

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДЕРЖАВНИЙ УНІВЕРСИТЕТ «ЖИТОМИРСЬКА ПОЛІТЕХНІКА»
ФАКУЛЬТЕТ ІНФОРМАЦІЙНО-КОМП'ЮТЕРНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

ПОЯСНЮВАЛЬНА ЗАПИСКА

до кваліфікаційної роботи освітнього ступеня «бакалавр»
за спеціальністю 121 «Інженерія програмного забезпечення»
(освітня програма «Інженерія програмного забезпечення»)

на тему:

«Мобільний додаток для вивчення іноземних мов»

Виконав студент групи ПЗ-22-3
СОКУЛЬСЬКИЙ Олег Ігорович

Керівник роботи:
ВАКАЛЮК Тетяна Анатоліївна

Рецензент:
ЛОБАНЧИКОВА Надія Миколаївна

Житомир – 2026

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДЕРЖАВНИЙ УНІВЕРСИТЕТ «ЖИТОМИРСЬКА ПОЛІТЕХНІКА»
ФАКУЛЬТЕТ ІНФОРМАЦІЙНО-КОМП'ЮТЕРНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

«ЗАТВЕРДЖУЮ»

Зав. кафедри інженерії
програмного забезпечення

Тетяна Вакалюк

«28» лютого 2026 р.

ЗАВДАННЯ
на кваліфікаційну роботу

Здобувач вищої освіти: **СОКУЛЬСЬКИЙ Олег Ігорович**

Керівник роботи: **ВАКАЛЮК Тетяна Анатоліївна**

Тема роботи: **«Мобільний додаток для вивчення іноземних мов»**,
затверджена Наказом закладу вищої освіти від **«27» лютого 2026 р., № 92/с**

Вихідні дані для роботи: об'єктом дослідження є процес цифрової взаємодії користувача з освітнім контентом, що включає механізми засвоєння лексичного матеріалу, систематизацію знань та персоналізацію навчання через мобільні інтерфейси.; предметом дослідження є архітектурні методи побудови мобільного застосунку, алгоритми адаптивного тестування користувачів та технологічний стек , що забезпечує стабільну роботу сервісу з вивчення мов.

Консультанти з бакалаврської кваліфікаційної роботи із зазначенням розділів, що їх стосуються:

Розділ	Консультант	Завдання видав	Завдання прийняв
1	Вакалюк Т.А.	01.03.2026р.	01.03.2026р
2	Вакалюк Т.А.	01.03.2026р	01.03.2026р
3	Вакалюк Т.А.	01.03.2026р	01.03.2026р

РЕФЕРАТ

Пояснювальна записка до бакалаврської кваліфікаційної роботи на тему «Мобільний додаток для вивчення іноземних мов» викладена на 100 сторінках друкованого тексту. Візуальний склад роботи включає 48 рисунків та 18 таблиць. Список використаної літератури охоплює 42 джерел, додатки розміщені на 5 сторінках. Повний обсяг пояснювальної записки – 110 сторінок.

Метою роботи є розробка та програмна реалізація мобільного додатка для вивчення іноземних мов, суттю якої є автоматизація освітнього процесу та засвоєння іноземної лексики й граматики за допомогою інтерактивної взаємодії користувача з автентичним аудіоконтентом, фіксації поточного прогресу та гнучкої персоналізації навчального треку в реальному часі.

Архітектура сервісу базується на використанні фреймворку React Native [1] для реалізації кросплатформеної клієнтської частини у поєднанні з розробкою серверного backend API [2] для надійної обробки, структурування та збереження користувацьких даних. Використання компонентного підходу та оптимізованих алгоритмів потокової передачі аудіо даних забезпечує високу швидкість завантаження навчальних матеріалів, низьке споживання ресурсів пристрою та нативний досвід взаємодії (Native UX), що є критичним для забезпечення безперервного й комфортного навчання.

Практична цінність. Результатом роботи є робочий прототип мобільного застосунку, адаптованого під потреби сучасних користувачів. Проект реалізовано з використанням сучасної мобільної розробки (наприклад, кросплатформеного підходу), що дозволяє досягти високої продуктивності та нативного досвіду взаємодії. повторень.

КЛЮЧОВІ СЛОВА: МОБІЛЬНИЙ ДОДАТОК, ІНОЗЕМНІ МОВИ, EDTECH, ІНТЕРФЕЙС КОРИСТУВАЧА, КРОСПЛАТФОРМЕНА РОЗРОБКА, АДАПТИВНЕ НАВЧАННЯ, ПЕРСОНАЛІЗАЦІЯ.

					ІПЗ.КР.Б – 121 – 26 – ПЗ			
Змн.	Арк.	№ докум.	Підпис	Дата				
Розроб.		Сокульський О.І.			Мобільний додаток для вивчення іноземних мов	Літ.	Арк.	Аркуші
Керівник		Вакалюк Т.А.					4	110
Рецензент		Лобанчикова Н.М.						
Зав. каф		Вакалюк Т.А.				Житомирська політехніка, група ІПЗ-22-3		

ABSTRACT

The explanatory note for the bachelor's qualification work on the topic "Mobile Application for Learning Foreign Languages" consists of 100 pages of printed text. The visual content of the work includes 48 figures and 18 tables. The reference list encompasses 42 sources, and the appendices occupy 5 pages. The total volume of the explanatory note is 110 pages.

The objective of the work is the development and software implementation of a mobile application for learning foreign languages. Its core purpose is the automation of the educational process and the acquisition of foreign vocabulary and grammar through the user's interactive engagement with authentic audio content, current progress tracking, and flexible real-time personalization of the learning track.

The service architecture is based on the React Native framework [1] for implementing the cross-platform client side, combined with the development of a server backend API [2] for reliable processing, structuring, and storage of user data. The use of a component-based approach and optimized audio data streaming algorithms ensures high loading speeds for educational materials, low device resource consumption, and a native user experience (Native UX), which is critical for ensuring continuous and comfortable learning.

Practical value. The result of the work is a functional prototype of a mobile application adapted to the needs of modern users. The project is implemented using modern mobile development techniques (e.g., a cross-platform approach), which allows achieving high performance and a native user experience. repetitions.

KEYWORDS: MOBILE APPLICATION, FOREIGN LANGUAGES, EDTECH, USER INTERFACE, CROSS-PLATFORM DEVELOPMENT, ADAPTIVE LEARNING, PERSONALIZATION.

					ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
						5
Змн.	Арк.	№ докум.	Підпис	Дата		

ЗМІСТ

РЕФЕРАТ.....	4
ABSTRACT	5
ЗМІСТ	6
ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ.....	8
ВСТУП.....	9
РОЗДІЛ 1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАДАЧІ	11
1.1 Постановка задачі та огляд існуючих систем-аналогів	11
1.2 Функціональні та нефункціональні вимоги до системи	19
1.3 Вибір і обґрунтування архітектурного рішення	23
1.4 Вибір стеку технологій та апаратного забезпечення	26
Висновки до першого розділу	28
РОЗДІЛ 2. ПРОЄКТУВАННЯ ПРОГРАМНОЇ СИСТЕМИ.....	30
2.1 Моделювання варіантів використання та поведінки системи	30
2.2 Проєктування логічної структури бази даних	37
2.3 Проєктування компонентної структури та взаємодії модулів	46
2.4 Розробка основних алгоритмів роботи платформи.....	58
Висновки до другого розділу	66
Розділ 3. ІНТЕРФЕЙС ТА ПОРЯДОК РОБОТИ З ІНФОРМАЦІОНОЮ СИСТЕМОЮ УПРАВЛІННЯ БУДІВЕЛЬНИМИ ПРОЄКТАМИ ТА РОБОЧИМИ ПРОЦЕСАМИ.....	68
3.1 Порядок встановлення та налаштування параметрів системи	68
3.2 Структура інтерфейсу інформаційної системи моніторингу та управління будівельними проєктами.....	72
3.3 Тестування роботи програмного продукту	86

Висновки до третього розділу	92
ВИСНОВКИ	94
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ.....	96
ДОДАТКИ	100

					ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
						7
Змн.	Арк.	№ докум.	Підпис	Дата		

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

БД – база даних

ПЗ – програмне забезпечення

СУБД – система управління базами даних

API (Application Programming Interface) – прикладний програмний інтерфейс

DTO (Data Transfer Object) – об'єкт передачі даних між шарами архітектури.

CRUD (Create, Read, Update, Delete) – базові функції управління даними (створення, читання, оновлення, видалення)

JSON (JavaScript Object Notation) – текстовий формат обміну даними

JWT (JSON Web Token) – відкритий стандарт для безпечної передачі інформації у вигляді JSON-об'єкта

MVVM (Model-View-ViewModel) – архітектурний патерн, що є стандартом для сучасної мобільної розробки

ORM (Object-Relational Mapping) – технологія програмування, яка зв'язує бази даних з об'єктно-орієнтованим кодом

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

REST (Representational State Transfer) – архітектурний стиль взаємодії компонентів розподіленого застосунку в мережі

SQL (Structured Query Language) – мова структурованих запитів до реляційних баз даних

UI (User Interface) – інтерфейс користувача

SDK (Software Development Kit) – набір засобів розробки для певної платформи (Android/iOS).

ВСТУП

Актуальність теми полягає у стабільно високому попиті на гнучкі та персоналізовані інструменти для вивчення іноземних мов в умовах глобальної цифровізації, а також у потребі користувачів мати зручний мобільний інструмент для безперервного доступу до знань. Особливої ваги сьогодні набуває можливість споживання освітнього матеріалу в аудіоформаті, що дозволяє зручно інтегрувати навчання у повсякденну рутину та розвивати навички сприйняття мови на слух. У сучасних умовах ринок EdTech-індустрії вимагає надійних рішень, оскільки існуючі продукти часто стикаються з проблемою перевантаженого інтерфейсу або відсутністю адаптивних алгоритмів, що враховують індивідуальний темп засвоєння матеріалу. Таке когнітивне перевантаження швидко знижує мотивацію користувачів і перешкоджає регулярним заняттям. Створення сучасного мобільного додатка для вивчення мов дозволяє автоматизувати процес подачі контенту, забезпечити інтерактивну взаємодію користувача з матеріалом та гарантувати ефективне відстеження прогресу в реальному часі. З технічної точки зору, проектування такого застосунку передбачає розробку оптимізованої архітектури з надійним API та продуманим клієнтським інтерфейсом, що робить цю задачу вкрай актуальною для сучасної інженерії програмного забезпечення.

Метою роботи є розробка та програмна реалізація мобільного додатка для вивчення іноземних мов, суттю якої є автоматизація процесу засвоєння лексики та граматики з використанням інтерактивного аудіоконтенту, систем тестування та інструментів персоналізації освітнього треку користувача в реальному часі. Для реалізації поставленої мети необхідно виконати низку завдань, зокрема: провести аналіз предметної області EdTech та існуючих мобільних рішень і сформулювати вимоги до функціональності системи, здійснити вибір засобів розробки з обґрунтуванням використання сучасного технологічного стеку кросплатформеної розробки, спроектувати базу даних та архітектуру API, а також розробити зручну клієнтську частину застосунку із впровадженням інтерактивних вправ та системи візуалізації навчального прогресу.

					ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
						9
Змн.	Арк.	№ докум.	Підпис	Дата		

Об'єктом дослідження є процес програмної реалізації створення мобільного додатка для інтерактивного навчання та управління освітнім контентом.

Предметом дослідження є моделі, методи та програмні засоби побудови кросплатформеного мобільного додатка для опанування іноземної мови. Дослідження також охоплює питання оптимізації обробки даних та забезпечення коректної синхронізації локального прогресу користувача з хмарним сервером.

Методи дослідження. У роботі застосовувався комплексний підхід, що базується на методах системного та порівняльного аналізу для дослідження ринку EdTech-рішень та вибору оптимального технологічного стеку. Для побудови логічної архітектури та розробки функціональних модулів застосовувалися методи об'єктно-орієнтованого проєктування, а для перевірки надійності алгоритмів засвоєння лексики та зручності користувацького інтерфейсу застосовувалися методи емпіричного тестування.

Практичне значення одержаних результатів. Одержані результати мають безпосереднє прикладне значення, оскільки розроблений програмний продукт може бути використаний як готова основа для запуску реального освітнього сервісу. Впровадження такого застосунку дозволить користувачам знизити поріг входу в навчальний процес та підвищити ефективність запам'ятовування, а гнучка архітектура системи забезпечить можливість легкого масштабування, додавання нових мовних пар та інтеграції зовнішніх API без глибокого рефакторингу.

РОЗДІЛ 1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАДАЧІ

1.1 Постановка задачі та огляд існуючих систем-аналогів

Предметною областю дипломного проєкту є сфера мобільних освітніх технологій (EdTech), зокрема сегмент програмного забезпечення для вивчення іноземних мов. В умовах глобалізації та переходу до концепції "Lifelong Learning" (навчання впродовж життя) володіння іноземними мовами стало базовою компетенцією. Проте ефективність опанування нових знань безпосередньо залежить від інструментарію, який використовує учень. Класичний процес вивчення мови за підручниками є фрагментарним і часто призводить до втрати мотивації. Основними перешкодами є відсутність персоналізації, коли універсальні програми не враховують індивідуальний темп учня, складність систематизації матеріалу та ручне ведення словників. Окремою проблемою є "крива забування": без алгоритмічного контролю за інтервалами повторень більша частина вивченого матеріалу втрачається вже за кілька днів. Усвідомлення цих бар'єрів зумовило потребу в інтелектуальних мобільних платформах, де додаток виступає як персональний тьютор, перекладаючи рутинні процеси на програмні алгоритми та надаючи користувачеві інтерактивне середовище.

У сучасному світі мобільні додатки вимагають надійної архітектури та високої швидкодії. Для їх створення існує велика кількість інструментів, що дозволяють реалізовувати найскладніші функціональні вимоги. Фронтенд (клієнтська частина) мобільного застосунку відповідає за інтерфейс користувача та взаємодію з ним. Сьогодні стандартом для створення складних і динамічних інтерфейсів є використання кросплатформеного підходу, лідером якого є фреймворк React Native [3]. Цей інструмент дозволяє розробляти нативні додатки, використовуючи єдину кодову базу. Для побудови інтерфейсу застосовується декларативний підхід та компонентна архітектура, що ідеально узгоджується з концепцією побудови мовних додатків: картки слів, прогрес-бари та модулі тестування динамічно змінюються залежно від успіхів учня.

					ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
						11
Змн.	Арк.	№ докум.	Підпис	Дата		

Бекенд (серверна частина) у сучасних мобільних додатках відповідає за обробку запитів, бізнес-логіку та роботу з API. Використання середовища виконання Node.js [4] дозволяє ефективно реалізовувати серверну логіку та обробку потокового аудіоконтенту. Для забезпечення надійності даних та суворої типізації на всіх рівнях додатку використовується мова TypeScript [5]. Для збереження словникових баз, профілів користувачів та синхронізації прогресу оптимальним є використання реляційних СУБД, таких як PostgreSQL [6] на стороні сервера, а також сучасних локальних рішень на клієнті для забезпечення безперебійної роботи в офлайн-режимі.

Важливим інженерним аспектом є впровадження архітектурного патерну MVVM (Model-View-ViewModel), що дозволяє інкапсулювати управління станом екрана та транзакції до баз даних. Механізми двосторонньої прив'язки даних (data binding) гарантують, що оновлення статусу вивчення слова в локальній базі автоматично синхронізується з візуальними індикаторами прогресу, виключаючи необхідність ручного оновлення. Проведений аналіз інструментів дозволив визначити переваги використання екосистеми React Native та сучасних серверних рішень, що створює міцне підґрунтя для розробки швидкого, адаптивного та персоналізованого освітнього застосунку.

Ринок EdTech-рішень для мобільного навчання демонструє стабільне зростання, оскільки користувачі шукають інструменти для ефективного використання коротких проміжків часу. Одним із найбільш репрезентативних рішень є платформа «Duolingo» (<https://www.duolingo.com/>) – беззаперечний лідер сегмента, функціонал якого базується на тотальній гейміфікації та алгоритмізованій подачі контенту.

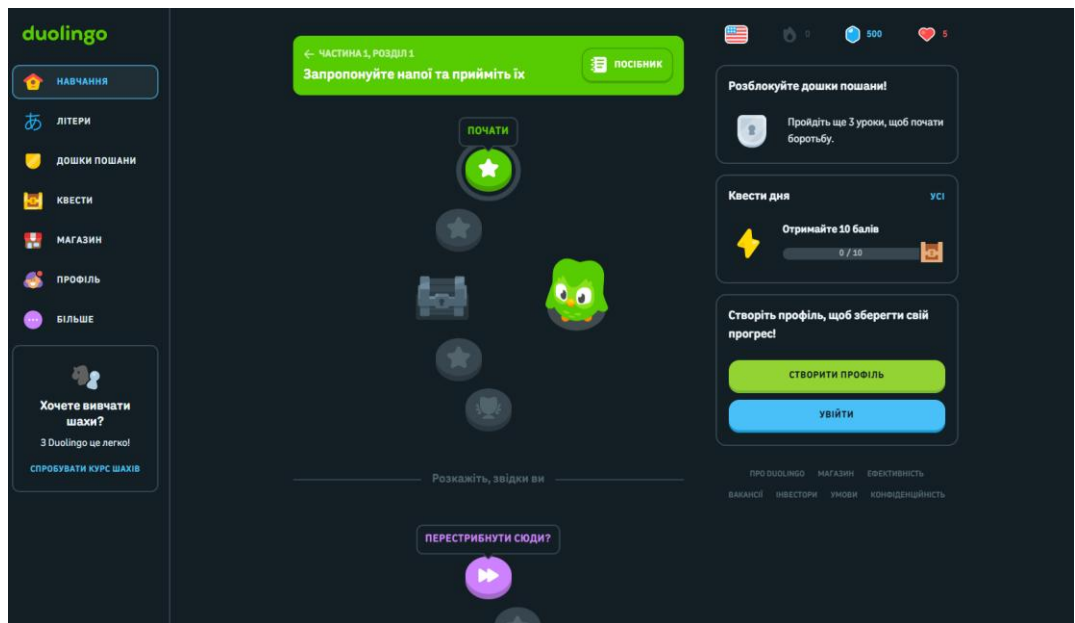


Рис. 1.1 – Інтерфейс головної сторінки додатку Duolingo

На головній сторінці платформи користувача одразу зустрічає структурований навчальний шлях (дерево навичок), згрупований за рівнями складності. Функціонально верхня навігаційна панель забезпечує швидкий доступ до статистики безперервного навчання (ударного режиму), балансу внутрішньої ігрової валюти та кількості доступних життів. Алгоритм подачі контенту реалізований через лінійну прогресію, що дозволяє користувачу фокусуватися на одному конкретному блоці завдань, мінімізуючи когнітивне перевантаження та значно знижуючи поріг входу для новачків.

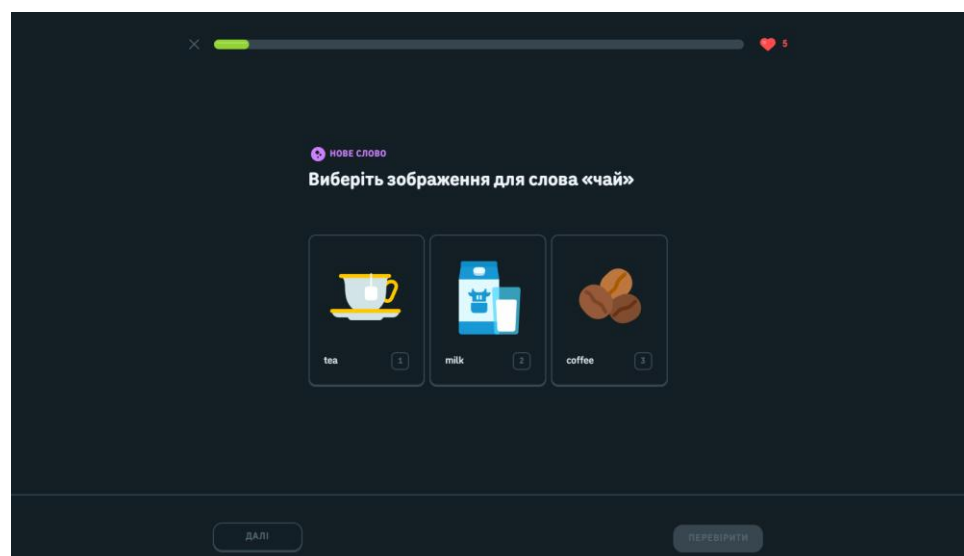


Рис. 1.2 – Сторінка інтерактивної вправи

На сторінці конкретного уроку зосереджено базовий функціонал взаємодії з навчальним матеріалом. Інтерфейс пропонує користувачу різноманітні механіки: переклад фраз, розпізнавання на слух та конструювання речень із готових текстових блоків. Важливим функціональним вузлом є модуль миттєвого зворотного зв'язку, який динамічно реагує на дії учня, підсвічуючи правильні чи неправильні відповіді та виводячи підказки у реальному часі. Ключовим елементом на цьому етапі є кнопка підтвердження, яка запускає логіку перевірки та записує результати у локальний стейт прогресу.

Загальна функціональна характеристика платформи «Duolingo» свідчить про високу здатність архітектури утримувати увагу користувачів через ігрові механіки (ліги, досягнення) та забезпечувати стабільний користувацький досвід. Водночас основним недоліком для досвідчених користувачів є надмірно жорстка лінійна структура. Користувач не має можливості самостійно формувати списки лексики для вивчення, інтегрувати власні словникові бази або фокусуватися виключно на поглибленому прослуховуванні аудіоконтенту. Система функціонує як «чорна скринька» із закритим алгоритмом подачі матеріалу, тоді як сучасні рішення вимагають більшої гнучкості, персоналізації освітнього треку та розширених можливостей для роботи з автентичними матеріалами.

Платформа «Anki» (<https://apps.ankiweb.net/>) є кросплатформним програмним забезпеченням, що базується на методі інтервальних повторень (Spaced Repetition System, SRS). На відміну від ігрових платформ з лінійною структурою, Anki не пропонує жорстко заданого курсу чи готових вправ, а функціонує як потужний інструмент для створення, налаштування та управління персональними словниковими базами у вигляді колод флеш-карток.

					ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
						14
Змн.	Арк.	№ докум.	Підпис	Дата		

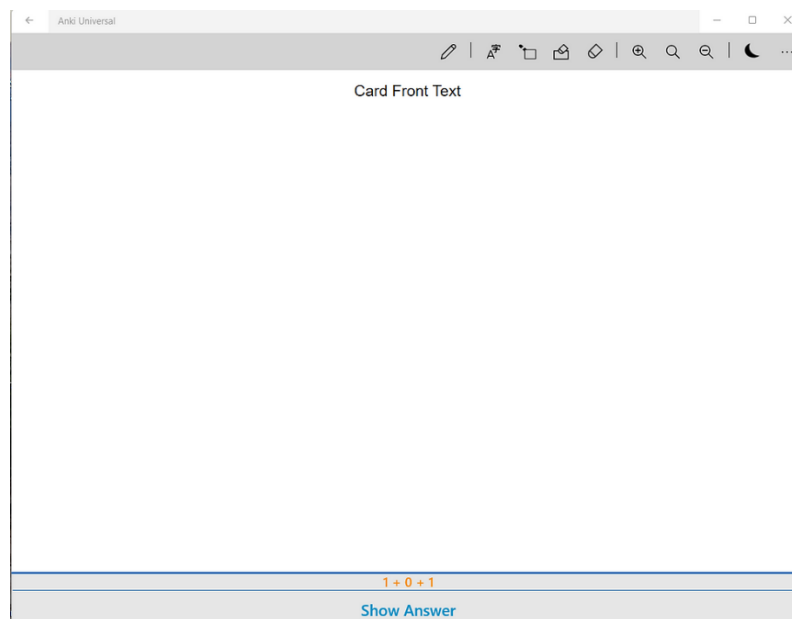


Рис. 1.3 – Головний екран мобільного додатку Anki

На головному екрані платформи користувачу надається доступ до списку створених або завантажених колод. Інтерфейс є максимально утилітарним і фокусується виключно на статистичних даних: навпроти кожної колоди відображаються числові індикатори, які вказують на кількість нових карток, карток у процесі вивчення та тих, що потребують повторення сьогодні. Основна логіка системи побудована на математично обґрунтованому алгоритмі (наприклад, SM-2), який аналізує самооцінку користувача після перегляду картки та автоматично розраховує оптимальний часовий інтервал до її наступного показу. Це забезпечує високу ефективність запам'ятовування лексики в довгостроковій перспективі.

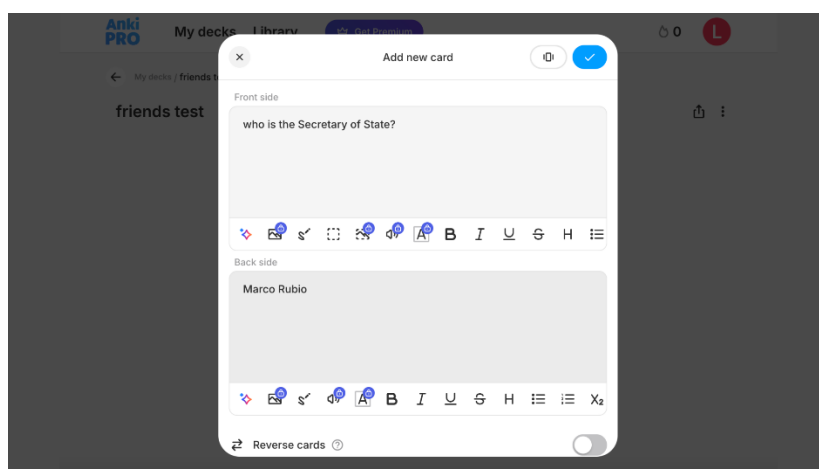


Рис. 1.4 – Інтерфейс створення нової картки в Anki

Ключовим функціональним вузлом додатку є модуль додавання та редагування контенту. На сторінці створення нової картки користувач отримує доступ до розширеного інструментарію. Інтерфейс дозволяє не лише заповнювати базові текстові поля (лицьова та зворотна сторони), але й додавати теги для категоризації, інтегрувати медіафайли (аудіо з вимовою, зображення) та навіть використовувати HTML/CSS для кастомізації зовнішнього вигляду картки.

