**МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ**

**Чорноморський національний університет імені Петра Могили**

**Факультет комп'ютерних наук**

**Кафедра інтелектуальних інформаційних систем**

ДОПУЩЕНО ДО ЗАХИСТУ

Завідувач кафедри інтелектуальних інформаційних систем

_____ Євген СІДЕНКО

«____»_____ 2026 р.

КВАЛІФІКАЦІЙНА РОБОТА

НА ЗДОБУТТЯ ОСВІТНЬОГО СТУПЕНЯ БАКАЛАВРА

# РОЗРОБКА ВЕБПЛАТФОРМИ ДЛЯ ВИВЧЕННЯ ІНОЗЕМНИХ МОВ ІЗ ВИКОРИСТАННЯМ ІНТЕЛЕКТУАЛЬНИХ ТЕХНОЛОГІЙ

Спеціальність 122 Комп'ютерні науки

Освітня програма «Комп'ютерні науки»

***Здобувач*** _____ Дмитро ЖУРАВСЬКИЙ

«___»_____ 2026 р.

***Керівник*** канд. фіз.-мат. наук, доцент _____ Інесса КУЛАКОВСЬКА

«___»_____ 2026 р.

**Миколаїв – 2026**

Чорноморський національний університет імені Петра Могили
(повне найменування закладу вищої освіти)

| | |
|---|---|
| Факультет | Комп’ютерних наук |
| Кафедра | Інтелектуальних інформаційних систем |
| Рівень вищої освіти | Перший (бакалаврський) |
| Освітній ступень | Бакалавр |
| Спеціальність | 122 Комп’ютерні науки |
| Освітня програма | Комп’ютерні науки |

ЗАТВЕРДЖУЮ

Завідувач кафедри інтелектуальних інформаційних систем

_____ Євген СІДЕНКО

«____» _____ 2025 р.

**ЗАВДАННЯ**
**на кваліфікаційну роботу здобувача**

**Журавського Дмитра Сергійовича**
((прізвище, ім’я, по батькові здобувача)

1. Тема кваліфікаційної роботи: «Розробка вебплатформи для вивчення іноземних мов із використанням інтелектуальних технологій».

Керівник роботи: Кулаковська Інесса Василівна, доцент кафедри інтелектуальних інформаційних систем, кандидат фіз.-мат. наук.

Затверджена наказом ЧНУ ім. Петра Могили від «25» грудня 2025 р. № 353.

2. Строк представлення кваліфікаційної роботи «___» _____ 2026 р.

3. Очікуваний результат роботи та початкові дані, якщо такі потрібні: розроблена інтелектуальна вебплатформа для вивчення іноземних мов з функціями гейміфікації, інтерактивним конструктором уроків та ансамблем моделей штучного інтелекту для автоматичного аналізу помилок і генерації адаптивного контенту. Початковими даними є вимоги до освітньої системи, результати аналізу існуючих аналогів на ринку та обрані засоби програмної розробки.

4. Перелік питань, що підлягають розробці: аналіз існуючих програмних рішень для дистанційного вивчення іноземних мов; вибір інструментів і технологій розробки системи; проєктування структури бази даних, архітектури програмного забезпечення та механізмів інтеграції NLP-модулів; реалізація функціональних можливостей освітньої платформи (кабінети викладача та студента, блоковий конструктор уроків, аналітичний дашборд); тестування системи та перевірка ефективності алгоритмів аналізу даних.

5. Перелік графічних матеріалів: презентація, рисунки, таблиці.

**Керівник роботи** _____ Інесса КУЛАКОВСЬКА
*(Особистий підпис)* *(Власне ім'я ПРІЗВИЩЕ)*

**Здобувач** _____ Дмитро ЖУРАВСЬКИЙ
*(Особистий підпис)* *(Власне ім'я ПРІЗВИЩЕ)*

Дата видачі завдання «24» грудня 2025 р.

# КАЛЕНДАРНИЙ ПЛАН
**кваліфікаційної роботи**

Тема: Розробка вебплатформи для вивчення іноземних мов із використанням інтелектуальних технологій

| № | Найменування роботи | Початок | Закінчення | Примітки |
|---|---|---|---|---|
| 1 | Отримання завдання на виконання КР | 21.12.2025 | 24.12.2025 | |
| 2 | Аналіз предметної області та постановка задачі | 25.12.2025 | 30.01.2026 | Виконано |
| 3 | Проведення системного аналізу наукових праць та комерційних рішень у сфері електронного навчання дослідження методологій застосування штучного інтелекту в освітньому процесі. | 31.01.2026 | 02.03.2026 | Виконано |
| 4 | Визначення та теоретичне обґрунтування стеку вебтехнологій і засобів машинного навчання, необхідних для створення багаторівневої освітньої екосистеми. | 02.03.2026 | 06.04.2026 | Виконано |
| 5 | Практична розробка архітектури та програмного коду платформи, інтеграція NLP-сервісів. | 02.04.2026 | 24.05.2026 | Виконано |
| 6 | Перший попередній захист КР на засіданні комісії кафедри | 25.05.2026 | 25.05.2026 | Виконано |
| 7 | Корегування роботи за результатами попереднього захисту | 26.05.2026 | 04.06.2026 | Виконано |
| 8 | Другий попередній захист КР на засіданні комісії кафедри | 05.06.2026 | 05.06.2026 | Виконано |
| 9 | Доробка та остаточне оформлення КР | 06.06.2026 | 14.06.2026 | Виконано |
| 10 | Подання КР, її електронної копії та інших документів (відгуку, рецензії) до захисту | 15.06.2026 | 19.06.2026 | Виконано |

**Керівник роботи** _____ Інесса КУЛАКОВСЬКА
*(Особистий підпис)* *(Власне ім'я ПРІЗВИЩЕ)*

**Здобувач** _____ Дмитро ЖУРАВСЬКИЙ
*(Особистий підпис)* *(Власне ім'я ПРІЗВИЩЕ)*

Дата складання календарного плану
«29» січня 2026 р.

# АНОТАЦІЯ

до кваліфікаційної роботи
здобувача групи 401 ЧНУ ім. Петра Могили

**Журавського Дмитра Сергійовича**

на тему: **«РОЗРОБКА ВЕБПЛАТФОРМИ ДЛЯ ВИВЧЕННЯ ІНОЗЕМНИХ МОВ ІЗ ВИКОРИСТАННЯМ ІНТЕЛЕКТУАЛЬНИХ ТЕХНОЛОГІЙ»**


**Актуальність роботи** визначається необхідністю створення освітніх систем нового покоління, які здатні забезпечити глибоку персоналізацію навчального процесу та досягти ефективної синергії між алгоритмами штучного інтелекту, викладачем і студентом.

**Об’єктом роботи** є процес дистанційного навчання іноземних мов із застосуванням сучасних інформаційних технологій.

**Предметом роботи** є моделі, методи та програмні засоби побудови інтелектуальної освітньої платформи, системи обробки природної мови (NLP), а також вебтехнології на базі фреймворку Next.js та СУБД PostgreSQL.

**Метою роботи** є розробка комплексної вебплатформи для вивчення іноземних мов, що поєднує гейміфікований інтерфейс, інтерактивний конструктор уроків та ансамбль моделей машинного навчання для автоматичного аналізу помилок користувачів і генерації адаптивного навчального контенту.

В результаті проведених досліджень та практичної реалізації поставлених завдань було створено програмний продукт, який оптимізує роботу викладача завдяки потужній системі аналітики, та підвищує ефективність навчання студентів за допомогою формування персоналізованих навчальних траєкторій на основі глобальної бази помилок.

Пояснювальна записка бакалаврської роботи складається зі вступу, чотирьох розділів, висновків і додатків. У першому розділі проаналізовано предметну область вивчення іноземних мов та існуючі освітні платформи, обґрунтовано необхідність створення нової системи з гібридним інтелектуальним підходом. У другому розділі обґрунтовано вибір технологічного стеку, спроєктовано

архітектуру системи, логічну структуру бази даних та алгоритми обробки природної мови для замкненого циклу аналізу помилок. У третьому розділі представлено проєктування інтерактивного інтерфейсу користувача з елементами гейміфікації, розробку блокового конструктора уроків та системи планування "Live Lesson". У четвертому розділі описано безпосередню програмну реалізацію вебплатформи, інтеграцію інтелектуальних модулів та наведено результати тестування системи.

Загальний обсяг роботи – [61] сторінок. Кваліфікаційна робота містить 2 додатки, [20] рисунків, [11] таблиць і [35] джерел посилання.

**Ключові слова:** вебплатформа, вивчення іноземних мов, штучний інтелект, NLP, Next.js, PostgreSQL, конструктор уроків, адаптивне навчання, гейміфікація.

# ABSTRACT

to the qualification work by the student of the group 401 of Petro Mohyla Black Sea National University

**Zhuravskyi Dmitry**

## " DEVELOPMENT OF A WEB PLATFORM FOR LEARNING FOREIGN LANGUAGES USING INTELLECTUAL TECHNOLOGIES "

A relevance of the topic is determined by the need to create new generation educational systems capable of providing deep personalization of the learning process and achieving effective synergy between artificial intelligence algorithms, teachers, and students.

An object of the work is the process of distance learning of foreign languages using modern information technologies.

A subject of the work is models, methods, and software tools for building an intelligent educational platform, natural language processing (NLP) systems, as well as web technologies based on the Next.js framework and PostgreSQL DBMS.

A purpose of the work is to develop a comprehensive web platform for learning foreign languages that combines a gamified interface, an interactive lesson builder, and an ensemble of machine learning models for automatic user error analysis and adaptive educational content generation.

As a result of the research conducted and the practical implementation of the tasks set, a software product was created that optimizes the teacher's work through a powerful analytics system, and increases the efficiency of student learning by forming personalized learning trajectories based on a global error database.

The explanatory note of the bachelor's thesis consists of an introduction, four sections, conclusions and appendices. The first section examines the subject area of foreign language learning and existing educational platforms, and justifies the need to create a new system with a hybrid intelligent approach. The second section justifies the choice of the technology stack, designs the system architecture, logical database structure, and NLP algorithms for a closed-loop error analysis. The third section presents the design of an interactive user interface with gamification elements, the development of a block-

based lesson builder, and the "Live Lesson" scheduling system. The fourth section describes the direct software implementation of the web platform, the integration of intelligent modules, and presents the results of system testing.

The overall scope of the work is [61] pages. Thesis contains [2] applications, [20] figures, [11] tables and [35] references in it.

**Keywords:** web platform, foreign language learning, artificial intelligence, NLP, Next.js, PostgreSQL, lesson builder, adaptive learning, gamification.

# ЗМІСТ

СКОРОЧЕННЯ ТА УМОВНІ ПОЗНАКИ ..... 4

ВСТУП ..... 5

1 АНАЛІЗ СУЧАСНОГО СТАНУ ПОСТАВЛЕНОЇ ЗАДАЧІ ..... 6

1.1 Опис предметної сфери ..... 6

1.2 Аналіз існуючих рішень та огляд наукових публікацій ..... 8

1.3 Постановка задачі ..... 12

Висновки до розділу 1 ..... 14

2 ПРОЄКТУВАННЯ АРХІТЕКТУРИ ТА ОБҐРУНТУВАННЯ ТЕХНОЛОГІЧНИХ РІШЕНЬ ..... 15

2.1 Вибір та обґрунтування технологічного стеку ..... 15

2.2 Технології розробки системи ..... 18

Висновки до розділу 2 ..... 21

3 РОЗРОБКА ВЕБЗАСТОСУНКУ ТА АНАЛІЗ ОТРИМАНИХ РЕЗУЛЬТАТІВ ..... 22

3.1 Проєктування архітектури та структури реляційної бази даних ..... 22

3.2 Об'єктно-орієнтоване проєктування та моделювання прецедентів ..... 26

3.3 Структура проєкту Next.js та клієнтська частина ..... 31

3.4 Розробка серверних маршрутів та контроль сесій ..... 36

3.5 Інтеграція штучного інтелекту: NLP мікросервіс та Gemini API ..... 38

Висновки до розділу 3 ..... 44

4 ІНТЕЛЕКТУАЛЬНІ АЛГОРИТМИ: ГІБРИДНА NLP-СИСТЕМА ТА RAG НА БАЗІ GEMINI ..... 46

4.1 Проблематика автоматичного аналізу відкритих відповідей та виклики розробки ..... 46

4.2 Реалізація локального NLP-мікросервісу на базі FastAPI ..... 48

4.3 Двомодельна ML-архітектура розпізнавання помилок: класифікатор Ariadna ..... 49

4.4 Двомодельна ML-архітектура розпізнавання помилок: токен-класифікатор Theseus ..... 51

4.5 Алгоритми обробки тексту на базі spaCy та dependency parsing ..... 53

4.6 База даних аналітики помилок та Prisma ORM ..... 56

4.7 Інтеграція LLM Gemini за технологією RAG для адаптивної генерації завдань ..... 56

Висновки до розділу 4 ..... 59

ВИСНОВКИ ..... 60

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ ..... 62

# СКОРОЧЕННЯ ТА УМОВНІ ПОЗНАКИ

| | | |
|---|---|---|
| БД | – | База даних |
| ПЗ | – | Програмне забезпечення |
| СУБД | – | Система управління базами даних |
| ШІ | – | Штучний інтелект |

| | | |
|---|---|---|
| API | – | Application Programming Interface |
| CSS | – | Cascading Style Sheets |
| DOM | – | Document Object Model |
| HTML | – | HyperText Markup Language |
| HTTP | – | HyperText Transfer Protocol |
| JSON | – | JavaScript Object Notation |
| JWT | – | JSON Web Token |
| LLM | – | Large Language Model |
| ML | – | Machine Learning |
| TTS | – | Text-to-Speech |
| NLP | – | Natural Language Processing |
| ORM | – | Object-Relational Mapping |
| SPA | – | Single Page Application |
| SSR | – | Server-Side Rendering |
| UI | – | User Interface |
| UX | – | User Experience |

# ВСТУП

Для сучасного етапу розвитку суспільства характерною є глобалізація економічних, культурних та освітніх процесів, що створює стабільний попит на вивчення іноземних мов. Особливий поштовх ця сфера отримала під час масового переходу на дистанційний формат роботи та навчання, коли онлайн-освіта стала основним стандартом індустрії. Сучасні студенти вимагають не просто доступу до матеріалів, а гнучкого графіку, інтерактивних завдань та постійного контролю свого прогресу. Проте більшість існуючих освітніх платформ не здатні забезпечити глибоку персоналізацію навчального процесу та синергію між викладачем, технологією і студентом. Як наслідок, учні часто стикаються з проблемою «фосилізації помилок», коли безперервне повторення дрібних неточностей без належного та систематичного контролю закріплює неправильні мовні патерни.

Для розв’язання зазначених проблем виникає необхідність у створенні освітніх платформ нового покоління, де штучний інтелект виступає не просто інструментом заміни викладача, а потужним аналітичним ядром системи. Інтеграція алгоритмів обробки природної мови (NLP) дозволяє безперервно відслідковувати кожну виконану вправу, формувати глобальну базу індивідуальних помилок та автоматично генерувати адаптивний навчальний контент. Такий гібридний підхід звільняє викладача від рутинної перевірки, надаючи йому потужні аналітичні дашборди для своєчасної корекції навчальної траєкторії під час живих занять.

У процесі виконання даної кваліфікаційної роботи спроєктовано та реалізовано комплексну інтелектуальну вебплатформу для вивчення іноземних мов. Було обґрунтовано вибір сучасного технологічного стеку, основу якого складають фреймворк Next.js для забезпечення миттєвого рендерингу та СУБД PostgreSQL для зберігання складної аналітики. Розроблено архітектуру замкненого циклу навчання (Feedback Loop), що забезпечує маршрутизацію користувацьких помилок та генерацію персоналізованих завдань.

# 1 АНАЛІЗ СУЧАСНОГО СТАНУ ПОСТАВЛЕНОЇ ЗАДАЧІ

## 1.1 Опис предметної сфери

Для сучасного етапу розвитку інформаційного суспільства характерною є стрімка глобалізація економічних, культурних та освітніх процесів, що формує стабільний і зростаючий попит на ефективне вивчення іноземних мов. Володіння іноземними мовами трансформувалося з інструменту особистого розвитку у критично важливу вимогу на міжнародному ринку праці. Відповідно, сфера освітніх технологій (EdTech) постійно модернізується, пропонуючи різноманітні підходи до організації навчального процесу. Особливо потужний поштовх ця галузь отримала внаслідок масового переходу на дистанційний формат взаємодії, що зробило онлайн-навчання основним стандартом сучасної індустрії.

Історично у предметній сфері дистанційного вивчення мов домінують два основні формати: індивідуальна робота з викладачем (репетитором) та самостійне навчання за допомогою мобільних застосунків чи інтерактивних тренажерів. Кожен із цих підходів має свої переваги, проте обидва демонструють суттєві обмеження у контексті довгострокової ефективності.

Традиційне навчання з викладачем забезпечує найвищий рівень персоналізації. Живий репетитор здатен миттєво адаптувати темп уроку під емоційний стан студента, пояснити специфічний контекст використання лексики та поставити правильну артикуляцію. Проте цей підхід повністю залежить від людського фактора. Викладач, маючи значну кількість учнів, фізично не здатен запам’ятати абсолютно всі дрібні помилки, допущені кожним студентом протягом місяців навчання. Крім того, підготовка індивідуалізованих матеріалів для усунення виявлених прогалин потребує величезної кількості неоплачуваного часу викладача, що робить процес масштабування такого навчання економічно неефективним.

З іншого боку, масові освітні онлайн-платформи пропонують високу доступність, елементи гейміфікації та стандартизовані курси, які часто базуються

на класичних алгоритмах інтервального повторення. Проте їх критичним недоліком є цілковита відсутність індивідуального підходу та живого зворотного зв'язку. Додаток фіксує лише факт правильної чи неправильної відповіді, не аналізуючи причину помилки. Це створює «ілюзію знань», коли користувач навчається проходити специфічні тести, але не здатний вільно комунікувати в реальних ситуаціях.

