Міністерство освіти і науки України
Харківський національний університет імені В. Н. Каразіна
Навчально-науковий інститут комп'ютерних наук та штучного інтелекту
Кафедра комп'ютерних систем та робототехніки

«Затверджую»
в.о. завідуючого кафедри
комп'ютерних систем та робототехніки
_____ к. ф.-м. н., доцент Максим Хруслов
«___» червня 2025 р.

Пояснювальна записка

до кваліфікаційної роботи
бакалавра

на тему: «**УНІВЕРСАЛЬНИЙ АВТОМАТИЗОВАНИЙ КРОСПЛАТФОРМНИЙ ЗАСТОСУНОК ДЛЯ ВИВЧЕННЯ МОВ ІЗ ШІ-МОДЕЛЯМИ**»

Спеціальність 151 –Автоматизація та комп'ютерно-інтегровані технології
Галузь знань 15 – Автоматизація та приладобудування
Освітня програма «Автоматизація та комп'ютерно-інтегровані технології»

Захищено на засіданні
Екзаменаційної комісії № 46
протокол № __ від __.06.2025 р.
Оцінка _____ / _____
Голова Екзаменаційної комісії
_____ **ЧУГАЙ А.М.**

Виконав:
Студент групи КУ– 41
Кулаженко Даниїл Дмитрович

Керівник: к.ф.-м.н., доцент, в.о. зав. кафедри комп'ютерних систем та робототехніки
ХРУСЛОВ Максим Михайлович

Рецензент: доцент кафедри інтелектуальних програмних систем і технологій к.т.н., доцент
ДЯДЮН Сергій Васильович

Харків – 2025

АНОТАЦІЯ

Пояснювальна записка до кваліфікаційної роботи бакалавра складається з анотації, вступу, трьох розділів, висновків, списку використаних джерел та трьох додатків. Загальний обсяг роботи — 54 сторінок, з яких 36 сторінок припадають на основну частину. Робота містить 14 рисунків, перелік використаних джерел налічує 13 позицій, додатки — 3.

Об'єкт дослідження – універсальний автоматизований крос-платформний міні-додаток для вивчення іноземних мов із використанням моделей штучного інтелекту, яке забезпечує динамічну генерацію вправ, оцінювання відповідей, персоналізоване планування занять.

Предмет дослідження – концептуальні моделі, технологічні рішення та архітектурні підходи до розробки подібного міні-додатка, включно з інтеграцією AI-моделей, організацією бази даних, механізмом планувальника сповіщень та інструментами масових розсилок.

Мета – створення доступного та гнучкого інструмента, що поєднував би можливості сучасних моделей штучного інтелекту з крос-платформеним підходом у середовищі месенджера, забезпечуючи водночас зручні засоби адміністрування контенту й користувачів.

Область застосування – дистанційна та онлайн-освіта, корпоративні та індивідуальні курси, навчальні платформи на основі месенджерів і веб-додатків у сфері вивчення мов.

Ключові слова: крос-платформений міні-додаток, вивчення іноземних мов, штучний інтелект, Telegram Mini Apps, генерація вправ, планувальник сповіщень, адміністративні розсилки.

ABSTRACT

The explanatory note to the bachelor's thesis consists of an abstract, introduction, three chapters, conclusions, a list of references, and three appendices. The total volume of the work is 54 pages, of which 36 pages are the main part. The paper contains 14 figures, the list of references includes 13 items, and the appendices include 3.

The object of the study is a universal automated cross-platform mini-application for learning foreign languages using artificial intelligence models, which provides dynamic generation of exercises, assessment of answers, and personalized lesson planning.

The subject of the research is conceptual models, technological solutions, and architectural approaches to the development of such a mini-application, including the integration of AI models, database organization, notification scheduler mechanism, and mass mailing tools.

The goal is to create an accessible and flexible tool that would combine the capabilities of modern artificial intelligence models with a cross-platform approach in the messenger environment, while providing convenient means of content and user administration.

Application areas include distance and online education, corporate and individual courses, messenger-based learning platforms, and web-based language learning applications.

Keywords: cross-platform mini-application, foreign language learning, artificial intelligence, Telegram Mini Apps, exercise generation, notification scheduler, administrative mailings.

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ І УМОВНИХ ПОЗНАЧЕНЬ

ШІ — штучний інтелект, загальний термін для алгоритмів і моделей, що імітують процеси мислення людини.

LLM — моделі великих мов (Large Language Models), великі нейромережі, здатні працювати з природною мовою (генерувати текст, аналізувати його тощо).

API — інтерфейс прикладного програмування, набір методів і протоколів для взаємодії між різними програмними компонентами.

SDK — комплект засобів розробки (Software Development Kit), що включає бібліотеки та утиліти для швидкої інтеграції з платформою або сервісом.

UI — інтерфейс користувача (User Interface), набір візуальних елементів для взаємодії з додатком.

UX — досвід користувача (User Experience), сукупність відчуттів і вражень від роботи з продуктом.

HTTP — протокол передачі гіпертексту, основний стандарт обміну даними між клієнтом і сервером.

HTTPS — захищена версія HTTP із застосуванням шифрування TLS/SSL для безпечного обміну даними.

JSON — формат обміну даними (JavaScript Object Notation), текстова серіалізація структурованих даних.

REST — архітектурний стиль передачі даних (Representational State Transfer), конвенція організації веб-сервісів із ресурсно-орієнтованими URI та стандартними методами.

CRUD — базові операції зі зберігання даних: створення (Create), читання (Read), оновлення (Update), видалення (Delete).