З інженерної точки зору платформа відрізняється безпрецедентною гнучкістю та розвиненою системою синхронізації локальних баз даних із хмарним сервером. Проте, попри потужну алгоритмічну базу, архітектура Anki має суттєві недоліки з точки зору користувацького досвіду (UX). Відсутність вбудованої гейміфікації, перевантажений налаштуваннями інтерфейс.

Платформа «Busuu» є ще одним вагомим представником на ринку мобільних освітніх додатків, який пропонує комплексний підхід, поєднуючи структуроване академічне навчання з активною соціальною взаємодією. На відміну від суто ігрової моделі Duolingo або вузькоспеціалізованого утилітарного підходу Anki, Busuu будує свою методологію на основі Загальноєвропейських рекомендацій з мовної освіти (CEFR) [7], пропонуючи збалансований розвиток лексики, граматики та розмовної практики.



Рис. 1.5 – Головний екран та структура навчального шляху в додатку Busuu

На головному екрані застосунку реалізовано послідовну архітектуру подачі матеріалу у вигляді хронологічного таймлайну. Інтерфейс чітко сегментований на рівні (від A1 до B2/C1) та тематичні розділи. Кожен такий розділ інкапсулює

					ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
						16
Змн.	Арк.	№ докум.	Підпис	Дата		

у собі комплексні заняття: вивчення нових слів, граматичні пояснення та практику аудіювання з використанням відеоматеріалів від реальних носіїв мови. З точки зору UX/UI дизайну, додаток підтримує баланс між інформативністю та візуальним навантаженням: користувач має чітке розуміння свого місцезнаходження у навчальному плані. Окрім цього, система використовує багаторівневу гейміфікацію (Layered gamification), де щоденні цілі та індикатори безперервного навчання інтегровані в інтерфейс без надмірної перевантаженості.

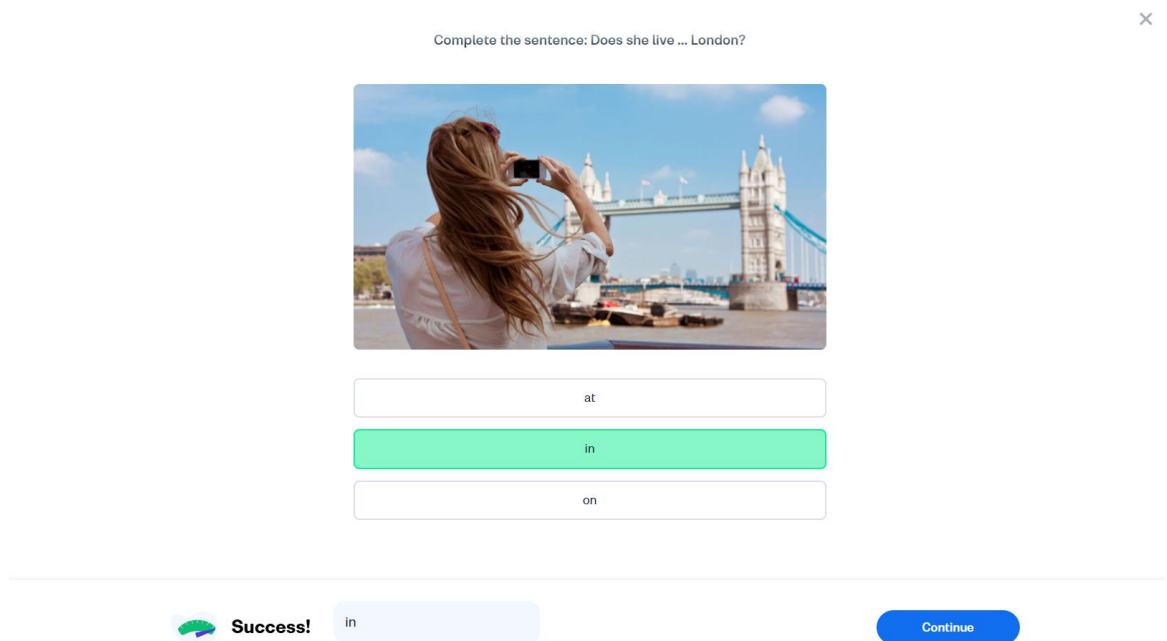


Рис. 1.6 – Інтерфейс модуля соціальної взаємодії (Community) та відеовправ

Унікальним функціональним вузлом архітектури Busuu є модуль спільноти (Community tab) та система взаємного оцінювання (Peer-to-Peer Review). Замість виключної опори на автоматизовані скрипти перевірки, платформа дозволяє користувачам надсилати свої письмові та усні відповіді на перевірку реальним носіям мови. З інженерної точки зору реалізація такого функціоналу вимагає розробки складної бекенд-інфраструктури для асинхронної маршрутизації завдань між мільйонами користувачів. Сучасні оновлення платформи також використовують гібридний підхід: інтеграцію AI-алгоритмів для миттєвого аналізу вимови [8] паралельно із соціальним зворотним зв'язком.

Загалом архітектура Busuu ефективно вирішує проблему ізолюваності учня, додаючи соціальний контекст та автентичні відеоматеріали, що імітують реальне спілкування. Однак система має і технічні та функціональні недоліки. Зокрема, глибока інтеграція високоякісного відеоконтенту вимагає стабільного та швидкого інтернет-з'єднання, а офлайн-режим та розширені інструменти повторення лексики приховані за жорстким монетизаційним бар'єром. Крім того, тривалий процес початкової адаптації користувача з обов'язковим тестуванням створює додаткові бар'єри під час першого запуску застосунку.

Для обґрунтування функціональних вимог до розроблюваного застосунку, порівняльні критерії зведено у таблицю 1.1.

Таблиця 1.1

Порівняння інформаційних систем

Критерії порівняння	Duolingo	Anki	Busuu	Додаток, що розробляється
Гейміфікація та ігрові механіки	Так	Ні	Так	Так
Створення власних наборів слів	Ні	Так	Ні	Так
Алгоритм інтервальних повторень	Так	Так	Так	Так
Офлайн-режим роботи	Обмежено	Так	За підпискою	Так
Тип інтерфейсу	Жорстко лінійний	Картковий	Послідовний таймлайн	Модульний (адаптивний)

Архітектурний підхід	Клієнт-сервер	Локальна БД	Клієнт-сервер	MVVM + Cloud Sync
Інтерактивні вправи (не лише картки)	Так	Ні	Так	Так
Можливість імпорту зовнішніх словників	Ні	Так	Ні	Так

Порівняльний аналіз програмних аналогів свідчить, що існуючі рішення часто мають суттєві обмеження: або пропонують занадто жорстку лінійну структуру контенту із прихованим за платною підпискою ключовим функціоналом (Duolingo, Busuu), або мають перевантажений застарілий інтерфейс та надто складну логіку налаштування (Anki). Жодна з проаналізованих платформ не забезпечує ідеального балансу між простотою ігрового мобільного додатка та гнучкістю професійного інструмента для створення власних освітніх траєкторій.

Розроблювана система вирішує ці проблеми, поєднуючи в собі сучасний архітектурний підхід (MVVM) для забезпечення високої швидкодії та надійної синхронізації з інтуїтивно зрозумілим адаптивним інтерфейсом. Гнучкість контенту дозволить користувачу самостійно керувати базою знань та працювати з аудіоматеріалами, що у комплексі з алгоритмами інтенсивного запам'ятовування робить застосунок найбільш ефективним інструментом для автоматизації навчального процесу та комфортного опанування іноземної мови.

1.2 Функціональні та нефункціональні вимоги до системи

Визначення цільової аудиторії та аналіз основних сценаріїв використання системи є фундаментом для проєктування архітектури мобільного застосунку. Розуміння потреб кінцевих користувачів дозволяє оптимізувати мобільний інтерфейс (UI), забезпечити позитивний досвід взаємодії (UX) та сфокусувати зусилля на розробці найбільш затребуваного функціоналу. Аналіз сегмента

EdTech дозволяє виділити три основні групи цільової аудиторії розроблюваної платформи, кожна з яких має власні болі та специфічні вимоги до програмного продукту.

Першим сегментом є активні здобувачі освіти, зокрема студенти та школярі. Це користувачі, для яких критично важливою є швидкість засвоєння матеріалу в умовах обмеженого часу. Їхня основна проблематика полягає у складності утримання великих обсягів лексики без систематичного повторення. Мобільний додаток вирішує це завдання через механізми гейміфікації та своєчасних push-сповіщень, надаючи доступ до інтерактивних карток та коротких навчальних сесій, що дозволяє легко інтегрувати процес вивчення мови у повсякденний графік.

Другий сегмент аудиторії фахівці, що вивчають професійну лексику (ESP – English for Specific Purposes). Це професіонали, такі як розробники, лікарі чи юристи, яким не потрібні загальні курси, а необхідна гнучкість у формуванні власної бази знань. Головною проблемою існуючих аналогів для них є закритість контенту. Розроблюваний застосунок пропонує потужний інструментарій для кастомізації, де користувач може самостійно створювати тематичні модулі, імпортувати власні списки слів та налаштовувати пріоритетність вивчення специфічних термінів.

Третій сегмент складають адміністратори та розробники контенту, які відповідають за наповнення глобальних баз даних та підтримку актуальності навчальних матеріалів. Для цієї групи система реалізує принцип централізованого джерела істини (Single Source of Truth). Це означає, що будь-які виправлення у словниковій базі на сервері автоматично синхронізуються з усіма мобільними клієнтами, гарантуючи достовірність та цілісність освітнього контенту.

На основі профілів цільової аудиторії сформовано базові сценарії взаємодії користувачів із системою, які підтверджують здатність платформи покривати різні потреби. Під час щоденного використання реалізується сценарій адаптивного тренування (Daily Review). Користувач запускає застосунок, і

					ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
						20
Змн.	Арк.	№ докум.	Підпис	Дата		

система на основі алгоритму інтервальних повторень автоматично формує чергу слів, які потребують закріплення. Цей процес охоплює проходження серії інтерактивних вправ, таких як вибір перекладу, аудіювання та написання. Після завершення сесії мобільний клієнт реактивно оновлює локальну базу даних, перераховуючи дату наступного повторення для кожної лексичної одиниці, та синхронізує поточний прогрес із хмарним сервером.

Інший характер взаємодії розкривається у сценарії створення персоналізованого модуля. Користувач ініціює створення нової групи слів, вводячи лексичні одиниці, а система, використовуючи інтеграцію зі сторонніми API (наприклад, Oxford Dictionary або Google Translate), автоматично пропонує релевантні варіанти перекладу та транскрипції. Збереження такого модуля генерує оптимізовану JSON-схему, роблячи його доступним для подальшого вивчення навіть у режимі офлайн. Крім того, для відстеження ефективності навчання користувач взаємодіє з розділом статистики, де система будує графіки на основі накопичених даних про успішність тестувань. За потреби реалізується сценарій експорту словникового запасу у форматах CSV або JSON для використання в інших освітніх інструментах, що забезпечує відкритість та мобільність зібраних даних.

Функціонал розроблюваного програмного комплексу базується на чіткому переліку завдань, виконання яких забезпечує задоволення потреб кінцевого користувача. Це дозволяє побудувати надійне середовище, де всі модулі системи працюють узгоджено для забезпечення безперебійного освітнього процесу. Архітектура мобільного застосунку спирається на систему управління доступом на основі ролей. Такий підхід гарантує безпеку даних, надійну авторизацію та гнучкість при взаємодії клієнтської частини з бекендом через API. Встановлені правила формують чітке розмежування прав і охоплюють повний життєвий цикл роботи з освітнім контентом: від етапу завантаження та адміністрування навчальних аудіоматеріалів до їх безпосереднього прослуховування та збереження персональної статистики. Детальний перелік можливостей для кожної ролі наведено у таблиці 1.2.

					ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
						21
Змн.	Арк.	№ докум.	Підпис	Дата		

Функціональні вимоги до мобільного застосунку

Роль користувача	Функціональні можливості у системі
Гість (неавторизований користувач)	Перегляд ознайомчого екрана (Onboarding). Ознайомлення з доступними мовними курсами. Реєстрація нового акаунта, авторизація через Email/Password або OAuth (Google/Apple).
Користувач (Учень)	Управління профілем та налаштування цілей навчання. Створення персональних словникових модулів. Проходження інтерактивних уроків та тестів. Використання режиму інтервальних повторень.
Контент-менеджер	Управління глобальною бібліотекою слів та фраз (CRUD). Модерація готових навчальних планів. Завантаження медіафайлів (аудіо-вимова, ілюстрації). Формування шаблонів вправ та тестів через адміністративну панель.
Адміністратор	Повний контроль над системою та базою користувачів. Призначення ролей. Аналітика загальної активності аудиторії. Налаштування параметрів безпеки та керування API-ключами зовнішніх сервісів перекладу.

На відміну від функціональних вимог, нефункціональні описують атрибути якості програмної системи, тобто те, як добре, швидко та безпечно вона повинна виконувати свої функції [9]. Для забезпечення високого рівня якості розробленого мобільного застосунку визначено чіткі інженерні метрики.

Вимоги до продуктивності та швидкодії визначають швидкість реакції системи на дії користувача в умовах мобільного середовища. Оскільки застосунок орієнтований на використання "на ходу", критичним є час холодного запуску, який не повинен перевищувати двох секунд. Час відгуку інтерфейсу на дії користувача (кліки, перемикання екранів) має бути миттєвим (до 100 мс).

					ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
						22
Змн.	Арк.	№ докум.	Підпис	Дата		

Рендеринг анімацій у навчальних модулях повинен відбуватися стабільно при 60 кадрах на секунду для забезпечення візуальної плавності UX на пристроях з різною потужністю.

Вимоги до надійності та відмовостійкості спираються на необхідність безперебійного доступу до матеріалів. Однією з ключових вимог є підтримка повноцінного офлайн-режиму, що дозволяє проходити завантажені уроки без доступу до мережі з подальшою синхронізацією результатів при відновленні з'єднання. Для економії мобільного трафіку та оптимізації заряду акумулятора передача даних має здійснюватися у стисненому форматі JSON, а кешування медіаконтенту є пріоритетним.

Вимоги до безпеки є критично важливими через специфіку роботи з персональними даними. Система повинна гарантувати цілісність даних користувача при раптовому закритті застосунку або втраті зв'язку. Для аутентифікації мають використовуватися безпечні токени (JWT), а збереження персональної інформації в локальній базі даних має відповідати стандартам шифрування на пристрої. Архітектура також повинна підтримувати масштабованість для легкого додавання нових мовних локалізацій завдяки патерну MVVM.

1.3 Вибір і обґрунтування архітектурного рішення

Ефективність мобільного застосунку для навчання критично залежить від надійної внутрішньої архітектури, швидкості доступу до лінгвістичних даних та цілісності збереження прогресу користувача. Для визначення оптимальної структури організації програмного коду проведемо аналіз архітектурних підходів. При проєктуванні мобільних систем часто використовуються класичні патерни, проте вони не завжди відповідають сучасним вимогам масштабування.

Запропонована Робертом Мартіном концепція Clean Architecture (Чиста архітектура) [10] ґрунтується на абсолютній незалежності ядра (доменної логіки) від інфраструктури, баз даних та зовнішніх фреймворків.

					ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
						23
Змн.	Арк.	№ докум.	Підпис	Дата		

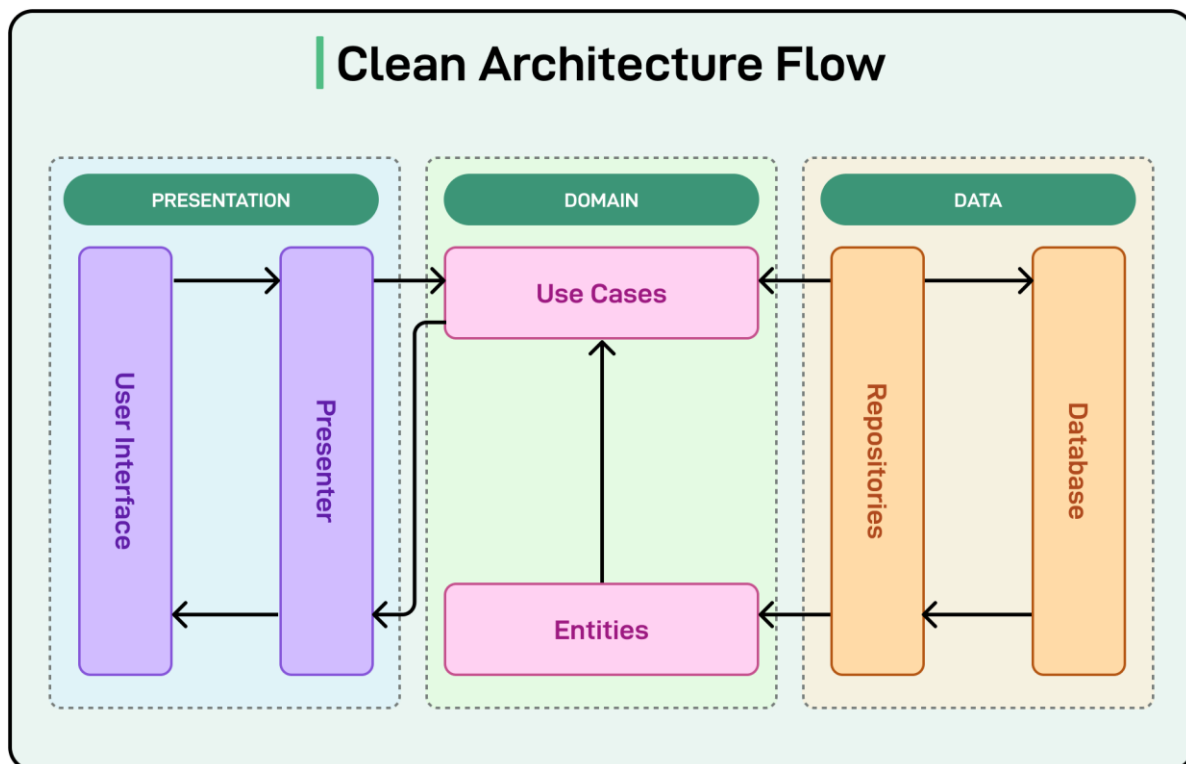


Рис. 1.7 – Концепція Clean Architecture

У цій архітектурі всі залежності спрямовані строго до центру внутрішній шар містить ключові бізнес-правила (Entities), далі йдуть сценарії використання (Use Cases), а зовнішні шари відповідають за інтерфейс користувача та взаємодію з мережею чи локальним сховищем. Для розроблюваного мобільного застосунку вибір Clean Architecture є найбільш технічно обґрунтованим рішенням. Це означає, що ядро системи – складні алгоритми інтервальних повторень та управління словниковим запасом – буде повністю ізольоване від візуальної частини. На рівні представлення (Presentation Layer) цей підхід доцільно поєднати з патерном MVVM (Model-View-ViewModel) [11], що дозволить реактивно оновлювати інтерфейс при зміні стану бази даних, забезпечуючи високу зв'язність коду та легкість покриття системи модульними тестами.

Логічним продовженням архітектурного проектування є вибір методів зберігання динамічних структур даних. Архітектурний виклик полягає у необхідності забезпечити сувору цілісність для словникових баз та гнучкість для збереження персоналізованих навчальних планів. Розгляд класичного патерну EAV (Entity-Attribute-Value) [12] у реляційних базах, де кожен атрибут слова зберігається в окремих рядках, показав, що для мобільного пристрою це є

критичним антипатерном. Велика кількість операцій об'єднання (JOIN) при спробі завантажити один урок створює надмірне навантаження на процесор смартфона, що призводить до затримок інтерфейсу та швидкого розряджання акумулятора. З іншого боку, використання виключно документо-орієнтованих баз даних (NoSQL) [13] дозволяє швидко читати урок як єдиний JSON-об'єкт, проте створює ризики для консистентності при синхронізації з хмарою через відсутність суворих реляційних зв'язків.

Для розв'язання цього компромісу у проєкті обрано гібридну комбіновану модель зберігання. На рівні мобільного клієнта використовується легковагова реляційна база SQLite [14], що є стандартом для спеціальності 121 та забезпечує стабільну роботу і оптимізацію в офлайн-режимі. На рівні сервера застосовується повноцінна реляційна СУБД (PostgreSQL) [15], яка гарантує ACID-транзакційність для профілів користувачів. Конфігурації складних інтерактивних вправ зберігаються у форматі JSON всередині реляційних таблиць. Таке архітектурне рішення дозволяє додавати нові типи ігрових механік без зміни схеми БД (Zero-migration), виконувати дельта-синхронізацію при відновленні мережі та отримувати повну структуру навчального модуля одним мережевим запитом.

Для забезпечення високої доступності, синхронізації даних у реальному часі та безпеки мобільного застосунку було обрано сучасну хмарну інфраструктуру. Це дозволяє делегувати підтримку серверної частини надійним провайдерам і зосередитися на бізнес-логіці освітнього процесу.

В якості базової хмарної платформи інтегровано екосистему Firebase від Google Cloud [16]. Сервіс Firebase Authentication [17] використовується для безпечної реєстрації та входу користувачів із підтримкою JWT-токенів та авторизації через соціальні мережі. Критично важливим інструментом для мобільного додатка є Firebase Cloud Messaging (FCM), який реалізує систему push-сповіщень для нагадування учневі про час інтервальних повторень, що суттєво підвищує рівень утримання (retention) аудиторії. Для оперативного

реагування на збої на різних моделях смартфонів підключено систему моніторингу помилок у реальному часі Firebase Crashlytics.

Оскільки мобільний пристрій має жорстко обмежену пам'ять, зберігання важких медіаданих (аудіофайлів з вимовою та ілюстрацій) безпосередньо в пакеті додатка є недоцільним. Для управління навчальним контентом інтегровано спеціалізований хмарний сервіс Cloudinary [18], який автоматично оптимізує аудіо та зображення під пропускну здатність мережі користувача, гарантуючи швидке завантаження уроків навіть при слабкому мобільному з'єднанні.

Хостинг серверної логіки (API) та розгортання глобальної бази даних реалізується на базі платформи Render [19]. Вона забезпечує розгортання контейнеризованого бекенду, автоматичне налаштування HTTPS-з'єднань та масштабування обчислювальних ресурсів під час пікових навантажень. Для захисту API-ендпоінтів від DDoS-атак та автоматизованих парсерів застосовується інфраструктура Cloudflare [20], яка виступає захисним проксі-шаром та забезпечує надійне шифрування трафіку між смартфоном і хмарию. На відміну від вебдодатків, дистрибуція мобільного продукту передбачає публікацію у цифрових магазинах, тому для автоматизації процесів тестування та збірки інсталяційних файлів (Continuous Integration / Continuous Deployment) застосовуються сучасні інструменти CI/CD, зокрема GitHub Actions [21].

1.4 Вибір стеку технологій та апаратного забезпечення

Вибір інструментальних засобів розробки є фундаментальним етапом проєктування, який безпосередньо визначає швидкість створення програмного продукту, його надійність, можливості подальшого масштабування та зручність підтримки. У цьому підрозділі послідовно розглянуто та обґрунтовано вибір технологій для клієнтської та серверної частин платформи, допоміжних середовищ розробки, а також визначено вимоги до апаратного забезпечення.

Для реалізації мобільного додатка обрано технологічний стек, що повністю базується на екосистемі JavaScript та її статично типізованого надмножини TypeScript [22]. Використання єдиної мовної парадигми для фронтенду та бекенду дозволяє суттєво оптимізувати процес розробки завдяки ефективному

					ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
						26
Змн.	Арк.	№ докум.	Підпис	Дата		

перевикористанню спільних типів даних та бізнес-логіки, що особливо важливо при роботі зі складними освітніми структурами та алгоритмами повторення.

Клієнтська частина системи є найбільш складною складовою мобільної платформи, оскільки саме вона реалізує логіку взаємодії користувача з навчальним контентом. В якості основного фреймворку було обрано React Native [23]. Цей вибір є ідеальним для розробки освітніх застосунків з кількох вагомих причин. По-перше, він базується на суворому компонентному підході, де кожен елемент інтерфейсу (картка слова, блок тестування, прогрес-бар) інкапсулюється в окремий модуль, що спрощує масштабування. По-друге, на відміну від вебрішень, React Native взаємодіє безпосередньо з нативними компонентами операційних систем iOS та Android через міст (Bridge). Це забезпечує стабільну частоту оновлення екрана на рівні 60 кадрів на секунду, плавність анімацій та миттєвий відгук на дії користувача, що гарантує позитивний досвід (Native UX).

Серверна частина відповідає за обробку API-запитів, виконання бізнес-логіки системи, автентифікацію та взаємодію з глобальною базою даних. В якості програмної платформи було обрано середовище Node.js [24]. Оскільки мобільний застосунок передбачає одночасну роботу багатьох користувачів, які надсилають на сервер запити для синхронізації свого прогресу, бекенд повинен ефективно витримувати високе навантаження. Асинхронна та подієво-орієнтована модель Node.js дозволяє серверу не блокувати потік виконання під час очікування запису в базу даних. В якості вебфреймворку використовується Express.js [25], який надає гнучкі інструменти для побудови архітектури REST API, швидкої маршрутизації та легкої інтеграції механізмів автентифікації.

Для забезпечення високої продуктивності процесу написання коду було підібрано відповідний інструментарій розробника. Осковим інтегрованим середовищем виступає Visual Studio Code [26]. Цей вибір зумовлений його легковажністю, широкою екосистемою розширень для налагодження кросплатформених застосунків та інтеграцією з лінтерами для перевірки синтаксису TypeScript. Для моделювання логічної архітектури програмного забезпечення обрано інструмент PlantUML [27]. Діаграми у ньому не малюються

					ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
						27
Змн.	Арк.	№ докум.	Підпис	Дата		

графічно, а описуються за допомогою спеціалізованої текстової мови. Це дозволяє зберігати архітектурні моделі у репозиторії системи контролю версій Git [28] разом із вихідним кодом проєкту та ефективно відстежувати історію їх змін.

Апаратні вимоги до робочої станції розробника зумовлені необхідністю одночасного запуску інтегрованого середовища розробки, локального сервера та емуляторів мобільних пристроїв, що потребує багатоядерного процесора та щонайменше 16 ГБ оперативної пам'яті. Для кінцевого користувача вимоги є більш лояльними: сучасний смартфон під управлінням Android або iOS із обсягом оперативної пам'яті від 2 ГБ та вільним місцем на накопичувачі для кешування навчальних медіафайлів. Стабільне підключення до мережі Інтернет є необхідним лише для періодичної синхронізації прогресу, тоді як основна логіка навчання доступна в автономному режимі.