Важливим аспектом є також когнітивна складова навчання. Згідно з дослідженнями нейробіології пам’яті, зокрема кривої забування Еббінгауза, процес втрати інформації має нелінійний характер, і без систематичного повторення знання швидко зникають. Однак стандартні платформи формують інтервали повторення на основі усереднених метрик, ігноруючи індивідуальні особливості сприйняття кожної людини. Відсутність системи, яка б враховувала ці відхилення, часто призводить до ефекту «плато», коли кількість витраченого часу перестає конвертуватися у якість навичок. Графічне відображення цього процесу наведено на рисунку 1.1.

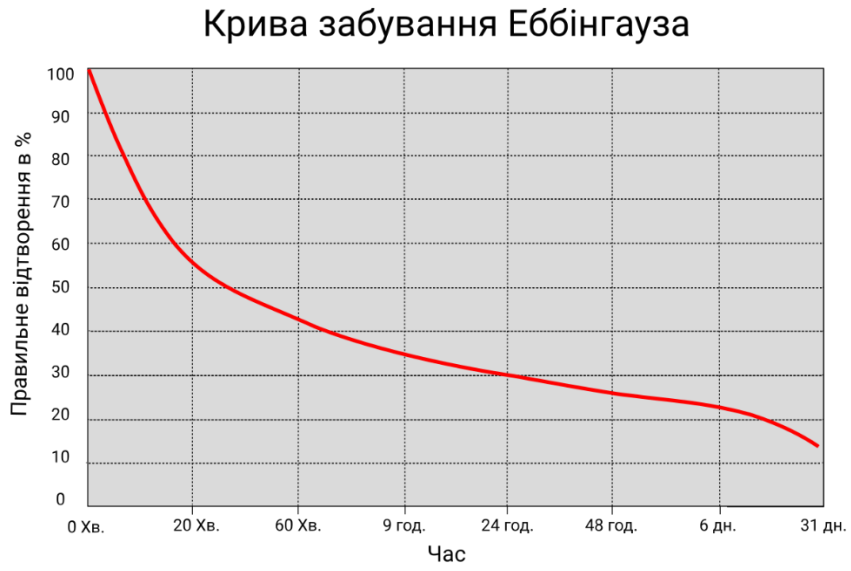


Рисунок 1.1 – Графік кривої забування Г. Еббінгауза [1]

Окремо слід виділити психологічний бар'єр – «страх помилки». Під час занять репетитор часто свідомо ігнорує дрібні граматичні неточності студента, щоб не перебивати його та зберегти динаміку розмови. З часом такі «пробачені» хиби

автоматизуються та закріплюються на нейронному рівні, що призводить до явища фосилізації помилок, викорінити які пізніше вкрай складно. Саме тому виникає необхідність у неупередженому машинному контролі, який працюватиме у фоновому режимі, скрупульозно фіксуючи кожну хибу та передаючи цю інформацію викладачу для обережної корекції.

Таким чином, ключова проблематика сучасного онлайн-навчання полягає у відсутності синергії між студентом, викладачем та технологіями. Існує гостра потреба у створенні комплексних систем нового покоління, де штучний інтелект виступатиме аналітичним ядром для агрегації помилок та генерації персоналізованого контенту, звільняючи викладача від рутини та дозволяючи йому зосередитися на менторстві та поясненні складних мовних концепцій.

### 1.2 Аналіз існуючих рішень та огляд наукових публікацій

Для обґрунтування доцільності розробки нової освітньої платформи було проведено аналіз існуючих рішень на ринку освітніх технологій (EdTech). Незважаючи на перенасиченість ринку різноманітними продуктами, більшість із них є вузькоспеціалізованими і фокусуються на вирішенні лише однієї конкретної проблеми. Сучасні цифрові інструменти для вивчення іноземних мов можна умовно поділити на три великі категорії: платформи для самостійного вивчення, маркетплейси репетиторів та інтелектуальні системи корекції тексту.

**1. Платформи для самостійного вивчення (Duolingo, Babbel, Busuu).**

Ця категорія представлена популярними мобільними та вебдодатками, орієнтованими на масового споживача. Їхньою головною перевагою є безпрецедентний рівень залучення користувачів завдяки глибокій гейміфікації навчального процесу. Такі системи використовують механіки підтримки серії днів (streaks), ліги, бали досвіду та ігрову валюту, що стимулює щоденну взаємодію з додатком.

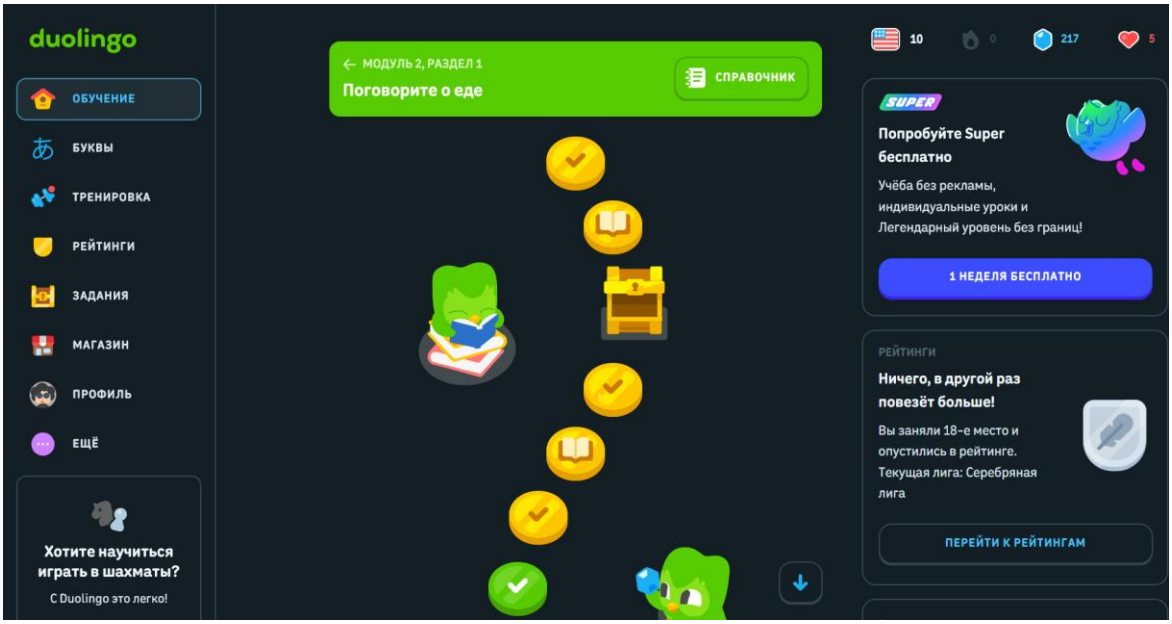


Рисунок 1.1 – Інтерфейс застосунку Duolingo [2]

Незважаючи на високий рівень утримання аудиторії (retention rate), ці платформи мають суттєві архітектурні та методологічні недоліки. Жорстка стандартизація курсів створює ілюзію навчання: користувач відточує навички проходження специфічних тестів, проте часто виявляється нездатним побудувати власне речення у реальній комунікації. Алгоритми платформи фіксують факт правильної або неправильної відповіді, проте не здійснюють семантичного розбору помилки. Додаток не здатен пояснити студенту, чому його логіка побудови фрази була хибною у даному контексті. Крім того, повна відсутність живого викладача унеможливлює розбір складних граматичних винятків.

**2. Платформи для роботи з репетитором (Preply, italki).**

Дана категорія вирішує проблему пошуку викладача та організації відеозв'язку. Платформи надають інфраструктуру для фільтрації фахівців за ціною, розкладом та відгуками, а також гарантують безпеку фінансових транзакцій.

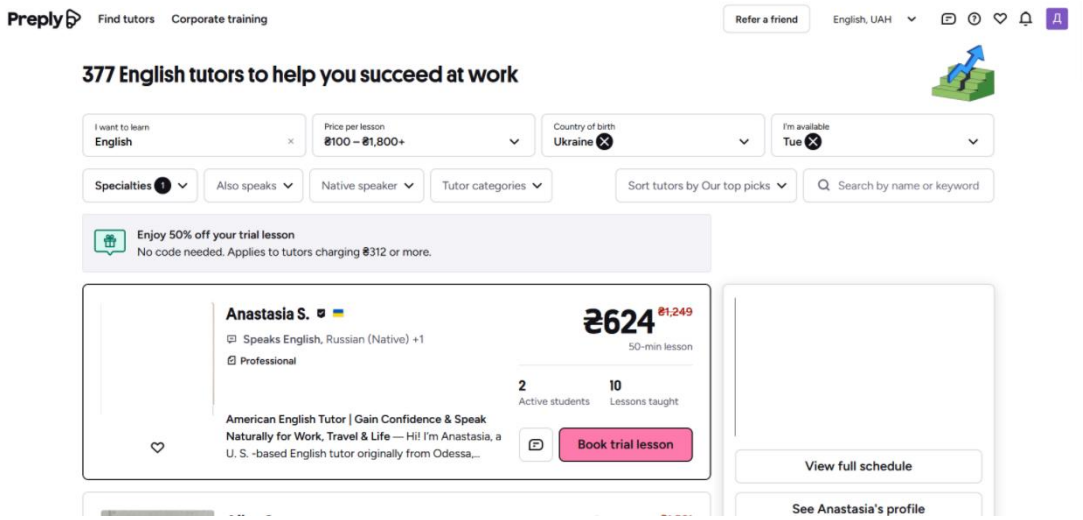


Рисунок 1.2 – Інтерфейс застосунку Preply [3]

Проте, окрім функції маркетплейсу, ці системи практично не пропонують викладачам серйозних аналітичних інструментів або вбудованих систем управління навчанням (LMS). Весь тягар відслідковування прогресу, фіксації помилок та підготовки персоналізованих завдань лягає виключно на репетитора. Через відсутність автоматизованої аналітики викладачі змушені використовувати сторонні таблиці (Excel, Google Sheets), що призводить до фрагментації даних та ускладнює системний аналіз навчальної траєкторії студента.

**3. Інструменти на базі штучного інтелекту (Grammarly, LLM-моделі).**

Третя категорія включає сучасні рішення на базі обробки природної мови, такі як Grammarly або інтерфейси до великих мовних моделей (ChatGPT). Вони демонструють високу ефективність у миттєвому виявленні орфографічних та синтаксичних помилок.

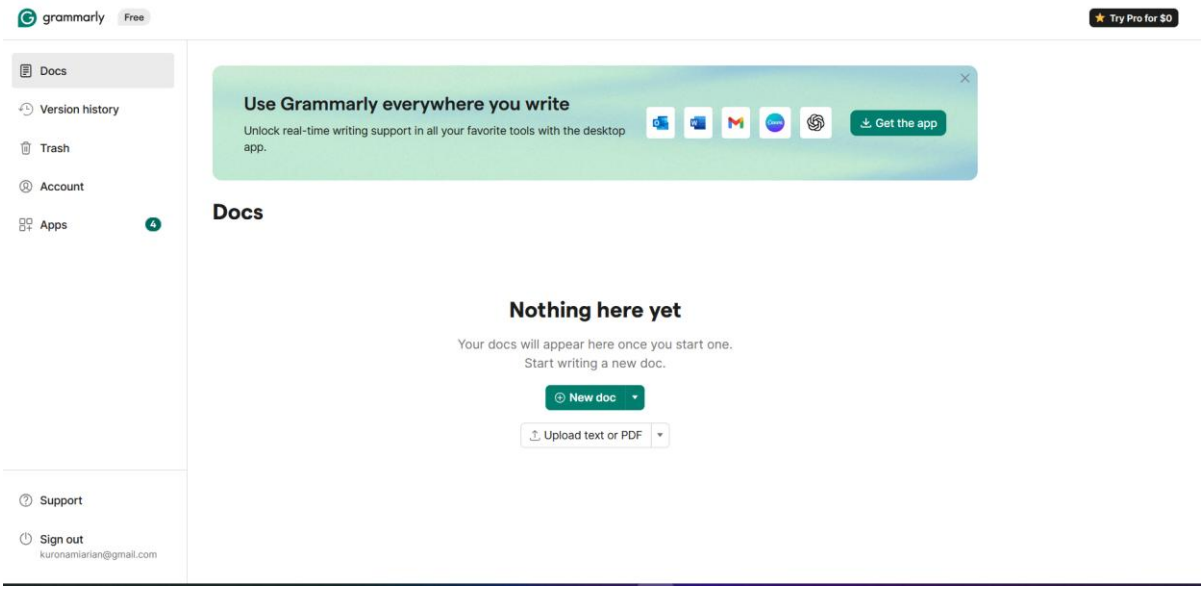


Рисунок 1.3 – Інтерфейс застосунку Grammarly [4]

Основним недоліком використання чистих LLM в освітніх цілях є відсутність довгострокової пам'яті щодо конкретного студента. Система виправляє помилки в моменті, але не формує глобальної бази вразливостей користувача. Відповідно, такі сервіси не можуть автоматично згенерувати адаптивний тест на повторення тих правил, з якими студент мав проблеми тиждень тому.

Окремої уваги заслуговує огляд сучасних наукових публікацій у галузі лінгводидактики та обробки природної мови. У дослідженнях, присвячених автоматизованому оцінюванню, зазначається, що використання виключно генеративних нейромереж має обмеження. Дослідники в галузі педагогічного дизайну наголошують, що мовні моделі часто порушують принцип «педагогічного скаффолдингу» [5]. Замість того, щоб через систему підказок підвести студента до самостійного висновку, модель видає готову відповідь, що блокує розвиток аналітичного мислення.

Більшість сучасних дослідників сходяться на думці, що майбутнє освітніх технологій – за розробкою гібридних систем. У науковій спільноті активно просувається концепція «Human-in-the-Loop» [6], згідно з якою штучний інтелект виконує функцію не замінника викладача, а його інтелектуального асистента. ШІ

повинен агрегувати дані та маркувати помилки, тоді як фінальне рішення щодо зміни навчальної стратегії залишається за живим ментором.

Для кращого розуміння позиціонування розроблюваної системи було створено порівняльну таблицю ключових функцій існуючих рішень (Таблиця 1.1).

Таблиця 1.1 – Порівняльна характеристика існуючих рішень на ринку

| **Функціонал / Сервіс** | **Duolingo** | **Preply** | **Grammarly** | **Пропонована система** |
|---|---|---|---|---|
| Наявність викладача | Відсутній | Присутній | Відсутній | Присутній |
| Автоматична перевірка вправ | Присутня | Відсутня | Присутня | Присутня |
| Глобальна база помилок студента | Відсутня | Відсутня | Відсутня | Присутня |
| Генерація адаптивного контенту | Частково | Відсутня | Відсутня | Присутня |
| Аналітичний дашборд для репетитора | Відсутній | Відсутній | Відсутній | Присутній |

Аналіз конкурентного середовища доводить наявність суттєвого технологічного розриву між інструментами автоматизованої ШІ-перевірки та платформами для роботи з репетитором. Відповідно, розробка системи, що об'єднає менторство з потужною фоновою NLP-аналітикою для безперервної маршрутизації помилок та генерації контенту, є актуальною та комерційно перспективною задачею.

## 1.3 Постановка задачі

**Актуальність теми.** Існуючі освітні платформи не забезпечують повноцінної синергії між викладачем, технологічною базою та студентом. Відсутність автоматизованих систем накопичення та аналізу індивідуальних

помилок користувача в довгостроковій перспективі ускладнює процес навчання та створює зайве неоплачуване навантаження на викладача. Тому розробка вебплатформи, що поєднує менторство з фоновою роботою алгоритмів штучного інтелекту для генерації адаптивного контенту, є актуальною науково-практичною задачею.

**Мета роботи полягає в** розробка комплексної вебплатформи для вивчення іноземних мов, яка забезпечить повну адаптацію та персоналізацію навчального процесу, а також оптимізацію роботи викладача за рахунок гібридної взаємодії з модулями обробки природної мови.

**Об’єктом дослідження є** процес дистанційного навчання іноземних мов із застосуванням сучасних інформаційних технологій.

**Предметом дослідження є** моделі, методи та програмні засоби побудови інтелектуальної освітньої платформи на базі ансамблю алгоритмів машинного навчання (NLP).

Для досягнення поставленої мети необхідно виконати такі **завдання:**

– проаналізувати предметну сферу, наявні освітні платформи та методи застосування штучного інтелекту в процесі вивчення іноземних мов;

– обґрунтувати вибір технологічного стеку та розробити загальну клієнт-серверну архітектуру інформаційної системи;

– спроєктувати реляційну базу даних для накопичення й глобального аналізу помилок користувачів;

– дослідити та розробити архітектуру інтелектуального модуля для генерації персоналізованого навчального контенту (Feedback Loop);

– здійснити програмну реалізацію платформи, включаючи блоковий конструктор уроків, дашборди та гейміфікований інтерфейс користувача;

– провести комплексне тестування розробленої системи та оцінити її продуктивність.

Виконання зазначених завдань дозволить реалізувати програмний продукт, який орієнтований на розв'язання проблеми розриву між самостійною роботою

студента та професійною підтримкою репетитора.

## Висновки до розділу 1

У першому розділі було проведено детальний системний аналіз предметної сфери дистанційного вивчення іноземних мов. Встановлено, що сучасні підходи до онлайн-навчання поділяються на самостійну роботу в додатках та взаємодію з живим репетитором. Незважаючи на широке різноманіття продуктів (Duolingo, Preply, Grammarly), жоден із них не забезпечує комплексної екосистеми, яка б поєднувала менторство з довгостроковою машинною аналітикою помилок.

На основі аналізу наукових публікацій доведено, що найбільш ефективним підходом у сучасних освітніх технологіях є створення гібридних систем за концепцією «Human-in-the-Loop», де штучний інтелект виконує роль асистента для агрегації даних, а фінальне коригування навчальної траєкторії здійснює викладач. Це дозволяє уникнути порушень педагогічного підходу та явища фосилізації помилок у студентів.