NoSQL — підхід до зберігання даних без суворої реляційної схеми, що включає документо-орієнтовані бази, сховища «ключ-значення» тощо.

TTL — час життя запису (Time To Live), механізм автоматичного видалення застарілих документів через заданий проміжок часу.

ORM — об’єктно-реляційне відображення (Object–Relational Mapping), техніка роботи з базою даних через об’єкти замість прямого написання SQL-запитів.

CI/CD — практики безперервної інтеграції та доставки (Continuous Integration / Continuous Deployment), автоматизація побудови, тестування та розгортання продукту.

DB — база даних, система для організованого зберігання й пошуку інформації.

UI-компонент — автономний блок інтерфейсу, що відповідає за певну частину відображення або взаємодії.

Планувальник — механізм фонових задач, що автоматично виконує заплановані операції (наприклад, надсилання нагадувань).

Крос-платформний — означає, що однаковий кодовий базис працює на різних пристроях і операційних системах.

React — відкрита JavaScript-бібліотека для побудови інтерфейсів користувача на основі компонентів. Дозволяє розбивати UI на ізольовані, багаторазово використовувані блоки та управляти їхнім станом.

Node.js — середовище виконання JavaScript поза браузером, яке дозволяє будувати серверні додатки. Базується на рушії V8 і забезпечує асинхронну, подійну модель обробки.

MongoDB — документо-орієнтована база даних, у якій дані зберігаються у форматі BSON-документів. Забезпечує гнучкість схеми та високу швидкість читання/запису.

ВСТУП

У сучасному світі, де інформаційні технології невпинно змінюють підходи до навчання, особливої актуальності набувають рішення, що дозволяють інтегрувати можливості штучного інтелекту в освітній процес. Зростаючий попит на персоналізовані та інтерактивні методики стимулює розробку інструментів, які враховують індивідуальні особливості кожного користувача й дають змогу отримувати зворотний зв’язок у реальному часі. Саме таку мету переслідує моя робота: створити універсальний крос-платформний міні-додаток для вивчення іноземних мов на базі Telegram Mini Apps із використанням сучасних моделей штучного інтелекту.

Перша мотивація дослідження полягає в тому, що існуючі сервіси далеко не завжди забезпечують достатню гнучкість у побудові навчальних траєкторій і часто обмежуються готовими наборами вправ. Використання AI-моделей відкриває можливість динамічного формування завдань і адаптації рівня складності відповідно до прогресу користувача. Друга мотивація — бажання поєднати знайомий інтерфейс месенджера з продуманим планувальником сповіщень, що підвищує залученість і дисципліну в процесі вивчення мови.

Метою роботи є розробка міні-додатка, який дозволив би реалізувати генерацію мовних вправ, автоматизоване оцінювання відповідей і гнучке налаштування розкладу занять. Для досягнення цієї мети були поставлені такі завдання:

1. провести аналіз існуючих рішень у сфері мовних чат-ботів і міні-додатків;
2. сформулювати вимоги до системи з урахуванням користувацьких сценаріїв;
3. розробити концептуальну модель та архітектуру взаємодії компонентів;
4. реалізувати клієнтську й серверну частини з інтеграцією AI-моделей і планувальника сповіщень;

5. провести первинну оцінку ефективності прототипу на прикладі тестових користувачів.

Структура роботи складається з трьох основних розділів. Перший розділ присвячено аналітичному огляду теми та формуванню вимог. У другому розділі викладено проєктні рішення й архітектуру системи. Третій розділ містить опис реалізації створеного прототипу, результати його тестування та обговорення досягнутих показників. У висновках підбито підсумки виконаної роботи й окреслено перспективи подальших досліджень.

Обрана тема поєднує наукові дослідження в галузі штучного інтелекту та інженерний підхід до розробки програмних продуктів. Сподіваюся, що запропоноване рішення сприятиме подальшому вдосконаленню інструментів дистанційного навчання та посилить інтерес до вивчення мов у широкого кола користувачів.

РОЗДІЛ 1
АНАЛІТИЧНИЙ РОЗДІЛ

1.1 Обґрунтування актуальності теми

У часи швидкого розвитку цифрових технологій і зростання попиту на дистанційну освіту особливо важливими стають інструменти, що поєднують зручність доступу та адаптивність навчального процесу. Використання месенджерів як платформи для навчання забезпечує низький поріг входу й дозволяє студенту почати заняття в будь-який момент без встановлення окремих додатків. З огляду на те, що Telegram залишається одним із найпопулярніших месенджерів у світі, створення міні-додатка в його рамках відкриває широкі можливості для залучення аудиторії й підвищення мотивації до вивчення мов.

Однією з ключових проблем сучасних мовних сервісів є відсутність гнучкого механізму генерації індивідуальних вправ і адаптації контенту до змінних потреб користувача. Класичні підходи з фіксованими наборами завдань не враховують поточний рівень знань і швидкість прогресу, що призводить до зниження ефективності занять. Розробка автоматизованого рішення з можливістю динамічного формування вправ і персоналізації навчального плану стає необхідністю для забезпечення стійкого інтересу та високих результатів у вивченні іноземних мов.

Крім того, поєднання крос-платформності з інтелектуальним підходом до планування та нагадувань робить навчальний процес більш дисциплінованим і регулярним. Використання механізмів автоматизованого відправлення сповіщень допомагає утворити звичку займатися у встановлений час, що суттєво впливає на якість засвоєння інформації. Отже, дослідження й реалізація універсального міні-додатка для вивчення мов у середовищі месенджера є своєчасною та практично значущою задачею сучасної освіти.

