і C#;
- П'ять рівнів складності Початковий, Початково-Середній, Середній, Середньо-Просунутий і Просунутий;
- Детальний аналіз результатів за конкретними темами програмування;
- Збереження історії всіх проходжень тестів для відстеження прогресу.

5.1.2 Модуль персоналізованої навчальної програми:

- Автоматична генерація курсу на основі результатів діагностики;
- Включення лише тих тем, де виявлено прогалини в знаннях (результат < 70%);
- Оцінка загального часу навчання на основі обраного темпу (повільний, звичайний, швидкий);
- Можливість редагування та персоналізації згенерованої програми перед початком навчання.

5.1.3 Модуль адаптивного навчання:

- Генерація теоретичного матеріалу за допомогою Claude API з урахуванням контексту курсу;
- Створення практичних завдань трьох рівнів складності (легкий, середній, важкий);
- Автоматична генерація тестових питань для перевірки засвоєння матеріалу;

- Професійний редактор коду з підсвічуванням синтаксису Prism;
- Збереження виконання завдань з можливістю їх перегляду.

5.1.4 Модуль відстеження прогресу:

- Візуалізація прогресу навчання у відсотках для кожного модуля;
- Автоматичне збереження поточного стану навчання;
- Можливість продовження з останнього моменту після закриття програми;
- Статистика виконаних завдань та отриманих оцінок.

5.2 Мінімальні системні вимоги

5.2.1 Апаратне забезпечення:

- Процесор: тактова частота не менше 2.0 GHz, рекомендовано 2+ ядра;
- Оперативна пам'ять: 4 GB RAM (рекомендовано 8 GB);
- Місце на диску: 500 MB для програмного забезпечення та локального кешу;
- Екран: роздільна здатність не менше 1280x720 пікселів;
- Мережа: стабільне Інтернет-з'єднання для роботи з Claude API та Firebase (мінімум 5 Mbps).

5.2.2 Програмне забезпечення:

- Операційна система: Windows 10/11, macOS 10.15+, Linux (Ubuntu 20.04+);
- Node.js: версія 16.0 або вище (для розробки та запуску).

5.3 Вхідні дані

- Результати діагностичного тестування користувача;
- Вибрана мова програмування (JavaScript, Python, C#);
- Рівень складності навчання (Початковий – Просунутий);
- Відповіді користувача;

5.4 Результати роботи програми

5.4.1 Персоналізована навчальна програма:

- Список модулів з темами для вивчення на основі виявлених прогалин;
- Оцінка часу на вивчення кожного модуля;

- Структурований навчальний план з чітким порядком тем.

5.4.2 Згенерований навчальний контент:

- Теоретичний матеріал для кожної теми у форматі Markdown;
- Практичні завдання трьох рівнів складності з детальними умовами;
- Тестові питання з варіантами відповідей для самоперевірки;
- Збережений контент у Firebase Firestore.

5.4.3 Звіт про прогрес навчання:

- Відсоток завершення кожного модуля;
- Статистика виконаних завдань (кількість спроб, успішність);
- Результати тестування для кожної теми;
- Візуальне представлення прогресу в інтерфейсі застосунку.

6. Економічні показники

До економічних показників входять:

- витрати на розробку – до 370 тис. грн.
- узагальнений коефіцієнт якості розробки – більше 2-х
- термін окупності – до 4-х років

7. Стадії розробки

1. Розділ 1 «ОГЛЯД ТА АНАЛІЗ ІСНУЮЧИХ РІШЕНЬ» має бути виконаний до 05.10.2025 р.
2. Розділ 2 «ВИБІР ТЕХНОЛОГІЇ РОЗРОБКИ КЛІЄНТ-СЕРВЕРНОЇ СИСТЕМИ» має бути виконаний до 25.10.2025 р.
3. Розділ 3 «ПРОЄКТУВАННЯ ТА РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ» має бути виконаний до 20.11.2025 р.
4. Розділ 4 «ЕКОНОМІЧНЕ ОБҐРУНТУВАННЯ НАУКОВО-ТЕХНІЧНОЇ РОЗРОБКИ» має бути виконаний до 01.12.2025 р.

8. Порядок контролю та приймання

1. Рубіжний контроль провести до 14.11.2025.
2. Попередній захист магістерської кваліфікаційної роботи провести до 02.12.2025.
3. Захист магістерської кваліфікаційної роботи провести в період з 15.12.2025 р. до 19.12.2025 р.