Висновки до першого розділу

У межах першого розділу здійснено комплексне дослідження предметної області та обґрунтовано актуальність розробки мобільного застосунку для вивчення іноземних мов. Аналіз популярних на ринку EdTech-рішень продемонстрував необхідність створення продукту, який би гармонійно поєднував у собі сучасні методики інтенсивного запам'ятовування (інтервальні повторення) з високим рівнем гнучкості при формуванні персоналізованого навчального контенту, чого не вистачає існуючим закритим або надто утилітарним системам.

Вивчення потреб цільової аудиторії дозволило чітко окреслити сегменти користувачів та формалізувати ключові сценарії взаємодії. На основі отриманих результатів було обґрунтовано вибір сучасного технологічного стеку, орієнтованого на кросплатформену мобільну розробку. Доведено доцільність використання фреймворку React Native у поєднанні з мовою TypeScript, що забезпечує стабільну продуктивність інтерфейсу та надійність програмного коду. Для розв'язання проблеми ефективного зберігання даних обрано гібридну архітектурну модель, яка передбачає використання локальної бази даних SQLite

					ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
						28
Змн.	Арк.	№ докум.	Підпис	Дата		

на пристрої для забезпечення офлайн-доступу та реляційної СУБД PostgreSQL на серверній частині для глобальної синхронізації.

Окрім цього, у розділі було детально сформульовано функціональні вимоги до системи, що базуються на рольовій моделі доступу, та визначено критичні нефункціональні метрики. Особливу увагу приділено параметрам швидкодії мобільного клієнта, безпеці передачі даних через хмарну інфраструктуру Firebase та стабільності роботи в умовах обмеженого мережевого з'єднання. Сукупність сформованих вимог, проаналізованих архітектурних підходів (Clean Architecture, MVVM) та обраних інструментів створює надійний теоретичний і технічний фундамент для переходу до етапу логічного проектування та програмної реалізації мобільного освітнього сервісу.

Загалом, сукупність сформованих вимог, критично проаналізованих архітектурних патернів (зокрема, Clean Architecture для надійної ізоляції бізнес-логіки та MVVM для реактивного зв'язування даних із графічним інтерфейсом), а також обґрунтований вибір інструментарію створюють надійний, науково та інженерно виважений теоретико-технічний фундамент. Це забезпечує плавний і безризиковий перехід до наступного етапу життєвого циклу розробки програмного забезпечення (SDLC) безпосереднього логічного проектування баз даних та програмної реалізації всіх модулів мобільного освітнього сервісу.

					ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
						29
Змн.	Арк.	№ докум.	Підпис	Дата		

РОЗДІЛ 2. ПРОЄКТУВАННЯ ПРОГРАМНОЇ СИСТЕМИ

2.1 Моделювання варіантів використання та поведінки системи

Use Case Modeling є концептуальним фундаментом для переходу від теоретичного збору вимог до практичного проєктування архітектури кросплатформного комплексу Speaker App. На цьому етапі об'єктно-орієнтованого аналізу визначаються межі системи (мобільний клієнт та вебпанель адміністратора), ідентифікуються ключові дійові особи (актори) та формалізуються прецеденти їхньої взаємодії. Застосування UML-діаграм дає змогу візуалізувати поведінкові аспекти застосунку, трансформуючи розрізнені функціональні вимоги у структуровану логічну схему. Детальний опис кожної ідентифікованої ролі платформи наведено у таблиці 2.1.

Таблиця – 2.1

Ідентифікація акторів системи

Актор	Опис ролі та зона відповідальності
Гість	Неавторизований користувач мобільного застосунку. Має доступ до ознайомчих екранів (Onboarding) та загальної інформації про можливості вивчення мов. Ініціює процедуру реєстрації або входу.
Учень (Авторизований користувач)	Авторизований користувач і основний суб'єкт взаємодії. Наділений повним доступом до навчального середовища: тематичних словників із транскрипцією та аудіо-вимовою, уроків, системи інтервальних повторень. Керує статусом власної Premium-підписки.
Контент-менеджер	Фахівець, що працює через вебпанель адміністратора (React Web Admin). Відповідає за наповнення БД (MongoDB) навчальним контентом: створює категорії, додає слова та конструює уроки.

Адміністратор	Особа, яка забезпечує технічне управління серверною інфраструктурою (Node.js/Express). Контролює політику безпеки, управляє правами доступу, моніторить навантаження та конфігурує інтеграції з API (Stripe, Google OAuth).
---------------	---

Відповідно до розробленої моделі управління доступом, в архітектурі Sreaker App виокремлено чотирьох незалежних акторів із чітко розмежованими зонами відповідальності. Враховуючи кросплатформну природу продукту, усі варіанти використання логічно розподілено на клієнтську частину (мобільний застосунок) та серверну панель адміністрування.

У межах публічного простору мобільного клієнта реалізуються прецеденти ознайомлення із застосунком, а також процеси ідентифікації: реєстрація, безпечна автентифікація (зокрема через інтеграцію з Google OAuth) та відновлення пароля. Ці базові сценарії є універсальними на початковому етапі використання платформи.

Навчальний простір охоплює основний функціонал для активного засвоєння іноземної мови авторизованим Учнем. До цієї групи належать проходження інтерактивних уроків, запуск сесій інтервальних повторень, переклад та прослуховування аудіо-вимови слів у тематичних словниках. Окремо виділяються варіанти використання, пов'язані з налаштуванням профілю, відстеженням персональної статистики прогресу та управлінням доступом до розширених функцій через модуль підписок (інтеграція зі Stripe).

Середовище адміністрування (Web Admin Panel) призначене для управління глобальними активами проєкту Контент-менеджером та Адміністратором. Тут реалізуються сценарії виконання CRUD-операцій над словниковими базами у MongoDB, створення шаблонів для нових уроків та завантаження медіафайлів. Адміністративна частина також включає системні прецеденти: моніторинг роботи бекенду, керування обліковими записами та

					ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
						31
Змн.	Арк.	№ докум.	Підпис	Дата		

конфігурацію ключів безпеки для зовнішніх сервісів. Подібна декомпозиція між мобільним клієнтом та вебпанеллю гарантує цілісність системи та дозволяє гнучко масштабувати проєкт.

На основі виявлених акторів та формалізованих прецедентів побудовано розширену діаграму варіантів використання системи (рис. 2.2).

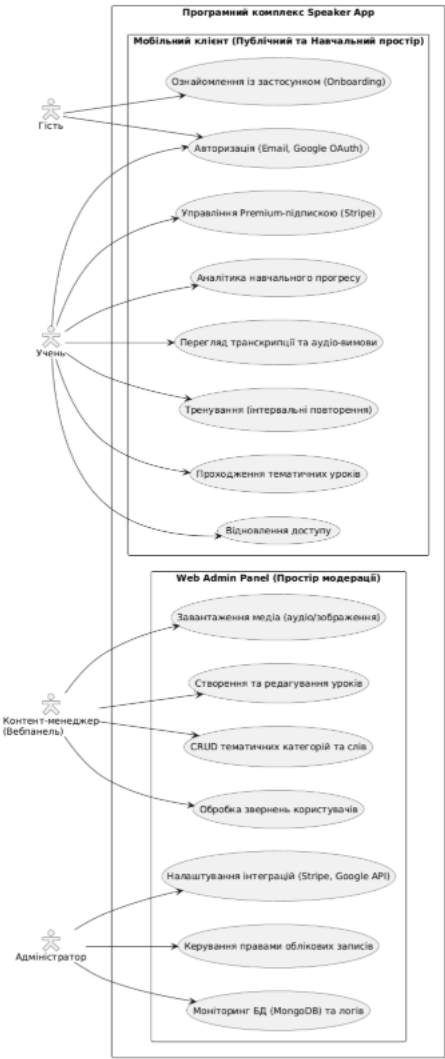


Рис. 2.2 – Діаграма варіантів використання системи

Подана візуалізація чітко розмежовує зони відповідальності: актори клієнтського рівня (Гість, Учень) взаємодіють виключно з інтерфейсом мобільного застосунку для навчання та управління підпискою. Водночас службові ролі наділені ексклюзивними правами доступу до вебпанелі для модерації словників та технічного налаштування бекенду. Спроектована модель слугує надійним фундаментом для подальшого формування об'єктної структури системи та реляційної схеми бази даних.

З метою підготовки до розробки алгоритмів серверного API та мобільного клієнта було складено специфікації для ключових прецедентів. Специфікація деталізує кожен крок взаємодії актора із системою, визначає необхідні передумови та описує альтернативні сценарії обробки помилок (наприклад, збої при оплаті через Stripe або відхилення токена Google OAuth). Це дає змогу зафіксувати мінімізуючи ризики при подальшому програмуванні. У таблиці 2.2 наведено специфікацію базового варіанта використання.

Таблиця 2.2

Специфікація прецеденту "Реєстрація та Авторизація"

Атрибут	Опис
Назва прецеденту	Реєстрація та Авторизація
Актор	Гість
Мета	Отримати персоналізований доступ до тематичних словників, інтерактивних уроків та аналітики прогресу в системі Speaker.
Передумови	Користувач відкрив мобільний застосунок Speaker та має активне підключення до мережі інтернет.
Основний сценарій (Успішний)	- Гість обирає метод реєстрації або входу (стандартно через Email або за допомогою Google OAuth). - Система відображає відповідний інтерфейс. - Гість вводить облікові дані або підтверджує доступ через Google-акаунт. - Клієнтська частина проводить первинну валідацію. - Сервер (Node.js) перевіряє дані в базі даних (MongoDB) або верифікує токен провайдера і генерує безпечний JWT-токен доступу. - Застосунок зберігає сесію та відкриває доступ до головного навчального екрана.

Альтернативний сценарій (Помилки)	4а. Некоректний формат даних: система виводить підказку щодо виправлення помилок (наприклад, невалідний email). 5а. Помилка на боці сервера (невірний пароль, блокування акаунту або відхилення Google OAuth): сервер повертає помилку автентифікації (HTTP 401), мобільний клієнт пропонує відновити пароль або змінити дані.
Постумови	Користувач переходить у статус авторизованого «Учня», активується механізм синхронізації його навчального прогресу та статусу підписки (Stripe) із хмарним сховищем.

Ключовим функціоналом розроблюваного застосунку Speaker є можливість розширення базового контенту шляхом формування індивідуальної освітньої траєкторії. У таблиці 2.3 наведено інженерну специфікацію прецеденту «Формування персоналізованого навчального модуля», який є найбільш складним з погляду реалізації клієнтської логіки (React Native) та синхронізації станів з бекендом.

Таблиця 2.3

Специфікація прецеденту «Формування персоналізованого навчального модуля»

Атрибут	Опис
Назва прецеденту	Формування персоналізованого навчального модуля(словника)
Актор	Учень
Мета	Створити власну підбірку навчального контенту (набір слів, перекладів, транскрипцій) для цільового вивчення.

					ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
						34
Змн.	Арк.	№ докум.	Підпис	Дата		

Передумови	Користувач (Учень) авторизований у мобільному застосунку Speaker та перейшов до розділу створення персональних словників.
Основний сценарій (Успішний)	1. Учень ініціює створення нового модуля. 2. Мобільний клієнт формує базову структуру об'єкта для нового-словника. 3. Користувач додає лексичні одиниці, використовуючи функцію пошуку або ручного введення (слова, переклади, аудіо-вимова). 4. Система автоматично інтегрує додані слова в алгоритм інтервальних повторень та генерує шаблони вправ. 5. Користувач зберігає створений модуль. 6. Застосунок серіалізує дані у формат JSON та відправляє запит на серверний REST API. 7. Сервер валідує отриману структуру та створює відповідний документ у колекції MongoDB, прив'язуючи його до ідентифікатора користувача.
Альтернативний сценарій (Помилки)	6а. Відсутність мережі: застосунок перехоплює помилку, сповіщає користувача та зберігає зміни в локальному кеші (наприклад, через AsyncStorage) для подальшої фонові синхронізації. 7а. Помилка валідації API: сервер відхиляє запит (HTTP 400), система вказує на некоректно заповнені поля.
Постумови	Оновлена структура персонального модуля збережена в базі даних та доступна для вивчення на будь-якому пристрої користувача.

Окрім клієнтського функціоналу, стабільність інформаційної системи Sreaker підтримується адміністративними процесами модерації та контролю якості контенту. Специфікація прецеденту модерації (табл. 2.4) демонструє логіку управління станами об'єктів у глобальній базі даних через вебпанель адміністратора.

Таблиця 2.4

Специфікація прецеденту «Призупинення доступу до публічного модуля»

Атрибут	Опис
Назва прецеденту	Призупинення доступу до навчального модуля (категорії/уроку)
Актор	Контент-менеджер, Адміністратор
Мета	Обмежити доступ користувачів до публічного навчального контенту, який містить грубі помилки або потребує актуалізації.
Передумови	Модератор успішно авторизований у вебпанелі (React Web Admin) та перейшов до розділу управління контентом.
Основний сценарій (Успішний)	1. Модератор знаходить модуль через систему фільтрації за ідентифікатором або автором. 2. Обирає дію «Деактивувати». 3. Система вимагає вказати причину блокування. 4. Після підтвердження сервер змінює статус об'єкта в БД з "active" на "suspended". 5. Система автоматично надсилає повідомлення автору з роз'ясненнями та інструкціями щодо виправлення.
Альтернативний сценарій (Помилки)	4а. Спроба блокування системного модуля: система перевіряє права доступу (RBAC) та відхиляє операцію для захищених сутностей.

Постумови	При спробі завантаження модуля мобільні клієнти отримують статус доступу «заборонено», а контент зникає із загального каталогу.
-----------	---

Розроблені специфікації прецедентів визначають чіткі архітектурні контракти між компонентами системи *Speaker* (мобільним клієнтом, вебпанеллю та сервером). Вони демонструють, що кожна дія актора супроводжується комплексною серверною логікою: безпечною аутентифікацією, валідацією прав доступу (RBAC), управлінням підписками, інтеграцією із зовнішніми API (Stripe, Google OAuth) та синхронізацією станів у базі даних (MongoDB). Ця деталізація слугує фундаментальною базою для наступного етапу – проєктування об'єктно-орієнтованої структури класів та маршрутів API бекенду.

2.2 Проєктування логічної структури бази даних

Для забезпечення надійного зберігання інформації та підтримки транзакційної цілісності освітніх даних застосунку *Speaker* було обрано реляційну систему управління базами даних [6]. У межах мобільної розробки це реалізується через використання локальної бази SQLite на боці клієнта [14] та PostgreSQL на серверному бекенді Node.js [4]. Такий підхід гарантує виконання вимог ACID, що є критично важливим для збереження прогресу навчання, словникових запасів та історії успішності користувачів, де будь-яка втрата даних може негативно вплинути на освітній процес.

Зважаючи на специфіку підсистеми персоналізації навчання, де структура навчального модуля може бути максимально динамічною (різна кількість вправ, типів медіаконтенту та варіативність тестів), для зберігання складних конфігурацій використано нативний тип даних JSON у PostgreSQL [15]. Це дозволяє ідеально поєднати надійність строгих реляційних зв'язків для системних сутностей із гнучкістю документо-орієнтованого підходу [13] для опису логіки інтерактивних завдань. Таке архітектурне рішення мінімізує

необхідність складних міграцій бази даних при додаванні нових типів ігрових механік та дозволяє уникнути антипатерну EAV (Entity-Attribute-Value) [12].

Логічна структура бази даних застосунку Speaker охоплює систему взаємопов'язаних таблиць, що забезпечують управління користувачами за рольовою моделлю (RBAC), лінгвістичними базами (слова, фрази, переклади), медіатекою та результатами тестувань. Для уникнення надлишковості інформації та підтримки цілісності даних більшість таблиць реляційної моделі було нормалізовано до третьої нормальної форми (3НФ). Це дозволяє ефективно масштабувати систему, додаючи нові мови або розширюючи статистичні показники без дублювання записів.

Наведений графічний концепт у вигляді ER-діаграми ілюструє ключові сутності системи, їхні атрибути, первинні та зовнішні ключі, а також типи логічних зв'язків. Зокрема, виділяються зв'язки «один-до-багатьох» між користувачами та їхніми персональними словниками, а також складні відношення між лексичними одиницями та мультимедійним контентом. Спроектowana схема формує єдиний інформаційний простір, який забезпечує швидку вибірку даних через REST API [2] для мобільного інтерфейсу та надійну синхронізацію між локальним сховищем смартфона і хмарним сервером [19].

Загалом, ретельно спроектована схема формує єдиний, надійно узгоджений інформаційний простір. Оптимізована структура таблиць забезпечує надшвидку вибірку даних, що є критично важливим для мінімізації затримок (latency) та безперебійної роботи мобільного інтерфейсу під час взаємодії із сервером.

Наведений графічний концепт у вигляді концептуальної ER-діаграми (Entity-Relationship) ілюструє ключові предметні сутності системи, їхні внутрішні атрибути, механізми суворої ідентифікації через первинні (Primary Keys) та зовнішні (Foreign Keys) ключі, а також встановлені між ними типи логічних зв'язків. Ця модель слугує не лише візуальним представленням інформаційних потоків, але й гарантує нормалізацію бази даних, запобігаючи виникненню аномалій чи надлишковості інформації.

					ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
						38
Змн.	Арк.	№ докум.	Підпис	Дата		

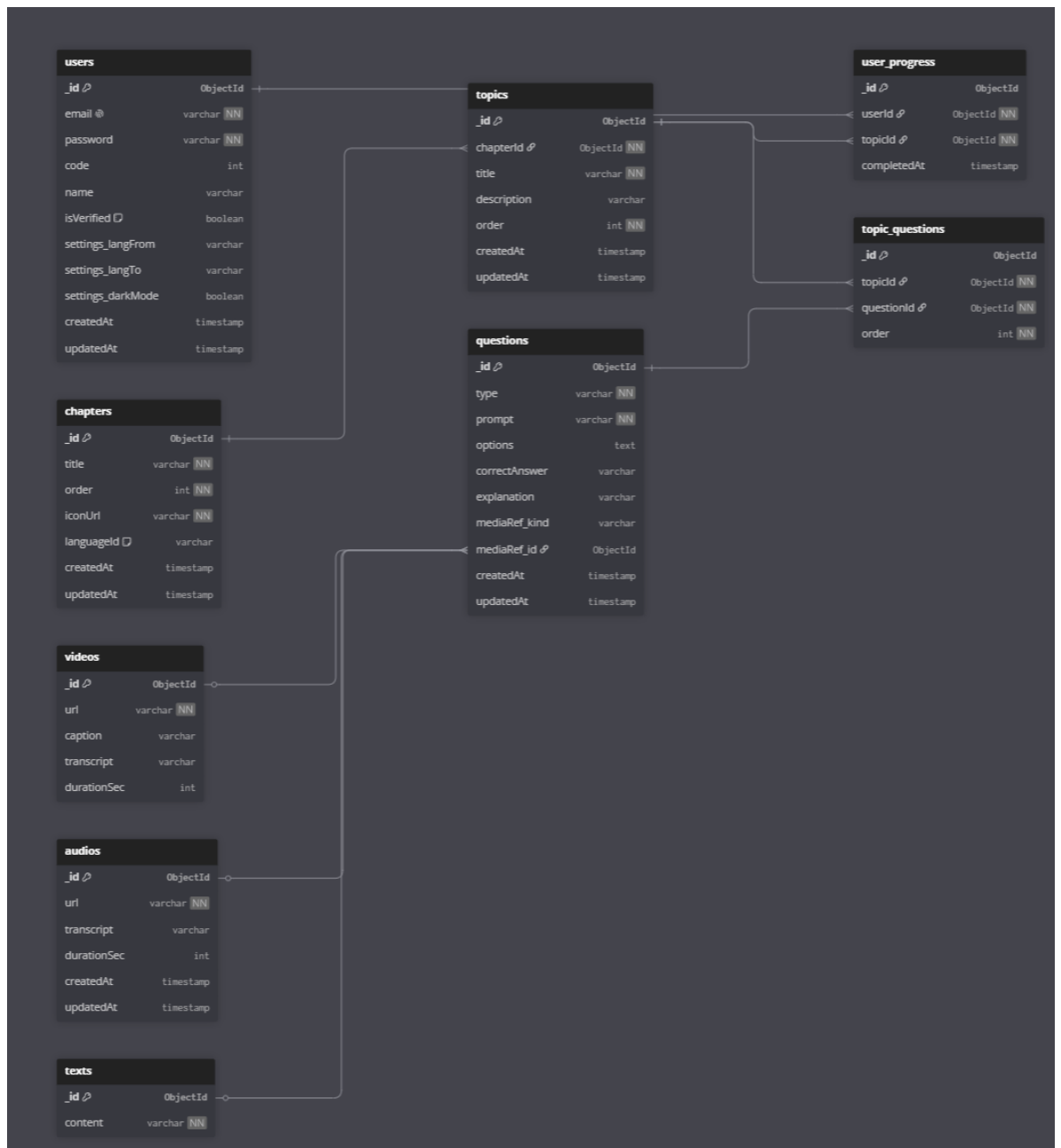


Рис. 2.3 – ER-діаграма реляційної бази даних

Для деталізації зв'язків між сутностями системи та опису архітектури даних було розроблено концептуальну ER-діаграму. Вона відображає ієрархічну структуру системи, де центральне місце займає взаємодія між профілем користувача та його індивідуальним навчальним контентом. Побудова логічних зв'язків базується на принципах цілісності та мінімізації надлишковості, що забезпечує стабільну роботу серверної частини при великих обсягах лінгвістичних даних.

Ключові зв'язки між сутностями системи побудовані за такими правилами. Між таблицями користувачів та їхніми персональними словниками встановлено

зв'язок «один-до-багатьох», що дозволяє кожному учневі формувати необмежену кількість тематичних модулів. При цьому видалення облікового запису ініціює каскадне видалення (Cascade Delete) всіх пов'язаних із ним персональних налаштувань та прогресу, що відповідає сучасним політикам конфіденційності.

Навчальні модулі пов'язані із сутністю вправ та карток слів також через зв'язок «один-до-багатьох». Для гнучкої класифікації матеріалу реалізовано зв'язок «багато-до-багатьох» між модулями та категоріями через транзитну таблицю, що дозволяє користувачам ефективно фільтрувати контент за рівнем складності згідно з рекомендаціями CEFR [7] або за тематикою.

Особливе місце в архітектурі займає модуль відстеження прогресу, який пов'язує результати тестувань із конкретними словами через таблицю статистики вивчення. Це дозволяє зберігати історію взаємодії користувача з кожною лексичною одиницею, фіксувати кількість помилок та дату наступного повторення згідно з алгоритмом інтервальних повторень. У таблиці 2.5 наведено специфікацію сутності, що відповідає за зберігання облікових записів та налаштувань доступу [17].

Таблиця 2.5

Структура таблиці users

Назва поля	Тип даних	Обмеження	Опис
id	INT	PK, AUTO_INCREMENT	Унікальний ідентифікатор користувача.
username	VARCHAR(255)	UNIQUE, NOT NULL	Публічне ім'я (нікнейм) учня в системі.
email	VARCHAR(255)	UNIQUE, NOT NULL	Електронна пошта для авторизації та сповіщень.

продовж. табл. 2.5

password_hash	VARCHAR(255)	NULL	Хешований пароль; може бути порожнім при вході через Google/Apple.
role	ENUM	NOT NULL, DEFAULT 'user'	Роль у системі ('user', 'moderator', 'admin').
language_level	ENUM	NOT NULL, DEFAULT 'A1'	Поточний рівень володіння іноземною мовою.
status	ENUM	NOT NULL, DEFAULT 'active'	Стан акаунта ('active', 'inactive', 'banned').
created_at	TIMESTAMP	DEFAULT CURRENT_TIMESTAMP	Дата створення профілю.

Центральною сутністю модуля управління контентом є навчальний модуль. Таблиця learning_modules містить загальні метадані освітнього блоку, параметри доступу та конфігураційні ключі для персоналізації. Фізична структура цієї таблиці (табл. 2.6) забезпечує гнучке керування видимістю матеріалів та дозволяє інтегрувати модулі у загальну освітню екосистему застосунку Speaker.

Структура таблиці learning_modules

Назва поля	Тип даних	Обмеження	Опис
id	INT	PK, AUTO_INCREMENT	Унікальний ідентифікатор навчального модуля.
user_id	INT	FK, NOT NULL	Посилання на власника (автора) модуля.
slug	VARCHAR(255)	UNIQUE, NOT NULL	Унікальний ідентифікатор для поширення модуля або офлайн-доступу.
title	VARCHAR(255)	NOT NULL	Назва навчального курсу або набору слів.
status	ENUM	NOT NULL, DEFAULT 'private'	Статус доступу ('private', 'public', 'shared').

продовж. табл. 2.6

module_config	JSON	NULL	Глобальні налаштування черговості вправ та логіки навчання у форматі JSON.
sync_key	VARCHAR(255)	NULL	Ключ для верифікації цілісності даних при хмарній синхронізації.

Для зберігання безпосереднього наповнення, яке користувач формує через інтерфейс редактора карток, використовується таблиця vocabulary_cards. Кожен запис у цій таблиці представляє ізольовану лексичну одиницю з набором асоціативних зв'язків.

Таблиця 2.7

Структура таблиці vocabulary_cards

Назва поля	Тип даних	Обмеження	Опис
id	INT	PK, AUTO_INCREMENT	Унікальний ідентифікатор картки.
module_id	INT	FK, NOT NULL	Посилання на модуль, до якого входить слово.
word_type	VARCHAR(50)	NOT NULL	Категорія (наприклад, 'noun', 'verb', 'phrase').

продовж. табл. 2.7

content_data	JSON	NOT NULL	Параметри лінгвістичного наповнення (переклад, приклади, транскрипція).
ui_settings	JSON	NOT NULL	Конфігурація відображення картки (стилі, посилання на медіафайли).

Для забезпечення безпеки, аудиту змін у лінгвістичних базах та контролю за діями модераторів реалізовано підсистему журналювання транзакцій. Структура таблиці audit_logs (табл. 2.8) дозволяє відстежувати історію модифікації критичних даних, що важливо для відновлення станів системи у разі несанкціонованого втручання.