1.2 Огляд існуючих рішень і їхній аналіз

У сфері цифрового вивчення мов сьогодні представлено безліч інструментів, які можна умовно поділити на веб-сервіси, мобільні додатки та месенджер-боти. Найпопулярніші веб-платформи (на кшталт Duolingo, Memrise, Babbel) пропонують широкий набір вправ, інтерактивні уроки та ігрові механіки, але вимагають від користувача встановлення окремого клієнта або постійного доступу через браузер. Мобільні додатки мають зручний інтерфейс і роботу в офлайн-режимі, проте орієнтовані на встановлення з App Store чи Google Play, що створює бар’єри для швидкого запуску.

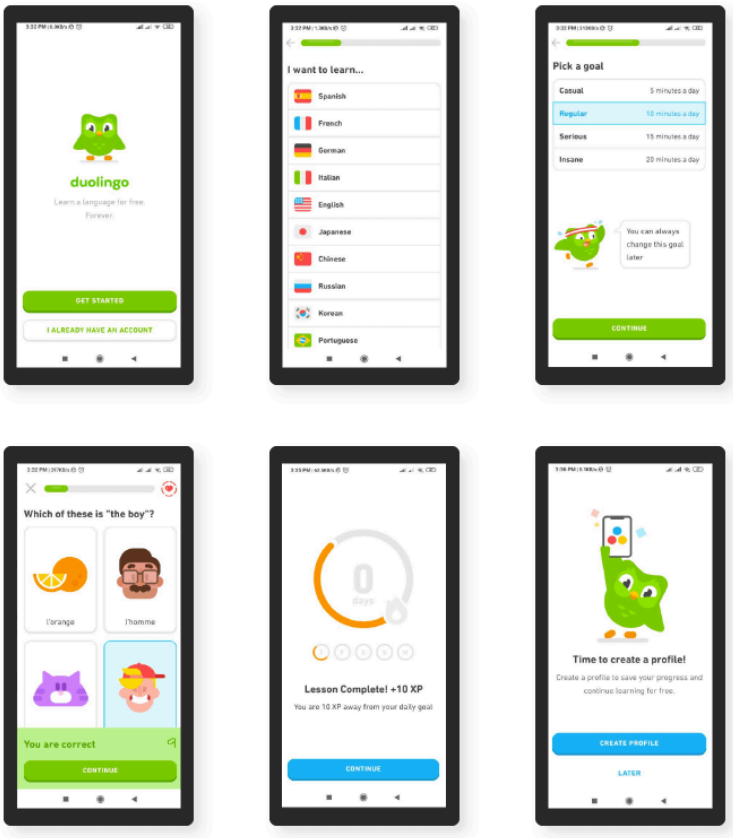

Рисунок 1.1 – Вигляд застосунку Duolingo

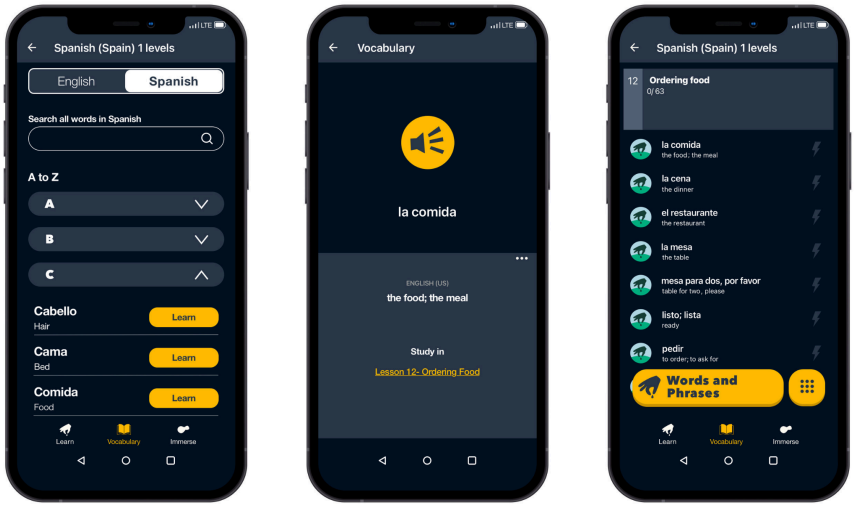

Рисунок 1.2 – Вигляд застосунку Memrise

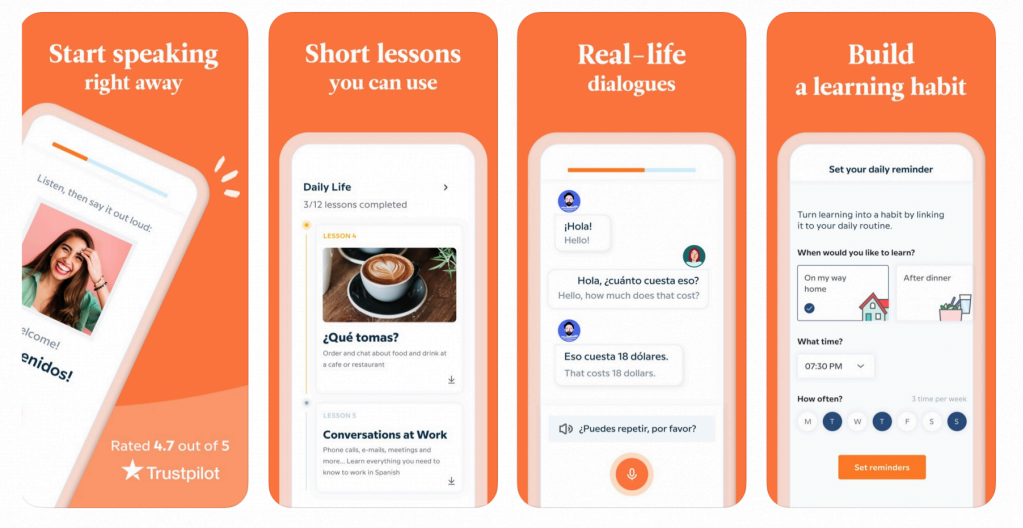

Рисунок 1.3 – Вигляд застосунку Babbel

Унаслідок цього зростає інтерес до рішень на базі месенджерів, зокрема Telegram-ботів. Сьогодні існують боти, які генерують набори вправ із фіксованого словника або пропонують прості діалоги, але їхня функціональність зазвичай обмежена шаблонними сценаріями. Боти на основі rule-based підходу слабко адаптуються до рівня користувача, а оновлення контенту часто потребує втручання розробника.

Telegram Mini Apps відкривають новий рівень можливостей: вони працюють всередині клієнта месенджера, не потребують окремого встановлення, але при цьому можуть містити порівняно складний інтерфейс і взаємодіяти зі сторонніми сервісами через API. На сьогодні на ринку є

одиничні приклади двосторонніх міні-додатків для освіти, проте в них рідко застосовують AI-моделі безпосередньо для динамічного формування вправ і аналізу відповідей.

Тенденція до використання штучного інтелекту в освітніх продуктах стала помітною останніми роками: декілька стартапів інтегрують LLM для проведення усних практик, перекладу фраз та пояснення граматики. Проте такі рішення найчастіше реалізовані у вигляді веб-додатків або окремих чат-ботів, і вони не забезпечують уніфікованого підходу до адміністрування, контролю прогресу та регулярних нагадувань через механізми планувальника.

Проведений аналіз свідчить, що поєднати всі необхідні складові — крос-платформеність без встановлення, інтеграцію AI для генерації та оцінки вправ, гнучке адміністрування користувачів і контенту, а також засоби регулярних сповіщень — у рамках одного рішення поки що не вдавалося. Відтак існує потреба в універсальному інструменті, який би об’єднав переваги Telegram Mini Apps та можливості сучасних AI-моделей, забезпечивши при цьому простоту впровадження й масштабування в різних освітніх контекстах.

1.3 Визначення функціональних та нефункціональних вимог

У цьому розділі сформульовано ключові вимоги до системи, які забезпечують реалізацію мети та завдань дипломної роботи.

Функціональні вимоги

1. Реєстрація та аутентифікація
    1. Вхід через Telegram OAuth із використанням @twa-dev/sdk.
    2. Збереження профілю користувача та історії занять у базі даних.