З урахуванням виявленої проблематики було сформульовано мету, визначено об'єкт і предмет дослідження, а також поставлено конкретні задачі розробки. Виконання цих задач дозволить створити інноваційну вебплатформу з гейміфікованим інтерфейсом, інтерактивним конструктором уроків та модулями NLP для формування глобальної бази користувацьких помилок.

# 2 ПРОЄКТУВАННЯ АРХІТЕКТУРИ ТА ОБҐРУНТУВАННЯ ТЕХНОЛОГІЧНИХ РІШЕНЬ

## 2.1 Вибір та обґрунтування технологічного стеку

При розробці високотехнологічної освітньої платформи з інтеграцією модулів штучного інтелекту критично важливим етапом є обґрунтований вибір технологічного стеку. Відповідно до спроєктованої архітектури, систему було логічно розділено на три взаємопов'язані рівні: клієнт-серверний додаток (фронтенд та бекенд), інфраструктура даних (СУБД та ORM) та окремий інтелектуальний модуль. У даному підрозділі проведено порівняльний аналіз альтернативних технологій для кожного з рівнів.

Для реалізації клієнтської та серверної частин вебзастосунку розглядалися три популярні підходи: класичні бекенд-фреймворки (Django), створення односторінкових застосунків (React SPA + Express) та мета-фреймворки з підтримкою серверного рендерингу (Next.js).

Використання Django вимагає побудови окремої клієнтської частини, що ускладнює розробку. Класичний React (SPA) чудово підходить для інтерактивних інтерфейсів, але повністю завантажується на клієнті, що збільшує час першого відмальовування. Натомість Next.js виступає найбільш збалансованим рішенням. Завдяки серверному рендерингу (SSR), сторінки з навчальними матеріалами завантажуються миттєво. Вбудовані API-маршрути дозволяють обробляти стандартні запити користувачів (реєстрація, збереження відповідей) без необхідності розгортати окремий бекенд-додаток.

У табл. 2.1 навдено приклад порівняння підходів до розробки вебзастосунку.

Таблиця 2.1 – Порівняння підходів до розробки вебзастосунку

| Критерій | Next.js | React SPA | Django |
|---|---|---|---|
| Архітектура | Монолітна | Роздільна | Монолітна |

Кінець таблиці 2.1

| Критерій | Next.js | React SPA | Django |
|---|---|---|---|
| Мова програмування | TypeScript | JavaScript/TypeScript | Python |
| Швидкість завантаження UI | Миттєво | Із затримкою | Швидко |

Для зберігання інформації розглядалися PostgreSQL, MySQL та документоорієнтована MongoDB. Оскільки основа системи – це глибока аналітика, де профілі студентів, транскрипції та лексичні помилки жорстко пов'язані між собою, використання NoSQL рішень (MongoDB) є недоцільним через відсутність суворої цілісності даних.

Між MySQL та PostgreSQL перевагу віддано PostgreSQL. Вона ідеально підходить для побудови складних зв'язків та виконання важких аналітичних запитів для формування дашборду репетитора, а також підтримує тип JSONB для зберігання нестандартних відповідей з інтерактивного конструктора уроків.

Таблиця 2.2 – Порівняння систем управління базами даних

| Критерій | PostgreSQL | MySQL | MongoDB |
|---|---|---|---|
| Тип | Реляційна | Реляційна | NoSQL |
| Підтримка складних зв'язків | Відмінна | Добра | Обмежена |
| Обробка масивів JSON | Нативна | Базова | Основна функція |

Для об'єктно-реляційного відображення (взаємодії коду з БД) порівнювались Prisma, TypeORM та Sequelize. Prisma ORM обрана завдяки унікальному підходу: вона використовує декларативну схему замість класів, автоматично генерує повністю типізований (Type-Safe) клієнт для TypeScript та суттєво мінімізує

помилки під час міграцій порівняно зі старішим Sequelize.

Таблиця 2.3 – Порівняння ORM-інструментів для TypeScript

| **Критерій** | **Prisma** | **TypeORM** | **Sequelize** |
|---|---|---|---|
| Підхід | Схема-орієнтований | Класи | Об'єктний |
| Безпека типів (Type Safety) | Абсолютна | Вимагає налаштувань | Слабка |
| Складність міграцій | Автоматизована | Ручна/напівавтомат | Ручна |

На рівні користувацького інтерфейсу постало завдання вибору інструментів стилізації та створення контенту. Для стилізації було обрано Tailwind CSS замість традиційного Bootstrap, оскільки утилітарний підхід Tailwind дозволяє створювати гнучкі та унікальні гейміфіковані інтерфейси без перевизначення громіздких стандартних класів. Для розробки конструктора уроків розглянуто TinyMCE та Editor.js. Вибір зупинено на Editor.js завдяки його блоковій структурі, яка видає чистий JSON-об'єкт замість HTML-коду. Це дозволяє легко маніпулювати окремими блоками (текст, аудіо, тест) як незалежними компонентами в базі даних.

У розрізі серверної інфраструктури розглядалися хмарні безсерверні функції та контейнеризація на базі віртуальних серверів. Від Serverless-рішень було свідомо відмовлено. Незважаючи на їх зручність, у реальному проєкті з постійною фоновою роботою алгоритмів аналітики вони створюють ліміти за часом виконання запиту та ускладнюють контроль над процесами. Тому обрано розгортання через Docker, що забезпечує повну ізоляцію середовища.

Окремим ключовим завданням був вибір методології інтеграції штучного інтелекту. Було прийнято рішення відмовитись як від суто алгоритмічного підходу, так і від використання виключно великих мовних моделей, на користь гібридної інтелектуальної архітектури. Цей гібридний підхід розділяє логіку на два етапи:

відстеження помилок та генерацію контенту.

Для відстеження та класифікації помилок (наприклад, визначення того, чи помилка є лексичною, граматичною чи помилкою аудіювання) використовуються власні донавчені ML-моделі. Такий підхід гарантує жорстку структурованість логів у базі даних, чого неможливо досягти при використанні виключно комерційних LLM.

Натомість для генерації нового навчального контенту ідеально підходять генеративні нейромережі. Зокрема, у системі використовується API Google Gemini. Якщо студент під час аудіювання помилково почув слово «dog» і написав «hog», власна ML-модель фіксує це як помилку аудіювання та відправляє у базу даних. Згодом система агрегує масив таких індивідуальних помилок і відправляє промпт до Gemini з вимогою згенерувати новий текст, який буде насичений проблемними словами студента. Оскільки система знає, що ці слова стосуються проблеми сприйняття на слух, згенерований текст додатково опрацьовується через модуль синтезу мовлення TTS, і студент отримує абсолютно нове унікальне аудіо-завдання. Така гібридна синергія дозволяє забезпечити найвищу ефективність закріплення матеріалу.

## 2.2 Технології розробки системи

Для програмної реалізації інформаційної системи було використано стек сучасних вебтехнологій, який забезпечує високу продуктивність, безпеку та можливість масштабування проєкту. Кожен компонент стеку виконує чітко визначену роль у загальній архітектурі.

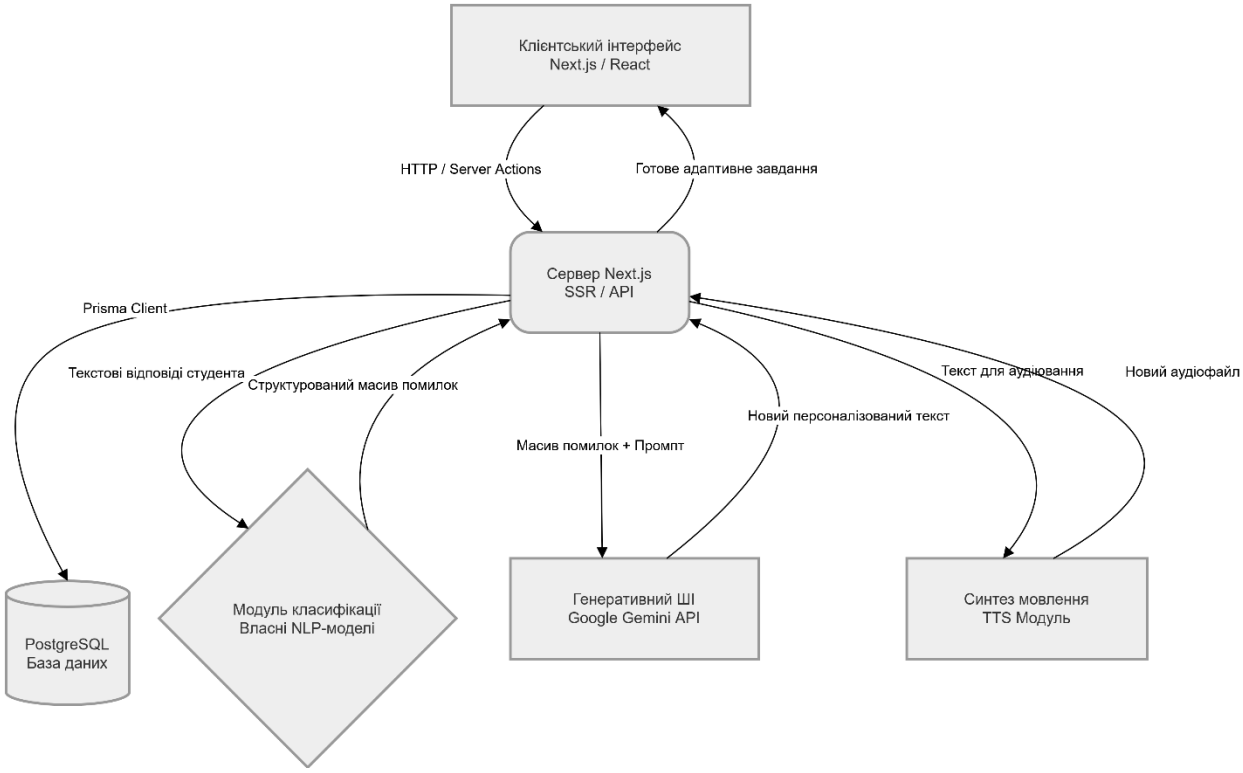


Рисунок 2.2 – Схема взаємодії технологічних компонентів вебплатформи

**Next.js та TypeScript.** Центральним елементом архітектури виступає фреймворк Next.js на базі мови програмування TypeScript.

Використання суворої типізації TypeScript дозволяє виявляти помилки на етапі компіляції та забезпечує надійність структур даних, що особливо важливо при роботі зі складними об'єктами відповідей від штучного інтелекту. Фреймворк Next.js використовується завдяки своїй гібридній природі рендерингу. Серверний рендеринг застосовується для миттєвої видачі готових HTML-сторінок з навчальним матеріалом, що гарантує високу швидкість роботи та SEO-оптимізацію. Водночас інтерактивні елементи, такі як тести та аудіоплеєри, рендеряться на стороні клієнта. Механізм Server Actions дозволяє безпечно обробляти форми реєстрації та записувати відповіді студентів у базу даних без створення окремого REST API.

**PostgreSQL та Prisma ORM.** Для забезпечення надійного збереження даних та побудови зв'язків використовується об'єктно-реляційна СУБД PostgreSQL у зв'язці з Prisma ORM.

PostgreSQL забезпечує транзакційність операцій та цілісність даних за принципом ACID. Ключовою перевагою цієї БД для даного проєкту є нативна підтримка типу даних JSONB. Це дозволяє ефективно зберігати динамічну структуру уроків (наприклад, непередбачувану кількість блоків тексту, аудіо та вікторин) в одній колонці таблиці з можливістю індексування. Prisma ORM, своєю чергою, автоматично генерує TypeScript-типи на основі схеми бази даних, що повністю виключає помилки розбіжності даних між бекендом та базою.

**Tailwind CSS.** Стилізація платформи здійснюється за допомогою утилітарного CSS-фреймворку Tailwind CSS.

На відміну від компонентних фреймворків, Tailwind надає низькорівневі класи, з яких формується унікальний дизайн проєкту. Це дозволяє швидко реалізовувати мікроанімації, динамічні кольорові теми та адаптивний дизайн під мобільні пристрої, що є критично важливим для забезпечення високого рівня гейміфікації (UX/UI) та залучення студентів.

**Editor.js.** Для розробки інтерактивного модуля створення уроків інтегровано бібліотеку Editor.js.

Традиційні текстові редактори генерують вихідні дані у форматі брудного HTML-коду, що ускладнює їх подальшу обробку та відображення на різних пристроях. Натомість Editor.js реалізує блоковий підхід: кожен елемент уроку є окремим блоком, який на виході формує чистий JSON-об'єкт. Під час рендерингу уроку Next.js зчитує цей JSON і перетворює кожен блок на повноцінний React-компонент. Це дозволяє викладачу легко комбінувати різні типи завдань без необхідності писати програмний код.

**Модулі ШІ та генерації контенту.** Система обробки природної мови NLP побудована як окремий інтелектуальний конвеєр. Вона приймає сирі текстові дані та класифікує їх за допомогою донавчених ML-моделей. Після фіксації помилок у базі даних (наприклад, хиб у сприйнятті на слух), масив цих даних відправляється до генеративної моделі через API Google Gemini. Згенерований нею текст, спеціально насичений проблемними словами студента, передається до модуля TTS,

який за допомогою нейромережевого синтезу голосу перетворює текст на нові аудіофайли. Це забезпечує повністю автоматизований цикл створення адаптивних вправ.

Таким чином, обраний технологічний стек формує цілісну екосистему, де Next.js відповідає за оркестрацію запитів та відображення інтерфейсу, PostgreSQL та Prisma забезпечують безпеку даних, а комбінація власних NLP-алгоритмів із генеративним API відповідає за адаптивну логіку навчання

## Висновки до розділу 2

У другому розділі було обґрунтовано вибір методів та технологій розробки інформаційної системи. На основі порівняльного аналізу доведено доцільність використання фреймворку Next.js порівняно з традиційними SPA та класичними бекенд-рішеннями, завдяки його можливостям серверного рендерингу та монолітній архітектурі.

Обґрунтовано вибір реляційної СУБД PostgreSQL та Prisma ORM для забезпечення суворої цілісності та безпеки типів. Ключовою особливістю розробленої архітектури є використання гібридного підходу до штучного інтелекту: власні ML-моделі відповідають за класифікацію та відстеження помилок, а генерація нового контенту та синтез мовлення TTS реалізуються через API Google Gemini.

Детально описано інтеграцію блокового редактора Editor.js для реалізації гнучкого конструктора уроків, що генерує JSON-дані замість HTML, та утилітарного фреймворку Tailwind CSS для створення гейміфікованого інтерфейсу користувача. Обраний технологічний стек створює надійний фундамент для переходу до безпосереднього проєктування архітектури та програмної реалізації вебзастосунку.

# 3 РОЗРОБКА ВЕБЗАСТОСУНКУ ТА АНАЛІЗ ОТРИМАНИХ РЕЗУЛЬТАТІВ

## 3.1 Проєктування архітектури та структури реляційної бази даних

Для забезпечення надійного зберігання даних, швидкого доступу до навчальних матеріалів та формування глибокої аналітики, ядром інформаційної системи виступає реляційна база даних під управлінням СУБД PostgreSQL. Структура бази даних спроєктована з урахуванням необхідності зберігання складних зв'язків між користувачами, навчальним контентом та класифікованими помилками.

Процес проєктування бази даних передбачав нормалізацію відношень для уникнення дублювання інформації та забезпечення цілісності. Логічну структуру бази даних у вигляді діаграми ER-діаграми, наведено на рисунку 3.1.

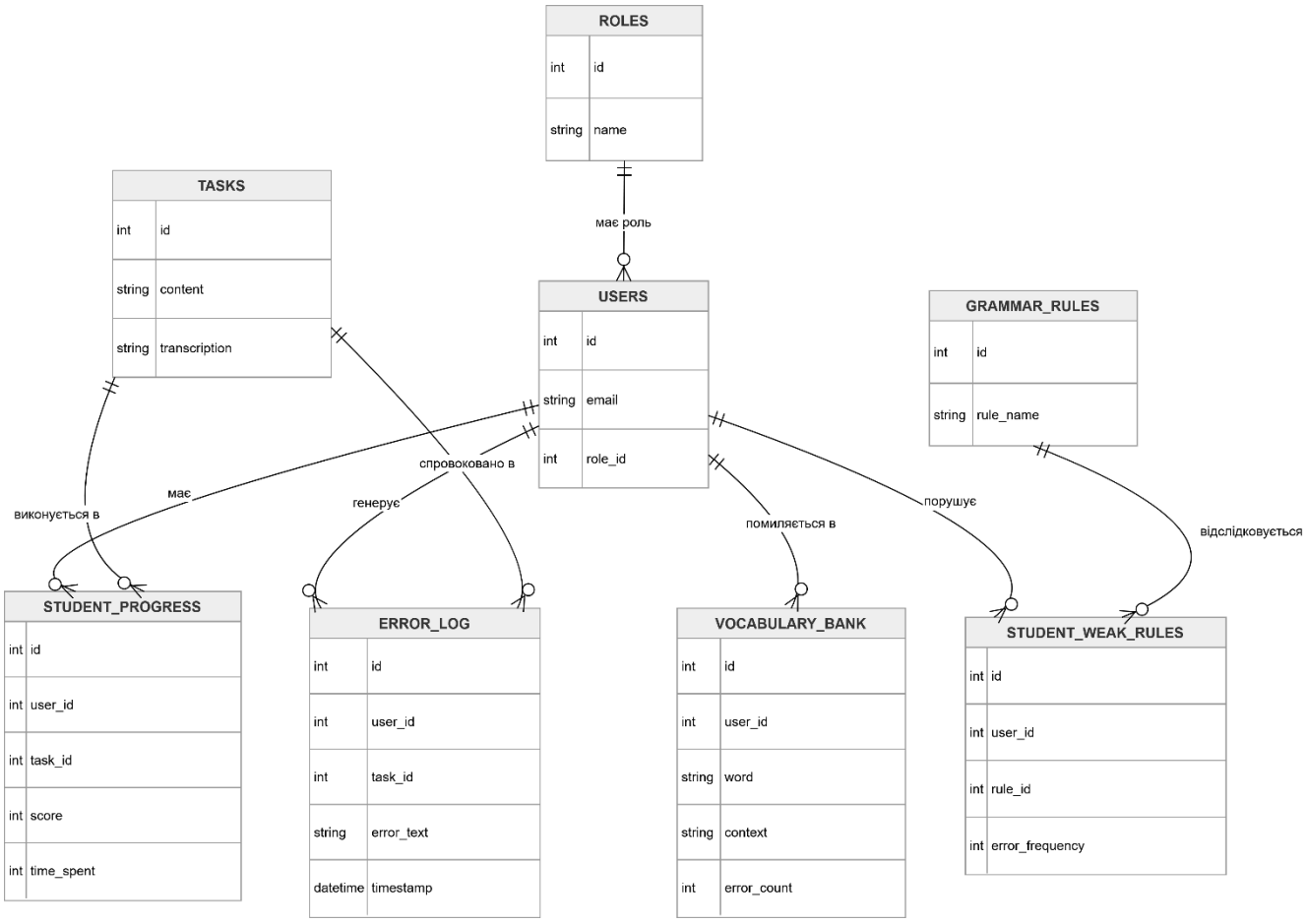


Рисунок 3.1 – Логічна структура реляційної бази даних (ER-діаграма)

Як видно з діаграми, архітектура бази даних чітко розділена на три взаємопов'язані модулі, кожен з яких виконує свою роль у загальному освітньому процесі.