Додаток Б (обов’язковий) Ілюстративна частина

Рисунок Б.1 – Use Case діаграма взаємодії клієнта з системою

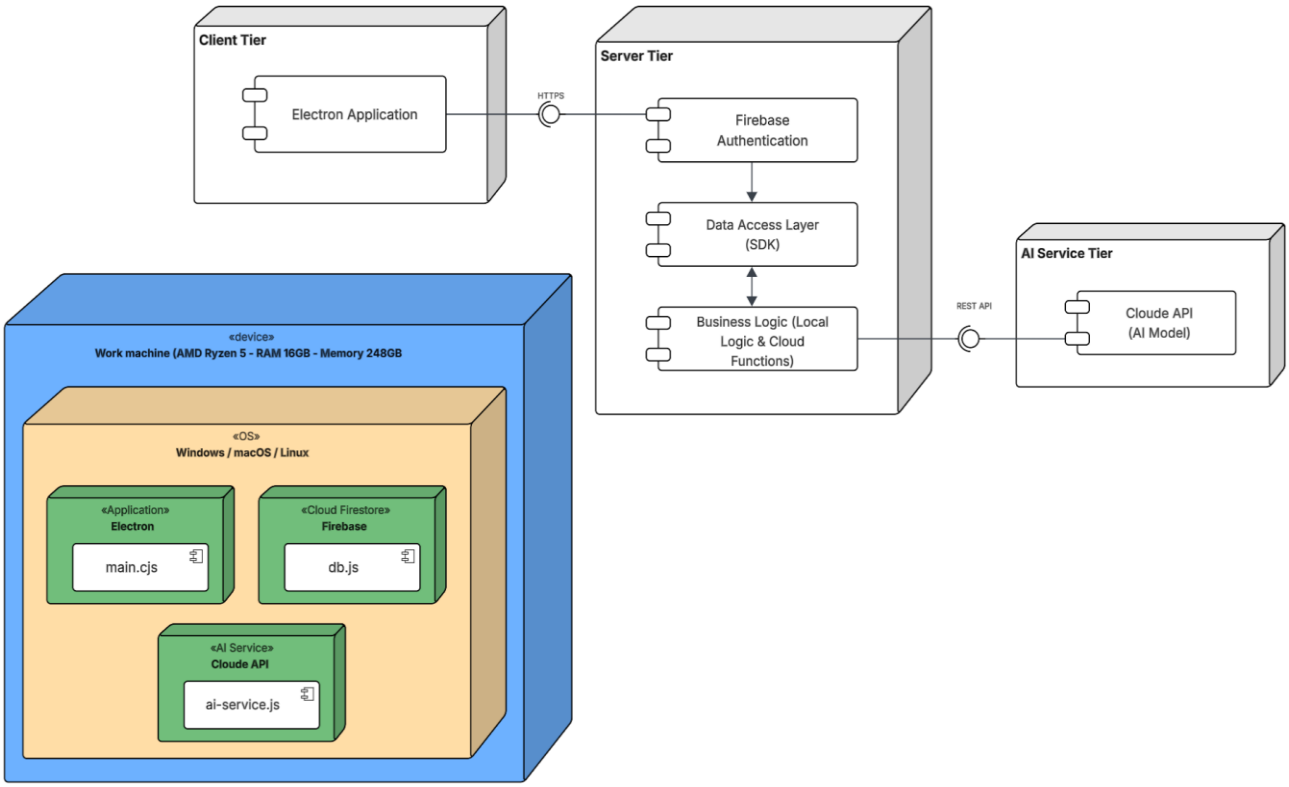

Рисунок Б.2 – Deployment діаграма

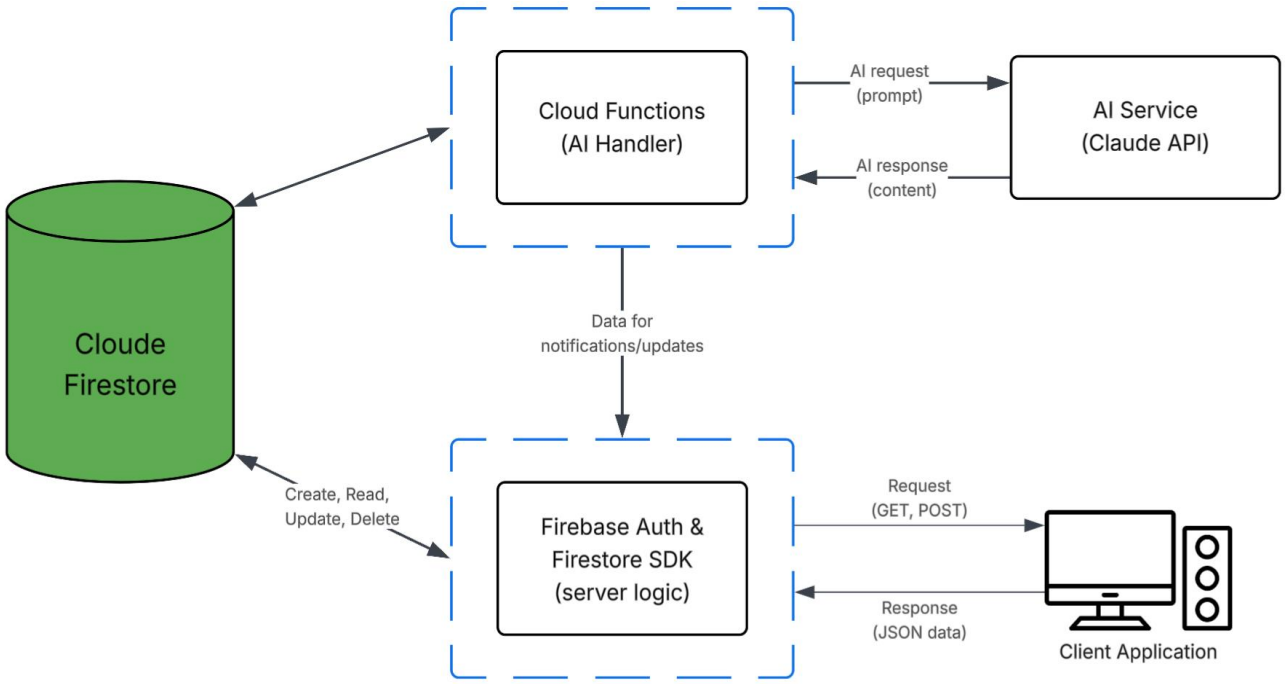

Рисунок Б.3 – Data Flow діаграма

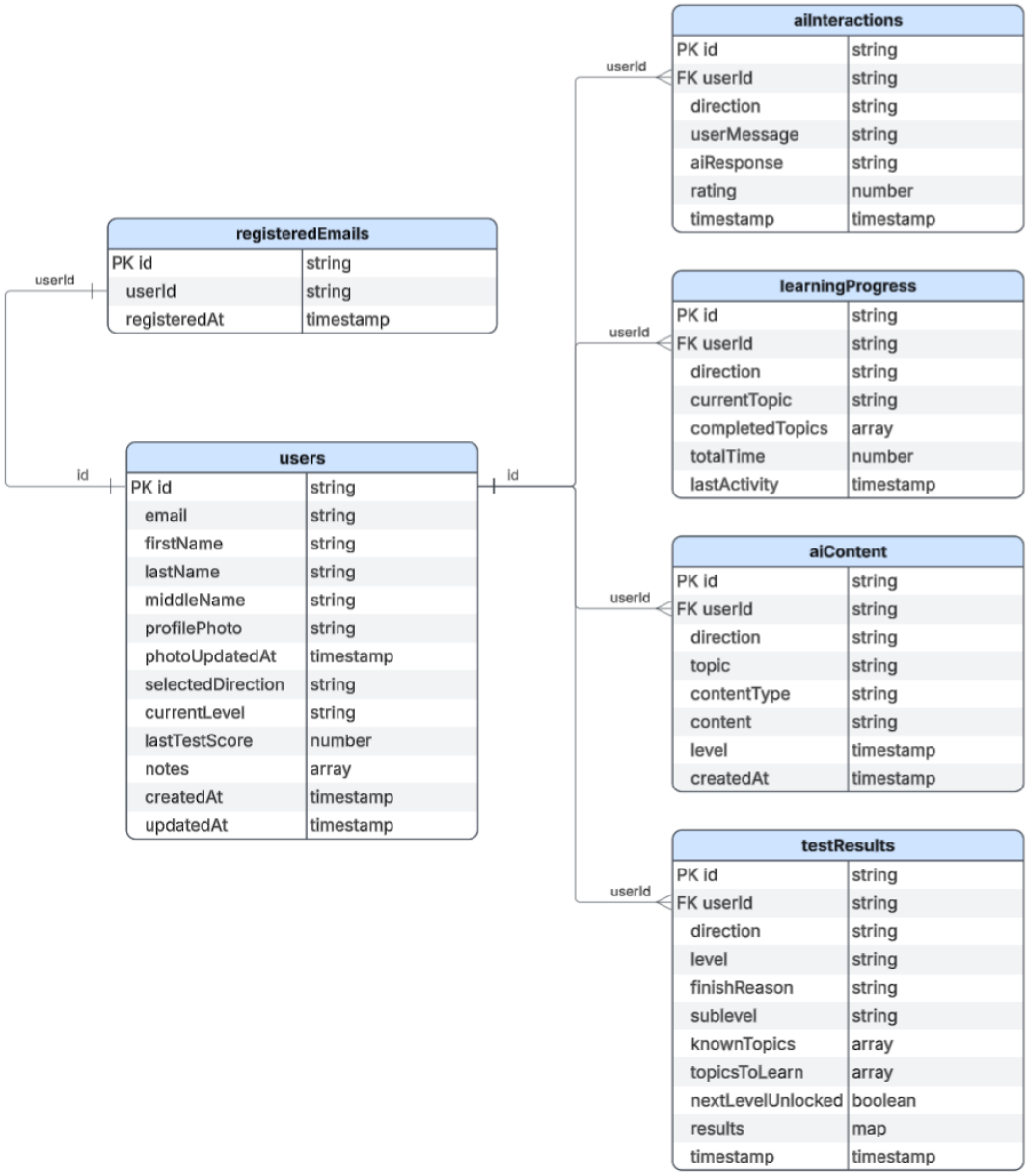

Рисунок Б.4 – Загальна ER-діаграма зв'язків між колекціями

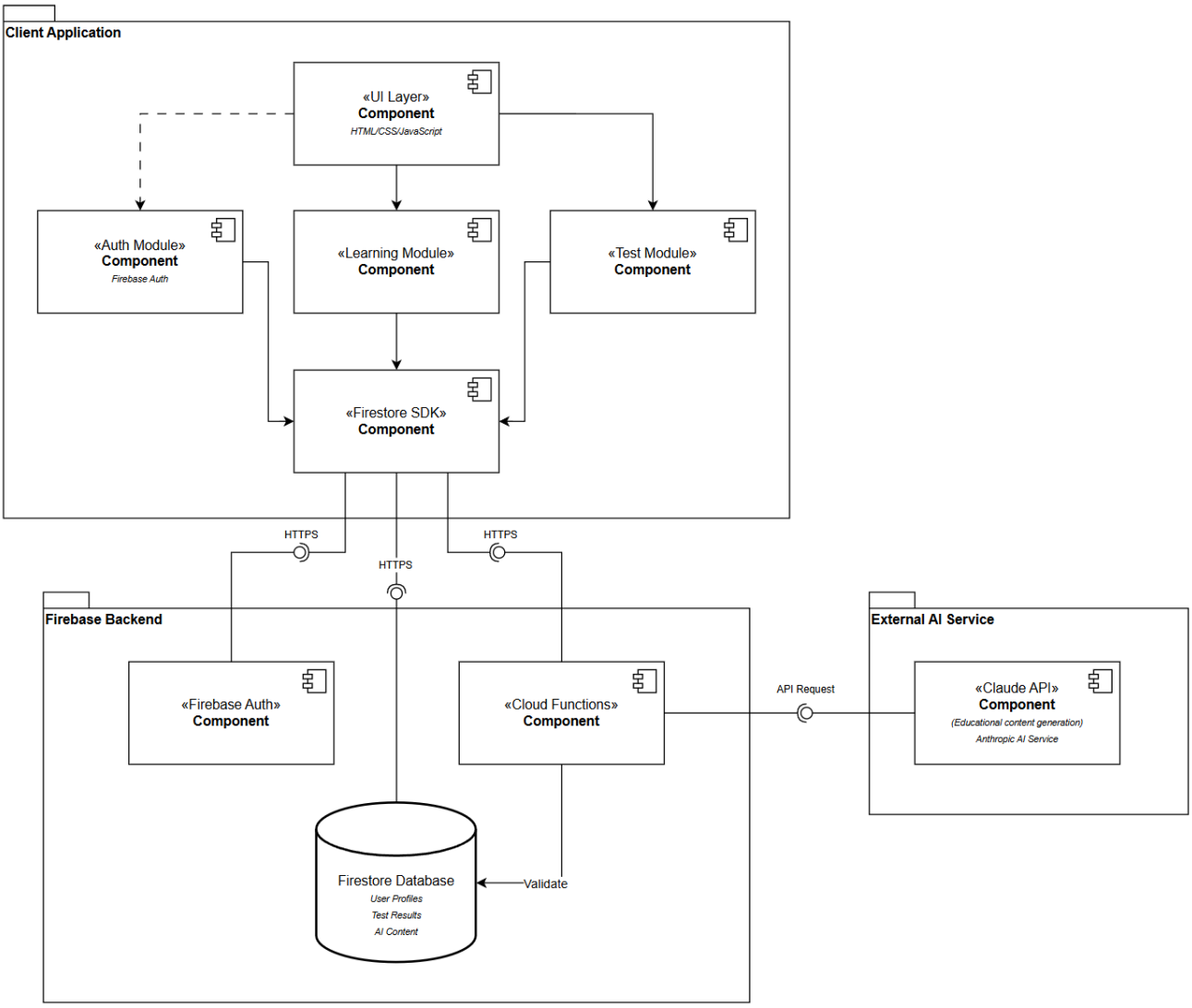

Рисунок Б.5 – Component діаграма архітектурних компонентів системи

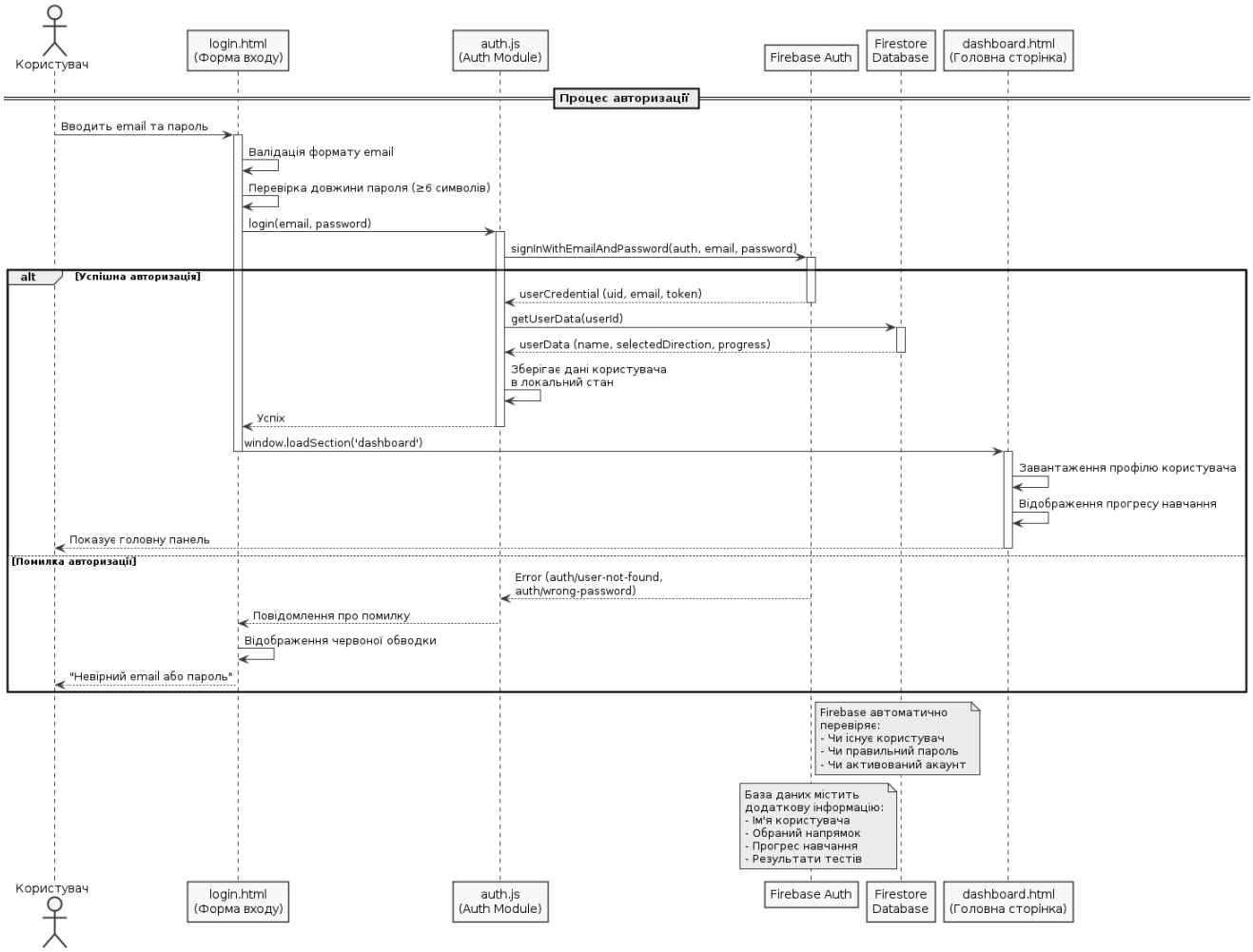

Рисунок Б.6 – Sequence діаграма авторизації

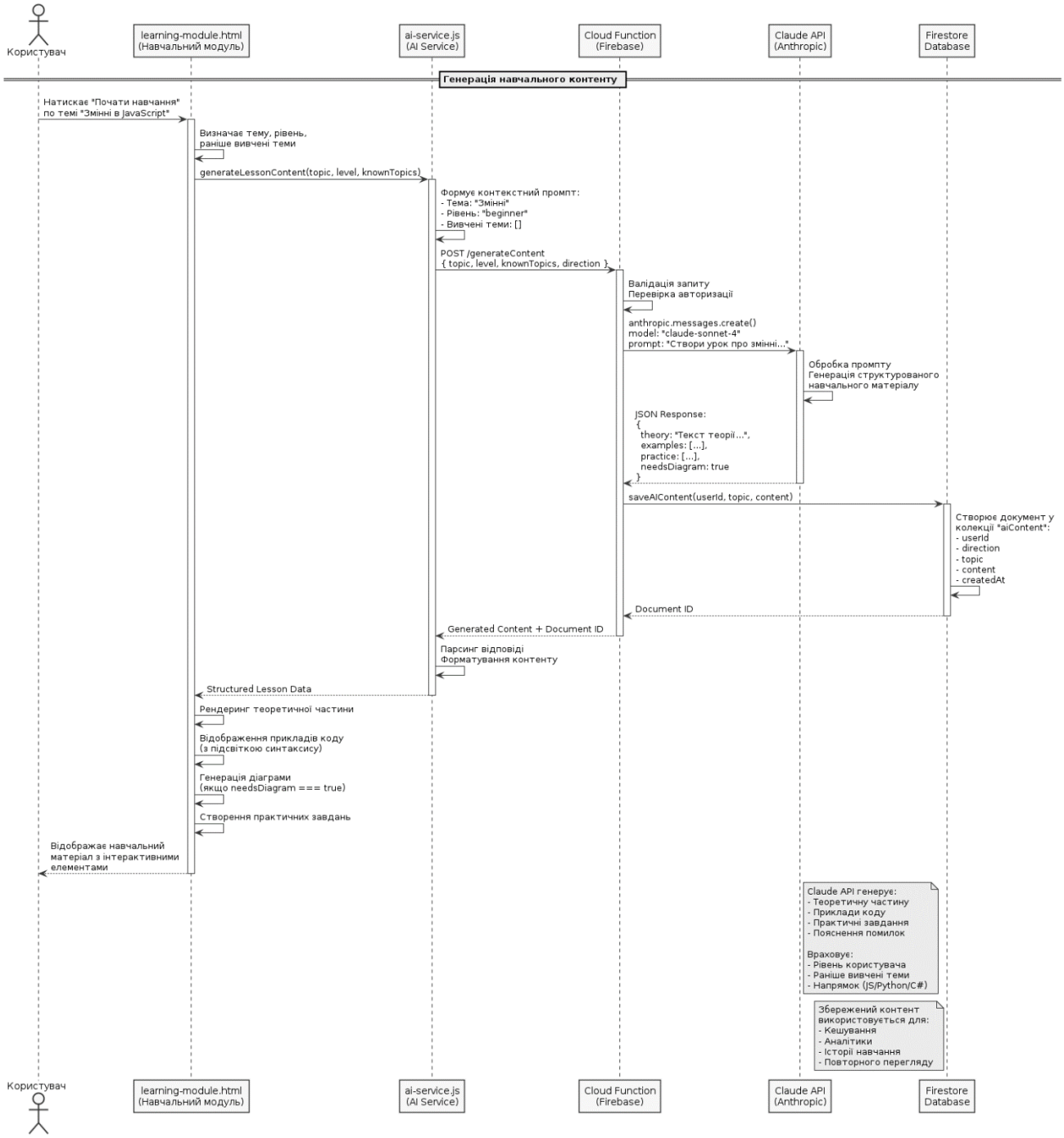

Рисунок Б.7 – Sequence діаграма генерації контенту ШІ

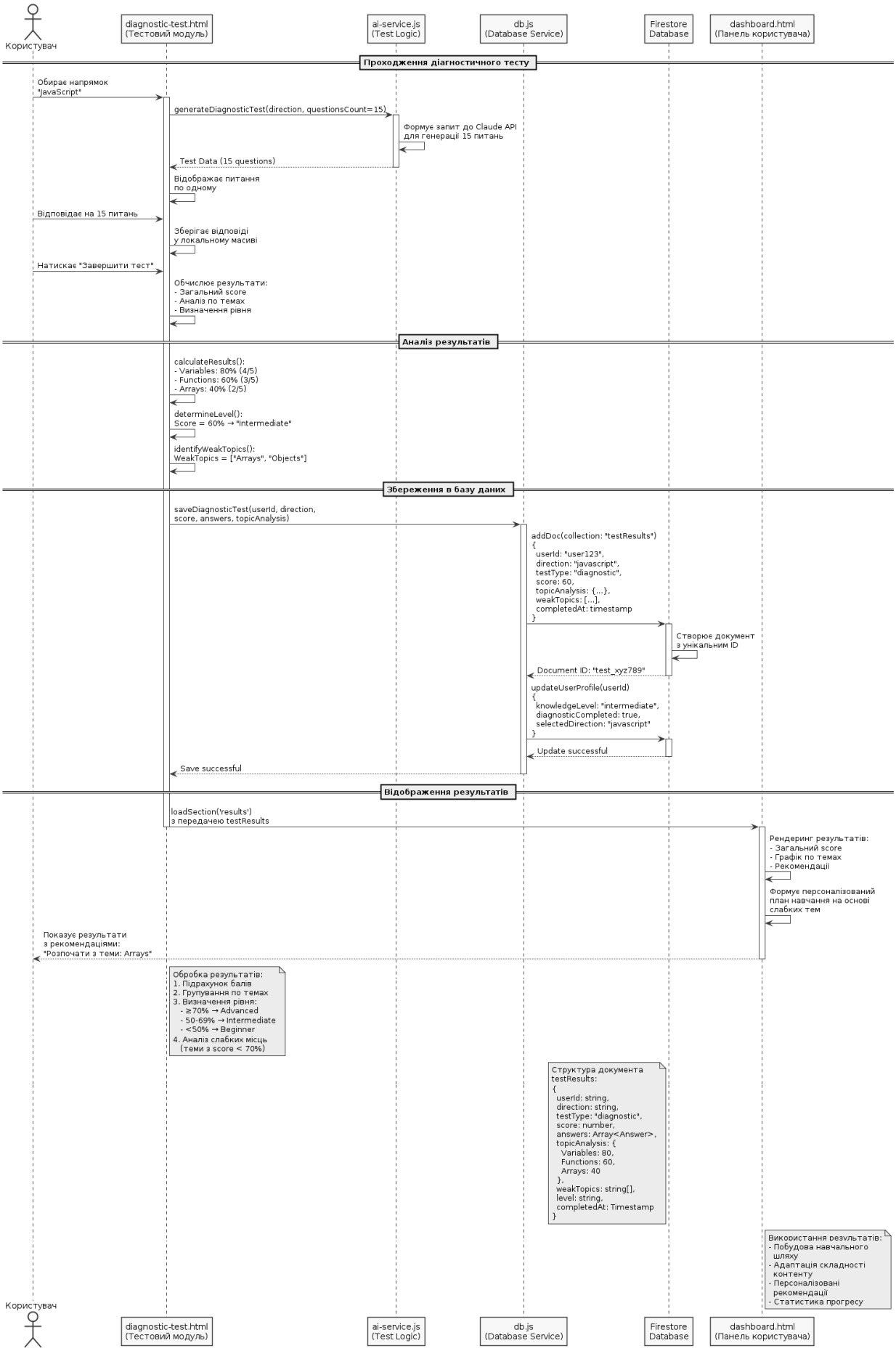

Рисунок Б.8 – Sequence діаграма збереження результатів тесту

Додаток В (обов’язковий) Лістинг модуля діагностичного тестування

```
<div class="test-container">
  ...
</div>
<script>
  (function() {
    const firebaseConfig = {
      ...
    };
    const db = window.db;
    let currentUser = null;
    function waitForAuth() {
      ...