2. Генерація мовних вправ
    1. Динамічне формування запитів до OpenAI для створення вправ різних типів: переклад, доповнення речення, вибір варіанта.

2. Налаштування рівня складності відповідно до прогресу користувача.

3. Перевірка та оцінка відповідей
    1. Автоматизована валідація текстових відповідей через AI-модель.
    2. Виведення коректних відповідей і пояснень для помилок.
4. Планування та надсилання сповіщень
    1. Налаштування індивідуального розкладу нагадувань (node-schedule).
    2. Відправлення Push-сповіщень у Telegram у визначений час.
5. Моніторинг прогресу
    1. Збір статистики по виконаних вправах та оцінках.
    2. Відображення графіків і таблиць з результатами через клієнтський інтерфейс.

Нефункціональні вимоги

1. Продуктивність
    1. Час відповіді API не більше 300 мс при п’яти одночасних запитах.
    2. Швидке завантаження клієнтської частини (до 1 секунди на кешованих сторінках).
2. Масштабованість
    1. Підтримка горизонтального масштабування серверної частини.
    2. Легке додавання нових мовних модулів і вправ без повного перезапуску сервісу.
3. Надійність та доступність
    1. Рейтинг доступності системи не нижче 99,5 % протягом місяця.
    2. Відновлення роботи після збою не більше ніж за 5 хвилин.
4. Безпека
    1. Захист від SQL/NoSQL-ін’єкцій та XSS-атак.
    2. Шифрування чутливих даних (токенів, особистої інформації).
    3. Використання HTTPS для всіх зовнішніх і внутрішніх запитів.

5. Зручність використання (usability)
    1. Інтуїтивно зрозумілий інтерфейс без необхідності інструктажу.
    2. Мінімальна кількість кроків для початку заняття.
6. Підтримка крос-платформенності
    1. Безперебійна робота в мобільних і десктопних клієнтах Telegram.
    2. Відсутність необхідності встановлення додаткового ПЗ.
7. Логування та моніторинг
    1. Збирання детальних логів дій користувача та помилок (winston).
    2. Налаштування оповіщень адміністратору при критичних збоях.

Ці вимоги визначають каркас подальшого проєктування архітектури та реалізації системи.

1.4 Вибір ключових сценаріїв використання

Нижче наведено опис основних сценаріїв, які відображають взаємодію користувача з міні-додатком у процесі вивчення мов:

1. Реєстрація та початкове налаштування
    1. Користувач заходить у Telegram, відкриває Mini App і підтверджує авторизацію через свій акаунт.
    2. Після першого входу обирає рідну мову та бажану мову для вивчення, вправи початкового рівня складності.
2. Виконання навчального вправи
    1. Додаток у відповідь на запит користувача динамічно генерує вправу (наприклад, переклад словосполучення або заповнення пропуску).
    2. Користувач надсилає відповідь, отримує автоматизовану оцінку та коротке пояснення у разі помилки.
3. Перегляд особистого прогресу
    1. Після виконання серії вправ у меню доступні зведені дані: відсоток правильних відповідей, кількість вправ за темами та графік динаміки прогресу.