**Модуль автентифікації та ролей.** Фундаментом системи є таблиці для управління доступом. Таблиця users містить ідентифікаційні дані та облікові записи.

Таблиця 3.1 – Опис полів таблиці Users

| **Назва атрибута** | **Тип даних** | **Опис призначення** |
|---|---|---|
| id | UUID / INT | Первинний ключ, унікальний ідентифікатор користувача |
| email | VARCHAR | Електронна адреса |
| password_hash | VARCHAR | Хешований пароль |
| role_id | INT | Зовнішній ключ на таблицю Roles |
| created_at | TIMESTAMP | Дата і час створення облікового запису |

Завдяки зв'язку з таблицею довідником roles, система розмежовує бізнес-логіку додатка: викладачі отримують доступ до дашбордів, а студенти бачать лише навчальний інтерфейс.

Таблиця 3.2 – Опис полів таблиці Roles

| **Назва атрибута** | **Тип даних** | **Опис призначення** |
|---|---|---|
| id | INT | Первинний ключ |
| name | VARCHAR | Назва ролі (STUDENT, TEACHER, ADMIN) |
| permissions | JSONB | Масив прав доступу для тонкого налаштування |

**Навчальний модуль.** Цей блок відповідає за збереження самих вправ та результатів їх виконання. Таблиця tasks є універсальним сховищем для всіх типів завдань.

Таблиця 3.3 – Опис полів таблиці Tasks

| **Назва атрибута** | **Тип даних** | **Опис призначення** |
|---|---|---|
| id | INT | Первинний ключ |
| content | JSONB | Вміст завдання (Editor.js) |
| transcription | TEXT | Текстова транскрипція для аудіо-завдань |

Продовження таблиці 3.3

| **Назва атрибута** | **Тип даних** | **Опис призначення** |
|---|---|---|
| type | VARCHAR | Тип (граматика, лексика, аудіювання) |

Таблиця student_progress виступає сполучною ланкою. Вона фіксує не лише факт виконання завдання, але й витрачений час (time_spent), що дозволяє оцінити когнітивне навантаження студента.

Таблиця 3.4 – Опис полів таблиці Student_Progress

| **Назва атрибута** | **Тип даних** | **Опис призначення** |
|---|---|---|
| id | INT | Первинний ключ |
| user_id | INT | Зовнішній ключ |
| task_id | INT | Зовнішній ключ |
| score | INT | Оцінка у відсотках (0-100) |
| time_spent | INT | Витрачений час у секундах |

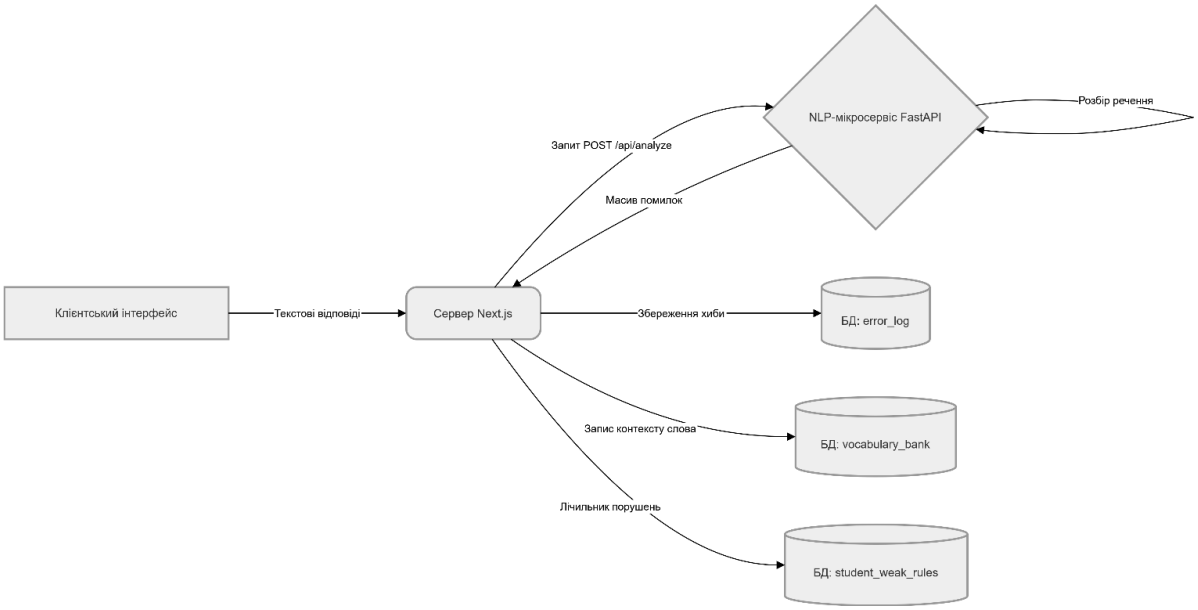


Рисунок 3.2 – Діаграма потоків даних між СУБД та NLP-модулем

**Модуль ШІ-аналітики та маршрутизації помилок.** Це ключовий елемент бази даних, який забезпечує роботу замкненого циклу навчання (Feedback Loop). Таблиця error_log фіксує абсолютно кожну хибу в системі в режимі реального часу.

Таблиця 3.5 – Опис полів таблиці Error_Log

| **Назва атрибута** | **Тип даних** | **Опис призначення** |
|---|---|---|
| id | BIGINT | Первинний ключ |
| user_id | INT | Хто допустив помилку |
| task_id | INT | У якому завданні |
| error_text | TEXT | Сирий текст хибної відповіді |
| timestamp | TIMESTAMP | Час фіксації |

Таблиця vocabulary_bank слугує для зберігання лексичних помилок. Замість того, щоб просто зберігати неправильне слово, система зберігає контекст його використання.

Таблиця 3.6 – Опис полів таблиці Vocabulary_Bank

| **Назва атрибута** | **Тип даних** | **Опис призначення** |
|---|---|---|
| id | INT | Первинний ключ |
| user_id | INT | Власник помилки |

Кінець таблиці 3.6

| **Назва атрибута** | **Тип даних** | **Опис призначення** |
|---|---|---|
| word | VARCHAR | Проблемне слово |
| context | TEXT | Речення, в якому допущено помилку |
| error_count | INT | Кількість повторень помилки |

Таблиця student_weak_rules динамічно пов'язує конкретного студента з тими граматичними правилами, які даються йому найважче.

Таблиця 3.7 – Опис полів таблиці Student_Weak_Rules

| **Назва атрибута** | **Тип даних** | **Опис призначення** |
|---|---|---|
| id | INT | Первинний ключ |
| user_id | INT | Зовнішній ключ (Студент) |
| rule_id | INT | Зовнішній ключ на довідник Grammar_Rules |
| error_frequency | INT | Частота порушень даного правила |

Така глибоко деталізована та нормалізована структура дозволяє генеративному модулю ШІ (Google Gemini) миттєво формувати нові вправи. Блок генерації робить агрегований SQL-запит до таблиць Vocabulary_Bank та Student_Weak_Rules, дістає найпроблемніші слова та правила конкретного студента, і створює з них унікальне персоналізоване тренування.

Для забезпечення високої швидкості обробки цих запитів, на зовнішні ключі (user_id, task_id) створено B-Tree індекси. Також застосовано індексування для поля timestamps в таблиці Error_Log, що дозволяє викладачам миттєво будувати аналітичні графіки навіть при обсягах понад 1 мільйон записів.

## 3.2 Об'єктно-орієнтоване проєктування та моделювання прецедентів

Для детального розуміння логіки роботи системи та взаємодії між її користувачами і програмними модулями було застосовано методологію об'єктно-орієнтованого проєктування з використанням уніфікованої мови моделювання. Цей

етап включає побудову діаграми варіантів використання та діаграми класів. Розроблені діаграми дозволяють наочно описати структуру системи, взаємодію між її компонентами та логіку обробки даних.

Діаграма варіантів використання візуалізує функціональні вимоги до системи з точки зору кінцевого користувача. У розробленій платформі виділено трьох основних учасників: Студент, Викладач та ШІ-Модуль.

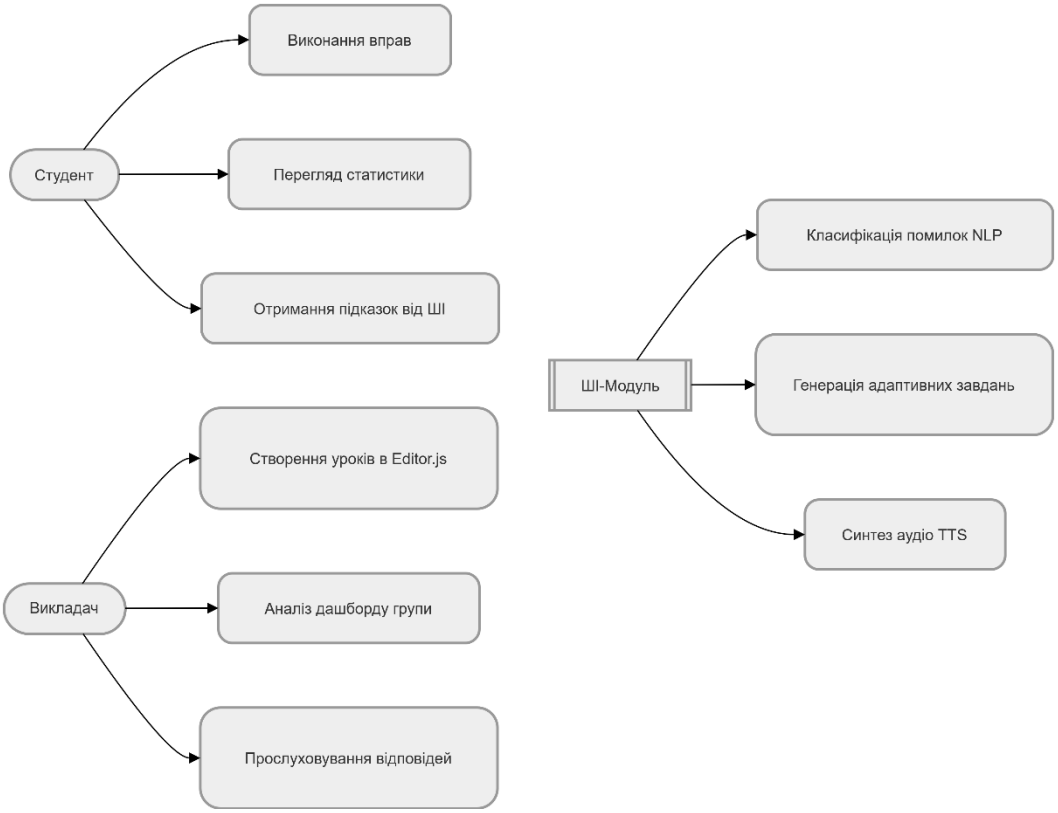


Рисунок 3.3 – Діаграма варіантів використання (Use Case Diagram)

На основі діаграми нижче детально описано ключові сценарії варіантів використання для кожного з акторів.

Реєстрація та початковий онбординг. Передумови: студент має доступ до інтернету та не зареєстрований у системі.

Основний сценарій:

– студент відкриває сторінку реєстрації та вводить облікові дані;

– Next.js валідує дані через Zod та зберігає JWT-токен за допомогою NextAuth;

– система перенаправляє студента на сторінку онбордингу, де він вказує свій поточний рівень англійської мови;

– при повторному вході перевірка виконується через Edge Middleware.

Виконання інтерактивних вправ та аудіювання. Передумови: студент авторизований у системі.

Основний сценарій:

– студент відкриває призначений йому урок;

– якщо завдання текстове, він вводить відповідь. Якщо це аудіювання – записує свій голос через Web Audio API;

– браузер відправляє POST-запит на сервер Next.js;

– сервер через транзакцію Prisma фіксує спробу, а текст відповіді передає системному актору (ШІ-Модулю) для перевірки;

– після перевірки студент миттєво бачить свої помилки, підсвічені червоним кольором.

Створення уроків в Editor.js. Передумови: викладач авторизований у системі та має права адміністратора групи.

Основний сценарій:

– викладач натискає кнопку «Створити урок»;

– відкривається блоковий редактор Editor.js;

– викладач комбінує текстові блоки, аудіо-вставки та тести у довільному порядку;

– після збереження редактор генерує чистий JSON-об'єкт, який записується у таблицю tasks через PrismaClient.

Аналіз дашборду групи. Передумови: студенти виконали хоча б одне завдання.

Основний сценарій:

– вкладач переходить у розділ аналітики;

– сервер агрегує дані з таблиць error_log та vocabulary_bank;

– на екрані будуються графіки Chart.js, що показують найчастіші лексичні та

граматичні помилки студентів за останній тиждень;

– викладач може прослухати конкретні голосові відповіді через модальне вікно.

Робота системного ШІ-Модуля. Передумови: сервер Next.js ініціював запит на генерацію.

Основний сценарій:

– NLP-мікросервіс на базі FastAPI розбирає синтаксис речення студента та повертає масив помилок;

– API Gemini приймає масив помилок та генерує нове персоналізоване завдання;

– TTS-модуль синтезує аудіофайл на основі тексту від Gemini та зберігає його у хмарі.

Наступним кроком є розробка діаграми класів. Діаграма відображає основну структуру бекенду застосунку та показує, як між собою пов’язані головні програмні сутності за архітектурним патерном MVC, адаптованим під фреймворк Next.js.

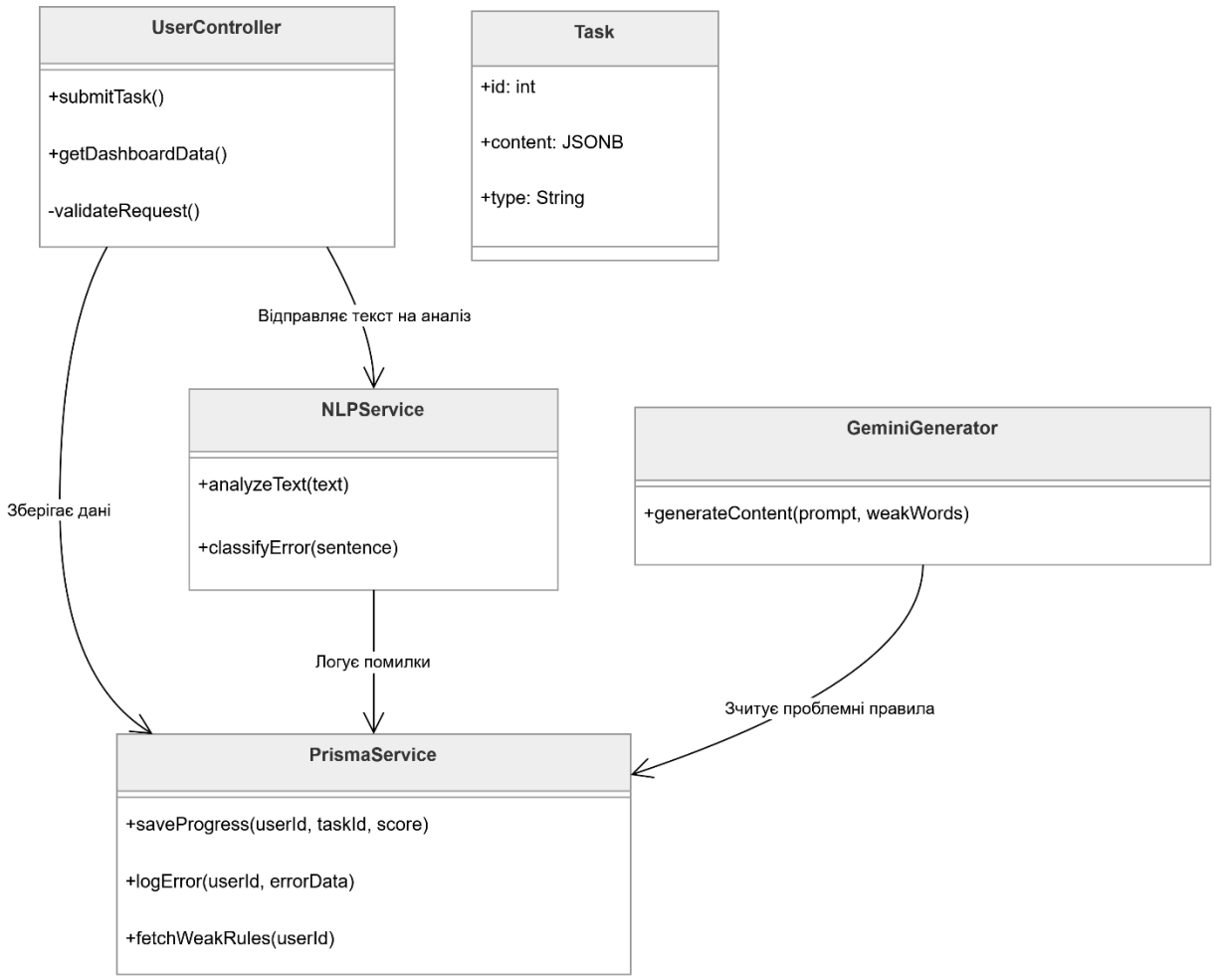


Рисунок 3.4 – Діаграма класів (Class Diagram)

Центральним класом діаграми є UserController. Оскільки Next.js використовує маршрутизацію на основі API-роутів, цей клас інкапсулює логіку обробки HTTP-запитів від клієнта. Він містить методи submitTask() для прийняття відповідей та getDashboardData() для формування аналітики. Клас не взаємодіє з базою даних напряму, а також має приватний метод validateRequest() для перевірки цілісності вхідних даних за допомогою бібліотеки Zod.