Таблиця 2.8

Структура таблиці audit_logs

Назва поля	Тип даних	Обмеження	Опис
id	BIGINT	PK, AUTO_INCREMENT	Унікальний ідентифікатор запису логу.
user_id	INT	FK, NOT NULL	Ідентифікатор особи, що виконала зміну.
action	VARCHAR(100)	NOT NULL	Тип операції (наприклад, 'EDIT_WORD', 'DELETE_MODULE').

target_table	VARCHAR(50)	NOT NULL	Сутність, яка зазнала змін.
record_id	INT	NOT NULL	Ідентифікатор конкретного об'єкта в базі.
created_at	TIMESTAMP	DEFAULT CURRENT_TIMESTAMP	Точний час виконання операції.

Ключовим архітектурним рішенням платформи є спосіб збереження структури навчального контенту через JSON-схеми. Замість створення надмірної кількості таблиць для кожного нового типу вправи чи формату медіаданих, поля module_config та content_data використовують нативний формат JSON. Це забезпечує гнучкість, характерну для NoSQL-рішень, зберігаючи при цьому всі переваги реляційної цілісності основної бази даних [13]. Такий підхід дозволяє розробнику додавати нові методики навчання (наприклад, алгоритми для аналізу вимови [8]) без необхідності складних маніпуляцій зі схемою бази даних, що значно прискорює цикл розробки та оновлення мобільного застосунку.

Дані у системі зберігаються у формі впорядкованого масиву об'єктів, де кожен елемент відповідає за конкретний функціональний блок або навчальний віджет у межах мобільного інтерфейсу. Такий підхід надає максимальну гнучкість при масштабуванні платформи: впровадження нових типів ігрових механік чи вправ не потребує виконання складних міграцій у реляційній структурі бази даних. Замість цього розробнику достатньо оновити логіку обробки JSON-схеми на боці клієнтського застосунку.

Кожен стандартний вузол навчальної схеми має суворо визначену структуру та містить набір ключових параметрів. Унікальний ідентифікатор блоку забезпечує коректну роботу алгоритмів рендерингу та дозволяє динамічно керувати станом елементів на екрані смартфона. Параметр типу константи визначає, який саме освітній компонент має бути завантажений фабрикою

рендерингу (картка для запам'ятовування, інтерактивний тест або аудіо-модуль). Вкладені об'єкти містять параметри лінгвістичного контенту (вибірки слів, переклади) та візуальні конфігурації.

Застосована архітектура гарантує високу продуктивність системи. Під час завантаження нового навчального модуля сервер виконує лише один атомарний запит до бази даних, отримує повну JSON-конфігурацію та миттєво передає її на мобільний пристрій [25]. Клієнтська частина на базі React Native відмальовує структуру інтерфейсу [1] та паралельно ініціює асинхронні запити на отримання медіафайлів із хмарного сервісу оптимізації (CDN) [18], що забезпечує безперервність користувацького досвіду без затримок та очікувань.

2.3 Проєктування компонентної структури та взаємодії модулів

Діаграма розгортання призначена для моделювання фізичної та інфраструктурної топології системи, відображаючи конфігурацію серверних потужностей та хмарних сервісів. Вона дозволяє оцінити розподіл навантаження між окремими компонентами платформи та деталізувати протоколи їхньої взаємодії [9]. Оскільки застосунок Sreaker проєктується як сучасне хмарне рішення, його архітектура базується на принципах розподіленої інфраструктури, що забезпечує відмовостійкість, масштабованість та швидку доставку навчального контенту [10].

На представленій діаграмі розгортання (рис. 2.4) чітко виділено ключові вузли екосистеми Sreaker. Клієнтський рівень (мобільний пристрій та браузер адміністратора) взаємодіє з бекендом через захищений шлюз [20]. Обробка бізнес-логіки відбувається на хмарній серверній платформі [19], тоді як збереження структурованих даних делеговано керованому сервісу баз даних (DBaaS). Окремі SaaS-рішення, такі як Cloudinary для мультимедійних файлів [18] та зовнішні API (Stripe, Google OAuth), інтегровані в єдину мережу, що дозволяє оптимізувати маршрутизацію запитів та гарантувати стабільну роботу застосунку навіть при високих навантаженнях.

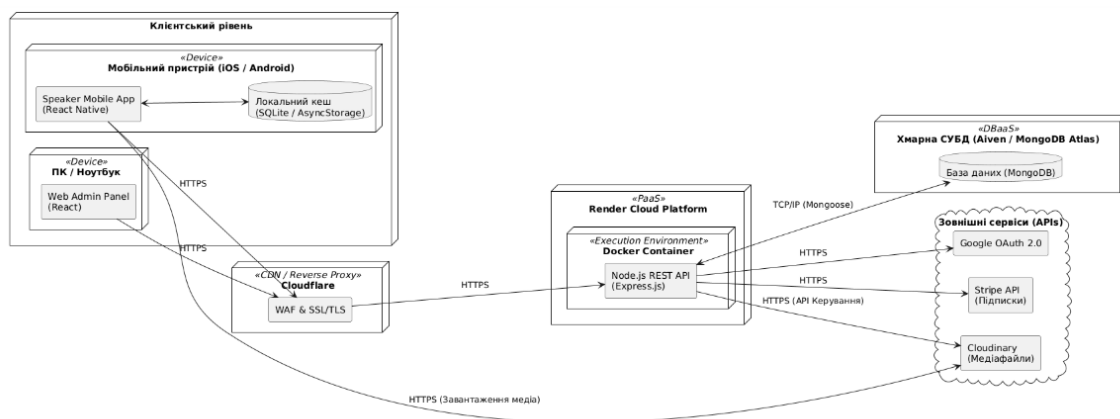


Рис. 2.4 – Діаграма розгортання системи

Архітектурна модель платформи базується на розподіленій хмарній інфраструктурі, що забезпечує стабільну роботу мобільного застосунку та надійну синхронізацію даних. Основним вузлом взаємодії є мобільний пристрій користувача, на якому виконується клієнтський застосунок, розроблений за допомогою фреймворку React Native [1, 23]. Цей компонент відповідає за рендеринг інтерфейсу навчання, управління локальним станом словників (наприклад, через SQLite або AsyncStorage [14]) та забезпечення миттєвої реакції на дії учня незалежно від якості мережевого з'єднання. Окремим клієнтським вузлом виступає вебпанель адміністратора, призначена для зручної модерації освітнього контенту.

Безпеку та маршрутизацію трафіку забезпечує мережа Cloudflare [20], яка виступає захисним бар'єром між клієнтом та серверною частиною. Цей вузол виконує функції реверс-проксі, фільтруючи автоматизований зловмисний трафік (WAF) та забезпечуючи шифрування даних за допомогою SSL/TLS-протоколів. Завдяки Cloudflare запити користувачів безпечно спрямовуються до відповідних хмарних ресурсів бекенду, що гарантує високу доступність системи та захист лінгвістичних баз даних від несанкціонованого доступу та DDoS-атак.

Серверна частина системи розгорнута на платформі Render [19], де функціонує основна бізнес-логіка Speaker у середовищі Node.js [4, 24] з використанням вебфреймворку Express.js [25]. Цей вузол обробляє запити до REST API [2], керує процесами автентифікації через Google OAuth [31], управляє життєвим циклом преміум-підписок за допомогою Stripe [30] та забезпечує

транзакційну взаємодію з базою даних. Хмарна інфраструктура (PaaS) дозволяє серверу автоматично масштабуватися при зростанні активності користувачів, підтримуючи стабільну швидкість обробки даних під час синхронізації прогресу навчання.

Зберігання структурованої інформації та словникових активів реалізовано за допомогою керованої документо-орієнтованої бази даних MongoDB (розгорнутої в хмарі MongoDB Atlas або Aiven) [13, 29]. Вузловий кластер бази даних працює в ізольованому віртуальному середовищі, що унеможливорює прямий зовнішній доступ та гарантує цілісність освітнього контенту. Для роботи з мультимедійними ресурсами, зокрема аудіофайлами з вимовою та ілюстраціями до карток, безпосередньо інтегровано сервіс Clouddinary [18]. Він забезпечує динамічну оптимізацію медіаконтенту (стиснення, зміну форматів) під параметри конкретного мобільного пристрою, що є критично важливим для економії мобільного трафіку та високої швидкодії застосунку.

Сценарій архітектурної взаємодії у системі ініціюється мобільним клієнтом Speaker. Коли користувач запускає застосунок, первинні запити проходять через захисні фільтри Cloudflare [20], які забезпечують безпечне з'єднання та перевірку цілісності сесії. Мобільний клієнт, побудований на базі React Native [1], звертається до хмарного бекенду за маршрутом /api/* для отримання актуальних навчальних планів або синхронізації прогресу вивчення мови. Cloudflare ідентифікує ці API-запити та безпечно проксіює їх безпосередньо на сервер Render [19], де розгорнута основна бізнес-логіка.

Node.js-сервер [4] приймає вхідний запит і маршрутизує його до відповідного функціонального модуля всередині структури модульного моноліта наприклад, до домену «Навчання» або «Словник». Виконуючи необхідні операції, бекенд ініціює оптимізований запит (наприклад, через ODM Mongoose) до керованої документо-орієнтованої бази даних MongoDB [29]. Отримавши результат (наприклад, перелік слів для інтервального повторення), сервер формує структуровану JSON-відповідь [2], яка повертається мобільному клієнту для миттєвого оновлення локального стану та візуалізації на екрані смартфона.

					ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
						48
Змн.	Арк.	№ докум.	Підпис	Дата		

Окремий архітектурний потік виділено для роботи з мультимедійним контентом. Під час додавання нових лінгвістичних матеріалів модератором через вебпанель, бекенд передає аудіо- та графічні файли до Cloudinary [18] через захищене API, зберігаючи у базі даних лише оптимізовані посилання. У процесі навчання мобільні пристрої завантажують ці ресурси (вимову слів, ілюстрації до карток) напряду з глобальної мережі доставки контенту (CDN) Cloudinary. Це дозволяє повністю оминати основний сервер платформи при завантаженні «важкого» контенту, що значно знижує мережеве навантаження та пришвидшує роботу інтерфейсу [9].

Така розподілена хмарна архітектура гарантує високу відмовостійкість, миттєву доставку освітніх матеріалів та надійний захист персональних даних учнів. Програмна реалізація системи Speaker базується на принципах модульності та суворого розділення відповідальності (Separation of Concerns). Для проєктування структури коду застосовано методології об'єктно-орієнтованого програмування та сучасні архітектурні патерни, що забезпечують слабку зв'язність компонентів [10].

Доменна модель описує фундаментальні сутності освітньої сфери (LearningModule, VocabularyCard, ProgressLog, User) та логічні зв'язки між ними, абстрагуючись від особливостей технічної реалізації. Вона формує ядро бізнес-логіки застосунку Speaker, визначаючи, як знання структурується та перетворюється на інтерактивний навчальний досвід. На рис. 2.5 представлено діаграму класів предметної області, яка відображає ієрархію об'єктів мобільного застосунку та їхню взаємодію в межах освітнього процесу [27].

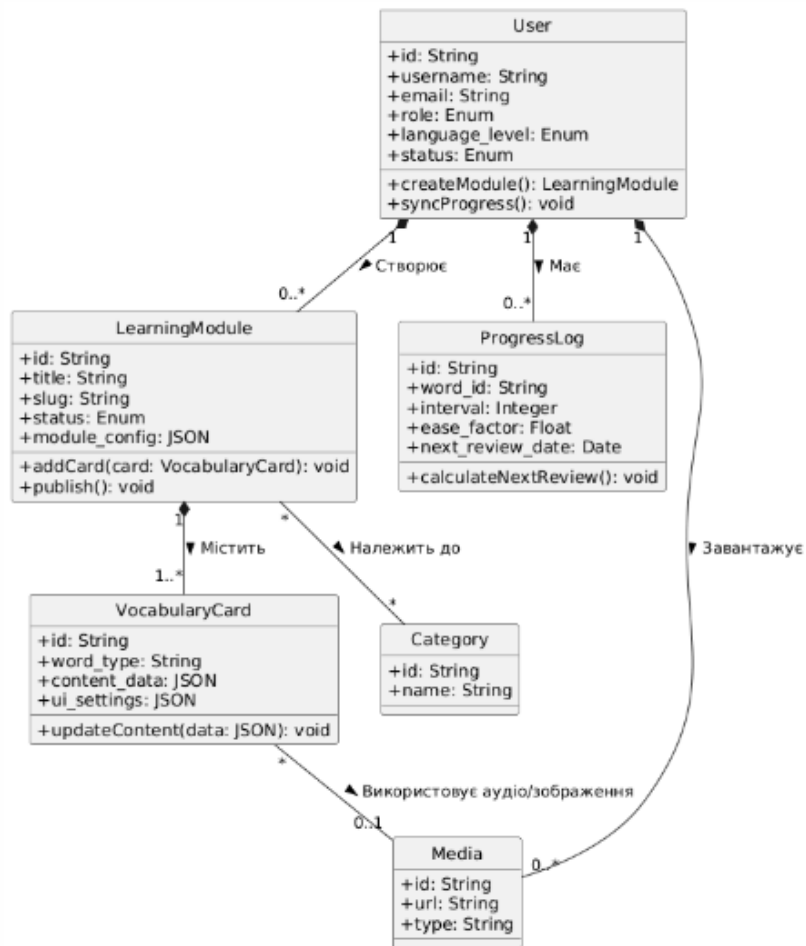


Рис. 2.5 – Діаграма класів предметної області

Центральною сутністю предметної області є клас User, який представляє обліковий запис учня. Він перебуває у відношенні композиції до класів LearningModule, Media та ProgressLog, оскільки ці об'єкти безпосередньо створюються користувачем та існують у межах його профілю (відповідно, видалення акаунта ініціює очищення пов'язаних сутностей). Клас LearningModule виступає агрегатором навчального контенту, перебуваючи у суворій композиції з класом VocabularyCard. Ключовими атрибутами карток є content_data та ui_settings типу JSON [13], де зберігається лінгвістичне наповнення (транскрипція, переклад, приклади) та візуальна конфігурація. Додатково модулі взаємодіють із глобальними категоріями (Category) та медіатекою, формуючи запити на отримання необхідних для навчання активів.

Для забезпечення масштабованості коду в межах архітектури та уникнення його дублювання, розробка спирається на використання сталих інженерних патернів [10]. Діаграма класів реалізації ілюструє проходження

потоків даних від мобільного застосунку до бази даних, відображаючи чіткий поділ відповідальності між компонентами системи Speaker.

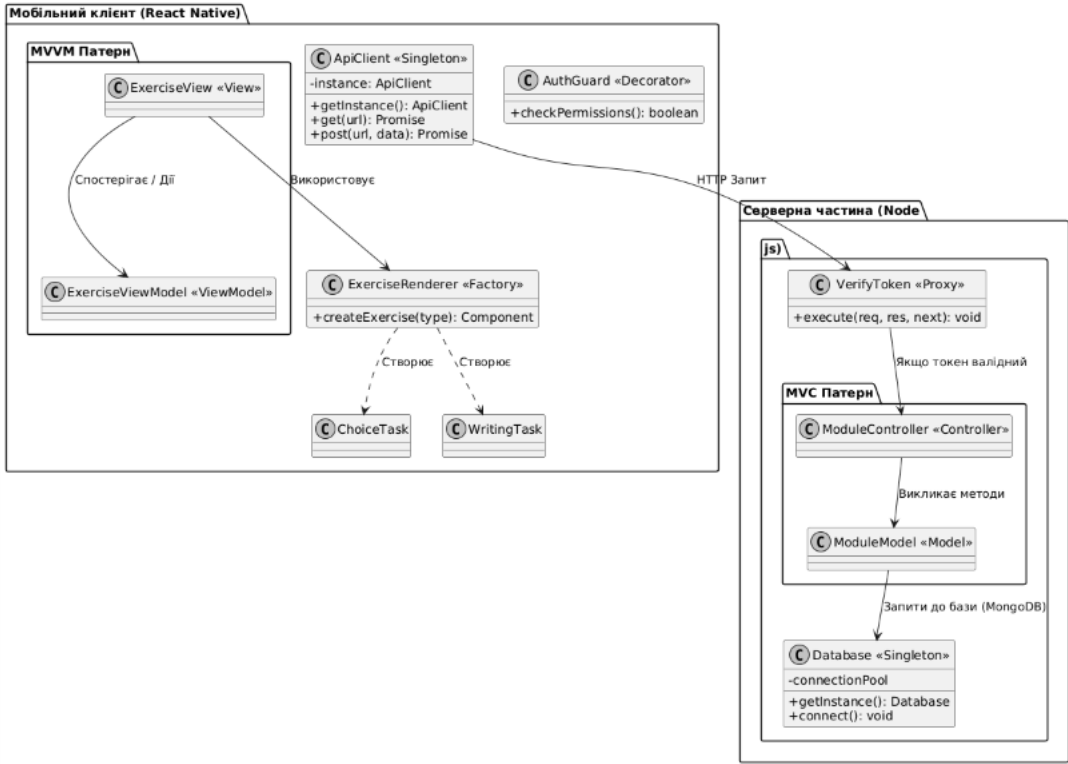


Рис. 2.6 – Діаграма класів реалізації з використанням патернів

В основу проектування системи закладено декілька ключових архітектурних шаблонів, що дозволяють ефективно розділити логіку програми та забезпечити її масштабованість. Зокрема, на серверному рівні використовується класичний патерн MVC, де контролер, наприклад, `ModuleController`, повністю керує бізнес-логікою, а модель забезпечує стабільну взаємодію з базою даних через інструментарій `Mongoose`. Натомість на мобільному клієнті застосовується архітектура MVVM [11], у якій роль відображення виконують компоненти `React Native` [1], а логіка керування станом винесена у відокремлену модель представлення із застосуванням сучасних менеджерів стану [32], що гарантує швидке та реактивне оновлення інтерфейсу під час будь-якої зміни даних. Для динамічного створення різних типів навчальних вправ на стороні фронтенду використовується патерн Фабрика, який реалізований через клас `ExerciseRenderer` і дозволяє автоматично інстанціювати відповідний компонент залежно від специфіки переданого завдання, такого як вибір правильної відповіді (`ChoiceTask`), письмове завдання (`WritingTask`) чи аудіо-вправа, тим самим

допомагаючи уникнути жорсткого та неефективного програмування кожного уроку в загальному макеті застосунку. Глобальні сервіси системи, що вимагають наявності виключно єдиного екземпляра в оперативній пам'яті, побудовані на базі патерну Одинак, яким у мобільному клієнті виступає спеціально налаштований екземпляр Axios під назвою ApiClient із вбудованими перехоплювачами JWT-токенів [17], тоді як на серверному боці цю роль виконує клас конектера бази даних, що ефективно керує пулом з'єднань до MongoDB [29] та запобігає критичній перевитраті системних ресурсів під час обробки інтенсивних запитів.

Крім того, для суворого обмеження доступу та перевірки прав застосовується патерн Замісник, який на клієнтському рівні втілений через компоненти вищого порядку, зокрема AuthGuard для захисту приватних маршрутів, а на сервері функціонує як проміжне програмне забезпечення VerifyToken у межах фреймворку Express.js [25], забезпечуючи надійну валідацію токенів безпеки перед наданням користувачу доступу до конфіденційних методів API.

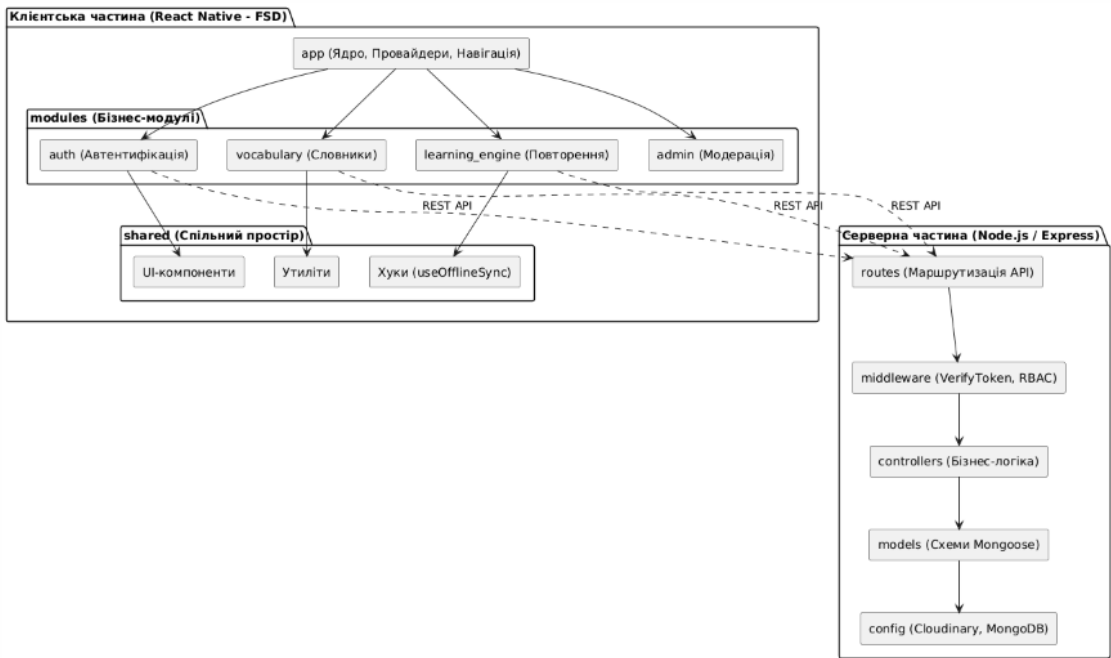


Рис. 2.7 – Діаграма компонентів програмного комплексу

Серверна частина мобільного застосунку, розроблена на базі Node.js [4] та Express, реалізована за принципами модульного моноліта. Структурна

організація коду передбачає чіткий поділ на рівні: каталог routes відповідає за декларацію кінцевих точок API [2], спрямовуючи вхідний трафік до відповідних контролерів (controllers). Безпека системи забезпечується шаром middleware, який виконує роль захисного екрана, здійснюючи перевірку JWT-токенів та валідацію ролей згідно з моделлю RBAC. Усі зовнішні інтеграції, включаючи конфігурацію медіасервісу Cloudinary [18], налаштування Stripe [30] та підключення до бази даних MongoDB, винесені в ізольований модуль config.

Клієнтська частина платформи, побудована мовою TypeScript [5] на базі фреймворку React Native, спроектована відповідно до методології Feature-Sliced Design (FSD), що дозволяє сформувати сувору модульну архітектуру та суттєво полегшити подальшу підтримку і масштабування вихідного коду. У межах цієї концепції весь програмний комплекс декомпоновано на три фундаментальні логічні рівні, кожен з яких відповідає за власну чітко визначену зону функціональності.

Першим рівнем є ядро мобільного застосунку (app), яке інкапсулює глобальні провайдери стану системи, включаючи централізовані механізми авторизації та локалізації, а також містить базову конфігурацію екранів навігації та логіку первинної ініціалізації локальної бази даних SQLite [14]. Наступний рівень представлений відокремленими незалежними предметними модулями (modules або features), які безпосередньо реалізують ключову бізнес-логіку застосунку. Серед них найскладнішим архітектурним елементом є рушій навчання (learning_engine), що повністю відповідає за математичний розрахунок і механіку інтервальних повторень, функціонування фабрики для динамічного рендерингу вправ та обробку результатів тестування користувачів, тоді як супутні модулі забезпечують процеси адміністрування та модерації контенту, управління словником і інтеграцію із зовнішніми сервісами автентифікації через Google OAuth [31].

Завершальним елементом цієї структури виступає спільний простір (shared), який об'єднує універсальні й перевикористовувані UI-компоненти, зокрема стандартні кнопки, поля введення та елементи анімації, а також

					ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
						53
Змн.	Арк.	№ докум.	Підпис	Дата		

ізолювані типи карток знань, загальні утиліти для обробки тексту та кастомні хуки, на кшталт useOfflineSync, розроблений для забезпечення надійної фонової синхронізації прогресу користувача з хмарою.

Така декомпозиція дозволяє незалежно тестувати та масштабувати окремі частини освітньої платформи без ризику порушення цілісності загальної логіки. Повна структура каталогів монорепозиторію Sreaker відображає фізичну організацію проєкту, де кожен компонент має чітко визначене місце.

Для деталізації процесів обміну даними між мобільним клієнтом та хмарним бекендом було розроблено діаграми взаємодії. Вони візуалізують життєвий цикл обробки HTTP-запитів у межах архітектури REST API [2], передачу управління між об'єктами та безпосередні звернення до документо-орієнтованої бази даних [13]. Це дозволяє проаналізувати ефективність маршрутизації та виявити можливі вузькі місця у логіці обробки інформації ще на етапі проєктування системи.

Діаграми послідовності відображають хронологічний порядок виклику методів та обміну повідомленнями між екземплярами класів під час виконання конкретних прецедентів. На рисунку 2.8 зображено діаграму послідовності процесу ініціалізації нового навчального модуля, що є базовим функціоналом застосунку Sreaker для формування індивідуального освітнього треку.

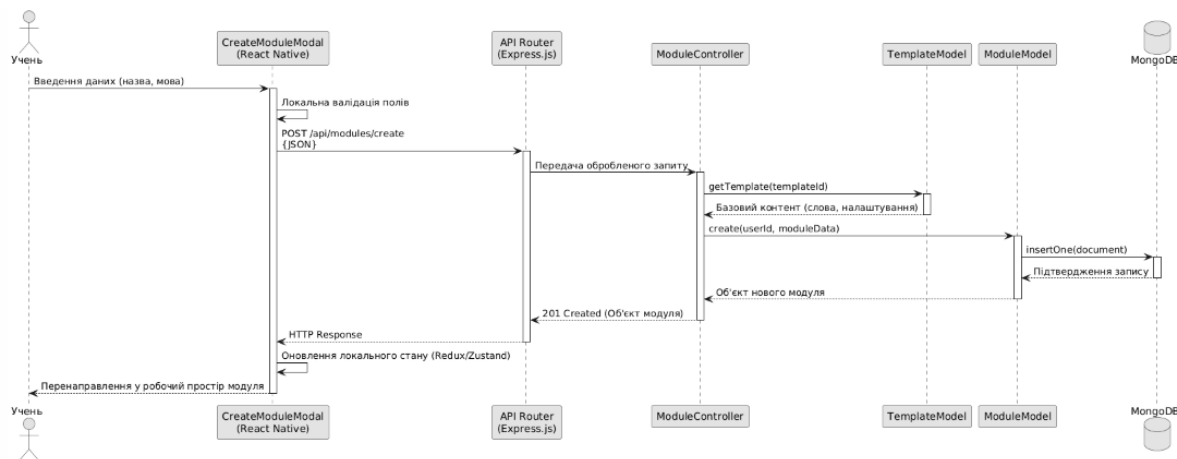


Рис. 2.8 – Діаграма послідовності процесу ініціалізації нового навчального модуля

Взаємодія між об'єктами під час ініціалізації нового навчального модуля або освітнього проєкту відбувається у чіткій хронологічній послідовності. Спочатку актор (Учень) взаємодіє з мобільним інтерфейсом CreateModuleModal у застосунку на базі React Native [1], заповнюючи форму базовими параметрами: назвою курсу, мовною парою та налаштуваннями приватності. Компонент інтерфейсу виконує локальну синхронну валідацію полів, перевіряючи коректність заповнення та наявність обов'язкових атрибутів.