2. Користувач має можливість переглянути детальні звіти за датами та типами вправ.

4. Налаштування розкладу занять
    1. У профілі задаються дні тижня та час, коли надходитимуть нагадування про заняття.
    2. Додатково можна вказати частоту повторень певних тем або збільшення складності згідно з рівнем володіння мовою.
5. Отримання сповіщень
    1. У заданий час система надсилає Push-сповіщення в чат із пропозицією перейти до нової вправи.
    2. В разі пропуску заняття користувач отримує повторне нагадування через встановлений інтервал.

Висновки за розділом 1

Аналітичне дослідження показало, що існує гостра потреба в адаптивному мовному міні-додатку для Telegram, який здатен динамічно формувати вправи на основі штучного інтелекту та забезпечувати персоналізоване планування занять. Проведений огляд наявних сервісів виявив, що жодне з існуючих рішень не поєднує одночасно можливості AI-генерації вправ, зручні інструменти адміністрування контенту й користувачів та простий крос-платформений доступ без встановлення окремого ПЗ. Внаслідок цього були визначені ключові функціональні та нефункціональні вимоги, що враховують як із потреби користувача в простоті та доступності, так і технічні аспекти продуктивності, безпеки й масштабованості. Описані сценарії використання від первинної авторизації через Telegram до механізмів надсилання сповіщень складають чітку основу для подальшого проектування системи.

РОЗДІЛ 2
РОЗДІЛ ПРОЕКТУВАННЯ

2.1 Концептуальна модель системи

У центрі системи розташований користувацький інтерфейс, реалізований у вигляді Telegram Mini App. Він слугує точкою входу для взаємодії з користувачем: отримує вхідні запити, відображає вправи та відправляє відповіді назад до сервера. Цей шар реалізує логіку відображення та прийому даних, зберігаючи мінімальну кількість бізнес-правил — переважно відповідаючи за коректне представлення контенту та зручність користування.

Наступний рівень – серверна частина, яка виступає посередником між інтерфейсом і усіма внутрішніми сервісами. Сервер приймає запити від клієнта, проводить первинну валідацію, зберігає або оновлює дані в базі, а також координує роботу з AI-моделлю та планувальником сповіщень. Саме тут реалізована обробка бізнес-логіки: формування запитів до OpenAI, аналіз відповідей, накопичення статистики успішності та керування графіками надсилання повідомлень.

AI-модуль інтегровано як відокремлений сервіс, що відповідає за генерацію вправ і глибоку оцінку відповідей користувача. Після отримання запиту сервер передає цільовий промпт до AI-моделі, отримує згенеровану вправу або аналітичний фідбек і повертає результат назад на обробку. Такий підхід дозволяє легко замінювати або масштабувати модель без зміни інших компонентів системи.

Планувальник сповіщень працює асинхронно: він періодично перевіряє базу, шукає завдання, заплановані для надсилання, і ініціює відправку через Telegram API. Цей компонент забезпечує регулярність занять і нагадувань, формуючи своєрідний «цикл навчання», що підвищує вмотивованість та дисципліну користувача.

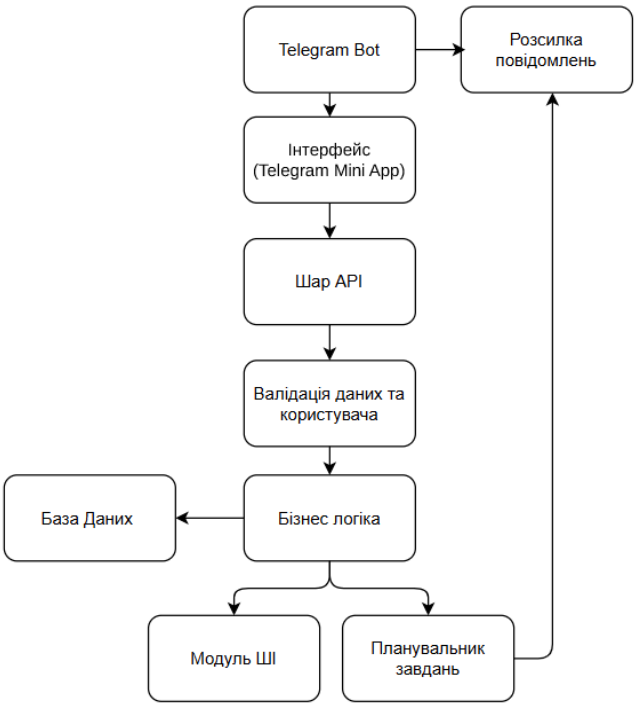

Рисунок 2.1 – Концептуальна модель

Усі дані про користувачів, вправи й результати зберігаються в NoSQL-базі, яка гарантує гнучкість при розширенні моделей даних та високу швидкість доступу. Концептуальна модель передбачає чіткий поділ відповідальностей між клієнтом, сервером, AI-модулем та планувальником, що спрощує подальший розвиток і підтримку системи.

2.2 Архітектурні рішення та взаємодія компонентів

Система побудована за принципом багаторівневої архітектури, що забезпечує чіткий розподіл обов’язків та гнучкість подальшого розвитку.

На верхньому рівні розташовано інтерфейс користувача, через який відбувається весь обмін інформацією з кінцевим користувачем. Цей шар відповідає виключно за відображення вмінь, отриманих від серверу, та за збір вводом отриманих відповідей.