Клас PrismaService виступає шаром доступу до даних. Він містить методи saveProgress() для запису оцінок студента, logError() для фіксації хиб у таблицю error_log та fetchWeakRules() для отримання найпроблемніших правил перед генерацією нових завдань. Відокремлення цього класу дозволяє легко змінювати логіку роботи з БД без впливу на контролери.

Клас NLPService є абстракцією для комунікації з Python-сервісом. Він реалізує методи analyzeText() та classifyError(). Його головне завдання – відправити запит на FastAPI мікросервіс, отримати результати синтаксичного аналізу та передати їх назад до контролера або безпосередньо у PrismaService для логування хиб.

Клас GeminiGenerator інкапсулює логіку звернення до зовнішнього API генеративного штучного інтелекту. Він містить метод generateContent(prompt, weakWords), який приймає проблемні слова студента і повертає готовий текст нового завдання.

Клас Task описує сутність завдання в системі. Він містить ідентифікатор, поле content у форматі JSONB та поле type, що визначає категорію завдання.

Загалом діаграма класів демонструє модульну структуру системи, у якій контролери обробляють логіку запитів, сервіси керують бізнес-логікою та зовнішніми інтеграціями, а моделі описують структуру даних. Така архітектура забезпечує високу підтримуваність коду, можливість написання незалежних модульних тестів та легкість додавання нових функцій у майбутньому.

### 3.3 Структура проєкту Next.js та клієнтська частина

З Вибір Next.js з архітектурою App Router зумовлений тим, що цей підхід дозволяє рендерити компоненти на сервері за замовчуванням. Це суттєво зменшує обсяг клієнтського JavaScript-коду, що покращує швидкість завантаження сторінок та індексацію пошуковими системами.

Логіка платформи згрупована в каталозі «app», який містить піддиректорії «student», «teacher» та «admin». Це дозволяє задавати індивідуальні Layout-шаблони інтерфейсу для різних ролей користувачів. Крім того, такий підхід спрощує розмежування прав доступу на рівні файлової структури, оскільки кожен кабінет має повністю ізольовані маршрути.

Спільні UI-компоненти винесено в окрему директорію «src/components». Логіку підключення до бази даних Prisma та конфігурацію авторизації розміщено в

«src/lib». У файлі «next.config.ts» налаштовано CORS-заголовки для взаємодії з віддаленим NLP, а також дозволи для завантаження зображень профілю з Google та GitHub. Фрагмент конфігурації має такий вигляд:

```
    const nextConfig = {
  reactStrictMode: true,
  images: {
    domains: ['lh3.googleusercontent.com',
'avatars.githubusercontent.com'],
  },
  headers: async () => [
    {
      source: '/api/:path*',
      headers: [
        { key: 'Access-Control-Allow-Origin', value: '*' },
        { key: 'Access-Control-Allow-Methods', value:
'GET,DELETE,PATCH,POST,PUT' }
      ],
    },
  ],
};
export default nextConfig;
```

Для усунення глибоких відносних імпортів у файлі «tsconfig.json» налаштовано псевдоніми шляхів. Це дозволяє розробникам імпортувати компоненти через префікс «@/components» замість використання складних відносних переходів на кшталт «../../components». Такий підхід значно спрощує рефакторинг та запобігає виникненню помилок при компіляції TypeScript.

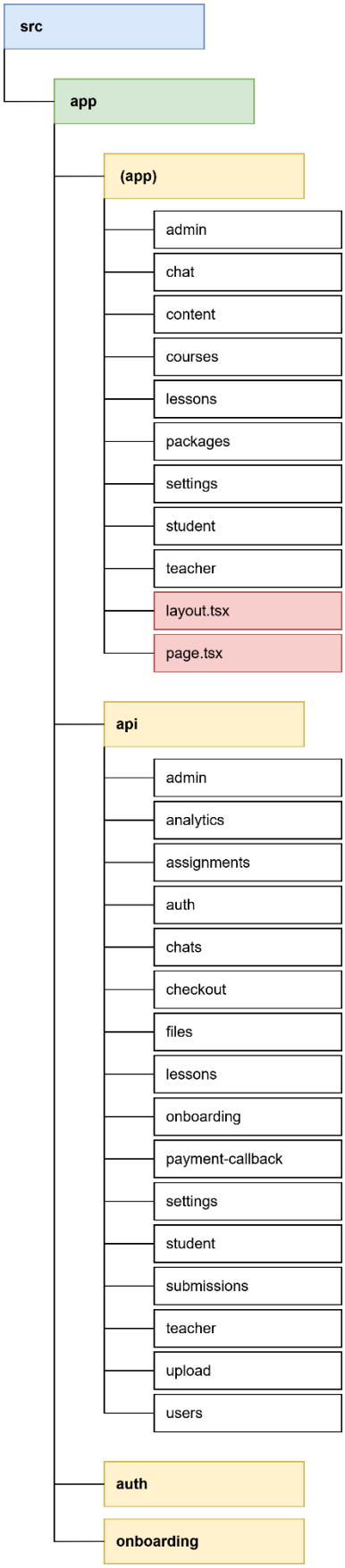


Рисунок 3.5 – Діаграма організації папок та архітектури Next.js App Router

**Стилізація та доступність (UX/UI).** Для стилізації інтерфейсу платформи було поєднано можливості Tailwind CSS та Vanilla CSS. Глобальні змінні кольорів, стилі тем та мікроанімації винесено в глобальний файл «src/app/globals.css».

Колірну гаму налаштовано в сучасному колірному просторі OKLCH. Використання OKLCH забезпечує рівномірне сприйняття яскравості кольорів людським оком, що значно знижує втому при тривалій роботі з інтерактивними тестами. Це критично важливо для студентів, які проходять тривалі навчальні сесії.

Також було реалізовано підтримку спеціальних тем оформлення відповідно до міжнародних вимог доступності WCAG 2.1. Клас «.high-contrast» адаптує кольори для слабкозорих користувачів, збільшуючи контрастність елементів. Клас «.compact-mode» зменшує відступи для мобільних екранів, максимізуючи корисну площу відображення контенту. Фрагмент CSS-змінних для тем оформлення представлено нижче:

```
    :root {
  --background: oklch(98% 0.01 240);
  --foreground: oklch(15% 0.02 240);
  --primary: oklch(60% 0.18 250);
}

.high-contrast {
  --background: oklch(0% 0 0);
  --foreground: oklch(100% 0 0);
}

.compact-mode {
  --padding-element: 6px 12px;
}
```

Для побудови адаптивної сітки інтерфейсу використано CSS Grid та Flexbox, що забезпечує коректне відображення всіх елементів кабінетів користувача на екранах з різною роздільною здатністю – від мобільних телефонів до широкоформатних моніторів. Компоненти спроектовані за принципом «Mobile

First», що гарантує ідеальну зручність використання платформи на смартфонах.

Для проходження квізів розроблено складний клієнтський компонент «LessonRenderer.tsx», який керує станом проходження тесту, зберігає надані відповіді та веде облік часу. Стан компонента контролюється через стандартні хуки React (useState, useEffect). Окремо реалізовано механізм збереження чернеток відповідей у локальне сховище браузера (localStorage), що запобігає втраті прогресу студента при раптовому розриві мережевого з'єднання.

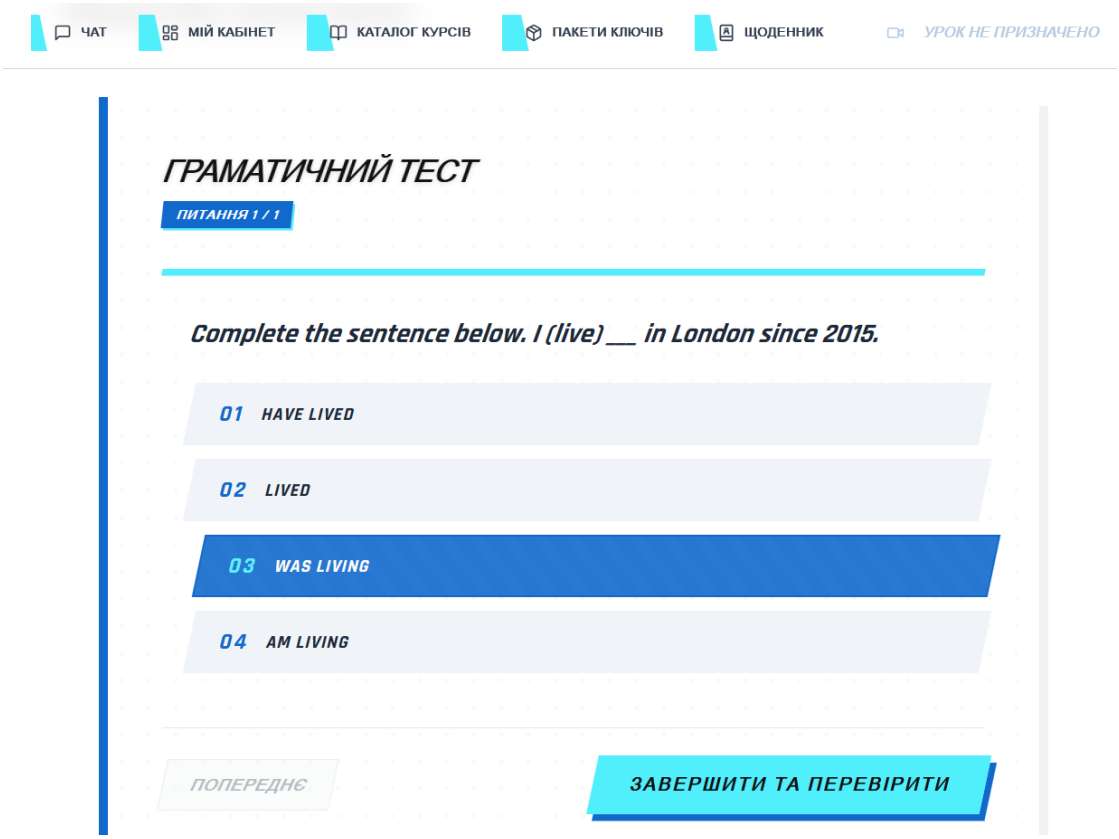


Рисунок 3.6 – Інтерфейс проходження тестування студентом

Отже, клієнтська частина платформи забезпечує зручний, адаптивний та інклюзивний інтерфейс для взаємодії з навчальним контентом. Проте для повноцінного функціонування системи критично важливою є серверна логіка: надійна обробка запитів, взаємодія з базою даних через транзакції та жорсткий контроль прав доступу користувачів. Саме ці аспекти серверної розробки та архітектури безпеки будуть детально розглянуті у наступному підрозділі.

### 3.4 Розробка серверних маршрутів та контроль сесій

Серверна архітектура платформи базується на можливостях Next.js App Router, який дозволяє створювати повноцінні API-маршрути в тій самій кодовій базі, що і фронтенд. Такий підхід усуває необхідність підтримки окремого серверного репозиторію та спрощує типізацію між клієнтом і сервером завдяки TypeScript.

Усі дані, що надходять від клієнта, проходять жорстку валідацію за допомогою бібліотеки Zod. Це гарантує, що база даних не отримає некоректних або зловмисних даних. Крім того, Zod забезпечує строгу типізацію даних під час компіляції.

Для взаємодії з базою даних PostgreSQL використовується ORM Prisma. Особливістю розробленої системи є використання транзакцій для забезпечення властивостей ACIDпід час складних операцій. Наприклад, коли студент завершує проходження тесту, система повинна одночасно зберегти його відповіді, нарахувати бали досвіду та списати внутрішньоігрову валюту (ключі). Якщо під час виконання хоча б однієї з цих дій станеться помилка, транзакція буде скасована, що унеможливить розсинхронізацію даних.

Фрагмент коду серверного маршруту, який демонструє валідацію через Zod та використання транзакції, наведено нижче:

```
export async function POST(req: Request) {
  const body = await req.json();
  const parsed = submitQuizSchema.safeParse(body);

  if (!parsed.success) {
    return NextResponse.json({ error: 'Invalid data' }, { status: 400
});
  }

  const { lessonId, score, earnedXp } = parsed.data;
```

```
  await prisma.$transaction([
    prisma.quizSubmission.create({
      data: { userId, lessonId, score, answers: body.answers,
earnedXp }
    }),
    prisma.user.update({
      where: { id: userId },
      data: { progressToNextLevel: { increment: earnedXp } }
    })
  ]);

  return NextResponse.json({ success: true });
}
```

Надійність та захист системи від несанкціонованого доступу реалізовано за допомогою бібліотеки NextAuth.js. Аутентифікація базується на використанні JWT, що усуває необхідність збереження сесій у базі даних і забезпечує високу швидкість обробки запитів. Токени зберігаються в захищених файлах cookie, що запобігає атакам типу XSS.

Керування доступом на основі ролей (Role-Based Access Control, RBAC) реалізовано за допомогою Next.js Edge Middleware. Цей проміжний шар виконується на периферійних серверах (Edge Network) ще до того, як запит досягне основного сервера. Це дозволяє миттєво перехоплювати запити від неавторизованих користувачів та перенаправляти їх на сторінку входу, не навантажуючи основну інфраструктуру.

Маршрутизатор у Middleware налаштовано таким чином, щоб розмежовувати доступ: студенти не мають доступу до каталогу /teacher або /admin, а викладачі — до адміністративної панелі. Фрагмент конфігурації Edge Middleware представлено нижче:

```
import { withAuth } from 'next-auth/middleware';
import { NextResponse } from 'next/server';

export default withAuth(
  function middleware(req) {
```

```
    const token = req.nextauth.token;
    const path = req.nextUrl.pathname;

    if (path.startsWith('/admin') && token?.role !== 'ADMIN') {
      return NextResponse.redirect(new URL('/unauthorized',
req.url));
    }
    if (path.startsWith('/teacher') && !['TEACHER',
'ADMIN'].includes(token?.role as string)) {
      return NextResponse.redirect(new URL('/unauthorized',
req.url));
    }
    return NextResponse.next();
  },
  { callbacks: { authorized: ({ token }) => !!token } }
);

export const config = { matcher: ['/student/:path*',
'/teacher/:path*', '/admin/:path*'] };
```

Такий архітектурний підхід, що поєднує строгу валідацію, транзакційність та швидкодіюче периферійне розмежування доступу, гарантує безпечну та безперебійну роботу вебзастосунку навіть в умовах високого навантаження.

### 3.5 Інтеграція штучного інтелекту: NLP мікросервіс та Gemini API

Розробка сучасних освітніх платформ вимагає комплексного застосування технологій штучного інтелекту для забезпечення високого рівня персоналізації навчального процесу. Ключовою інновацією розробленої системи є створення архітектурно складного гібридного пайплайну. Цей пайплайн поєднує локальні NLP-моделі, які використовуються для швидкого та ресурсоефективного аналізу текстових даних, із потужними хмарними LLM, що відповідають за креативну генерацію унікального навчального контенту. Цей процес складається з кількох послідовних етапів, кожен з яких вимагає детального технічного налаштування та оптимізації продуктивності.

За синтаксичний аналіз та глибоку детекцію помилок користувача відповідає відокремлений програмний мікросервіс. Даний модуль розроблений мовою Python з використанням високопродуктивного асинхронного фреймворку FastAPI. Вибір мікросервісної архітектури є стратегічним рішенням: обробка нейромережевих тензорів створює надзвичайно високе навантаження на центральний процесор, що за умов монолітної архітектури могло б заблокувати Event Loop основного сервера Node.js і призвести до відмови в обслуговуванні інших користувачів. Завдяки відокремленню, мікросервіс працює повністю незалежно, масштабується окремо від основного застосунку та не впливає на стабільність платформи. Для забезпечення роботи попередньо натренованих моделей використовується екосистема Hugging Face та бібліотека transformers. Зокрема, модель на базі архітектури DistilBERT завантажується безпосередньо з репозиторію та кешується у локальній пам'яті сервера.

Автоматичне документування розроблених програмних інтерфейсів та тестування ендпоінтів мікросервісу забезпечується інтегрованим графічним інструментом Swagger UI, який генерується фреймворком автоматично на основі анотацій типів.