Після підтвердження дії користувачем, мобільний клієнт формує асинхронний HTTP POST запит на маршрут /api/modules/create і передає JSON-пакет із конфігураційними даними. Запит проходить через шари безпеки фреймворку Express.js [25], обробляється маршрутизатором та делегується до ModuleController. На цьому етапі контролер звертається до моделі TemplateModel для отримання базового лінгвістичного контенту або стартової структури вправ, якщо було обрано готовий шаблон навчання.

Отримавши контент шаблону, контролер викликає метод create() моделі ModuleModel, передаючи ідентифікатор користувача та параметри майбутнього модуля. Після успішного виконання операції збереження документа в базі даних MongoDB [29], сервер формує відповідь і повертає об'єкт нового навчального блоку клієнту через архітектуру REST API [2]. Мобільний застосунок завершує транзакцію, оновлює локальний кеш стану і перенаправляє учня безпосередньо до робочого простору створеного модуля для початку навчання.

На рисунку 2.9 представлена діаграма послідовності створення звернення (тікету) до контент-менеджерів або служби підтримки через мобільний клієнт чи вебпанель. Цей процес є критично важливим для забезпечення високої якості освітнього контенту, оскільки дозволяє користувачам миттєво повідомляти про помилки в перекладах, некоректну вимову або пропонувати додавання нових лексичних тем.

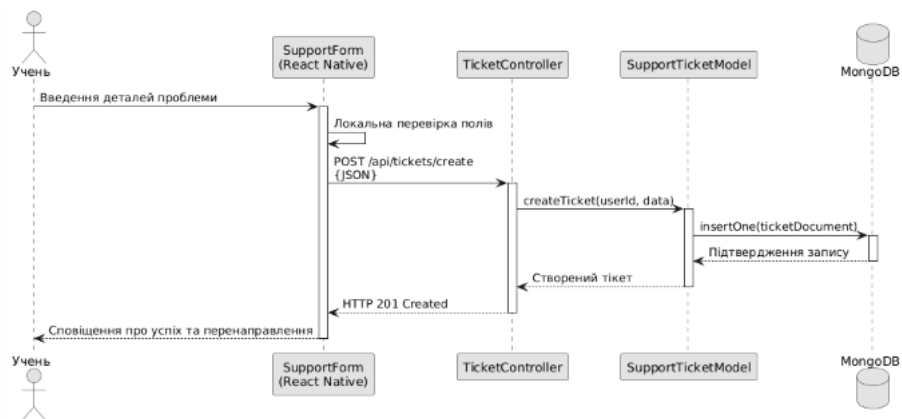


Рис. 2.9 – Діаграма послідовності "Створення звернення до служби підтримки"

Логіка обміну даними при формуванні звернення до підтримки передбачає кілька послідовних етапів. Актор (Учень) вводить деталі проблеми у відповідному розділі застосунку, після чого інтерфейс здійснює локальну перевірку полів. Користувач підтверджує відправку форми, що ініціює HTTP POST запит до TicketController. Контролер передає отримані дані моделі SupportTicketModel, яка виконує транзакційний запис у документо-орієнтовану базу даних [13]. Після успішного завершення транзакції сервер повертає статус 201 Created, а мобільний застосунок повідомляє користувача про успіх і перенаправляє його до переліку активних запитів для відстеження статусу модерації звернення.

На рисунку 2.10 ілюструється складна бізнес-транзакція процесу модерації та призупинення публічного доступу до навчального контенту адміністратором платформи Speaker. Цей сценарій містить розгалуження та умовні виклики, що забезпечують захист екосистеми застосунку від неякісного або шкідливого наповнення.

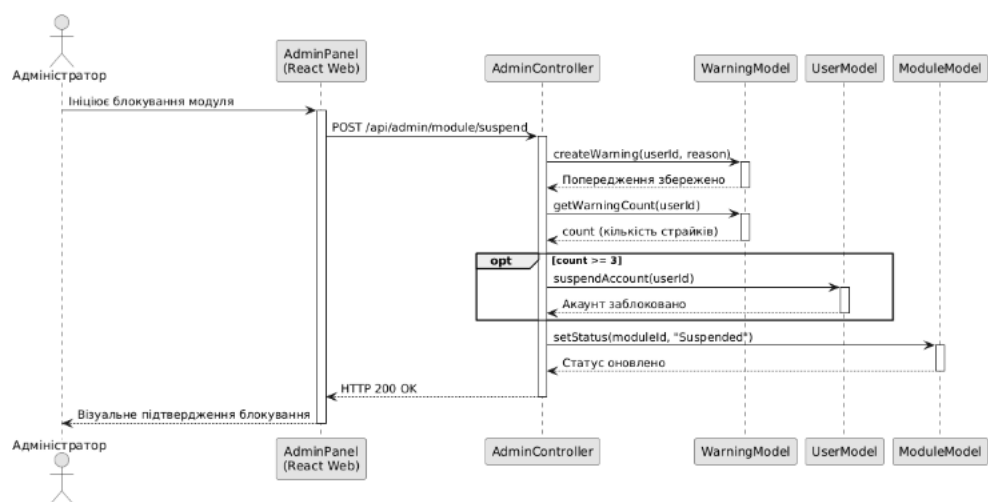


Рис. 2.10 – Діаграма послідовності "Призупинення аналітичного дашборда"

Хронологія адміністративної модерації розпочинається з того, що Адміністратор через вебпанель керування ініціює блокування модуля-порушника. Система відправляє захищений HTTP POST запит до AdminController, який першочергово звертається до моделі WarningModel для фіксації офіційного попередження (страйку) в профілі автора контенту. Це дозволяє накопичувати історію порушень для подальшого прийняття системних рішень.

Після фіксації попередження контролер виконує повторний запит до моделі для отримання загальної кількості активних страйків (попереджень) даного користувача. На цьому етапі реалізовано умовне розгалуження: якщо кількість порушень досягає критичної межі (три або більше), система автоматично звертається до UserModel для призупинення дії всього облікового запису. Це запобігає масовому поширенню неякісного контенту від одного джерела та порушенню правил платформи.

Незалежно від результату перевірки ліміту попереджень, контролер надсилає команду до ModuleModel для негайної зміни статусу конкретного навчального ресурсу на Suspended у базі даних MongoDB [29]. Після успішного оновлення записів серверний бекенд на Node.js [4] повертає статус 200 OK клієнтському інтерфейсу. На завершальному етапі панель адміністратора

синхронізується з хмарою, візуально позначаючи навчальний блок як заблокований.

Для глибшого розуміння структурних зв'язків між об'єктами під час виконання складних транзакцій використовується діаграма кооперації (комунікації). Вона трансліює ту ж логіку обміну повідомленнями, що й діаграма послідовності, проте зміщує акцент із хронології на просторову архітектуру та топологію зв'язків між програмними модулями [10].

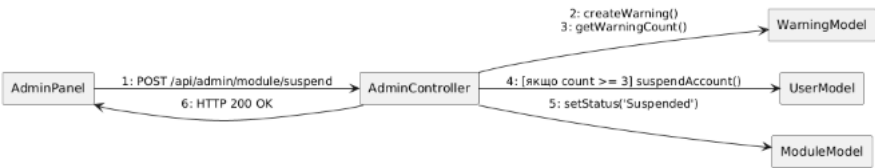


Рис. 2.11 – Діаграма кооперації "Призупинення аналітичного дашборда"

Діаграма кооперації, представлена на рисунку 2.11, демонструє радіальну топологію управління, де центральним вузлом виступає AdminController. Він виконує функцію фасадного інтерфейсу (патерн Facade), що ізолює клієнтський застосунок від прямої взаємодії з шаром даних. Така архітектура дозволяє контролеру ефективно координувати роботу трьох незалежних моделей: WarningModel, UserModel та ModuleModel. Це забезпечує суворе дотримання принципу єдиної відповідальності (Single Responsibility Principle) [10] та гарантує транзакційну цілісність під час модерації навчального контенту.

2.4 Розробка основних алгоритмів роботи платформи

Для детального розуміння внутрішньої логіки застосунку Speaker було розроблено алгоритми ключових бізнес-процесів, візуалізовані за допомогою діаграм активності. Ці моделі відображають динаміку поведінки системи, ілюструючи паралельні обчислення, логічні розгалуження та чіткий розподіл зон відповідальності між мобільним клієнтом та серверною частиною за допомогою функціональних доріжок (swimlanes).

Алгоритм процесу «Формування навчального модуля» описує послідовність дій системи та уповноваженого користувача (адміністратора або

контент-менеджера) під час створення, наповнення та публікації нового освітнього контенту. Процес має транзакційну природу та супроводжується суворими перевітками бізнес-логіки на рівні домену для забезпечення цілісності навчального матеріалу.

Даний алгоритм описує один із фундаментальних процесів програмної платформи ініціалізацію, комплексну валідацію та подальше налаштування нового освітнього блоку на базі попередньо визначеного шаблону структури. Застосування такого шаблонного підходу (патерну проєктування) не лише мінімізує кількість рутинних операцій з боку користувача чи адміністратора, але й забезпечує високий рівень архітектурної гнучкості при формуванні різних типів навчальних завдань. Завдяки абстрагуванню бізнес-логіки від рівня представлення, система отримує здатність динамічно генерувати інтерфейси та адаптуватися під специфіку конкретного навчального матеріалу без необхідності дублювання коду.

Життєвий цикл процесу розпочинається з ініціації клієнтського запиту користувачем, після чого система проходить через багаторівневий конвеєр обробки. Цей конвеєр охоплює етапи суворої валідації вхідних параметрів, санітизації даних, вибору оптимальної конфігурації та безпечної асинхронної синхронізації з хмарним сховищем. На цьому етапі програмно реалізовано критично важливі механізми перевірки цілісності завантажених медіафайлів (зокрема аудіо- та відеоконтенту) за допомогою порівняння контрольних сум та хешування. Це гарантує коректну передачу кожного байту даних, їхній захист від пошкоджень та стабільну роботу застосунку навіть в умовах нестабільного мережевого з'єднання або тимчасової втрати зв'язку на клієнтському пристрої. Вбудовані механізми повторних запитів (retry policies) додатково підвищують загальну відмовостійкість цього вузла.

Така комплексна візуалізація потоку управління (control flow) та потоку даних (data flow) є критично необхідною для глибокого розуміння прихованих асинхронних процесів у системі. Крім того, формалізований опис цих станів слугує надійною інженерною та архітектурною основою для подальшого

					ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
						59
Змн.	Арк.	№ докум.	Підпис	Дата		

масштабування програмного комплексу. Це дозволить у майбутньому безперешкодно інтегрувати нові модулі, розширювати формати освітніх завдань та витримувати підвищені навантаження при зростанні активної бази користувачів.

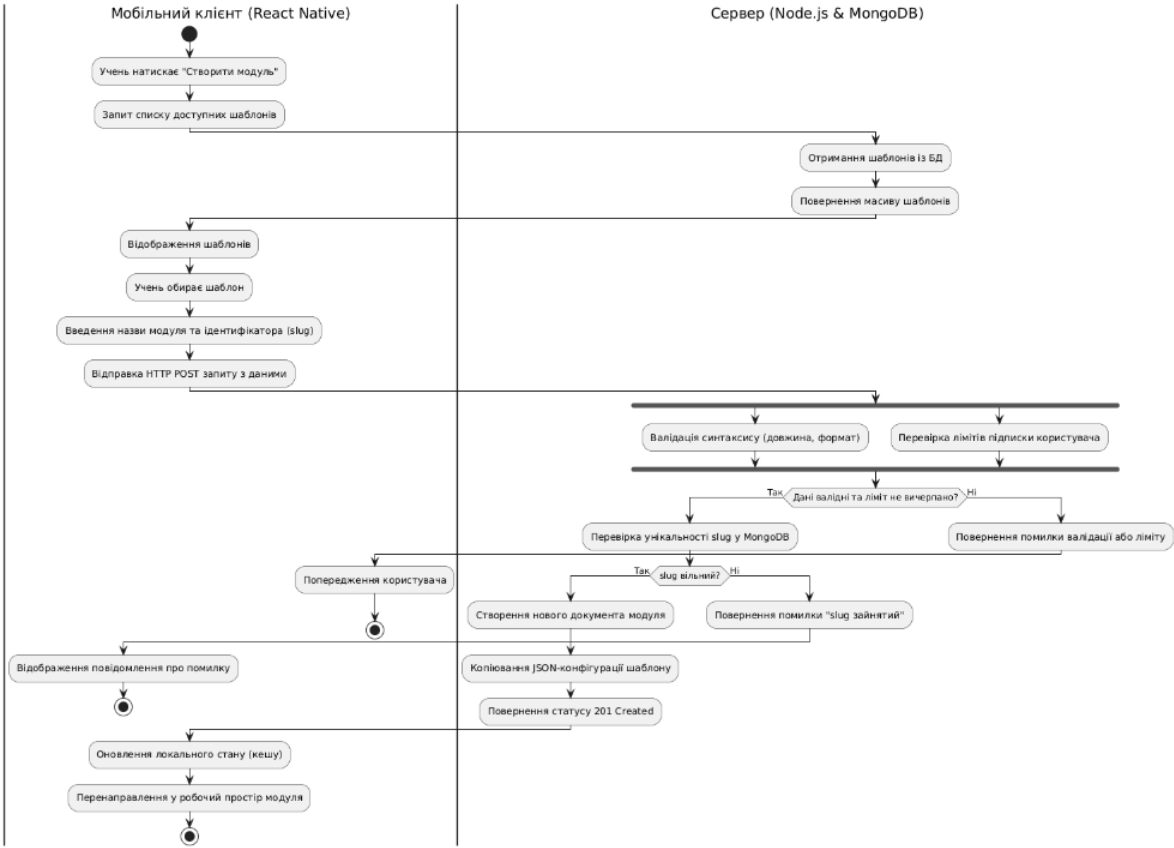


Рис. 2.12 – Діаграма активності " Створення навчального модуля"

Покрокове виконання алгоритму створення нового навчального модуля розпочинається з ініціалізації процесу користувачем, дія якого стимулює мобільний клієнт миттєво звернутися до серверного REST API [2] для завантаження переліку доступних освітніх шаблонів та мовних конфігурацій із відповідної колекції бази даних MongoDB [29]. Отримавши необхідну інформацію, учень обирає релевантний шаблон та вводить базові параметри, зокрема назву модуля і його унікальний текстовий ідентифікатор (slug), який у подальшому використовуватиметься для формування прямих посилань та забезпечення коректної синхронізації контенту. Після відправки цих даних на стороні серверного бекенду в середовищі Node.js [4] автоматично запускаються два паралельні процеси валідації, що включають детальну перевірку

синтаксичної коректності введеної інформації щодо формату тексту і довжини назви, а також верифікацію користувацьких лімітів для контролю допустимої кількості активних проєктів згідно з поточним рівнем доступу чи наявністю преміум-підписки через платіжний шлюз Stripe [30]. У разі успішного проходження цих попередніх етапів сервер ініціює додатковий пошуковий запит до колекції `learning_modules` з метою підтвердження абсолютної унікальності обраного ідентифікатора в межах усієї екосистеми застосунку, що дозволяє превентивно уникнути маршрутизаційних конфліктів під час публічного поширення матеріалів. За умови дотримання всіх перелічених вимог у базі даних MongoDB формується новий документ, після чого система автоматично імпортує дефолтну JSON-конфігурацію типів вправ із раніше обраного шаблону [13], а клієнтський застосунок, розроблений на базі фреймворку React Native [1], миттєво перенаправляє користувача до новоствореного робочого простору для безпосереднього початку наповнення модуля лексичними одиницями.

Алгоритм процесу «Редагування аналітичного контенту» регламентує послідовність дій системи та уповноваженого користувача під час модифікації існуючих статистичних показників, метрик успішності або конфігурацій аналітичних звітів. Оскільки аналітичні дані безпосередньо впливають на відстеження прогресу та формування персоналізованих рекомендацій, цей процес супроводжується суворими механізмами контролю цілісності та версіонування.

Цей процес демонструє роботу з візуальним редактором навчальних модулів та словників, де основне навантаження лягає на клієнтську частину (React Native [1]), а звернення до сервера мінімізовані для забезпечення плавності інтерфейсу. Такий архітектурний підхід дозволяє реалізувати механізми оптимістичного оновлення (Optimistic UI), коли зміни відображаються користувачеві миттєво за рахунок управління локальним станом програми. Завдяки ефективному використанню пам'яті та кешуванню даних на мобільному пристрої, система здатна обробляти складні маніпуляції з контентом без відчутних мережових затримок. Усі необхідні транзакції з базою даних

виконуються асинхронно у фоновому режимі, переважно у вигляді пакетних запитів (batch requests) під час збереження або фінальної синхронізації. Це не лише суттєво знижує кількість надлишкових HTTP-викликів і зменшує навантаження на серверну інфраструктуру, але й гарантує високу чутливість (responsiveness) застосунку, утримуючи концентрацію користувача безпосередньо на освітньому процесі.

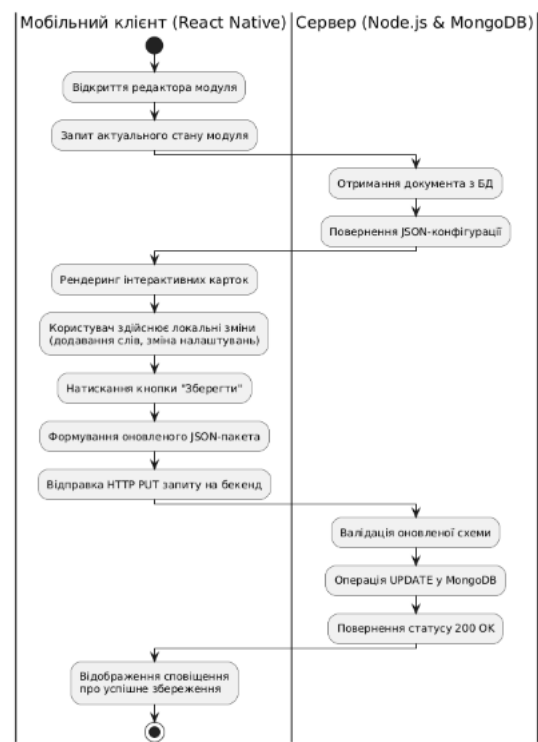


Рис. 2.13 – Діаграма активності "Редагування навчального контенту"

Під час ініціалізації редактора серверна частина (Node.js [4]) передає мобільному клієнту актуальний масив JSON-конфігурацій модулів із бази даних. Це дозволяє застосунку отримати повну структуру уроку для подальшої локальної маніпуляції. Отримавши дані, клієнтська частина виконує рендеринг інтерактивних компонентів (карток, вправ, медіаблоків) відповідно до отриманої схеми, підготовлюючи робочу область для користувача.

Безпосередня взаємодія та маніпуляції з контентом здійснюються користувачем у локальній пам’яті пристрою. Цей процес включає додавання або видалення лексичних одиниць, зміну параметрів вправ та перевпорядкування карток знань. Завдяки використанню локального менеджера стану (наприклад,

Zustand або Redux [32]), усі внесені зміни відображаються на екрані миттєво, без затримок на мережеві запити.

Після завершення редагування та натискання кнопки збереження, мобільний застосунок формує фінальний оновлений JSON-пакет, який відображає нову структуру всього модуля. Цей пакет передається на серверну частину, яка приймає оновлені дані, виконує атомарну операцію UPDATE у колекції MongoDB [29] та повертає статус успішного виконання. Після отримання підтвердження від сервера інтерфейс застосунку виводить користувачеві сповіщення про успішне збереження змін.

Алгоритм процесу «Отримання доступу до розширеного освітнього контенту» описує логіку взаємодії користувача з платформою при спробі отримати доступ до спеціалізованих навчальних матеріалів (наприклад, професійні словники або деталізована аналітика вимови), враховуючи перевірку рівня підписки та прав доступу. Оскільки цей процес передбачає роботу з преміальним контентом та зміну статусів профілю, він супроводжується багаторівневою авторизацією. Під час ініціації запиту система не лише перевіряє актуальність підписки через серверний модуль управління доступом, але й гарантує безпечне проведення транзакцій у базі даних (наприклад, при списанні внутрішніх балів).

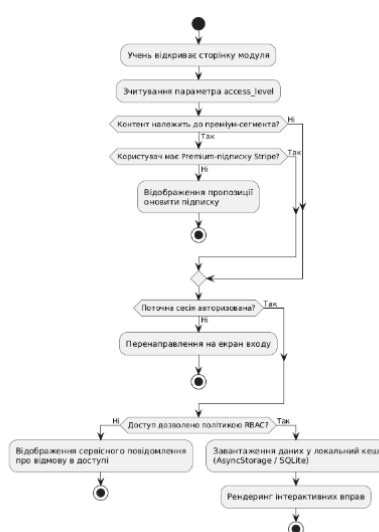


Рис. 2.14 – Діаграма активності "Отримання доступу до розширеного освітнього контенту"

Процес розпочинається з того, що учень відкриває сторінку курсу або конкретного мовного модуля. У цей момент клієнтська частина застосунку автоматично зчитує параметр `access_level` для кожного елемента контенту. Якщо обраний навчальний матеріал належить до преміум-сегмента (наприклад, корпоративні ІТ-глосарії), а користувач має лише базовий (безкоштовний) рівень доступу, система одразу відображає неактивну кнопку з пропозицією оновити підписку через платіжний шлюз Stripe [30].

При спробі активувати розширений функціонал застосунок першочергово перевіряє стан поточної сесії, аналізуючи чинність та валідність JWT-токена [17]. У разі відсутності авторизації користувач автоматично перенаправляється на екран входу. Додатково, для забезпечення захисту інформації, система бекенду перевіряє права доступу до специфічних баз даних відповідно до політики управління ролями (RBAC). Якщо доступ для даного користувача обмежений, на екран виводиться відповідне сервісне повідомлення. У разі ж успішного проходження всіх етапів верифікації, необхідні дані кешуються в локальному сховищі пристрою (AsyncStorage або SQLite [14]), після чого активується рендеринг інтерактивних вправ та графіків прогресу навчання.

Алгоритм процесу «Призупинення навчального модуля Адміністратором» є ключовим інструментом системної модерації, який забезпечує автоматизоване реагування на публікацію неякісного або недостовірного освітнього контенту в застосунку Sreaker. Впровадження такого інструментарію є критично важливим для підтримки високих стандартів платформи, особливо зважаючи на специфіку роботи з аудіоматеріалами, де якість звуку та точність вимови безпосередньо впливають на результативність сприйняття на слух. Архітектурно цей процес інтегрований на рівні серверної логіки та використовує систему тригерів. У разі фіксації порушень (наприклад, перевищення порогу скарг або виявлення невідповідностей у структурі модуля), проблемний контент автоматично переводиться у статус «Призупинено» (Suspended).

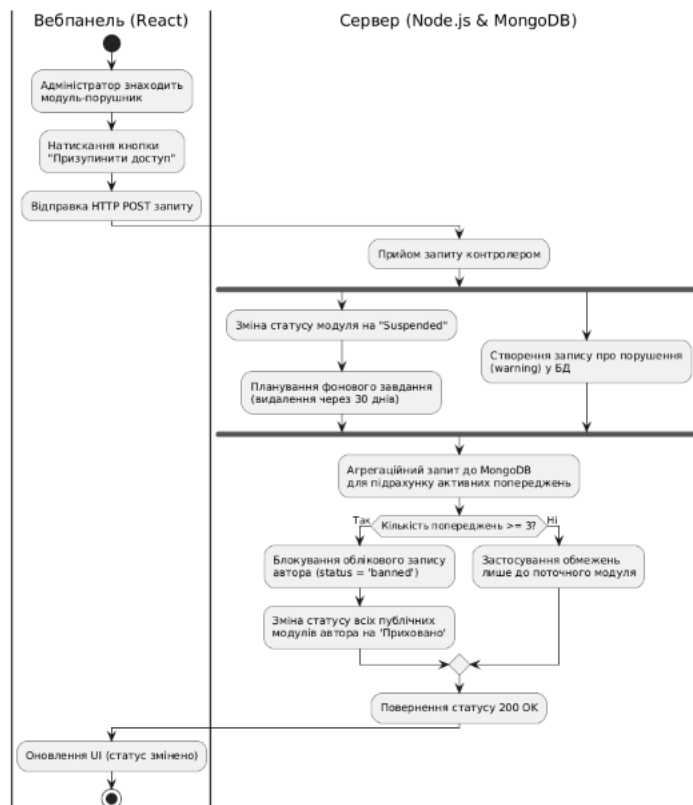


Рис. 2.15 – Діаграма активності " Призупинення навчального модуля Адміністратором"

Процес модерації розпочинається з етапу ініціації адміністратор платформи через спеціалізовану вебпанель керування (React) ідентифікує навчальний модуль, що порушує правила спільноти, та ініціює процедуру його призупинення. Відповідний мережевий запит передається на серверний бекенд платформи Speaker.

Після отримання та валідації дії Node.js-сервер одночасно запускає два паралельні процеси. З одного боку, система оновлює статус проблемного модуля на suspended, автоматично плануючи фонове завдання (cron job) щодо його безповоротного видалення через 30 днів для оптимізації та очищення бази даних. З іншого боку, паралельно формується новий документ у колекції попереджень (warnings) бази MongoDB [29] для фіксації порушення безпосередньо в історії автора контенту.