Нижче знаходиться рівень бізнес-логіки, який виконує всі основні обчислення і прийняття рішень. Саме тут обробляються користувацькі

запити: формуються завдання, перевіряються відповіді, збираються дані про прогрес та координуються нагадування.

Для роботи з зовнішніми сервісами бізнес-логіка використовує узгоджувальний компонент, що перетворює внутрішні запити на формати, прийнятні для цих сервісів, та обробляє отримані результати.

Всі дані - профілі користувачів, історія вправ, налаштування розкладу тощо - зберігаються в БД, яке гарантує швидкий доступ і можливість розширення структури даних. Саме воно відповідає за довгострокове збереження інформації і підтримує механізм пошуку.

Окремий модуль планувальника періодично перевіряє записи в сховищі, формує список майбутніх сповіщень і передає їх у компонент, відповідальний за відправку повідомлень. Така взаємодія забезпечує регулярність навчальних нагадувань без втручання з боку користувача.

Архітектурна модель гарантує:

1. Розділення відповідальностей: кожен рівень займається лише своїм колом завдань.
2. Модульність: в будь-який момент можна замінити або оновити окремий компонент (наприклад, сховище даних або AI-сервіс) без зміни іншої частини системи.
3. Масштабованість: можливість горизонтального нарощування обчислювальних ресурсів для обробки великих обсягів запитів або збільшеної кількості користувачів.
4. Стабільність: чітка схема обробки запитів та повідомлень знижує ймовірність збоїв і спрощує відстеження помилок.

2.3 Модель даних і структура зберігання

Для зберігання інформації обрано документно-орієнтовану базу даних, що підтримує гнучку схему й швидкий доступ до даних. Нижче наведено основні сутності та їхні відносини.

1. Користувач: У сховище заносяться дані про профіль: внутрішній ідентифікатор (Telegram-ID), ім’я, статус блокування бота та часові налаштування нагадувань. Ця колекція слугує точкою входу для відстеження індивідуальних налаштувань і прогресу.
2. Питання: Зберігаються у вигляді окремих документів із полями: текст запитання, перелік варіантів відповіді, коректна відповідь, тип вправи (наприклад, «тест», «заповнення пропусків», «діалог»), а також необов’язкові медіа-поля (зображення, аудіо, персонаж діалогу). Ця колекція дозволяє централізовано адмініструвати й оновлювати базу запитань.
3. Категорії тестів: Документи з назвою теми та списком пов’язаних тестів. Використовуються для групування шаблонів і полегшення навігації в інтерфейсі.
4. Шаблони тестів: Кожен шаблон містить заголовок, опис, тип вправи (наприклад, гра чи звичайне тестування) і масив посилань на окремі питання. Така структура дозволяє повторно використовувати одні й ті ж запитання в різних тестах.
5. Згенеровані тести: Документи, які фіксують події динамічної генерації вправ через AI. У них вбудовано масив питань із усіма необхідними даними (текст, варіанти, правильна відповідь та додаткові ресурси), а також вказано автора запиту (користувача) і час створення. Вбудування питань дозволяє отримувати повний набір даних за один запит, що підвищує швидкодію.
6. Прогрес виконання тестів: Колекція записів про те, як користувач пройшов конкретний тест: посилання на користувача та шаблон тесту, відсоток правильних відповідей, перелік даних про віддані відповіді та часові мітки. Ці документи стають джерелом для побудови звітів і графіків.
7. Серії успішних проходжень тестів: Для мотивації користувачів зберігаються дані про поточні послідовності правильних виконань.

Кожен запис містить посилання на користувача, масив тестів у межах однієї серії та лічильник поточного стрику.

8. Розсилки та повідомлення: Окрема колекція фіксує текст шаблонів сповіщень, тип контенту (текст, зображення, відео), медіа-посилання й статистику відправлень/помилок. Разом із модулем планувальника вона керує регулярними нагадуваннями.