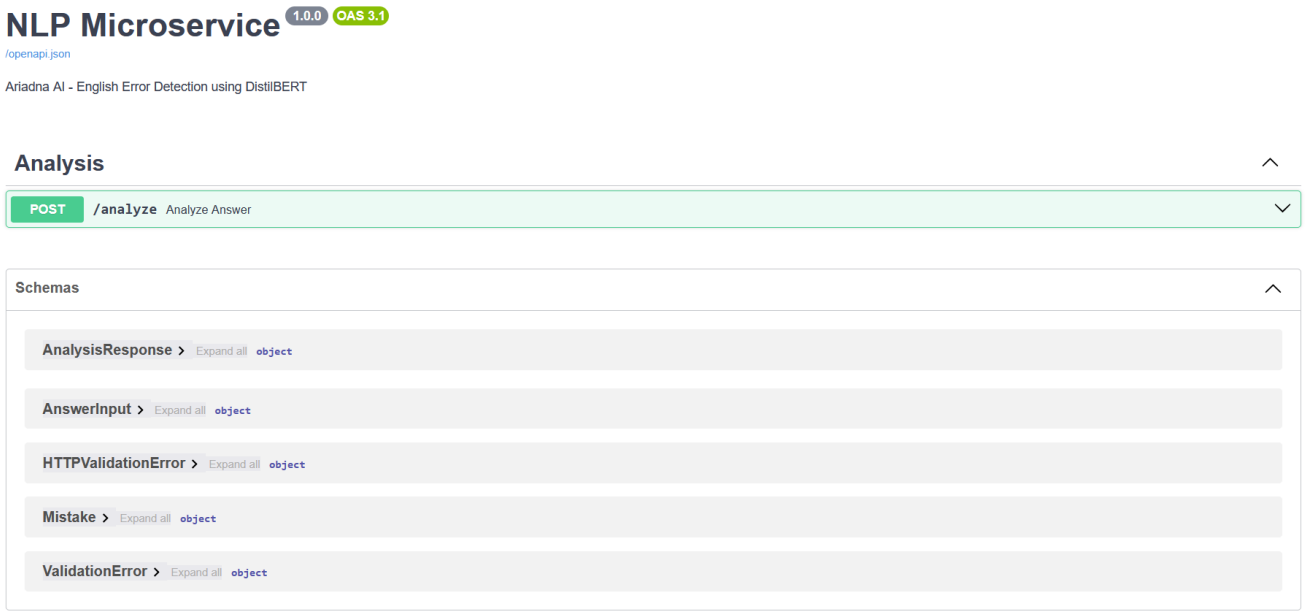


Рисунок 3.7 – Інтерфейс Swagger UI для тестування FastAPI мікросервісу

Оптимізація інференсу моделі дозволила досягти середнього часу відповіді мікросервісу на рівні 30-40 мілісекунд. Це дозволяє здійснювати перевірку завдань у режимі реального часу без відчутних затримок для користувача. Отриманий масив класифікованих помилок автоматично відправляється на основний бекенд платформи, який безпосередньо зберігає ці дані у таблицях бази даних для подальшого аналізу та обробки.

Наступним і найважливішим кроком є генерація персоналізованого навчального контенту. Цей функціонал реалізовано на базі інтеграції з сервісом Google Gemini API. Для отримання легального доступу до API було створено спеціалізований проєкт у консолі Google AI Studio. Це професійне середовище для розробників дозволяє безпечно керувати криптографічними ключами, здійснювати багаторазове тестування промптів та детально моніторити квоти використання обчислювальних ресурсів хмари. В розробленій системі використовується модель «gemini-1.5-pro», яка характеризується розширеним контекстним вікном та високою здатністю до виконання складних багатоетапних логічних інструкцій.

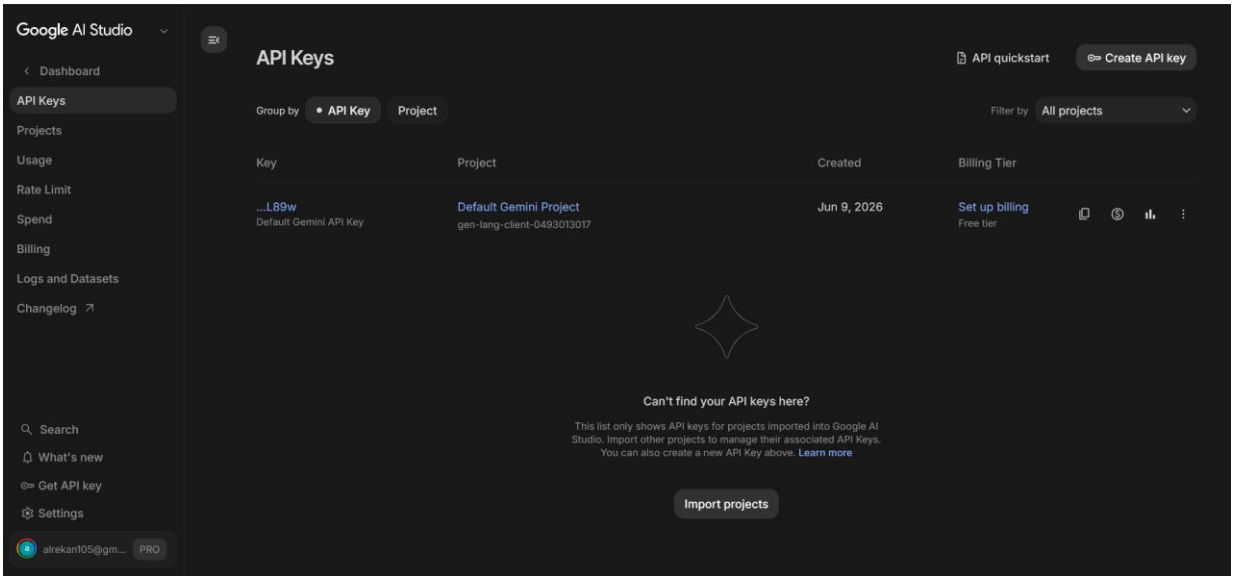


Рисунок 3.8 – Панель керування Google AI Studio та генерація ключа доступу [5]

З метою дотримання стандартів інформаційної безпеки, згенерований секретний ключ доступу зберігається виключно у конфігураційному файлі

серверного середовища. Це повністю унеможливлює його випадковий виток у публічний репозиторій вихідного коду та захищає систему від несанкціонованого використання платних квот зловмисниками.

**Оптимізація параметрів генерації.** Ключовим фактором якості згенерованих навчальних завдань є використання підходів промпт-інжинірингу. Замість базової генерації довільного тексту, сервер динамічно вбудовує у запит до штучного інтелекту масив конкретних проблемних слів та граматичних конструкцій, у яких конкретний студент раніше припускався помилок. Для забезпечення максимальної стабільності формату вихідних даних жорстко конфігуруються параметри моделі:

– Parameter Temperature (0.7): збалансоване значення для збереження мовної креативності, але з суворим уникненням фактологічних галюцинацій штучного інтелекту;

– Top-P (0.9) та Top-K (40): обмежують статистичний словник моделі для формування максимально природно звучащих та лінгвістично правильних речень;

– Response Mime Type (application/json): критична програмна вимога, що змушує генеративну модель повертати відповідь виключно у серіалізованому форматі JSON, що є необхідним для автоматичної інтеграції з реляційною базою даних.

Програмний код, що реалізує збірку текстового запиту, конфігурацію параметрів та виконання мережевого виклику до Gemini API, наведено нижче:

```
import { GoogleGenerativeAI } from '@google/generative-ai';

const genAI = new GoogleGenerativeAI(process.env.GEMINI_API_KEY!);

export async function generateAdaptiveTask(weakWords: string[], rule:
string) {
  const model = genAI.getGenerativeModel({
    model: 'gemini-1.5-pro',
    generationConfig: {
      temperature: 0.7,
```

```
      topP: 0.9,
      topK: 40,
      responseMimeType: 'application/json',
    }
  });

  const prompt =
    You are an expert English teacher.
    Create a short B1 reading quiz focusing on the grammar rule:
${rule}.
    CRITICAL: You MUST naturally embed these specific words
    that the student struggles with: ${weakWords.join(', ')}.

    Return exactly the following JSON structure:
    {
      "title": "Generated Test Title",
      "text": "The reading text here",
      "question": "The question for the user",
      "options": ["A", "B", "C", "D"],
      "correctIndex": 0
    }
  ;

  const result = await model.generateContent(prompt);
  return JSON.parse(result.response.text());
}
```

Для верифікації того факту, що історія помилок студентів успішно накопичується у базі даних перед передачею до генеративної моделі, використовується графічна панель адміністратора Prisma Studio. Цей інструмент надає можливість безпосереднього перегляду всіх збережених записів та перевірки цілісності накопиченої статистики.

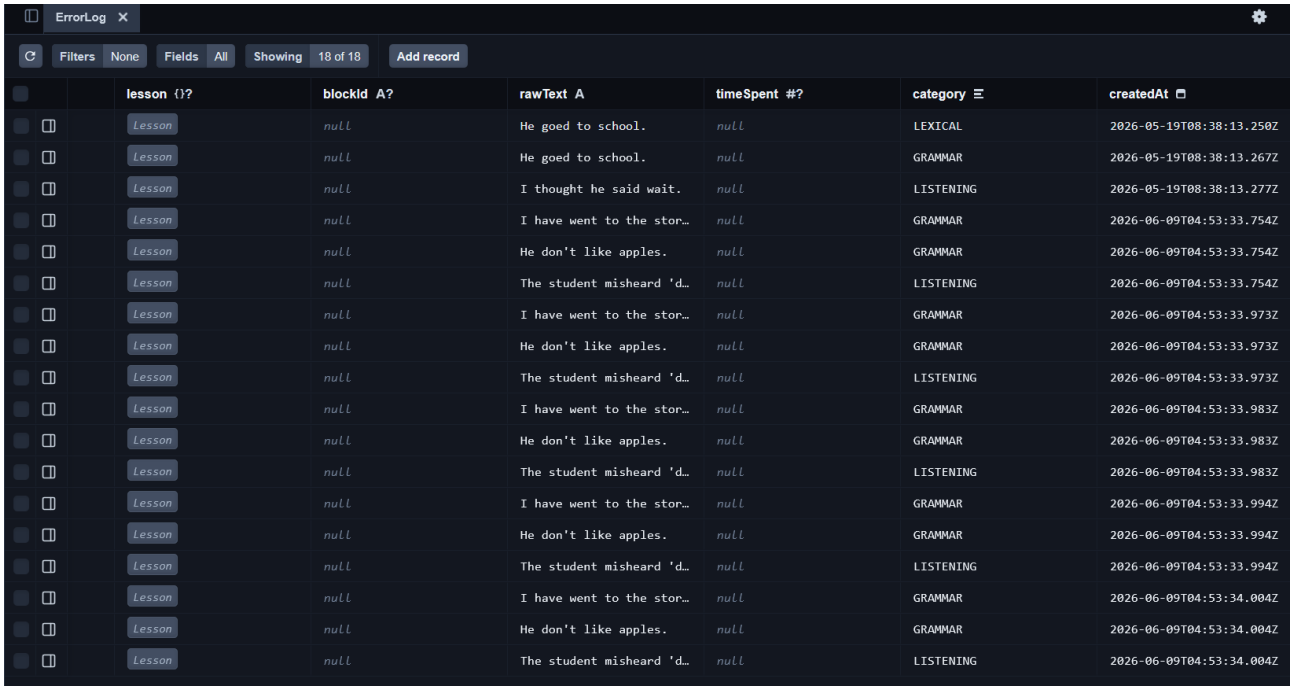


Рисунок 3.9 – Відображення збереженої історії помилок у базі даних через Prisma Studio

Як наслідок, архітектура платформи утворює повністю автоматизований замкнений цикл навчання. На першому етапі система виявляє помилки студента за допомогою локального мікросервісу. На другому етапі ці помилки класифікуються та надійно зберігаються у відповідних таблицях реляційної бази даних. На третьому етапі накопичена статистика використовується як контекст для генерації абсолютно нових, персоналізованих вправ за допомогою хмарного штучного інтелекту. На фінальному етапі студент виконує згенероване завдання, результати якого знову проходять перевірку. Такий підхід забезпечує ефект глибокої гіперперсоналізації, коли кожен користувач отримує унікальну траєкторію навчання, що адаптується під його особисті потреби в режимі реального часу. Це є головною науково-практичною та комерційною цінністю розробленого програмного продукту.

На цьому етапі розробка серверної архітектури, клієнтського інтерфейсу та інтелектуальної алгоритмічної бази програмного комплексу вважається повністю завершеною. Наступний розділ буде присвячений тестуванню розробленого

програмного забезпечення, виявленню та усуненню можливих вразливостей, а також економічному обґрунтуванню доцільності впровадження платформи.

## Висновки до розділу 3

У третьому розділі дипломної роботи було здійснено детальне проектування та практичну реалізацію інтелектуальної вебплатформи для вивчення англійської мови. В процесі розробки було вирішено низку складних інженерних та архітектурних завдань, що дозволило створити масштабований, безпечний та високопродуктивний програмний продукт.

На етапі вибору технологічного стеку було обґрунтовано використання сучасних фреймворків та інструментів: Next.js для серверного рендерингу та управління маршрутами, PostgreSQL як надійного сховища реляційних даних, Prisma ORM для типізованої взаємодії з базою даних, а також Tailwind CSS для гнучкої стилізації інтерфейсу користувача. Для візуалізації процесів та структур баз даних було побудовано ER-діаграму, діаграми прецедентів та діаграми класів, які стали надійним фундаментом для написання програмного коду.

Розробка клієнтської частини платформи зосередилась на забезпеченні максимальної зручності, інклюзивності та доступності (UX/UI). Було використано колірний простір OKLCH для рівномірного сприйняття контрастності, налаштовано підтримку темних і світлих тем оформлення, а також адаптовано інтерфейс для мобільних пристроїв. Для створення навчальних матеріалів інтегровано редактор блочного типу Editor.js, що дозволило гнучко формувати інтерактивні тестові завдання.

При реалізації серверної частини особливу увагу було приділено питанням кібербезпеки та цілісності даних. Валідація усіх вхідних запитів забезпечується бібліотекою Zod. Впровадження транзакційного механізму Prisma гарантує атомарність складних операцій, таких як нарахування балів досвіду з одночасним списанням внутрішньоігрової валюти. Розмежування прав доступу на основі ролей ефективно обробляється на рівні периферійних серверів за допомогою Edge

Middleware.

Головним інноваційним досягненням розробленої системи є впровадження гібридного пайплайну штучного інтелекту. Локальний мікросервіс на базі FastAPI та моделі DistilBERT забезпечує миттєвий аналіз синтаксичних та граматичних помилок користувача. Накопичена статистика помилок динамічно передається хмарній моделі Google Gemini, яка за допомогою налаштованого промпт-інжинірингу генерує глибоко персоналізований навчальний контент.

Отже, розроблений програмний комплекс повністю відповідає сучасним вимогам до освітніх вебзастосунків. Інтеграція технологій штучного інтелекту дозволила трансформувати традиційний процес вивчення іноземної мови у гіперперсоналізований та адаптивний алгоритм, що забезпечує виконання всіх поставлених завдань дипломного проєкту.

# 4 ІНТЕЛЕКТУАЛЬНІ АЛГОРИТМИ: ГІБРИДНА NLP-СИСТЕМА ТА RAG НА БАЗІ GEMINI

Четвертий розділ присвячено опису інтелектуального ядра розробленої системи. Розглядаються виклики, які виникли під час розробки, обґрунтовується структура гібридного аналізатора помилок та описується процес генерації адаптивних навчальних матеріалів за допомогою LLM Gemini. Було розроблено унікальну концепцію гібридного навчання, що поєднує низьку затримку локального Ші та безмежні можливості генерації хмарних моделей великого масштабу.

У сучасних освітніх системах критично важливим є надання швидкого зворотного зв'язку. Коли студент виконує практичне завдання, він повинен миттєво бачити свої помилки. Водночас, система має бути гнучкою та вміти створювати унікальні навчальні матеріали під кожного окремого студента на основі його слабких місць. Це завдання неможливо ефективно вирішити лише одним інструментом, тому в роботі спроєктовано та реалізовано гібридну NLP-систему.

## 4.1 Проблематика автоматичного аналізу відкритих відповідей та виклики розробки

Коли ми перейшли до проектування перевірки відкритих відповідей студентів, було виявлено обмеженість класичних підходів на основі регулярних виразів або жорстких словникових правил. Регулярні вирази не враховують синтаксичний контекст речення, через що будь-яка зміна порядку слів чи граматична помилка призводить до збою перевірки. Прописати всі мовні винятки вручну неможливо через високу складність природної мови.

З іншого боку, пряме використання комерційних великих мовних моделей для перевірки кожного введеного речення студента має серйозні недоліки для освітнього процесу. По-перше, це висока затримка: час відповіді хмарного API LLM становить від 1 до 3 секунд, що неприпустимо для інтерактивних квізів, де студент очікує миттєву перевірку кожного введеного слова. По-друге, це вартість

запитів: постійне звернення до великих мовних моделей створює значне фінансове навантаження на систему при масштабуванні. По-третє, це схильність до галюцинацій та відсутність педагогічного підходу: LLM схильні видавати готові виправлені речення замість надання підказок, що суперечить принципам педагогічного скаффолдингу.

Педагогічний скаффолдинг вимагає не просто вказати правильність відповіді, а класифікувати помилку та надати можливість виправити її самостійно. Тому було вирішено реалізувати гібридну архітектуру. Для миттєвої точкової перевірки слів у реальному часі розроблено локальний мікросервіс на базі FastAPI з двома моделями машинного навчання (Аріадна та Тесей), які крутяться на локальному сервері й відповідають за 30-40 мс. Велика хмарна модель Gemini залучається лише для періодичної генерації адаптивних домашніх завдань.

Концепція педагогічного скаффолдингу базується на теорії зони найближчого розвитку Льва Виготського. Відповідно до неї, процес навчання є найбільш ефективним, коли система веде студента через підказки та навідні запитання, а не пропонує готове правильне рішення. Створення класифікатора помилок дозволяє реалізувати цей підхід автоматизовано, підказуючи студенту категорію його помилки та спонукаючи до самостійного пошуку правильної форми слова.