    }
    const CLAUDE_API_KEY = '...';
    const CLAUDE_API_URL = 'https://api.anthropic.com/v1/messages';
    async function callClaudeAPI(prompt, systemPrompt = '', maxTokens = 4000) {
      ...
    }
    const TOPICS_BY_LEVEL = {
      javascript: {
        ...
      },
      python: {
        ...
      },
      csharp: {
        ...
      }
    };
    async function initTest() {
      ...
```

```
}
async function startTesting() {
    ...
}
async function loadNextQuestion() {
    ...
}
async function generateQuestionForTopic(topic, level) {
    ...
}
function formatQuestionText(text) {
    ...
}
function displayQuestion(question) {
    ...
}
function selectOption(index) {
    ...
}
async function processAnswer() {
    ...
}
function showAnswerFeedback(isCorrect, isUnknown) {
    ...
}
function updateLevelIndicator(level) {
    ...
}
function updateProgressBar() {
    ...
}
function finishTest() {
    ...
}
function determineFinalLevel() {
```

```
        ...
    }
    async function saveResults(finalLevel) {
        ...
    }
    function showResults(finalLevel) {
        ...
    }
    async function generateLevelReasoning(direction, level, correctAnswers, totalQuestions,
mistakesCount) {
        ...
    }
    function getDirectionName(direction) {
        ...
    }
    function showError(message) {
        ...
    }
    nextBtn.addEventListener('click', processAnswer);
    initTest();
  })();
</script>
```

Додаток Г (обов’язковий) Лістинг модуля адаптивного навчання

```
<div class="learning-container">
    <aside class="sidebar">
        ...
    </aside>
    <main class="main-content">
        ...
    </main>
</div>
<script>
    (function () {
        function waitForPrism(callback) {
            ...
        }
        waitForPrism(() => {
            ...
        });
    })();
</script>
<script>
    (function () {
        function initMermaid() {
            ...
        }
        function tryInitMermaid() {
            ...
        }
        async function callClaudeAPI(prompt, systemPrompt = '', maxTokens = 4000) {