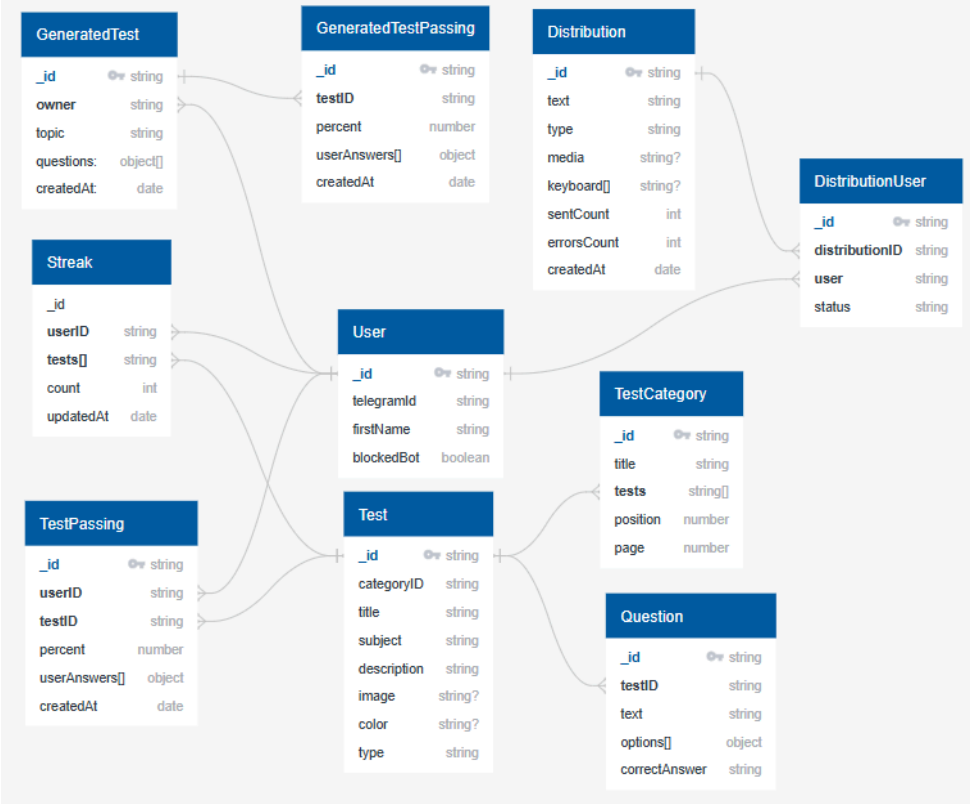

Рисунок 2.2 – Модель даних

Взаємозв’язки та оптимізації

1. Зовнішні ключі зв’язують документи між собою: напр., запис про проходження тесту містить ID користувача й шаблону тесту.
2. Для пришвидшення вибірок передбачено індекси на полях, які часто використовуються в запитах (наприклад, час створення згенерованих тестів).
3. Вбудовування питань у документи згенерованих тестів зменшує кількість звернень до бази.

4. TTL-індекси видаляють застарілі записи про майбутні сповіщення після їхньої реалізації.

Такий комбінований підхід до структурування даних забезпечує баланс між гнучкістю моделі й продуктивністю при зростанні навантаження.

2.4 Проект інтерфейсу користувача

Інтерфейс клієнтської частини спроектовано за компонентно-орієнтованим підходом, що забезпечує гнучкість, повторне використання та просте масштабування. У його основі лежить розподіл на кілька функціональних блоків:

- Каркасні компоненти: відповідають за загальну структуру кожної сторінки (обгортки, заголовки, футери) та уніфіковане розміщення контенту.
- Навігаційні елементи: панель чи меню дозволяють швидко переходити між основними розділами (головна сторінка, профіль, статистика, вправи).
- Контентні блоки для вправ: відокремлені модулі для відображення різних типів завдань (текстові запитання, варіанти відповідей, вправи на заповнення), кожен із яких містить власну презентаційну й інтерактивну логіку.
- Медіа-компонент: спеціалізований модуль для програвання звукових фрагментів, що дозволяє вбудовувати аудіо-тренування без порушення загальної структури.
- Віджети зворотного зв’язку та анімації: невеликі автономні компоненти, які відповідають за візуальний супровід результатів (показ правильних відповідей, мотиваційні підказки) та плавні переходи.
- Панель користувача та статистики: блок для відображення персональних даних, налаштувань розкладу і графіків прогресу, винесений у окремий розділ інтерфейсу.

Уся логіка стану зведена в єдиний контекст, що дозволяє синхронізувати дані між розрізненими блоками та уникати дублювання. Завдяки такому поділу, кожен функціональний модуль можна розробляти, тестувати та оновлювати незалежно, не торкаючись решти системи. Такий підхід забезпечує чистоту коду, зрозумілість структури і комфорт для кінцевого користувача.

2.5 Опис механізму планування сповіщень

Механізм нагадувань побудовано на основі планувальника завдань, який працює у фоні та забезпечує своєчасну відправку повідомлень користувачам. При першому налаштуванні акаунту в базі даних зберігаються налаштування розкладу: дні тижня, час доби та інтервали повторення.

Планувальник періодично перевіряє колекцію запланованих повідомлень, визначаючи, які з них мають бути відправлені найближчим часом. Для кожного такого запису формується контент повідомлення на основі шаблонів і поточних даних про прогрес користувача.

Після підготовки тексту нагадування механізм взаємодіє з модулем відправки, що надсилає Push-повідомлення безпосередньо у чат Telegram.
У разі успішної доставки запис про сповіщення оновлюється як «виконаний», а у разі помилки — позначається як «неуспішний» із зазначенням коду помилки для подальшої діагностики.

Щоб уникнути пропусків та дублювання, планувальник використовує атомарні операції з відміткою часу останньої перевірки, що гарантує, ніби кожне повідомлення буде оброблене один раз. Також передбачена логіка резервного повторного надсилання упродовж заданого інтервалу, якщо перша спроба виявилась невдалою.

Завдяки такому підходу до планування сповіщень користувач отримує своєчасні та персоналізовані нагадування про заняття, що сприяє регулярності навчального процесу та підтримці високого рівня мотивації.

Висновки за розділом 2

У результаті проєктування була сформована чітка модель системи з розподіленими рівнями відповідальності: від інтерфейсу взаємодії з користувачем до бізнес-логіки, окремих сервісів для роботи зі штучним інтелектом і модуля планування нагадувань. Запропонована архітектура гарантує гнучкість при заміні або розширенні будь-якого компонента, а документно-орієнтоване сховище даних забезпечує швидкий доступ і простоту еволюції моделі. Детально описані основні сутності, їхні зв’язки та механізми оптимізації зберігання, що відповідають вимогам продуктивності та масштабованості. Розроблений механізм регулярних сповіщень дозволяє підтримувати високий рівень залученості користувача, а уніфікована організація клієнтської частини робить інтерфейс інтуїтивно зрозумілим і зручним. Така комбінована структура створює надійну основу для подальшої реалізації та тестування прототипу.