Наступним кроком система автоматично виконує агрегаційний запит до бази даних для підрахунку загальної кількості активних попереджень (страйків) у автора даного модуля. На основі отриманих даних спрацьовує логічне

розгалуження алгоритму. Якщо сумарна кількість попереджень досягла або перевищила критичну межу (три і більше), система ініціює каскадне блокування всього облікового запису користувача та превентивно приховує всі його публічні навчальні матеріали із загального каталогу застосунку. У випадку, якщо встановлений ліміт порушень не перевищено, відповідні обмеження застосовуються виключно до поточного (одного) модуля.

Висновки до другого розділу

У другому розділі було виконано комплексне проєктування програмної архітектури мобільної інформаційної системи *Speaker* для вивчення іноземних мов. Процес проєктування охопив усі ключові етапи: від побудови високорівневої моделі серверної інфраструктури до глибокої деталізації алгоритмів обробки лінгвістичних даних та синхронізації навчального прогресу між мобільним клієнтом і сервером.

У ході роботи було побудовано детальну модель розгортання системи. Зокрема, обґрунтовано доцільність розподіленого хмарного підходу із використанням сучасних PaaS та SaaS рішень, таких як *Render* [19] для хостингу *Node.js*-сервера, *MongoDB Atlas* для керованої бази даних та *Cloudinary* [18] для глобальної оптимізації медіафайлів. Обрана конфігурація інфраструктури гарантує високу відмовостійкість під час роботи з мультимедійним контентом, швидку доставку аудіоматеріалів через *CDN*.

Крім того, було формалізовано варіанти використання застосунку. Визначено ключові ролі акторів (Учень, Контент-менеджер, Адміністратор) та розроблено інженерні специфікації для базових прецедентів. Особливу увагу приділено логіці формування персоналізованих навчальних модулів та механізмам багаторівневої модерації контенту, що забезпечує неухильне дотримання освітніх стандартів та високий рівень безпеки платформи *Speaker*.

Для реалізації бізнес-вимог успішно розроблено об'єктно-орієнтовану структуру проєкту. Побудовані діаграми класів ілюструють практичне застосування таких патернів проєктування, як *MVC*, *Factory*, *Proxy* та *Singleton* [10]. Використання архітектурної методології *Feature-Sliced Design*

					ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
						66
Змн.	Арк.	№ докум.	Підпис	Дата		

(FSD) для клієнтської частини на React Native [1] дає змогу ізольовано розвивати навчальні модулі та ігрові механіки інтервальних повторень, не порушуючи цілісності ядра мобільної екосистеми.

Також було спроектовано гібридну модель даних. Логічна структура бази даних поєднує переваги суворих зв'язків для профілів користувачів та статистики транзакцій (інтеграція Stripe [30]) із гнучкістю документо-орієнтованого формату JSON у MongoDB для зберігання динамічних конфігурацій інтерактивних вправ [13]. Такий підхід гарантує, що застосунок зможе оперативно адаптуватися до інтеграції нових типів освітнього контенту без складних та ризикованих міграцій архітектури.

Нарешті, було докладно описано динамічну поведінку системи. За допомогою діаграм активності, послідовності та кооперації деталізовано бізнес-логіку створення контенту, модерації ресурсів та проходження тестів. Було візуалізовано хронологію взаємодії між мобільним клієнтом, вебпанеллю, серверними контролерами та шаром даних у межах REST API архітектури [2].

РОЗДІЛ 3. ІНТЕРФЕЙС ТА ПОРЯДОК РОБОТИ З ІНФОРМАЦІОНОЮ СИСТЕМОЮ УПРАВЛІННЯ БУДІВЕЛЬНИМИ ПРОЄКТАМИ ТА РОБОЧИМИ ПРОЦЕСАМИ

3.1 Порядок встановлення та налаштування параметрів системи

Система Sreaker розгортається у розподіленому хмарному середовищі, а процес від коміту до появи оновленої версії у виробництві є повністю автоматизованим. Розробник не виконує жодних ручних дій для публікації достатньо інтегрувати зміни до основної гілки репозиторію. Решту бере на себе конвеєр безперервної інтеграції та розгортання (CI/CD) [21].

Система використовує п'ять спеціалізованих хмарних служб (табл. 3.1). Кожна з них відповідає за окремий аспект роботи платформи і може масштабуватися незалежно від решти компонентів інфраструктури..

Таблиця 3.1

Хмарна інфраструктура системи

Служба	Платформа	Призначення
Render App Service	Render	Хостинг серверної частини (Node.js/Express)
Versel	Versel	Хостинг клієнтської вебпанелі адміністратора (React)
MongoDB Atlas	MongoDB Inc.	Керована документо-орієнтована база даних
Cloudinary	Cloudinary Ltd	Сховище та CDN для медіафайлів (аудіо, графіка)
GitHub Actions	GitHub	Реєстр коду та платформа автоматизації CI/CD

Клієнтські застосунки функціонують в умовах суворої ізоляції. Будь-які прямі транзакції з мобільного застосунку або браузера до бази даних фізично заблоковані на рівні архітектури. Єдиною дозволеною точкою взаємодії виступає серверне REST API [2] на базі платформи Render [19]. Саме цей вузол бере на себе всю оркестрацію: він керує структурованими документами у MongoDB [29] та маніпулює бінарними об'єктами (аудіофайлами вимови) через API Cloudinary [18]. Маршрут доставки готового коду проходить через репозиторій GitHub. CI-конвеєр тестує та компілює проєкт, після чого середовище Render отримує автоматичний webhook-сигнал на розгортання оновленого сервера.

Серверна частина пакується у Docker-образ із використанням багатоетапної збірки (Multi-stage build). На першому етапі завантажуються залежності та виконується транспіляція вихідного коду з TypeScript [5] у JavaScript. На другому етапі до фінального образу копіюються лише скомпільовані артефакти та необхідні production-залежності, без вихідного коду і засобів розробки. Такий підхід суттєво зменшує розмір кінцевого образу і виключає потрапляння зайвих інструментів у виробниче середовище. Базовим образом є node:18-alpine мінімальний оптимізований образ із середовищем виконання Node.js [4].

Конвеєр безперервної доставки реалізовано засобами GitHub Actions (рис. 3.1). Механізм розбито на лінійні етапи із жорсткою залежністю. Логіка пайплайну не допускає компромісів: провал перевірки типів або лінтера миттєво обриває процес збірки, а будь-яка помилка під час проходження модульних тестів автоматично скасовує розгортання. Додатково налаштовано сувору фільтрацію за гілками репозиторію. Стадії фінального деплою ініціюються виключно під час злиття змін в основну гілку (main). Усі інші гілки проходять лише фазу аудиту якості та ізольованого тестування, без жодного впливу на виробниче середовище.

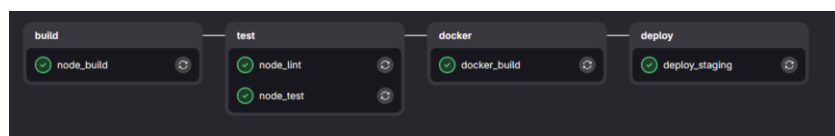


Рис. 3.1 – Діаграма конвеєра CI/CD

Стадії конвеєра

Стадія	Завдання	Зміст
1	2	3
lint	npm run lint	Перевірка стилю коду (ESLint) та статичний аналіз типізації (TypeScript)
test	Npm test	Модульні тести серверної бізнес-логіки та клієнтських компонентів
build_api	docker build	Збірка Docker-образу для серверної частини (Node.js)
build_web	npm run build	Компіляція статичних файлів вебпанелі (React)
deploy_api	Render Webhook	Оновлення Docker-образу та автоматичний перезапуск бекенду
deploy_web	Vercel CLI	Завантаження оновленої версії панелі на хостинг Vercel

Стадія lint перевіряє відповідність коду встановленим правилам форматування і виявляє потенційно небезпечні конструкції ще до запуску тестів. Модульні тести перевіряють доменну логіку (наприклад, алгоритми обчислення інтервалів для словника) без прямого підключення до продакшен-бази даних. Стадія автоматичного розгортання гарантує, що нова версія коду безшовно замінює попередню.

Чутливі параметри конфігурації рядок підключення до бази даних, ключі підпису токенів, облікові дані служб Stripe та Cloudinary категорично не зберігаються у вихідному коді. Вони безпечно зберігаються у вигляді секретів (GitHub Secrets) і встановлюються безпосередньо у панелях керування Render та Vercel (табл. 3.3). Ці захищені змінні не відображаються у логах конвеєра навіть при явному виведенні платформа автоматично маскує їхні значення. Під час виконання застосунку вони стають доступними через глобальні змінні середовища операційної системи (process.env).

					ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
						70
Змн.	Арк.	№ докум.	Підпис	Дата		

Таблиця 3.3

Параметри конфігурації виробничого середовища

Параметр	Призначення
MONGO_URI	Рядок підключення до хмарного кластера бази даних MongoDB Atlas
JWT_SECRET	Криптографічний ключ для підпису токенів доступу та авторизації
CLOUDINARY_URL	Облікові дані для доступу до сервісу зберігання та оптимізації медіафайлів
STRIPE_SECRET_KEY	Секретний ключ для управління преміум-підписками через платіжний шлюз [30]
GOOGLE_CLIENT_ID	Ідентифікатор застосунку для забезпечення авторизації через Google OAuth [31]

Для розробки та локального тестування система запускається без підключення до хмарного хостингу. Розробнику достатньо створити конфігураційний файл `.env` та виконати команду `npm run dev` у робочій директорії, яка ініціює локальний сервер.

Команда запускає три контейнери, що утворюють повну локальну копію системи (табл. 3.4).

Таблиця 3.4

Сервіси локального середовища

Контейнер	Образ	Порт	Призначення
speaker-api	speaker/api:local	5000	Серверна частина
Speaker-web	speaker/web:local	3000	Клієнтська вебпанель адміністратора
speaker-db	mongo:6	27017	Документо-орієнтована база даних MongoDB

Процес локального розгортання ініціює автоматичне виконання скриптів наповнення бази даних базовим тестовим контентом (seeding). Одразу після формування стартових колекцій документів клієнтський інтерфейс вебпанелі активується за адресою <http://localhost:3000>. Взаємодія з бекендом маршрутизується через локальний порт 5000. Таке архітектурне рішення перетворює робочу станцію розробника на точну структурну копію бойового (production) сервера. Ризик виникнення критичних розбіжностей між локальним та виробничим середовищами нівелюється повністю.

Обробка мультимедійного контенту під час розробки також відбувається автономно. Локальна конфігурація системи примусово відсікає будь-які мережеві запити до зовнішніх хмарних провайдерів, таких як Cloudinary. Натомість бінарний потік аудіофайлів чи зображень спрямовується безпосередньо на фізичний накопичувач робочої станції та фіксується в ізольованому каталозі `uploads/`. Це дає змогу розробнику тестувати завантаження медіа без використання хмарного трафіку.

Щодо демонтажу локальної інфраструктури, то повне очищення середовища, включно зі знищенням контейнерів та прив'язаних томів збереження даних (volumes), ініціюється єдиним консольним запитом:

```
docker compose -f docker-compose.local.yml down -v
```

Після повторного запуску системи скрипти наповнення бази даних застосовуються заново, і платформа починає роботу з чистою (скинутою до дефолтного стану) базою MongoDB. Це гарантує стабільність тестування та відсутність залишкових артефактів від попередніх сесій розробки.

3.2 Структура інтерфейсу інформаційної системи моніторингу та управління будівельними проєктами

Інфраструктура платформи Speaker включає повноцінну вебчастину та мобільний клієнт. Головна сторінка сайту (Landing Page) виконує презентаційну та дистрибутивну функції (рис. 3.2). Вона оформлена у сучасному мінімалістичному стилі з використанням масивної типографіки та градієнтних акцентів. Заголовок «Speak fluently. Without fear.» транслює основний емоційний

					ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		72

посил бренду. На сторінці розміщено прямі посилання на завантаження інсталяційного файлу («Download APK»), навігаційне меню (Features, Methodology, Pricing) та точку входу до закритої адміністративної панелі («Admin Panel»).

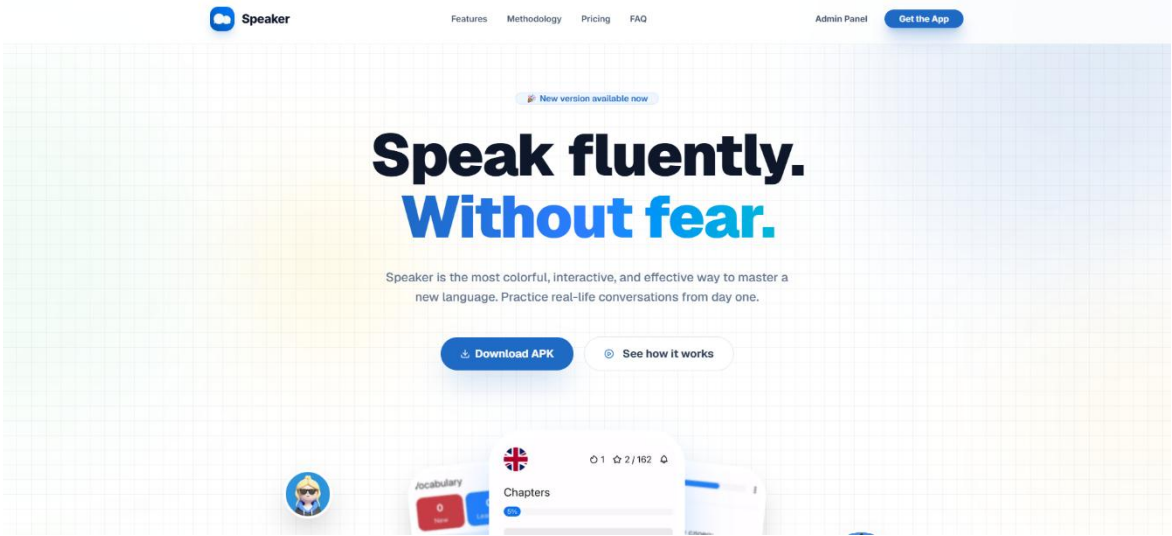


Рис. 3.2 – Головна сторінка вебсайту платформи Speaker

Інтерфейс безпосередньо мобільного застосунку Speaker спроектовано з урахуванням сучасних вимог до User Experience (UX) та User Interface (UI), що забезпечує швидку та безшовну взаємодію з освітнім контентом. При першому запуску мобільного клієнта користувача зустрічає екран привітання (рис. 3.3). Центральним елементом є яскрава тематична ілюстрація, яка візуалізує глобальність платформи. Екран містить головну кнопку (Call-to-Action) «Start learning» для нових користувачів та текстове посилання «Log in» для зареєстрованих учнів.

Центральним елементом є яскрава тематична ілюстрація, яка візуалізує глобальність платформи та задає мотиваційний тон. Використання оптимізованої графіки гарантує високу чіткість відображення на екранах мобільних пристроїв із різною щільністю пікселів без втрати продуктивності застосунку. Екран містить головну кнопку (Call-to-Action) «Start learning» для нових користувачів та текстове посилання «Log in» для зареєстрованих учнів.



Рис. 3.3 – Екран привітання мобільного застосунку

Процес реєстрації та автентифікації користувачів спроектовано з акцентом на мінімізацію когнітивного навантаження (рис. 3.4). На екрані створення профілю («Sign up») застосовуються стандартні патерни введення даних. Крім класичного методу реєстрації, реалізовано можливість швидкої авторизації через сторонні сервіси (зокрема, «Sign up with Google» через технологію OAuth 2.0).

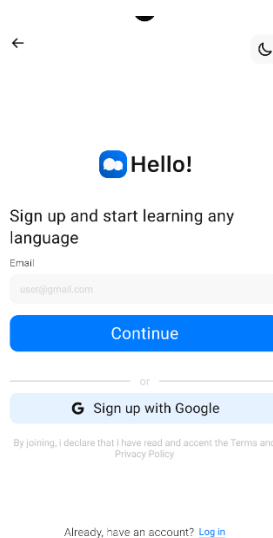


Рис. 3.4 – Екран реєстрації нового користувача

Аналогічний мінімалістичний підхід застосовано і на екрані авторизації (рис. 3.5), де передбачено поля для електронної пошти та пароля. Усі ці екрани оснащені зручною навігацією та посиланнями на політику конфіденційності.

Рис. 3.5 – Екран авторизації в систему

Після успішної авторизації користувач потрапляє на головний екран навчального треку (рис. 3.6). Верхня панель (Header) містить інформацію про обрану мову вивчення, поточний прогрес у вигляді щоденного «стрику» (streak) та кількість накопичених балів. Центральна частина екрана це вертикальна дорожня карта (Roadmap) уроків у межах поточного розділу. Активний або ключовий етап навчання виділено насиченим синім кольором. У нижній частині розташовано глобальне навігаційне меню (Bottom Navigation Bar).

Головний екран є візитною карткою застосунку та відіграє вирішальну роль у формуванні користувацького досвіду (UX), оскільки саме сюди потрапляє учень після успішної авторизації.

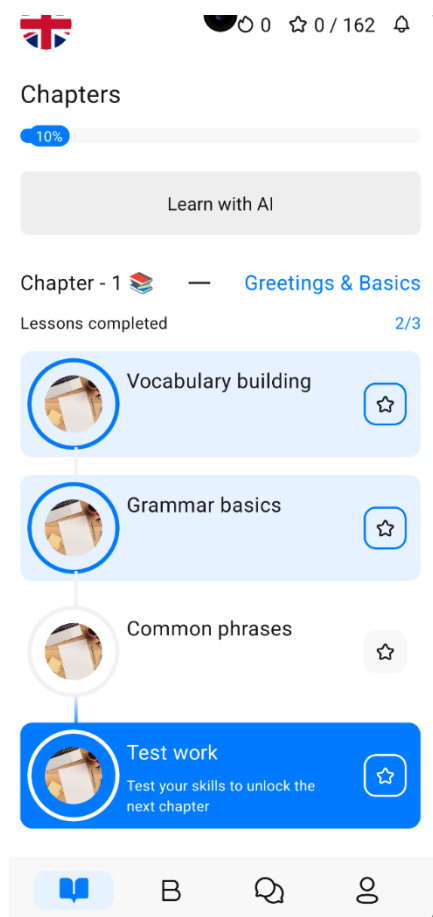


Рис. 3.6 – Головний екран навчального треку

Основою взаємодії зі *Speaker* є проходження інтерактивних лінгвістичних вправ (рис. 3.7). На екрані тестування відображається умова завдання та варіанти відповідей у вигляді клікабельних карток. Ключовою особливістю є механізм миттєвого зворотного зв'язку. При виборі неправильної відповіді система активує нижню модальну панель (Bottom Sheet) із червоним індикатором («Wrong»), автоматично виводить правильний контекст та пропонує перехід до наступного кроку.

Ключовою функціональною та методичною особливістю даного екрана є імплементація механізму миттєвого зворотного зв'язку (Immediate Feedback Loop). З точки зору сучасного педагогічного дизайну, своєчасна реакція системи на дії учня є критично важливою для формування правильних мовних патернів та запобігання закріпленню помилок у пам'яті.

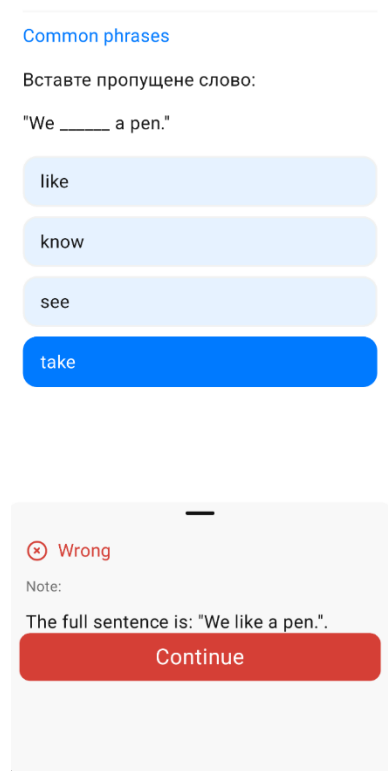


Рис. 3.7 – Екран виконання інтерактивної вправи з обробкою помилки

Після проходження всіх тестових завдань користувач потрапляє на екран результатів. У разі невдалого проходження з'являється екран із заголовком «Урок не пройдено», підбадьорливим текстом та відповідною ілюстрацією сумного персонажа (рис. 3.8). Користувачеві пропонується єдина кнопка «Try Again».

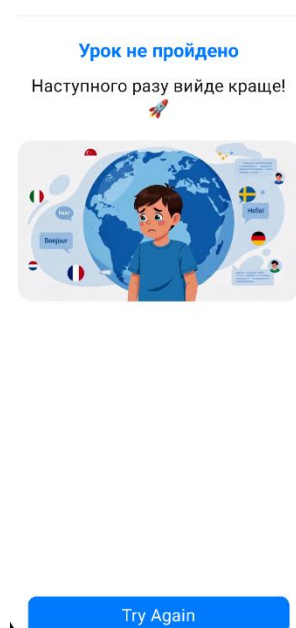


Рис. 3.8 – Екран неуспішного завершення урок

Якщо ж модуль пройдено успішно, інтерфейс кардинально змінюється на позитивний: з'являється радісна ілюстрація, заголовок «Урок завершено!» та кнопка «Close» (рис. 3.9). Цей візуальний контраст допомагає знизити стрес від помилок та стимулює вироблення дофаміну.



Рис. 3.9 – Екран успішного завершення уроку

Важливою складовою екосистеми є вбудований розумний словник (рис. 3.10). У верхній частині екрана розташовано аналітичні віджети з інтуїтивним кольоровим кодуванням: нові слова (червоний блок), у процесі вивчення (синій) та засвоєні (зелений). Основний контент організовано у вигляді списку тематичних категорій (Travel, Restaurant тощо).

Основний навчальний контент під аналітичним блоком логічно організовано у вигляді ієрархічного списку тематичних категорій (наприклад, «Travel», «Restaurant» тощо). Відмова від класичного алфавітного сортування на користь ситуативного групування базується на законах асоціативного запам'ятовування. Вивчення слів у контексті конкретних життєвих сценаріїв та єдиного семантичного поля значно підвищує ефективність їх засвоєння і готує користувача до практичного використання лексики в реальних комунікативних ситуаціях.

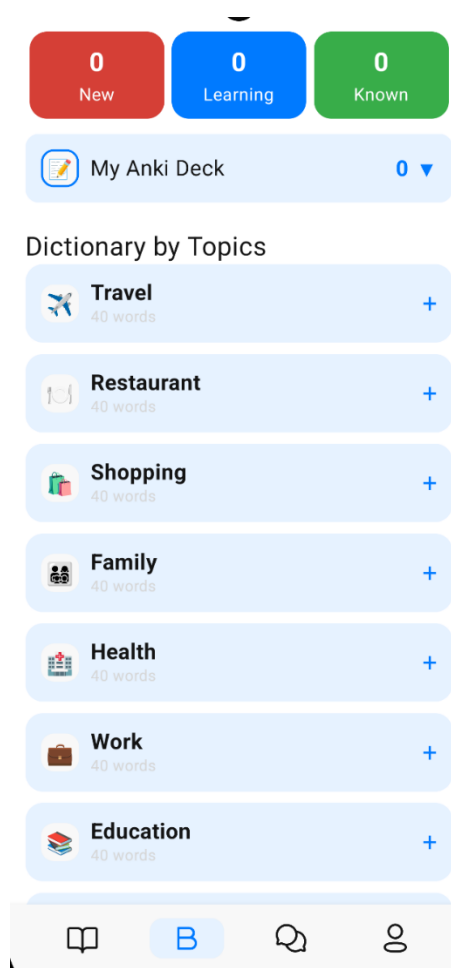


Рис. 3.10 – Структурований тематичний словник та панель статистики

Користувач може розгорнути будь-яку лексичну категорію для перегляду детального переліку слів (рис. 3.11). Кожна лексична одиниця супроводжується транскрипцією (ІРА), перекладом та контекстними прикладами використання у реченнях («Examples»).

Для забезпечення ефективної структуризації великих обсягів навчального контенту в застосунку реалізовано принцип поступового розкриття інформації (Progressive Disclosure). Користувач має інтуїтивну можливість розгорнути будь-яку обрану лексичну категорію для перегляду детального переліку слів. Використання ієрархічних списків, що розгортаються (Accordion UI), дозволяє максимально раціонально використовувати обмежений робочий простір дисплея мобільного пристрою та ефективно запобігає когнітивному перевантаженню учня надмірною кількістю даних на одному екрані.

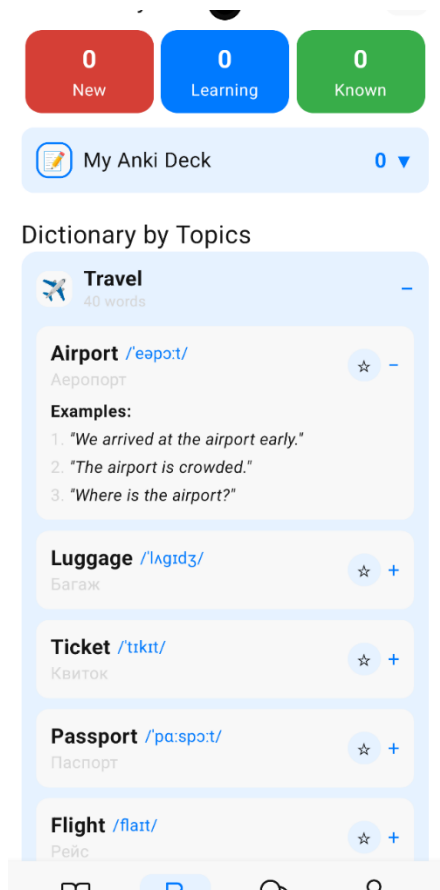


Рис. 3.11 – Розгорнута картка слова з фонетичною транскрипцією та прикладами використання

Однією з ключових інновацій застосунку є глибока інтеграція модуля штучного інтелекту (рис. 3.12). Інтерфейс екрана «AI Tutor» реалізовано у форматі месенджера. Розумний чат-бот ініціює діалог, а учень вільно вводить свій запит. ШІ миттєво адаптує сценарій бесіди, створюючи імерсивне мовне середовище для практики.