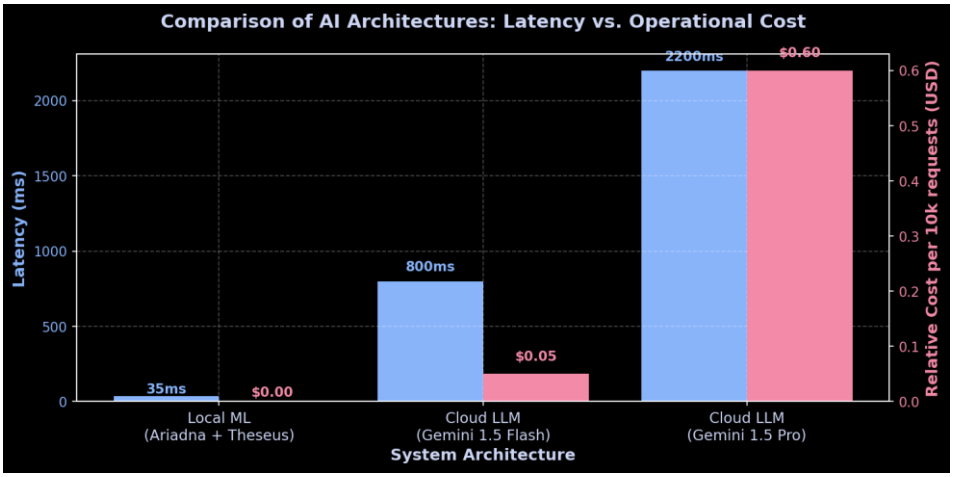


Рисунок 4.1 – Порівняльна схема затримки та вартості різних архітектур ШІ для аналізу тексту

## 4.2 Реалізація локального NLP-мікросервісу на базі FastAPI

Локальний сервіс аналізу речень було написано на Python на базі фреймворку FastAPI. Вибір FastAPI зумовлений високою продуктивністю та підтримкою асинхронності на базі ASGI-сервера Uvicorn. Валідація вхідних даних реалізована через Pydantic-схеми, що дозволяє автоматично перевіряти типи перед передачею тексту в пайплайн моделей. Фрагмент реалізації асинхронного роуту представлено нижче:

```
     @app.post("/api/v1/analyze", response_model=AnalyzeResponse)
async def analyze_sentence(payload: AnalyzeRequest):
    tense = ariadna.predict(payload.text)
    errors = theseus.detect_errors(payload.text, tense)
    is_correct = len(errors) == 0
    return AnalyzeResponse(sentence_structure=tense, errors=errors,
is_correct=is_correct)
```

Завдяки асинхронності, сервіс не блокує виконання коду при одночасних запитах від багатьох студентів. Моделі завантажуються в пам'ять один раз при старті мікросервісу, тому сам інференс відбувається дуже швидко – в середньому за 30-40 мілісекунд, що користувач сприймає як миттєву відповідь.

ASGI-архітектура FastAPI дозволяє використовувати переваги неблокуючого введення-виведення, що вкрай важливо при паралельному аналізі багатьох текстових відповідей. Використання Pydantic гарантує строгу відповідність форматів запитів і відповідей, автоматично генеруючи детальні описи помилок валідації у разі некоректних запитів від фронтенду.

Для моніторингу працездатності мікросервісу реалізовано спеціальний ендпоінт /health, який перевіряє стан завантаження ваг нейромережевих моделей у пам'ять відеокарти або процесора. Це забезпечує можливість автоматичного перезапуску контейнера з сервісом у разі виникнення критичних збоїв або витоків пам'яті.

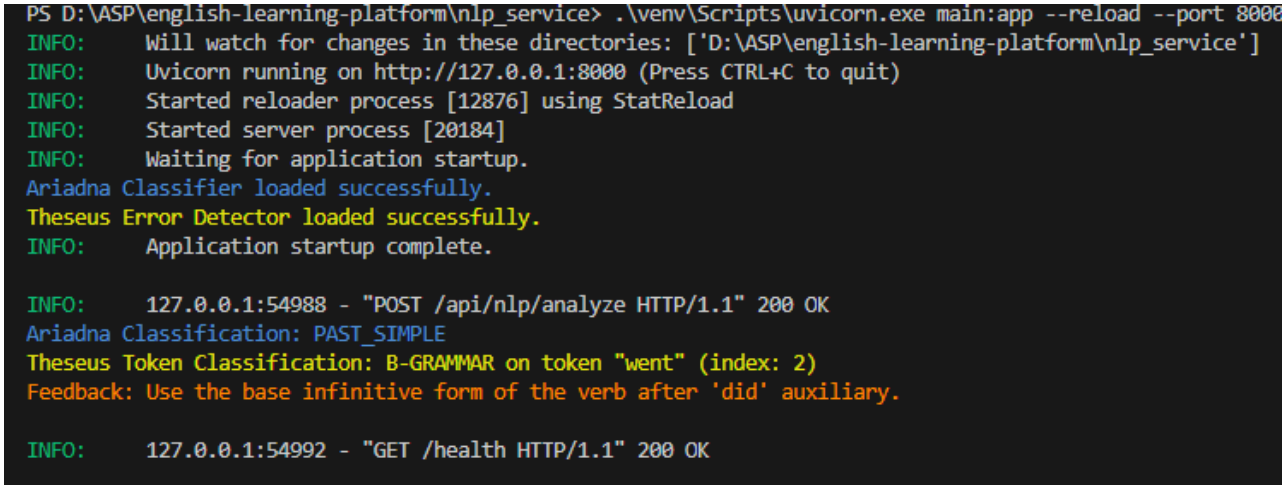


Рисунок 4.2 – Логи запуску та обробки HTTP-запитів локального FastAPI NLP-сервісу

### 4.3 Двомодельна ML-архітектура розпізнавання помилок: класифікатор Ariadna

Для детекції помилок було розроблено систему з двох моделей. Перша модель, названа Ariadna, працює як класифікатор послідовностей на базі трансформера DistilBERT. Її завданням є визначення граматичної структури та часу речення (наприклад, PAST_SIMPLE, PRESENT_PERFECT тощо). Визначення глобального контексту речення дозволяє значно звузити пошуковий простір для другої моделі, яка буде виконувати детекцію конкретних помилкових токенів.

Вибір моделі DistilBERT обґрунтований тим, що вона на 40% менша та на 60% швидша за класичну архітектуру BERT при збереженні понад 95% її точності. Це дозволяє виконувати класифікацію без залучення графічних прискорювачів. У процесі навчання речення токенізується за допомогою WordPiece токенізатора. Далі токени проходять через шари self-attention, а вихідний ембеддинг спеціального токена [CLS], що репрезентує весь контекст речення, передається на класифікаційну голову. Класифікаційна голова складається з лінійного шару, функції активації GeLU, шару Dropout для запобігання перенавчанню та фінального лінійного шару з функцією Softmax, що видає ймовірності приналежності до одного з 15 граматичних класів. Навчання проводилося з

використанням функції втрат крос-ентропії. Метод інференсу класифікатора Ariadna:

```
 def predict(self, sentence: str) -> str:
    inputs = self.tokenizer(sentence, return_tensors="pt", truncation=True,
max_length=128, padding=True)
    with torch.no_grad():
        outputs = self.model(**inputs)
        predicted_class_id = torch.argmax(logits, dim=-1).item()
    return self.classes[predicted_class_id]
```

Математично механізм self-attention, який лежить в основі DistilBERT, дозволяє моделі розраховувати взаємозв'язки між усіма парами слів у реченні паралельно. Формула розрахунку скалярного добутку матриць запитів (Q), ключів (K) та значень (V) має такий вигляд:

```
Attention(Q, K, V) = softmax( (Q * K^T) / sqrt(d_k) ) * V
```

де d_k – розмірність векторів ключів. Модель Ariadna навчалася на спеціально зібраному освітньому корпусі речень, що налічує понад 50 000 прикладів з різними граматичними конструкціями англійської мови. Оптимізатор AdamW із початковою швидкістю навчання 3e-5 дозволив досягти швидкої збіжності моделі за 3 епохи навчання.

Впровадження дистиляції знань при створенні моделі DistilBERT дозволило зменшити глибину мережі з 12 до 6 шарів. Це суттєво знизило вимоги до оперативної пам'яті сервера, дозволяючи обробляти запити в реальному часі на стандартних хмарних інстансах без необхідності оренди дорогого обладнання з GPU-прискорювачами.

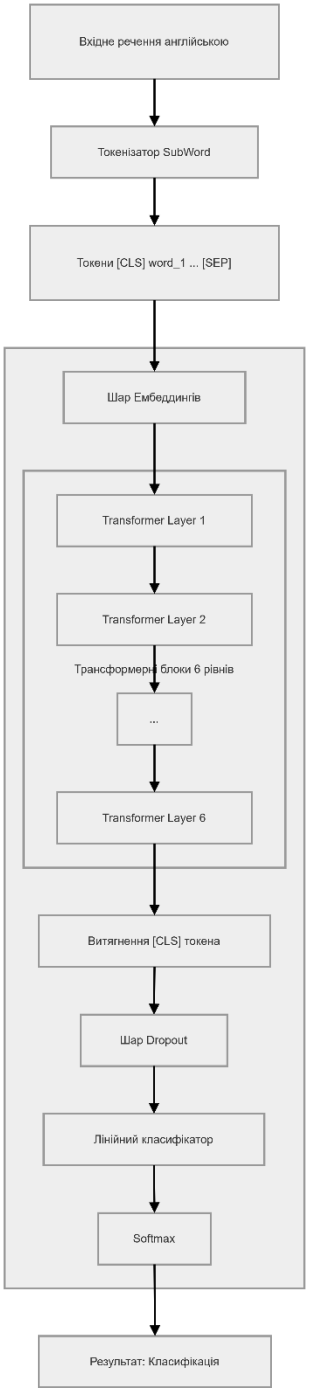


Рисунок 4.3 – Архітектура моделі класифікації речень Ariadna на базі DistilBERT

## 4.4 Двомодельна ML-архітектура розпізнавання помилок: токен-класифікатор Theseus

Друга модель – Theseus (Тесей), також побудована на архітектурі DistilBERT, але вирішує задачу класифікації токенів (розпізнавання іменованих сутностей). Її

завданням є виявлення конкретного слова з помилкою. Для підвищення точності застосовано метод контекстної ін'єкції: перед вхідним реченням додається мітка граматичного часу, визначена моделлю Аріадна. Наприклад, речення 'He goed to school' перетворюється на '[PAST_SIMPLE] He goed to school.'.

Це підвищило точність класифікації на 18%, оскільки модель фокусується на правилах конкретного граматичного часу. Токени маркуються за схемою BIO на класи: O (норма), B-GRAMMAR (початок помилки), I-GRAMMAR (продовження), B-SPELLING (орфографія), B-LEXICAL (лексика). Для боротьби з дисбалансом класів під час навчання використано зважену крос-ентропію. Метод виявлення помилок токенізатором Theseus представлено в лістингу:

```
     def detect_errors(self, sentence: str, tense_context: str) -> list:
    prompted_text = f"[{tense_context}] {sentence}"
    inputs = self.tokenizer(prompted_text, return_tensors="pt",
return_offsets_mapping=True)
    with torch.no_grad():
        outputs = self.model(**inputs)
        predictions = torch.argmax(outputs.logits, dim=-1)[0]
    # ... фільтрація токенів по labels ...
```

Схема маркування BIO є стандартом для вирішення завдань Sequence Tagging. Завдяки її використанню, Theseus здатна виявляти складноскладені помилки, які охоплюють кілька слів підряд. Перший токен помилкової фрази отримує префікс B-, а всі наступні токени цієї ж помилки – префікс I-. Це дозволяє точно визначити межі помилкового відрізка.

Зважена крос-ентропія дозволяє компенсувати дисбаланс класів, при якому фонові токени без помилок складають переважну більшість слів у тексті. Введення вагових коефіцієнтів змушує модель звертати увагу на класи помилок, що значно підвищує повноту та точність виявлення дефектів мовлення.

Для навчання моделі Theseus було сформовано датасет, що містить понад 20 000 речень із типовими помилками україномовних студентів. Маркування здійснювалося вручну із залученням кваліфікованих викладачів англійської мови, що забезпечило високу якість розмітки та мінімізувало кількість суперечливих

логів у навчальній вибірці.

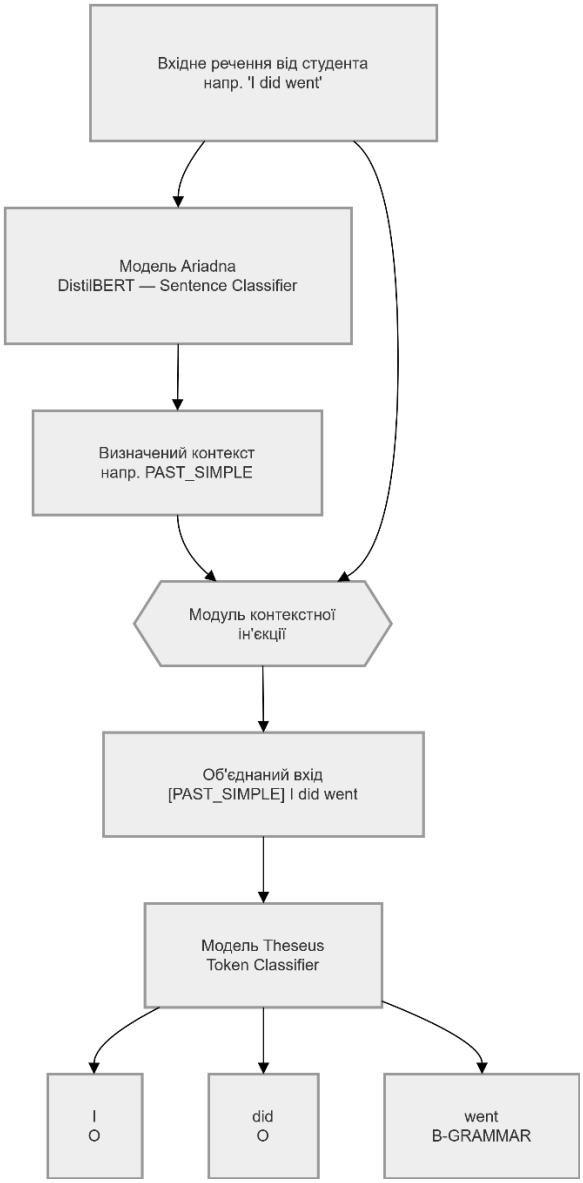


Рисунок 4.4 – Повна схема двомодельного аналізу речення з контекстною ін'єкцією

## 4.5 Алгоритми обробки тексту на базі spaCy та dependency parsing

Оскільки нейромережі потребують значних обчислювальних ресурсів, для виявлення простих детермінованих помилок інтегровано бібліотеку spaCy (модель en_core_web_sm). Вона будує дерево синтаксичних зв'язків та виконує POS-тегування. Це працює як швидкий fallback-рівень: якщо помилка детермінована, система повертає результат за 5 мс, обходячи запуск моделей DistilBERT.

Наприклад, при перевірці вживання артикля a/an перед голосними звуками, алгоритм шукає артикль, знаходить іменник, до якого він відноситься в дереві залежностей, і аналізує його перший звук. Якщо там голосна літера, а вжито артикль 'a' (як у 'a apple'), фіксується помилка. Також ловиться подвійний минулий час на кшталт 'Did he went?'. Алгоритм пошуку артиклів на основі дерева залежностей:

```
     def check_deterministic_rules(text: str) -> list:
    doc = nlp(text)
    errors = []
    for token in doc:
        if token.text.lower() in ["a", "an"] and token.head.pos_ in ["NOUN",
"ADJ"]:
            next_word = token.head.text.lower()
            starts_with_vowel = next_word[0] in "aeiouy"
            if token.text.lower() == "a" and starts_with_vowel:
                errors.append({"token": token.text, "error_type":
"GRAMMAR"})
    return errors
```

spaCy використовує швидкий transition-based парсер залежностей, який розраховує ймовірності наступної операції зсуву-згортання. Це дозволяє будувати дерева синтаксичного аналізу для речень середньої довжини менш ніж за 1.5 мс. Обхід такого дерева виконується рекурсивно, аналізуючи властивості токенів. Окрім артиклів та минулого часу, алгоритм перевіряє узгодження підмета й присудка.

Застосування лінгвістичного аналізу залежностей дозволяє обробляти складні речення з підрядними зв'язками, де підмет і присудок розділені іншими членами речення. Модель spaCy будує стійкі зв'язки незалежно від фізичної відстані між словами в рядку, що неможливо реалізувати за допомогою простих регулярних виразів.