            ...
        }
        const topicsByDirection = {
            javascript: [
                ...
```

```
    ],
    python: [
        ...
    ],
    csharp: [
        ...
    ]
};
function waitForAuth() {
    ...
}
function displayPracticeForm(practiceData) {
    ...
}
function initCodeEditors() {
    ...
}
window.saveTestAnswers = async function () {
    ...
};
window.saveTask1 = async function () {
    ...
};
window.saveTask2 = async function () {
    ...
};
async function updatePracticeProgress() {
    ...
}
function updateSubmitButton() {
    ...
}
async function savePracticeToFirestore() {
    ...
}
```

```
async function loadPracticeFromFirestore() {
    ...
}
function restoreSavedPracticeData(savedPractice) {
    ...
}
function displaySubmittedFeedback(savedPractice) {
    ...
}
async function initLearning() {
    ...
}
async function loadCompletedTopics() {
    ...
}
async function loadPersonalizedTopics() {
    ...
}
async function getPreviousLevelTopics(direction, currentLevel) {
    ...
}
async function generateTopicsForNextLevel(direction, level) {
    ...
}
function updateHeader() {
    ...
}
function loadTopics() {
   ...
}
function updatePracticeButton() {
    ...
}
function displayContent(topic, htmlContent) {
    ...
```

```
    }
    window.selectTopic = async function(topic) {
                        ...
    }
    window.proceedToNextLevel = async function(nextLevel) {
      ...
    };
    window.markTopicCompleted = async function (topicId) {
      ...
    };
    window.showPractice = async function() {
      ...
    }
    async function generatePracticeContent() {
      ...
    }
    function updateSubmitButton() {
      ...
    }
    async function saveTestAnalysisToCourseAnalytics(userId, direction, mistakes) {
      ...
    }
    async function savePracticeFeedbackToCourseAnalytics(userId, direction, taskTitle, code,
feedback) {
      ...
    }
    window.submitPractice = async function () {
      ...
    };
    async function analyzeCode(code) {
      ...
    }
    initLearning();
  })();
</script>
```