Технічна реалізація взаємодії з модулем базується на алгоритмах обробки природної мови (NLP) та відкритості діалогової системи. Розумний чат-бот здатний проактивно ініціювати бесіду та ставити контекстні запитання, тоді як учень не обмежений жорсткими рамками тестів і може вільно вводити свій запит природною мовою. Штучний інтелект миттєво аналізує семантику повідомлення, рівень підготовки користувача та динамічно адаптує подальший сценарій бесіди. Завдяки цьому генерується унікальний навчальний контент, який перетворює

стандартний додаток на імерсивне мовне середовище, що імітує реальне занурення у контекст та забезпечує високоефективну розмовну практику.

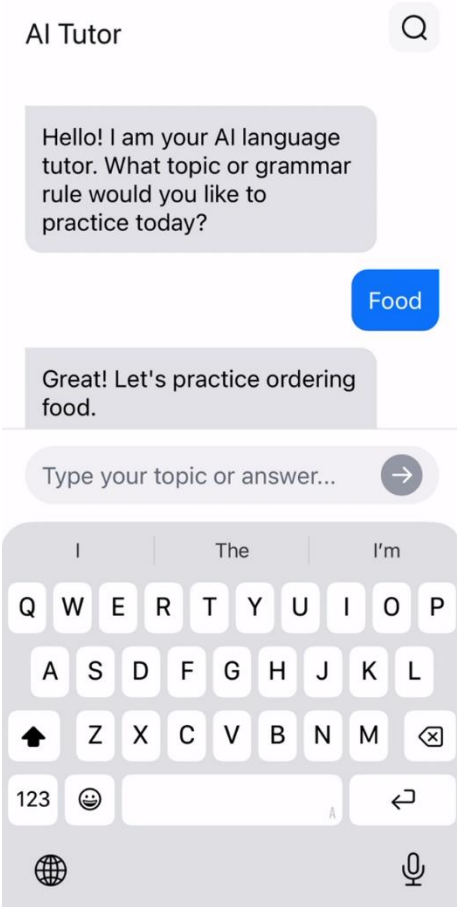


Рис. 3.12 – Інтерактивний чат зі штучним інтелектом

Для забезпечення соціальної взаємодії реалізовано модуль комунікації (рис. 3.13). Екран розділено на вкладки пошуку нових користувачів та списку друзів. Для кожного учасника відображається його рівень володіння мовою за шкалою CEFR (наприклад, B2 Upper-Int), індикатор прогресу та маркер мережевого статусу (онлайн).

З точки зору архітектури користувацького інтерфейсу, робочий простір екрана логічно розділено за допомогою системи вкладок (Tabs). Такий навігаційний патерн дозволяє ефективно структурувати інформацію, розмежовуючи інструменти пошуку нових користувачів (для розширення соціального графа та пошуку мовних партнерів) і список уже доданих друзів (для

швидкого доступу до активних зв'язків), не перевантажуючи при цьому візуальний простір.

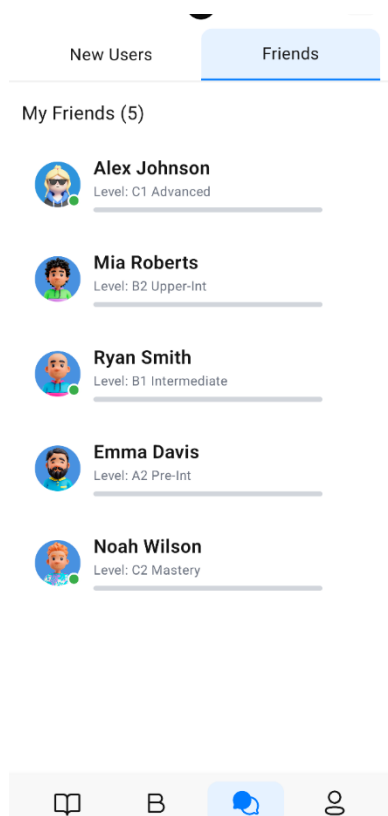


Рис. 3.13 – Соціальний модуль: список друзів та моніторинг їхнього прогресу

Особистий кабінет учня є центральним хабом для відстеження індивідуального прогресу (рис. 3.14). У вкладці прогресу візуалізовано кругову діаграму загального успіху та дашборди вивченої лексики. Також тут розміщено кнопку «Random lesson» для генерації випадкового уроку (інструмент інтервального повторення).

Окремої уваги заслуговує інтеграція в архітектуру кабінету кнопки «Random lesson», яка відповідає за динамічну генерацію випадкового уроку. З інженерної та педагогічної точок зору, цей функціонал є не просто елементом гейміфікації, а потужним інструментом, що базується на науково обґрунтованій методиці інтервального повторення (Spaced Repetition). Використання автоматизованого алгоритму випадкової вибірки пройденого матеріалу стимулює довгострокову пам'ять учня, ефективно протидіючи природному процесу

забування (так званий «кривій Еббінгауза»). Таким чином, мобільний застосунок бере на себе активну роль у закріпленні пройденої лексики та її надійному переведенні в активний словниковий запас користувача.

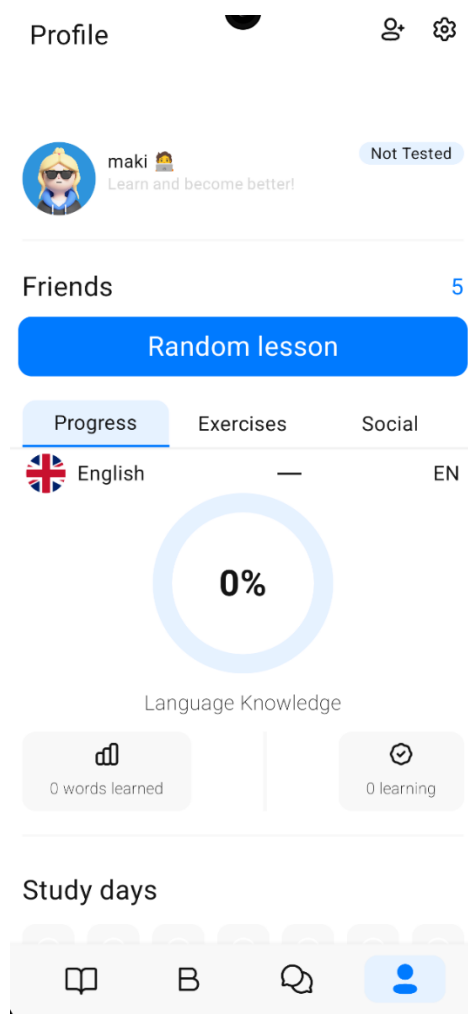


Рис. 3.14 – Особистий профіль користувача та статистика прогресу

Для забезпечення високого рівня гнучкості та персоналізації взаємодії з додатком, у системі передбачено спеціальний розділ конфігурації. Через стандартний графічний елемент управління іконку шестірні, яка є інтуїтивно зрозумілою та відповідає сучасним патернам проєктування UI/UX інтерфейсів, користувач отримує прямий доступ до головного меню налаштувань (рис. 3.15). Цей екран є важливим компонентом системи, оскільки він дозволяє адаптувати програмний продукт під індивідуальні потреби та вподобання кожного окремого клієнта.

Структурно дане меню логічно розділене на кілька тематичних категорій для забезпечення швидкої та зручної навігації. Зокрема, блок «Account»

виступає центральним вузлом для управління персональною інформацією. Він дозволяє користувачеві оперативно переглядати та редагувати власні дані профілю, підтримуючи таким чином актуальність інформації в системі.

Рис. 3.15 – Меню налаштувань облікового запису та параметрів застосунку

Процес перемикання між різними мовними курсами реалізовано максимально ергономічно (рис. 3.16). Замість перенаправлення на окремі екрани застосовується спливаюче модальне вікно (Bottom Sheet) із переліком доступних мов та відповідними прапорами.

Натомість для реалізації цієї функції застосовується інтерактивне спливаюче модальне вікно нижнього рівня (Bottom Sheet). Такий архітектурний підхід має низку суттєвих переваг з точки зору UX-дизайну. По-перше, використання Bottom Sheet дозволяє користувачеві здійснювати зміну курсу «на льоту», не втрачаючи візуального контакту з основним екраном, що робить процес навігації більш плавним та безперервним. По-друге, даний елемент

управління розташований у нижній частині екрана, що робить його ідеально адаптованим для комфортного використання однією рукою на сучасних мобільних пристроях із великою діагоналлю дисплея.

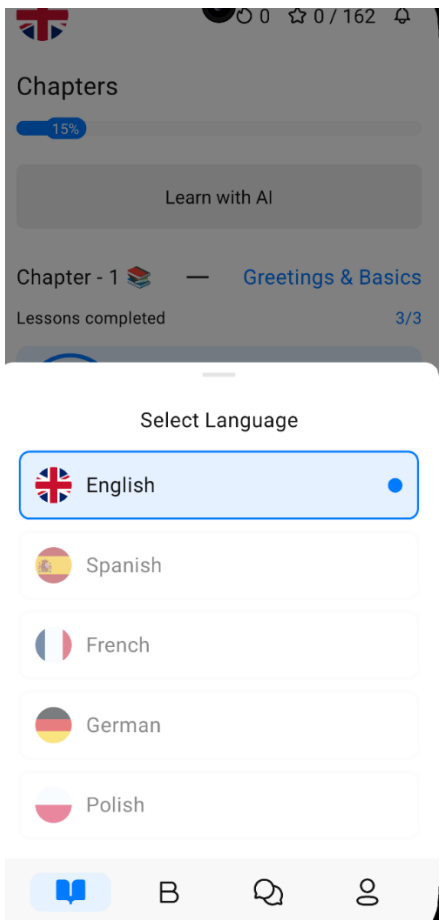


Рис. 3.16 – Інтерфейс вибору мови вивчення

Для повноцінного управління освітнім контентом розроблено спеціалізовану вебпанель адміністратора Speaker OS (рис. 3.17). Інтерфейс побудовано за класичним шаблоном дашборда з боковим навігаційним меню (Sidebar), яке надає доступ до модулів курсу, бази словника та профілів користувачів. У розділі «Course Modules» реалізовано зручну систему фільтрації за мовами (вкладки з прапорами) та ієрархічне відображення карток модулів для кожного рівня (наприклад, A1).

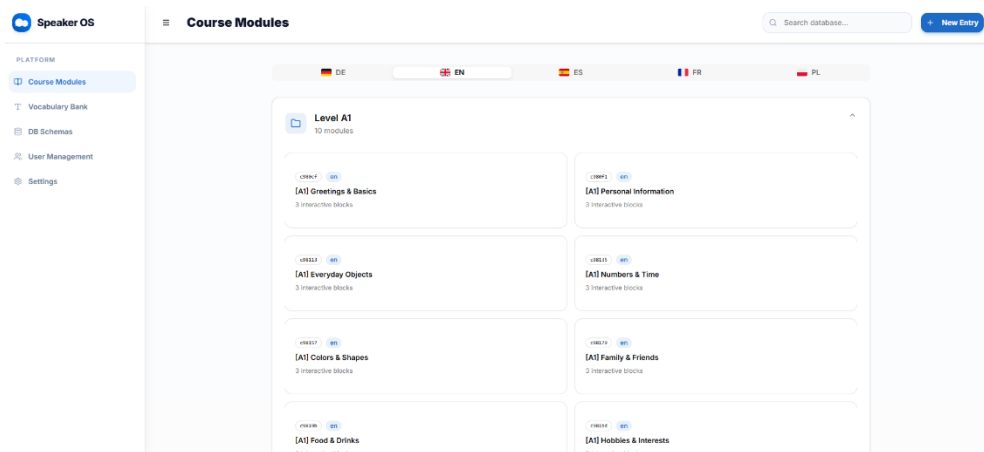


Рис. 3.17 – Вебпанель адміністратор: управління навчальними модулями

Процес створення або модифікації контенту відбувається через інтуїтивне модальне вікно редагування (рис. 3.18). Адміністратор може налаштувати базові метадані уроку (назву, мовний код, рівень складності) та керувати його внутрішньою структурою за допомогою функціональних блоків («Vocabulary building», «Grammar basics»). Такий візуальний конструктор дозволяє методологам платформи швидко оновлювати навчальну програму без прямого втручання в програмний код.

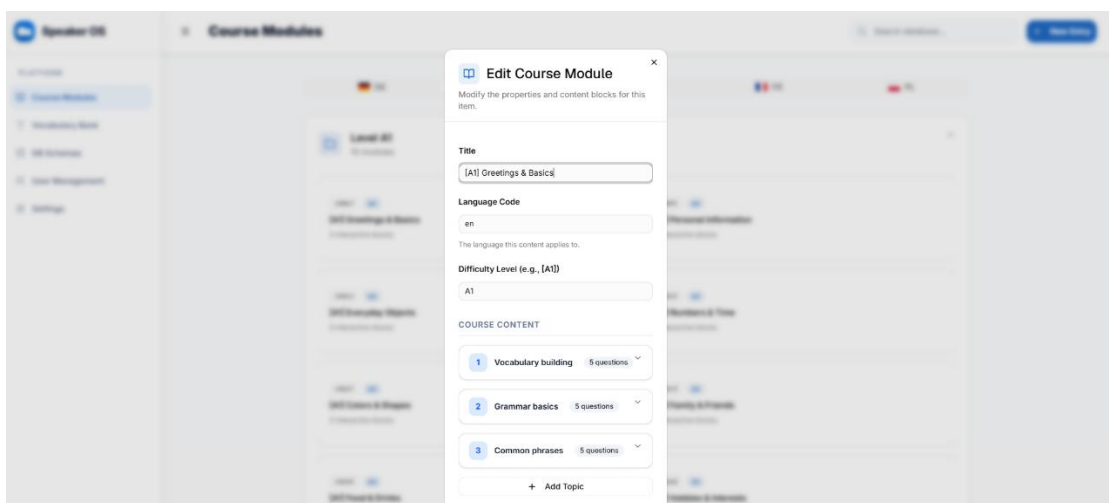


Рис. 3.18 – Інтерфейс редагування навчального модуля у вебпанелі

3.3 Тестування роботи програмного продукту

Тестування системи Speaker охоплює три взаємопов'язані рівні: модульні та інтеграційні тести серверної частини (Node.js) за допомогою фреймворку Jest і бібліотеки Supertest, модульні тести клієнтських інтерфейсів (мобільний

					ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк. 86
Змн.	Арк.	№ докум.	Підпис	Дата		

застосунок на React Native та вебпанель на React) з використанням React Testing Library, а також тестування взаємодії з базою даних через MongoDB Memory Server. Кожен рівень вирішує свою задачу і навмисно не дублює інші. Серверна частина покрита понад 70 модульними тестами, які успішно проходять під час кожної збірки (рис. 3.19). Клієнтська частина містить тестові набори, що охоплюють ключові UI-компоненти, хуки навігації та сторінки авторизації.

```
Test Suites: 2 passed, 2 total
Tests:      18 passed, 18 total
Snapshots:  0 total
Time:       2.786 s, estimated 21 s
Ran all test suites.
```

Рис. 3.19 – Результати виконання модульних тестів серверної частини

Доменні тести (перевірка бізнес-логіки) найважливіша частина серверного покриття. Саме вони гарантують, що ключові правила системи виконуються безпомилково, забезпечуючи стабільність ядра застосунку. Вони перевіряють інваріанти сутностей без прив'язки до HTTP-контролерів чи маршрутизації. Така архітектурна ізоляція дозволяє запускати їх значно швидше, уникаючи зайвих витрат ресурсів на імітацію мережевих запитів чи підняття повноцінного серверного середовища. Наприклад, система блокує створення профілю з некоректним форматом електронної пошти на рівні моделі Mongoose. Цей підхід є критично важливим для збереження суворої цілісності бази даних ще до моменту обробки запиту. Адміністратор не може опублікувати навчальний модуль, якщо в ньому немає жодного тестового завдання, що унеможливорює доступ користувачів до «сирого» або логічно незавершеного матеріалу. Призупинений (заблокований) контент гарантовано не віддається мобільному клієнту під час синхронізації (табл. 3.5). Завдяки глибокому доменному тестуванню формується надійна «сітка безпеки», яка дозволяє розробникам впевнено проводити рефакторинг коду та масштабувати функціонал без ризику появи регресійних помилок.).

Тести доменного рівня

Клас	Тест	Що перевіряється
1	2	3
AuthServiceTests	Register_InvalidEmail_ShouldThrowError	Некоректний формат email недопустимий під час створення облікового запису
AuthServiceTests	Login_WrongPassword_ShouldReturn401	Авторизація неможлива при невірному хеші пароля
LessonModelTests	Publish_WithoutExercises_ShouldThrow	Публікація порожнього уроку (без завдань) суворо заборонена
VocabularyTests	AddWord_DuplicateInDeck_ShouldIgnore	Подвійне додавання одного й того ж слова до словника ігнорується системою
ProgressTests	CompleteLesson_AlreadyCompleted_NoDoubleXP	Повторне нарахування балів (досвіду) за вже пройдений урок неможливе
AdminModerationTests	SuspendModule_ShouldHideFromClients	Призупинений або заблокований модуль ігнорується клієнтським API

Тестування API-маршрутів (контролерів) влаштовано інакше. Для цього використовується бібліотека Supertest, яка дозволяє симулювати HTTP-запити до екземпляра Express-застосунку. Перевіряється, що запит на отримання уроків правильно фільтрує контент за обраною мовою, та що аналітичний дашборд повертає достовірні відсотки успішності (табл. 3.6).

Таблиця 3.6

Тестові класи API-контролерів

Тестовий набір	Кількість тестів	Що охоплює
AuthController.test.ts	15	Реєстрація, авторизація, валідація JWT-токенів, обробка помилок входу
LessonController.test.ts	24	Отримання списку уроків, фільтрація за мовним кодом, публікація, блокування
ProgressController.test.ts	12	Синхронізація загальної статистики, оновлення «стриків» (streaks), нарахування балів

Тести клієнтської частини написані на Jest з використанням екосистеми React Testing Library (зокрема RNTL для мобільного клієнта). Ключовий елемент тестування захищені маршрути (Protected Routes) та глобальний стан користувача. Перевіряється, що неавтентифікований учень примусово перенаправляється на екран входу («Log in»), а авторизований безперешкодно бачить свій навчальний трек. Окремо тестується поведінка інтерфейсу: наприклад, поява червоного модального вікна (Bottom Sheet) при виборі неправильної відповіді у вікторині (рис. 3.20, табл. 3.7).

```

pp> npm run test -- --watchAll=false

> speaker-app@1.0.0 test
> jest --watchAll --watchAll=false

PASS  __tests__/VocabularyList.test.tsx
PASS  __tests__/useAuth.test.ts
PASS  __tests__/QuizEngine.test.tsx
PASS  __tests__/LanguageSelector.test.tsx
PASS  __tests__/AuthNavigator.test.tsx

Test Suites: 5 passed, 5 total
Tests:      10 passed, 10 total
Snapshots:  0 total
Time:       1.071 s, estimated 4 s
Ran all test suites.

```

Рис. 3.20 – Результати виконання тестів клієнтської частини

Таблиця 3.7

Тестові файли клієнтської частини

Файл	Що перевіряється
AuthNavigator.test.tsx	Захист маршрутів: анонімний стан проти авторизованого профілю
LanguageSelector.test.tsx	Відмальовування списку мов у модальному вікні та збереження вибору
QuizEngine.test.tsx	Логіка валідації відповідей, показ модального вікна помилки (Bottom Sheet)
VocabularyList.test.tsx	Рендеринг списку слів, розгортання карток з фонетичною транскрипцією
useAuth.test.ts	Робота кастомного хука авторизації та безпечне збереження JWT у сховищі

Інтеграційні тести з базою даних останній рівень надійності. Інструмент MongoDB Memory Server при кожному запуску тестового циклу створює чисту базу даних у оперативній пам'яті та застосовує актуальні Mongoose-схеми. Кожен тест отримує ізольовану порожню базу залишки від попередніх операцій не впливають на результат. Перевіряється збереження складних вкладених JSON-

структур (наприклад, масивів запитань усередині модулів) та коректність роботи агрегаційних конвеєрів (Aggregation Framework) для збору персональної статистики.

```
===== Coverage summary =====
=====
Statements   : 34.67% ( 207/597 )
Branches     : 16.82% ( 35/208 )
Functions    : 21.05% ( 8/38 )
Lines        : 34.15% ( 195/571 )
=====

Test Suites: 2 passed, 2 total
Tests:       18 passed, 18 total
Snapshots:   0 total
Time:        5.42 s
Ran all test suites.
Force exiting Jest: Have you considered using `--detectOpenHandles`
to detect async operations that kept running after all tests finished?
```

Рис. 3.21 – Результати виконання інтеграційних тестів

Зведені дані щодо обраних інструментів та результатів покриття коду зібрано нижче (табл. 3.8).

Таблиця 3.8

Загальне покриття тестами

Рівень	Інструмент	Результат
Бізнес-логіка (Моделі)	Jest	Пройдено
API-контролери	Jest + Supertest	Пройдено
Компоненти клієнта	Jest + React Testing Library	Пройдено
Інтеграція з БД	MongoDB Memory Server	Пройдено

Загалом для перевірки працездатності проєкту написано понад 100 тестів. Такий комбінований підхід дозволив протестувати систему з різних боків. Серверна частина перевірялася фреймворком Jest. Щоб тестувати контролери ізольовано від хмарної БД, але максимально наближено до реальності,

залежності емулювалися через MongoDB Memory Server, а запити генерувалися через Supertest. На клієнтській стороні застосовано зв'язку Jest та Testing Library. Цей набір перевірів правильність відмальовування UI-компонентів, роботу інтерактивних вправ та надійність захисту роутів. Оскільки під час прогону жоден тест не видав критичної помилки, поточну версію системи можна вважати стабільною та готовою до експлуатації.

Висновки до третього розділу

Програмний продукт розгортається у сучасній хмарній інфраструктурі. Архітектуру платформи Speaker розділено на незалежні компоненти: серверна частина (Node.js/Express бекенд), мобільний клієнт (React Native), адміністративна вебпанель (React) та документо-орієнтована база даних (MongoDB). Щоб автоматизувати та пришвидшити випуск оновлень, налаштовано CI/CD пайплайн за допомогою GitHub Actions. Кожен коміт у головну гілку репозиторію автоматично ініціює перевірку якості стилю (Linting), запуск тестів та збірку коду. Після успішного проходження пайплайну нова версія бекенду розгортається на хмарній платформі Render, а вебпанель на Vercel (або аналогічному хостингу). Для комфортної розробки на локальній машині передбачено конфігурацію Docker Compose, яка дозволяє швидко ізольовано запустити екземпляр бази даних.

На стороні клієнтських застосунків реалізовано чітке розмежування доступу. Інтерфейс та його функціонал адаптуються залежно від ролі: звичайний користувач (учень) взаємодіє з мобільним застосунком для проходження уроків та ведення словника, тоді як Адміністратор використовує вебпанель Speaker OS для глобального управління контентом, аналітики та модерації навчальних матеріалів. Завдяки цьому екосистему вдалося логічно розділити на зону споживання контенту та зону його адміністрування.

Якість коду підтверджено багаторівневим тестуванням. Серверна частина вкрита модульними тестами для перевірки бізнес-правил та ізольованої логіки валідації. На клієнтських платформах тести контролюють навігаційні маршрути та реакції UI-компонентів. Найважливіший етап інтеграційне тестування API за

					ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
						92
Змн.	Арк.	№ докум.	Підпис	Дата		

допомогою інструменту MongoDB Memory Server, що дозволяє перевірити реальну роботу контролерів із базою даних у швидкому віртуальному середовищі без створення мережових запитів. Увесь тестовий набір виконується успішно, що доводить надійність обраних архітектурних рішень.

					ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
						93
Змн.	Арк.	№ докум.	Підпис	Дата		

ВИСНОВКИ

Результатом виконання дипломної роботи є створена інформаційна екосистема *Speaker*. Система призначена для інтерактивного вивчення іноземних мов та комплексного моніторингу успішності користувачів. Вона переводить у сучасний цифровий формат увесь освітній цикл: від базового ознайомлення з лексикою за допомогою системи інтервальних повторень до фінального тестування знань та розмовної практики зі штучним інтелектом.

Огляд існуючого програмного забезпечення виявив низку проблем у цій ніші. Наявні на ринку аналоги або є надмірно монетизованими з агресивною рекламою, або мають застарілий та перевантажений інтерфейс, який швидко знижує мотивацію учнів до навчання. Крім того, більшість розглянутих сервісів не має гнучкого механізму управління контентом із чітким поділом архітектури на мобільний клієнт для користувача та адміністративну вебпанель для модератора. Це стало головним аргументом для розробки власного модульного рішення.

Бекенд проєкту базується на сучасних патернах проєктування із використанням екосистеми *Node.js/Express*. В основу бізнес-логіки покладено гнучку документо-орієнтовану модель на базі NoSQL-сховища *MongoDB*. Інфраструктура бази даних надійно контролює збереження прогресу, синхронізацію щоденних «стриків» (*streaks*) та захищає систему від некоректних дій (наприклад, спроб штучного накручування рейтингу), незалежно від джерела HTTP-запиту. Завдяки сучасному підходу до розгортання, релізи нових версій серверної частини та вебпанелі виконуються повністю автоматично через налаштований CI/CD-пайплайн у хмарних сервісах.

Головна практична цінність створеної системи полягає у формуванні єдиного централізованого освітнього хабу. Учень отримує безшовний, гейміфікований досвід вивчення мови у зручному мобільному форматі (*React Native*), маючи під рукою вікторини, розумний словник та AI-репетитора. Водночас адміністратор проєкту може в реальному часі створювати, редагувати або блокувати навчальні модулі через спеціалізовану вебпанель *Speaker OS*

					ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
						94
Змн.	Арк.	№ докум.	Підпис	Дата		

(React), без необхідності випускати оновлення застосунку в маркетах. Стабільність платформи доведено на практиці: для перевірки коду написано понад 100 тестів, які успішно покривають бізнес-правила моделей, інтеграційний рівень API-контролерів та клієнтські компоненти інтерфейсу.

					ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
						95
Змн.	Арк.	№ докум.	Підпис	Дата		

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. React Native: A framework for building native apps using React : офіційна документація. URL: <https://reactnative.dev> (дата звернення: 06.06.2026).
2. Pautasso C., Zimmermann O., Leymann F. Restful web services vs. big web services: making the right architectural decision. Proceedings of the 17th International Conference on World Wide Web. 2008. P. 805–814. DOI: 10.1145/1367497.1367606.
3. Biørn-Hansen A., Grønli T. M., Ghinea G. A taxonomy and survey of cross-platform mobile app development. IEEE Transactions on Software Engineering. 2018. Vol. 45, No 9. P. 841–854. DOI: 10.1109/TSE.2018.2804446.
4. Node.js: JavaScript runtime built on Chrome's V8 JavaScript engine : офіційна документація. URL: <https://nodejs.org> (дата звернення: 06.06.2026).
5. TypeScript: Typed JavaScript at Any Scale : офіційна документація. URL: <https://www.typescriptlang.org> (дата звернення: 06.06.2026).
6. PostgreSQL: The World's Most Advanced Open Source Relational Database : офіційна документація. URL: <https://www.postgresql.org> (дата звернення: 06.06.2026).
7. Common European Framework of Reference for Languages: Learning, teaching, assessment : електронний ресурс / Council of Europe. URL: <https://www.coe.int/en/web/common-european-framework-reference-languages> (дата звернення: 06.06.2026).
8. Roll I., Wylie R. Evolution and revolution in artificial intelligence in education. International Journal of Artificial Intelligence in Education. 2016. Vol. 26, No 2. P. 582–599. DOI: 10.1007/s40593-016-0110-3.
9. Chung L., Nixon B. A., Yu E. Non-functional requirements in software engineering : monograph. Boston : Springer, 2000. 476 p.
10. Martin R. C. Clean Architecture: A Craftsman's Guide to Software Structure and Design. Boston : Prentice Hall, 2017. 432 p.
11. Smith J. Patterns - WPF Apps With The Model-View-ViewModel Design Pattern. MSDN Magazine. 2009. Vol. 24, No 2. P. 1–12.

					ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
						96
Змн.	Арк.	№ докум.	Підпис	Дата		

12. Dinu V., Nadkarni P. Guidelines for the effective use of entity-attribute-value modeling for biomedical databases. International Journal of Medical Informatics. 2007. Vol. 76, No 11. P. 769–779. DOI: 10.1016/j.ijmedinf.2006.09.023.
13. Survey on NoSQL database / J. Han, E. Hailong, G. Le, J. Jian. Proceedings of the 6th International Conference on Pervasive Computing and Applications. 2011. P. 363–366. DOI: 10.1109/ICPCA.2011.6106531.
14. SQLite: Small. Fast. Reliable. Choose any three : офіційна документація. URL: <https://www.sqlite.org> (дата звернення: 06.06.2026).
15. Codd E. F. A relational model of data for large shared data banks. Communications of the ACM. 1970. Vol. 13, No 6. P. 377–387. DOI: 10.1145/362384.362685.
16. Firebase: Build and run successful apps : офіційна документація. URL: <https://firebase.google.com> (дата звернення: 06.06.2026).
17. Firebase Authentication : електронний ресурс. URL: <https://firebase.google.com/docs/auth> (дата звернення: 06.06.2026).
18. Cloudinary: Image and Video Upload, Storage, Optimization : електронний ресурс. URL: <https://cloudinary.com> (дата звернення: 06.06.2026).
19. Render: Cloud Application Hosting for Developers : офіційна документація. URL: <https://render.com> (дата звернення: 06.06.2026).
20. Cloudflare: The Web Performance & Security Company : електронний ресурс. URL: <https://www.cloudflare.com> (дата звернення: 06.06.2026).
21. How do software developers use GitHub actions to automate their workflows? / N. Kinsman, M. Wessel, M. A. Gerosa, C. Treude. Empirical Software Engineering. 2021. Vol. 26, No 4. P. 1–35. DOI: 10.1007/s10664-021-09956-1.
22. Bierman G., Abadi M., Torgersen M. Understanding typescript. European Conference on Object-Oriented Programming (ECOOP). 2014. P. 257–281. DOI: 10.1007/978-3-662-44202-9_11.
23. Hanssen T., Sjøberg D. I. K. Evaluating cross-platform mobile development using React Native. Proceedings of the 12th ACM/IEEE International

					ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
						97
Змн.	Арк.	№ докум.	Підпис	Дата		

- Symposium on Empirical Software Engineering and Measurement. 2018. P. 1–4. DOI: 10.1145/3239235.3267428.
24. Tilkov S., Vinoski S. Node.js: Using JavaScript to build high-performance network programs. IEEE Internet Computing. 2010. Vol. 14, No 6. P. 80–83. DOI: 10.1109/MIC.2010.145.
25. Express: Fast, unopinionated, minimalist web framework for Node.js : електронний ресурс. URL: <https://expressjs.com> (дата звернення: 06.06.2026).
26. Visual Studio Code: Code editing. Redefined : офіційна документація. URL: <https://code.visualstudio.com> (дата звернення: 06.06.2026).
27. PlantUML: Open-source tool that uses simple textual descriptions to draw UML diagrams : електронний ресурс. URL: <https://plantuml.com> (дата звернення: 06.06.2026).
28. Chacon S., Straub B. Pro Git. 2nd ed. New York : Apress, 2014. 456 p.
29. MongoDB: The Developer Data Platform : офіційна документація. URL: <https://www.mongodb.com> (дата звернення: 06.06.2026).
30. Stripe API Reference : офіційна документація. URL: <https://stripe.com/docs/api> (дата звернення: 06.06.2026).
31. Hardt D. The OAuth 2.0 Authorization Framework : RFC 6749. Internet Engineering Task Force. 2012. 76 p. URL: <https://datatracker.ietf.org/doc/html/rfc6749> (дата звернення: 06.06.2026).
32. Zustand: Bear necessities for state management in React : електронний ресурс. URL: <https://github.com/pmndrs/zustand> (дата звернення: 06.06.2026).
33. Garousi V., Felderer M. Worlds apart: industrial and academic focus areas in software testing. IEEE Software. 2017. Vol. 34, No 5. P. 38–45. DOI: 10.1109/MS.2017.3571565.
34. Jest: Delightful JavaScript Testing : офіційна документація. URL: <https://jestjs.io> (дата звернення: 06.06.2026).

					ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
						98
Змн.	Арк.	№ докум.	Підпис	Дата		

35. SuperTest: Node.js library for testing HTTP servers : електронний ресурс.
URL: <https://github.com/ladjs/supertest> (дата звернення: 06.06.2026).
36. React Testing Library : офіційна документація. URL: <https://testing-library.com/docs/react-testing-library/intro> (дата звернення: 06.06.2026).
37. MongoDB Memory Server for Node.js : електронний ресурс.
URL: <https://github.com/nodkz/mongodb-memory-server> (дата звернення: 06.06.2026).
38. Mongoose: Elegant MongoDB object modeling for Node.js : офіційна документація. URL: <https://mongoosejs.com> (дата звернення: 06.06.2026).
39. Jones M., Bradley J., Sakimura N. JSON Web Token (JWT) : RFC 7519. Internet Engineering Task Force. 2015. 30 р.
URL: <https://datatracker.ietf.org/doc/html/rfc7519> (дата звернення: 06.06.2026).
40. GitHub Actions Documentation : офіційний сайт.
URL: <https://docs.github.com/en/actions> (дата звернення: 06.06.2026).
41. Vercel: Develop. Preview. Ship : офіційний сайт.
URL: <https://vercel.com> (дата звернення: 06.06.2026).
42. Merkel D. Docker: lightweight linux containers for consistent development and deployment. Linux Journal. 2014. Vol. 2014, No 239. P. 2.
URL: <https://dl.acm.org/doi/10.5555/2600239.2600241> (дата звернення: 06.06.2026).

ДОДАТКИ

					ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
						100
Змн.	Арк.	№ докум.	Підпис	Дата		

Додаток А

Діаграма варіантів використання

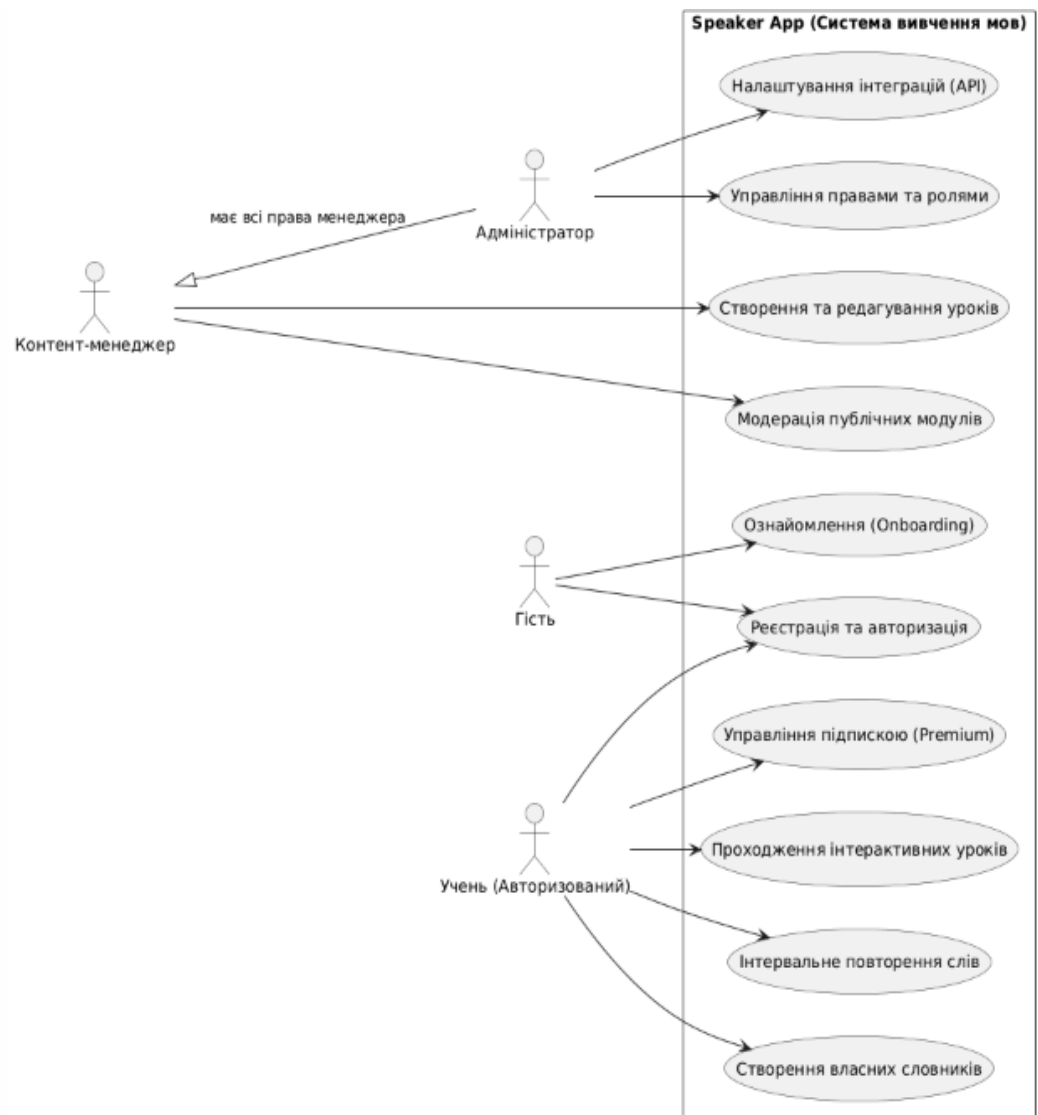


Рис. А.1 – Діаграма варіантів використання

Додаток Б

Діаграма компонентів

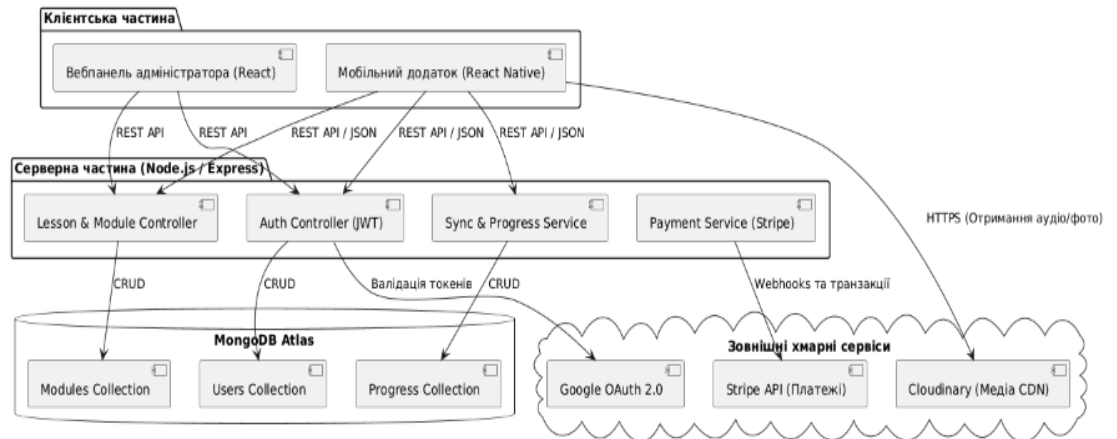


Рис. Б.1 – Діаграма компонентів

Додаток В

Діаграма шару застосунку (Application Layer)

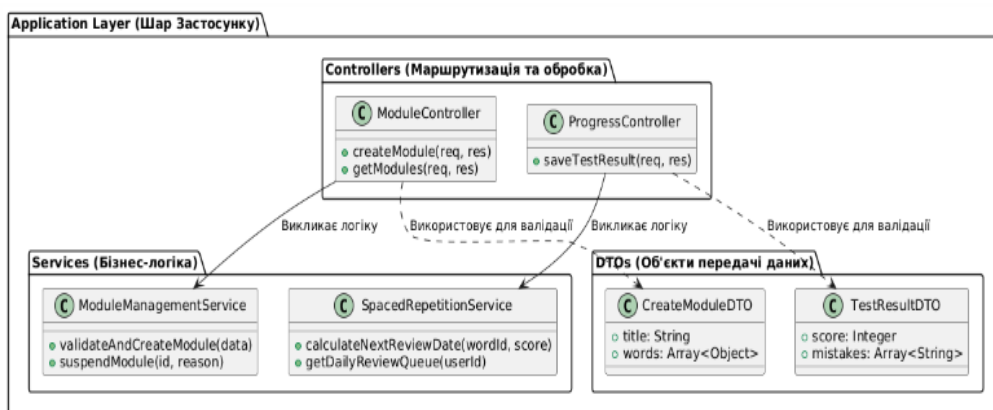


Рис. В.1 – Діаграма шару застосунку

Додаток Г

Діаграма шару інфраструктури (Infrastructure Layer Diagram)

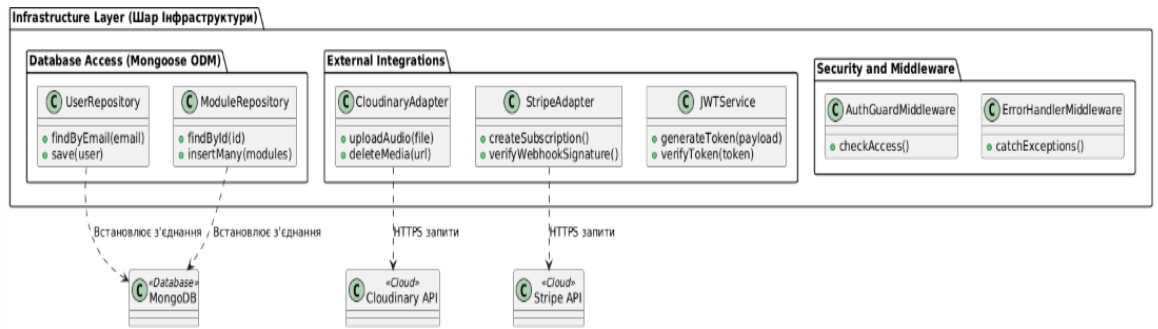


Рис. Г.1 – Діаграма шару інфраструктури

Додаток Д

Діаграма послідовності подання результатів тесту

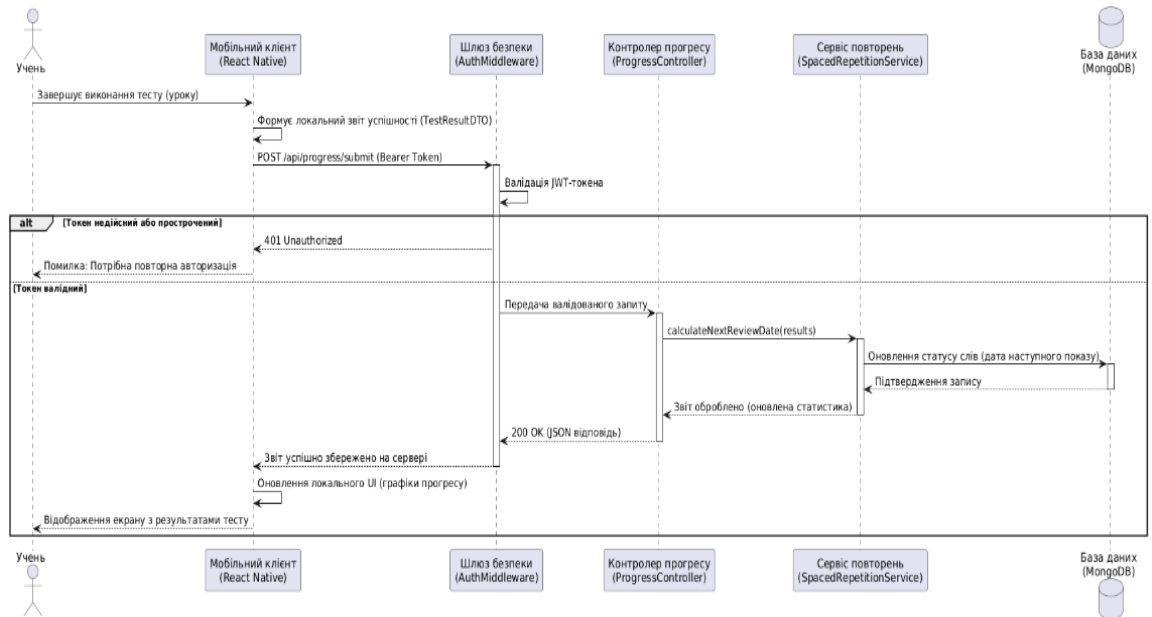


Рис. Д.1 – Діаграма послідовності подання звіту

Додаток Е

Діаграма послідовності рецензування модулю

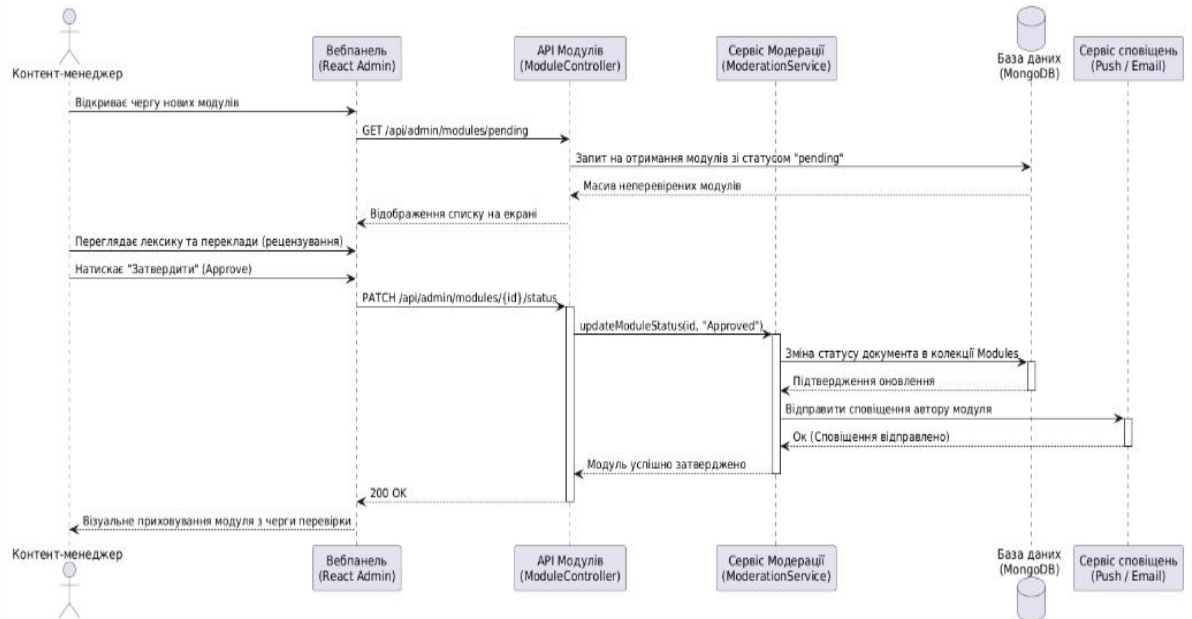


Рис. Е.1 – Діаграма послідовності рецензування звіту

Фрагменти коду серверної частини

Ж.1 Команди створення навчального модуля

```
export interface CreateModuleDto
{
    userId: string;

    title: string;

    category: string;

    isPublic: boolean;

    words: Array<{
        original: string;

        translation: string;

        transcription?: string;
    }>;
}
```

Ж.2 Обробник (Service) логіки збереження навчального модуля

```
using System.Reflection;
using static System.Runtime.InteropServices.JavaScript.JSType;

export class ModuleService
{
    public async createModule(data: CreateModuleDto): Promise<IModule> {
        // Валідація бізнес-правил
        if (!data.title || data.words.length === 0) {
            throw new ValidationException('Модуль повинен мати назву та мінімум одне слово.');
```

```
        }

        // Створення нового об'єкта в колекції бази даних
        const newModule = new ModuleModel({
            author: data.userId,
            title: data.title,
            category: data.category,
            isPublic: data.isPublic,
            words: data.words,
            createdAt: new Date(),
        });

        await newModule.save();
```

					ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
						107
Змн.	Арк.	№ докум.	Підпис	Дата		

```

return newModule;
}
}

```

Ж.3 Валідатор запиту на створення модуля (схема Zod)

```

import { z } from 'zod';
export const createModuleSchema = z.object({
  body: z.object({
    title: z.string().min(3, 'Назва має містити щонайменше 3 символи'),
    category: z.string().min(2),
    isPublic: z.boolean().default(false),
    words: z.array(
      z.object({
        original: z.string().min(1, 'Слово є обов\'язковим'),
        translation: z.string().min(1, 'Переклад є обов\'язковим'),
        transcription: z.string().optional(),
      })
    ).min(1, 'Необхідно додати хоча б одне слово'),
  }),
});

```

Ж.4 Метод контролера для обробки HTTP-запиту створення модуля

```

using static System.Net.Mime.MediaTypeNames;
using static System.Runtime.InteropServices.JavaScript.JSType;

import { Request, Response, NextFunction } from 'express';
export class ModuleController
{
  constructor(private moduleService: ModuleService) {}
  public createModuleHandler = async(
    req: Request,
    res: Response,
    next: NextFunction
  ) : Promise<void> => {
    try {
      const userId = req.user.id; // Отримується з JWT middleware після
авторизації
      const moduleData: CreateModuleDto = { ...req.body, userId
    };
    const createdModule = await this.moduleService.createModule(moduleData);

    res.status(201).json({
      success: true,
      data: createdModule,
    });
  } catch (error) {
    next(error); // Передача помилки в глобальний обробник
(ErrorHandlerMiddleware)
  }
};
}

```

					ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
						108
Змн.	Арк.	№ докум.	Підпис	Дата		

Тестовий клас для перевірки доменного рівня (Unit Tests)

```

import { ModuleService } from '../src/services/ModuleService';
import { ValidationError } from '../src/exceptions/ValidationError';

describe('ModuleService - Unit Tests', () => {
  let moduleService: ModuleService;

  beforeEach(() => {
    moduleService = new ModuleService();
  });

  it('повинен викидати помилку, якщо масив слів порожній', async () => {
    const invalidData = {
      userId: 'user-123',
      title: 'English Basics',
      category: 'General',
      isPublic: true,
      words: [], // Порожній масив слів порушує бізнес-логіку
    };

    await
    expect(moduleService.createModule(invalidData)).rejects.toThrow(ValidationError)
    ;
    await expect(moduleService.createModule(invalidData)).rejects.toThrow(/
    мінімум одне слово /);
  });

  it('повинен викидати помилку, якщо відсутня назва модуля', async () => {
    const invalidData = {
      userId: 'user-123',
      title: '', // Порожня назва
      category: 'General',
      isPublic: false,
      words: [{ original: 'Hello', translation: 'Привіт' }],
    };

    await
    expect(moduleService.createModule(invalidData)).rejects.toThrow(ValidationError)
    ;
  });
});

```

					ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		109

Тестовий клас обробника API маршрутів (Integration Tests)

```

import request from 'supertest';
import app from '../src/app';
import { generateMockToken } from '../helpers/jwtHelper';

describe('Module API Endpoints', () => {
  const userToken = generateMockToken({ id: 'user-123', role: 'user' });

  it('повинен успішно створювати новий модуль при валідних даних', async () => {
    const newModulePayload = {
      title: 'Advanced IT Vocabulary',
      category: 'IT',
      isPublic: false,
      words:
        [
          { original: 'Deployment', translation: 'Розгортання' }
        ]
    };

    const response = await request(app)
      .post('/api/modules')
      .set('Authorization', `Bearer ${userToken}`) // Симуляція авторизованого запиту
      .send(newModulePayload);

    expect(response.status).toBe(201);
    expect(response.body.success).toBe(true);
    expect(response.body.data.title).toBe('Advanced IT Vocabulary');
    expect(response.body.data.words).toHaveLength(1);

    it('повинен повертати помилку 401 (Unauthorized) без токена', async () => {
      const response = await request(app)
        .post('/api/modules')
        .send({ title: 'Test Module' });

      expect(response.status).toBe(401);
      expect(response.body.message).toMatch(/необхідна авторизація / i);

      it('повинен повертати помилку 400 (Bad Request) при невірній схемі', async () => {
        const invalidPayload = {

```

					ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
						110
Змн.	Арк.	№ докум.	Підпис	Дата		

```

        category: 'IT', // Пропущено обов'язкове поле 'title'
        words: []
    };

const response = await request(app)
    .post('/api/modules')
    .set('Authorization', `Bearer ${userToken}`)
    .send(invalidPayload);

expect(response.status).toBe(400);
expect(response.body.errors[0].message).toMatch(/ Назва має містити /);
});
});

```

					ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
						111
Змн.	Арк.	№ докум.	Підпис	Дата		