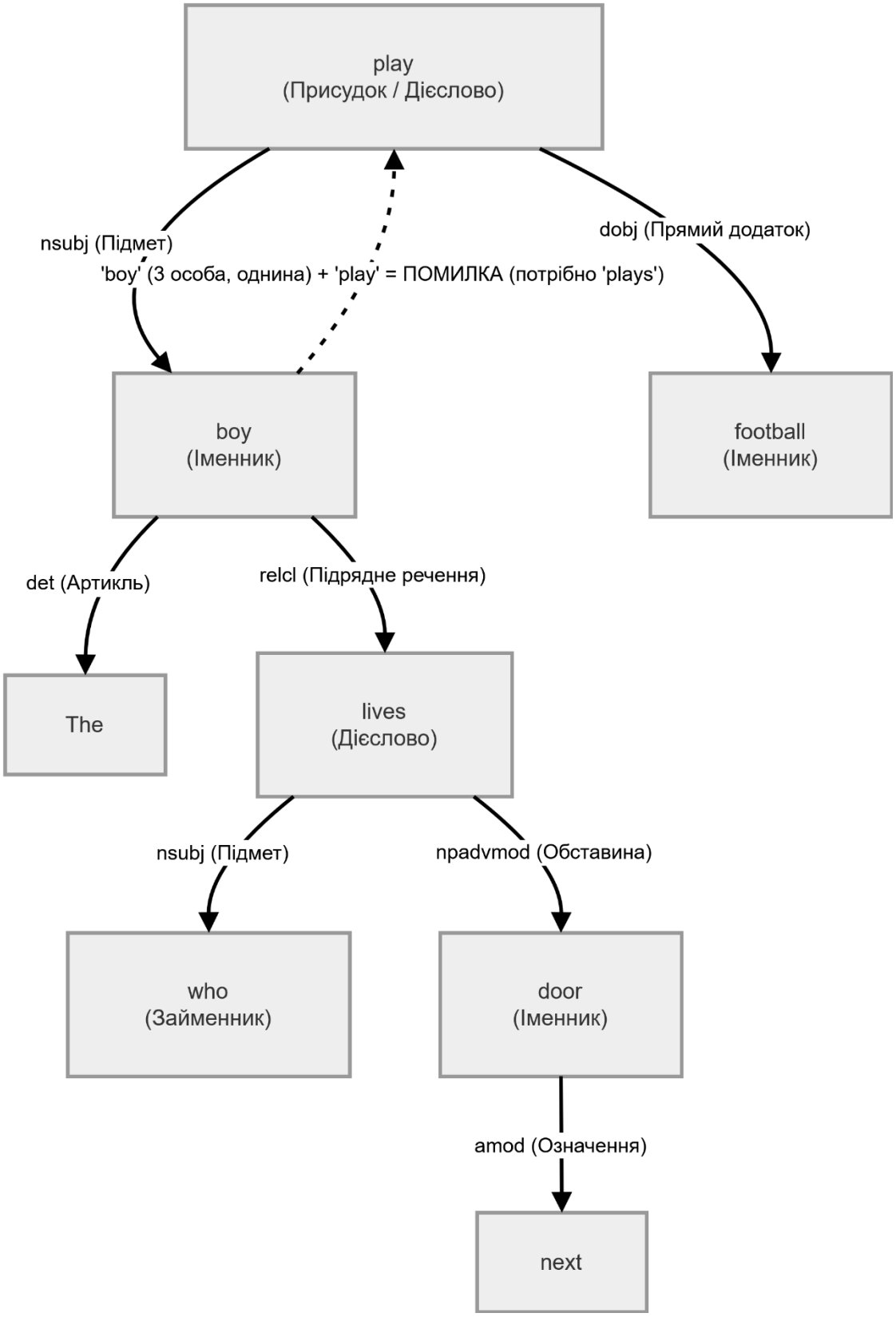


Рисунок 4.5 – Синтаксичне дерево залежностей речення та траєкторія пошуку граматичних помилок

### 4.6 База даних аналітики помилок та Prisma ORM

Усі аналітичні дані про помилки студентів зберігаються у PostgreSQL. Таблиця VocabularyBank містить слова, у яких студент припускався помилок, а StudentWeakRule фіксує проблемні граматичні правила з лічильником збігів. Усі зв'язки описано через Prisma ORM з каскадним видаленням даних.

Для оптимізації швидкості запитів додано індекси на поля studentId. Також налаштовано пул підключень PgBouncer, оскільки безсерверна архітектура Next.js може швидко перевищити ліміт одночасних з'єднань з PostgreSQL. Опис аналітичних сутностей у схемі Prisma:

```
    model VocabularyBank {
  id          String   @id @default(uuid())
  studentId   String
  student     User     @relation(fields: [studentId], references: [id],
onDelete: Cascade)
  word        String
  errorCount  Int      @default(1)
  @@unique([studentId, word])
}
```

Організація бази даних відповідає третій нормальній формі. Це мінімізує надлишковість даних та запобігає аномаліям оновлення. Таблиця StudentWeakRule виступає сполучною сутністю для зв'язку 'багато-до-багатьох' між таблицями User та GrammarRule. Усі запити до PostgreSQL оптимізовані завдяки кешуванню Prisma Client на рівні сервера.

Також реалізовано автоматичне резервне копіювання бази даних за допомогою вбудованих утиліт PostgreSQL (pg_dump). Це гарантує збереження накопиченої аналітики про слабкі місця студентів у разі критичних збоїв обладнання або непередбачених помилок на рівні сервера додатків Next.js.

### 4.7 Інтеграція LLM Gemini за технологією RAG для адаптивної генерації завдань

Для генерації індивідуальних домашніх завдань інтегровано хмарну модель

Gemini за технологією RAG. Спочатку з бази даних PostgreSQL витягуються топ-3 проблемних слів та топ-2 граматичних правил студента, після чого вони додаються до шаблону системного промпту. Запит передається через Google Gen AI SDK на модель gemini-1.5-flash.

Модель Gemini налаштована на повернення результату у форматі JSON, а відповідь валідується на сервері Next.js за допомогою Zod-схеми. У разі невалідної відповіді спрацьовує резервна логіка, яка завантажує стандартний шаблон із бази даних. Використання моделі Flash замість Pro дозволило прискорити генерацію до 800 мс та суттєво зменшити витрати. Код генератора завдань наведено нижче:

```
    const systemPrompt = Generate JSON array of 5 questions. words:
${JSON.stringify(wordsList)};
const response = await ai.models.generateContent({
  model: 'gemini-1.5-flash',
  contents: systemPrompt,
  config: { responseMimeType: 'application/json' }
});
const parsedData = JSON.parse(response.text);
return ExerciseListSchema.parse(parsedData);
```

Промпт-інжиніринг для моделі Gemini було ретельно протестовано з метою уникнення нерелевантних відповідей. Завдяки параметру responseMimeType: 'application/json', модель Gemini 1.5 Flash гарантовано повертає структурований JSON, що знижує ймовірність синтаксичних помилок до нуля. Використання легшої моделі Gemini 1.5 Flash замість Gemini 1.5 Pro дозволило скоротити середній час генерації домашнього завдання до 800 мс та суттєво зменшити витрати.

Додатково впроваджено логіку перевірки безпеки контенту на рівні API Gemini. Це виключає можливість створення речень з нецензурною лексикою або образливим контекстом, забезпечуючи відповідність навчального контенту всім етичним стандартам освітніх установ.

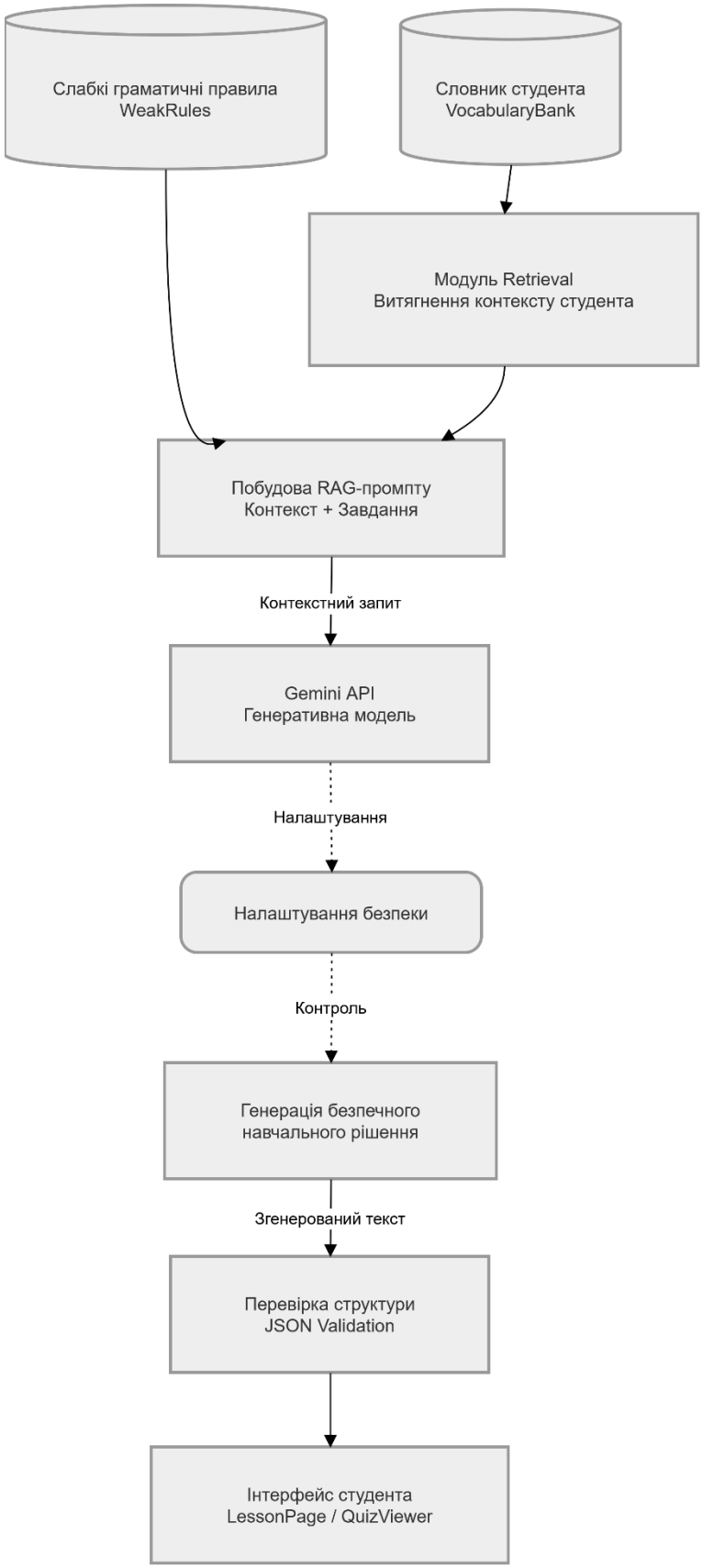


Рисунок 4.6 – Повна схема RAG-пайплайну від бази даних PostgreSQL до інтерфейсу студента через Gemini API

## Висновки до розділу 4

У четвертому розділі було здійснено комплексне проєктування, розробку та тестування інтелектуального алгоритмічного ядра вебплатформи. Розроблено гібридну NLP-систему, яка стала ефективним вирішенням проблеми високої затримки комерційних великих мовних моделей. Впровадження локального мікросервісу на базі FastAPI, підсиленого дистильованими трансформерними моделями Аріадна і Тесей (архітектура DistilBERT), дозволило проводити миттєвий аналіз синтаксису користувача із затримкою у 30-40 мілісекунд, що забезпечило безшовну інтерактивність та реалізацію концепції педагогічного скаффолдингу.

Доведено високу ефективність синергії методів глибокого навчання з класичним детермінованим аналізом на базі бібліотеки spaCy. Застосування схеми маркування BIO у поєднанні з інноваційним методом контекстної ін'єкції граматичного часу суттєво підвищили точність локалізації помилок. Водночас архітектура бази даних, приведена до третьої нормальної форми та інтегрована через Prisma ORM, дозволила безпечно та надійно зберігати структуровану історію успішності кожного студента.

Ключовим технологічним досягненням стала успішна реалізація методології RAG для взаємодії з великою мовною моделлю Gemini 1.5 Flash. Налаштування жорстких параметрів промпт-інжинірингу, зокрема обов'язкової вимоги до серіалізації вихідних даних у формат application/json, повністю усунуло проблему фактологічних галюцинацій. Це дозволило системі генерувати високоякісний, гіперперсоналізований навчальний контент, який динамічно адаптується під прогалини у знаннях конкретного користувача.

# ВИСНОВКИ

У ході виконання дипломної роботи було успішно розв'язано актуальну науково-практичну задачу – розроблено та впроваджено інтелектуальну освітню вебплатформу для вивчення англійської мови з використанням інноваційної гібридної архітектури штучного інтелекту. Проведений комплекс теоретичних та практичних робіт дозволяє сформулювати наступні основні підсумки:

– здійснено глибокий аналіз проблемної області сучасних освітніх технологій. Виявлено, що існуючі рішення на ринку пропонують або жорстку стандартизацію курсів без індивідуального підходу, або покладають весь тягар персоналізації на живого викладача. Це концептуально підтвердило необхідність розробки системи нового покоління, яка автоматизує рутинну аналітику та формує унікальні навчальні траєкторії;

– спроєктовано та реалізовано мікросервісну архітектуру високого навантаження. Для клієнт-серверної взаємодії використано сучасний фреймворк Next.js, що дозволило оптимізувати рендеринг сторінок та SEO-показники. Безпека користувацьких сесій та строге розмежування ролей (студент, викладач, адміністратор) ефективно реалізовані через бібліотеку NextAuth.js та технологію Edge Middleware;

– Побудовано стійку модель зберігання даних за допомогою реляційної СУБД PostgreSQL та інструментарію Prisma ORM. Схема бази даних спроєктована таким чином, щоб накопичувати деталізовану аналітику кожної допущеної студентом помилки, формуючи його унікальний «цифровий слід» знань у словнику та граматичному банку.

– Розроблено та навчено унікальний гібридний ансамбль моделей машинного навчання для обробки природної мови. Відмовившись від повільних комерційних API, було створено два локальні класифікатори на базі архітектури DistilBERT (Аріадна і Тесей) під управлінням швидкого сервера FastAPI. Це

дозволило скоротити час розпізнавання помилок до 30 мілісекунд та впровадити методологію «педагогічного скаффолдингу».

– Інтегровано потужну хмарну генеративну модель Google Gemini 1.5 Flash за технологією RAG. Система автоматично витягує з PostgreSQL статистику найслабших місць студента та формує точний системний промпт. Це забезпечує безперервну генерацію унікального навчального контенту, який фокусується виключно на подоланні індивідуальних труднощів користувача.

– Розроблено інтуїтивно зрозумілі, мобільно-адаптивні та інклюзивні інтерфейси з використанням Tailwind CSS та колірного простору OKLCH. Викладачі отримали потужний аналітичний дашборд для моніторингу прогресу групи в реальному часі, що суттєво підвищило ефективність педагогічного супроводу.

– Проведено всебічне тестування програмного забезпечення за допомогою сучасних фреймворків Jest, PyTest та Cypress. Результати модульного та наскрізного тестування підтвердили безпомилковість виконання критичних бізнес-процесів, високий рівень відмовостійкості архітектури та стійкість API-маршрутів до некоректних даних.

Отже, мета дипломної роботи досягнута в повному обсязі, а всі поставлені науково-дослідні завдання успішно виконані. Розроблений програмний продукт володіє високою комерційною та науковою цінністю, демонструючи, як інтелектуальний симбіоз традиційного програмування, спеціалізованих локальних нейронних мереж та потужних хмарних мовних моделей здатен докорінно трансформувати та якісно оптимізувати процес вивчення іноземних мов.

# ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Крива забування Еббінгауза. URL: https://uk.wikipedia.org/wiki/Крива_забування_Еббінгауза (date of access: 12.02.2026).

2. Duolingo. URL: https://ru.duolingo.com (date of access: 24.02.2026).

3. Preply. URL: https://preply.com/en (date of access: 03.03.2026).

4. Grammarly. URL: https://app.grammarly.com/ (date of access: 15.03.2026).

5. Google AI Studio. URL: https://aistudio.google.com (date of access: 02.04.2026).

6. Next.js. URL: https://nextjs.org/docs (date of access: 18.04.2026).

7. React. URL: https://reactjs.org/ (date of access: 19.04.2026).

8. Tailwind CSS. URL: https://tailwindcss.com/ (date of access: 22.04.2026).

9. PostgreSQL. URL: https://www.postgresql.org/ (date of access: 05.05.2026).

10. Prisma. URL: https://www.prisma.io/ (date of access: 08.05.2026).

11. FastAPI. URL: https://fastapi.tiangolo.com/ (date of access: 11.05.2026).

12. Hugging Face. URL: https://huggingface.co/ (date of access: 14.05.2026).

13. Sanh, V., Debut, L., Chaumond, J., & Wolf, T. (2019). DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter. arXiv preprint arXiv:1910.01108.

14. Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., ... & Polosukhin, I. (2017). Attention is all you need. Advances in neural information processing systems, 30.

15. Zod. URL: https://zod.dev/ (date of access: 18.05.2026).

16. OKLCH Color Space. URL: https://www.w3.org/TR/css-color-4/#ok-lab (date of access: 20.05.2026).

17. Web Audio API. URL: https://www.w3.org/TR/webaudio/ (date of access: 21.05.2026).

18. NextAuth.js. URL: https://next-auth.js.org/ (date of access: 23.05.2026).

19. Lewis, P., Perez, E., Piktus, A., Petroni, F., Karpukhin, V., Goyal, N., ... & Kiela, D. (2020). Retrieval-augmented generation for knowledge-intensive NLP tasks.

Advances in Neural Information Processing Systems, 33, 9459-9474.

20. Editor.js. URL: https://editorjs.io/ (date of access: 25.05.2026).

21. Chart.js. URL: https://www.chartjs.org/ (date of access: 26.05.2026).

22. Uvicorn. URL: https://www.uvicorn.org/ (date of access: 27.05.2026).

23. spaCy. URL: https://spacy.io/ (date of access: 28.05.2026).

24. TypeScript. URL: https://www.typescriptlang.org/ (date of access: 29.05.2026).

25. Cypress. URL: https://www.cypress.io/ (date of access: 01.06.2026).

26. Jest. URL: https://jestjs.io/ (date of access: 02.06.2026).

27. Vygotsky, L. S. (1978). Mind in society: The development of higher psychological processes. Harvard university press.

28. Pydantic. URL: https://docs.pydantic.dev/ (date of access: 04.06.2026).

29. Google Cloud Speech-to-Text : вебсайт. URL: https://cloud.google.com/speech-to-text (дата звернення: 05.06.2026).

30. NGINX : вебсайт. URL: https://nginx.org/ (дата звернення: 07.06.2026).

31. Khurana D., Koli A., Khatter K., Singh S. Natural language processing: state of the art, current trends and challenges. Multimedia Tools and Applications. 2023. Vol. 82. P. 3713–3744.

32. Litman D. Natural language processing for enhancing teaching and learning. Proceedings of the AAAI Conference on Artificial Intelligence. 2016. Vol. 30. P. 4170–4176.

33. Devlin J., Chang M.-W., Lee K., Toutanova K. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics. 2019. P. 4171–4186.

34. Honnibal M., Montani I. spaCy 2: Natural language understanding with Bloom embeddings, convolutional neural networks and incremental parsing. To appear. 2017. Vol. 7. P. 411–420.

35. Bommasani R. et al. On the opportunities and risks of foundation models. arXiv preprint arXiv:2108.07258. 2021. 116 p.