Додаток Д (обов’язковий) Лістинг модуля інтеграції з Claude API

```
(function() {
    function waitForRAG() {
        ...
    }
    function initializeAIService() {
        async function callClaudeAPI(prompt, systemPrompt = '', maxTokens = 4000) {
            const requestBody = {
                ...
            };
            if (systemPrompt) {
                ...
            }
            try {
                const response = await fetch(CLAUDE_API_URL, {
                    ...
                });
                if (!response.ok) {
                    ...
                }
            } catch (error) {
                throw error;
            }
        }
        window.generateLearningContentWithRAG = async function(userId, direction, topic, level,
contentType = 'theory') {
            try {
                const relevantMaterials = await window.searchRelevantLearningContent(topic, direction,
3);
                const context = window.formatContextForAI(relevantMaterials);
                const systemPrompt = `...`;
                let mainPrompt = '';
```

```
            if (contentType === 'theory') {
                mainPrompt = `...`;
            } else if (contentType === 'practice') {
                mainPrompt = `...`;
            }
            const generatedContent = await callClaudeAPI(mainPrompt, systemPrompt, 4000);
            await window.saveAIContentWithKeywords(userId, direction, topic, contentType,
generatedContent, level);
            return {
                ...
            };
        } catch (error) {
            throw error;
        }
    };
    window.answerQuestionWithRAG = async function(userQuestion, direction, userId) {
        try {
            const relevantMaterials = await window.searchRelevantLearningContent(userQuestion,
direction, 3);
            const context = window.formatContextForAI(relevantMaterials);
            const systemPrompt = `...`;
            const mainPrompt = `...`;
            const answer = await callClaudeAPI(mainPrompt, systemPrompt, 3000);
            return {
                answer: answer,
                usedContext: relevantMaterials.length > 0,
                contextCount: relevantMaterials.length
            };
        } catch (error) {
            throw error;
        }
    };
    window.AI_RAG_READY = true;
  }
})();
```

Додаток Е (обов'язковий) Лістинг модуля RAG-системи

```
(function() {
    'use strict';
    function waitForSystems() {
        return new Promise((resolve) => {
            const checkSystems = setInterval(() => {
                const keywordReady = typeof window.searchRelevantLearningContent === 'function';
                const vectorReady = typeof window.vectorSearchAPI === 'function';
                const saveReady = typeof window.saveAIContentWithKeywords === 'function';
                if (keywordReady && vectorReady && saveReady) {
                    clearInterval(checkSystems);
                    resolve();
                }
            }, 100);
        });
    }
    waitForSystems().then(() => {
        window.hybridSearch = async function(queryText, direction, topK = 5) {
            try {
                const [keywordResults, vectorResults] = await Promise.all([
                    window.searchRelevantLearningContent(queryText, direction, topK)
                        .catch(error => {
                            return [];
                        }),
                    window.vectorSearchAPI(queryText, direction, topK)
                        .catch(error => {
                            return [];
                        })
                ]);
                const combinedResults = combineAndRankResults(
                    keywordResults,
                    vectorResults,
```

```
            topK
        );
        if (combinedResults.length > 0) {
            combinedResults.slice(0, 3).forEach((result, i) => {
                console.log(` ${i+1}. ${result.topic} (score: ${result.hybridScore.toFixed(3)},
sources: ${result.sources.join('+')})`);
            });
        }
        return combinedResults;
    } catch (error) {
        return window.searchRelevantLearningContent(queryText, direction, topK);
    }
};
function combineAndRankResults(keywordResults, vectorResults, topK) {
    const resultsMap = new Map();
    keywordResults.forEach((result, index) => {
        const id = result.id || result.topic;
        const keywordRank = keywordResults.length - index;
        resultsMap.set(id, {
            ...
        });
    });
    vectorResults.forEach((result, index) => {
        const id = result.id || result.topic;
        const vectorRank = vectorResults.length - index;
        if (resultsMap.has(id)) {
            ...
        } else {
            ...
        }
    });
    const results = Array.from(resultsMap.values()).map(result => {
        const KEYWORD_WEIGHT = 0.4;
        const VECTOR_WEIGHT = 0.6;
        const BOTH_SYSTEMS_BONUS = 0.3;
```

```
            const normalizedKeyword = result.keywordScore > 0 ?
                (result.keywordScore / Math.max(...keywordResults.map(r => r.relevanceScore || 1))) :
0;
            const normalizedVector = result.vectorScore > 0 ? result.vectorScore : 0;
            let hybridScore = (normalizedKeyword * KEYWORD_WEIGHT) + (normalizedVector *
VECTOR_WEIGHT);
            if (result.sources.length === 2) {
                hybridScore += BOTH_SYSTEMS_BONUS;
            }
            return {
                ...
            };
        });
        results.sort((a, b) => b.hybridScore - a.hybridScore);
        return results.slice(0, topK);
    }
    window.saveAIContentHybrid = async function(userId, direction, topic, contentType, content,
level) {
        try {
            const docId = await window.saveAIContentWithKeywords(
                ...
            );
            try {
                if (typeof window.indexDocumentAPI === 'function') {
                    await window.indexDocumentAPI(docId, content, {
                        userId,
                        direction,
                        topic,
                        contentType,
                        level
                    });
                }
            } catch (vectorError) {
                console.warn('Pinecone індексація не вдала:', vectorError);
            }
```

```
            return docId;
        } catch (error) {
            console.error('Помилка збереження:', error);
            throw error;
        }
    };
    window.HYBRID_RAG_READY = true;
}).catch(error => {
    console.error('Помилка ініціалізації гібридного RAG:', error);
});
})();
```

Додаток Ж (обов'язковий) Лістинг модуля управління базою даних

```
import { initializeApp } from 'firebase/app';
import {
    ...
} from 'firebase/firestore';
import {
    ...
} from 'firebase/auth';
const firebaseConfig = {
    ...
};
export async function createUserDocument(user, additionalData = {}) {
    if (!user) return;
    const userRef = doc(db, 'users', user.uid);
    try {
        ...
    } catch (error) {
        ...
        throw error;
    }
}
export async function getUserData(userId) {
    try {
        ...
    } catch (error) {
        ...
    }
}
export async function updateUserDirection(userId, direction) {
    try {
        ...
    } catch (error) {
```

```
        ...
    }
}
export async function updateUserLevel(userId, level) {
    try {
        ...
    } catch (error) {
        ...
    }
}
export async function saveDiagnosticTest(userId, direction, score, answers) {
    try {
        ...
    } catch (error) {
        ...
    }
}
export async function saveTopicTest(userId, direction, topic, score, answers) {
    try {
        ...
    } catch (error) {
        ...
    }
}
export async function getUserTests(userId) {
    try {
        ...
    } catch (error) {
        ...
    }
}
export async function saveAIContent(userId, direction, topic, contentType, content, level) {
    try {
        ...
    } catch (error) {
```

```
        ...
    }
}
export async function getAIContent(userId, direction, topic, contentType) {
    try {
        ...
    } catch (error) {
        ...
    }
}
export async function updateLearningProgress(userId, direction, topic, timeSpent = 0) {
    try {
        ...
    } catch (error) {
        ...
    }
}
export async function markTopicCompleted(userId, direction, topic) {
    try {
        ...
    } catch (error) {
        ...
    }
}
export async function getLearningProgress(userId, direction) {
    try {
        ...
    } catch (error) {
        ...
    }
}
export async function saveAIInteraction(userId, direction, userMessage, aiResponse, rating = null)
{
    try {
        ...
```

```
    } catch (error) {
        ...
    }
}
export async function getAIInteractions(userId, limitCount = 10) {
    try {
        ...
    } catch (error) {
        ...
    }
}
export async function addActiveCourse(userId, direction) {
    try {
        ...
    } catch (error) {
        ...
    }
}
export async function removeActiveCourse(userId, direction) {
    ...
}
export async function saveDiagnosticResults(userId, direction, diagnosticData) {
    ...
}
export async function addTestAnalysis(userId, direction, testData) {
    ...
}
export async function addPracticeFeedback(userId, direction, practiceData) {
    ...
}
export async function getCourseAnalytics(userId, direction) {
    ...
}
export { db, auth };
```

Додаток К (обов'язковий) Протокол перевірки МКР

ПРОТОКОЛ ПЕРЕВІРКИ КВАЛІФІКАЦІЙНОЇ РОБОТИ НА НАЯВНІСТЬ ЗАПОЗПОЗИЧЕНЬ

Назва роботи: «Розробка клієнт-серверної системи персоналізованого навчання з адаптивним контентом на основі штучного інтелекту»

Тип роботи: магістерська кваліфікаційна робота
(бакалаврська кваліфікаційна робота / магістерська кваліфікаційна робота)

Підрозділ: кафедра АІІТ
(кафедра, факультет, навчальна група)

Коефіцієнт подібності текстових запозичень, виявлених у роботі системою StrikePlagiarism (КП1) 0.08 %

Висновок щодо перевірки кваліфікаційної роботи (відмітити потрібне)

- Запозичення, виявлені у роботі, оформлені коректно і не містять ознак академічного плагіату, фабрикації, фальсифікації. Роботу прийняти до захисту.
- У роботі не виявлено ознак плагіату, фабрикації, фальсифікації, але надмірна кількість текстових запозичень та/або наявність типових розрахунків не дозволяють прийняти рішення про оригінальність та самостійність її виконання. Роботу направити на доопрацювання.
- У роботі виявлено ознаки академічного плагіату та/або в ній містяться навмисні спотворення тексту, що вказують на спроби приховування недобросовісних запозичень. Робота до захисту не приймається.

Експертна комісія:

Бісікало О.В., зав. каф. АІІТ _____
(прізвище, ініціали, посада) (підпис)

Овчинников К.В., доц. каф. АІІТ _____
(прізвище, ініціали, посада) (підпис)

Особа, відповідальна за перевірку _____ Маслій Р.В.
(підпис) (прізвище, ініціали)

З висновком експертної комісії ознайомлений(-на)

Керівник _____ Коцюбинський В.Ю., к.т.н. доц. каф. АІІТ
(підпис) (прізвище, ініціали, посада)

Студент _____ Морозов П.В.
(підпис) (прізвище, ініціали)