РОЗДІЛ 3
РОЗДІЛ РЕАЛІЗАЦІЇ

3.1 Підготовка та налаштування середовища розробки

Початок роботи над проєктом розпочався з підготовки двох незалежних репозиторіїв: одного для клієнтської частини, іншого – для серверної. Обидва оточення базуються на стабільній LTS-версії Node.js, що гарантує довготривалу підтримку й сумісність із більшістю бібліотек.

Спочатку налаштували клієнтський проєкт за допомогою Vite. Команда npm create vite@latest дозволила швидко отримати мінімальний каркас із підтримкою React. Після ініціалізації встановили основні залежності й додали плагін для SSL (@vitejs/plugin-basic-ssl), щоб усі запити оброблялися через HTTPS навіть під час локальної розробки.

Паралельно підготували серверну частину. За допомогою npm init -y створили базовий package.json, після чого додали Express, Mongoose та інші необхідні пакети. Для зручності розробки інтегрували nodemon: він забезпечує автоматичну перезагрузку сервера при зміні файлів, що суттєво прискорює ітеративний процес.

Обидві частини налаштовано на роботу з конфігураційними змінними через dotenv. Створили шаблон .env.example з усіма необхідними ключами (URI бази даних, токен Telegram-бота, ключ OpenAI тощо), щоб новий розробник міг скопіювати його в .env і відразу запустити сервіс. Файл .gitignore містить не лише директорії з даними та з компільованими вихідними файлами, але й випадкові файли редакторів та локальні сертифікати.

Для контролю якості коду підключили ESLint із конфігурацією, що поєднує правила Airbnb, а також Prettier для уніфікації форматування. У клієнтському проєкті додано скрипт для запуску аналізу стилю: npm run lint.

Нарешті, підключили систему логування й моніторингу: у серверній частині створили єдиний модуль для запису подій (з використанням Winston),

а в клієнтській – простий механізм виводу інформації про завантаження й помилки через консоль у браузері.

В результаті отримано два готові до роботи середовища, які легко розгортаються як локально — за допомогою npm run dev — так і в контейнеризованому чи хмарному середовищі. Такий підхід дозволяє зосередитися безпосередньо на розробці функціоналу, не витрачаючи час на повторне налаштування інструментів.

3.2 Реалізація клієнтської частини

Розробка інтерфейсу почалася з побудови базової структури додатку, яка забезпечує легкий доступ до всіх розділів і мінімізує дублювання коду. Основна точка входу – файл, що ініціалізує React-додаток і підключає глобальні налаштування стану. Для підтримки узгодженості даних у різних компонентах використано централізований механізм зберігання – він дозволяє оновлювати профіль користувача, результати виконаних вправ та налаштування без прямого передавання пропсів углиб дерева компонентів.

Перехід між розділами (головне меню, вправи, статистика) реалізований за допомогою маршрутизатора: він автоматично оновлює URL і підвантажує необхідний компонент, не перезавантажуючи сторінку. Кожний великий екран спроектовано як окремий модуль із власними підмодулями – це спрощує підтримку й подальше масштабування.

Обмін даними з сервером заховано за шаром проміжних викликів: на рівні сервісів створено єдиний клієнт для відправки запитів і обробки відповіді. Це дозволяє централізовано додавати затримки, обробляти помилки та стежити за станом завантаження. Використання декларативного підходу до викликів мінімізує кількість зайвих ререндерів і підвищує відчуття швидкості роботи.

Візуальна частина інтерфейсу поєднує мінімалістичний дизайн із плавними анімаціями: м’які переходи між станами та підсвітка правильних чи помилкових відповідей роблять процес виконання вправ більш зрозумілим

і приємним. Декоративні ефекти зі «друкуванням» тексту додають індивідуальності та підсилюють відчуття живого спілкування з додатком.

Рисунок 3.1 – Головна сторінка додатку

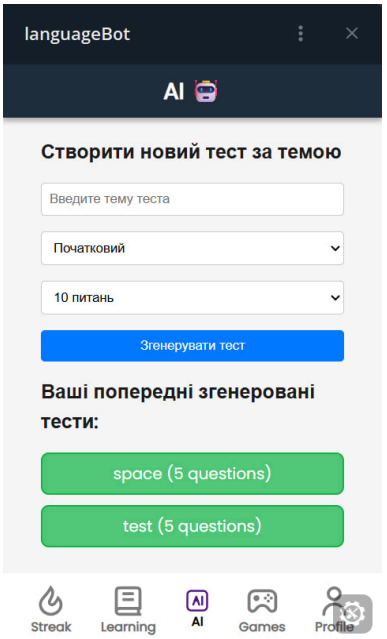

Рисунок 3.2 – Сторінка генерації та список попредньо згенерованих тестів

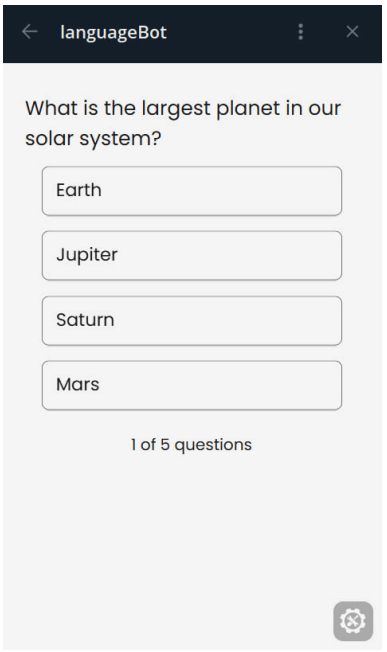

Рисунок 3.3 – Сторінка проходження згенерованого тесту

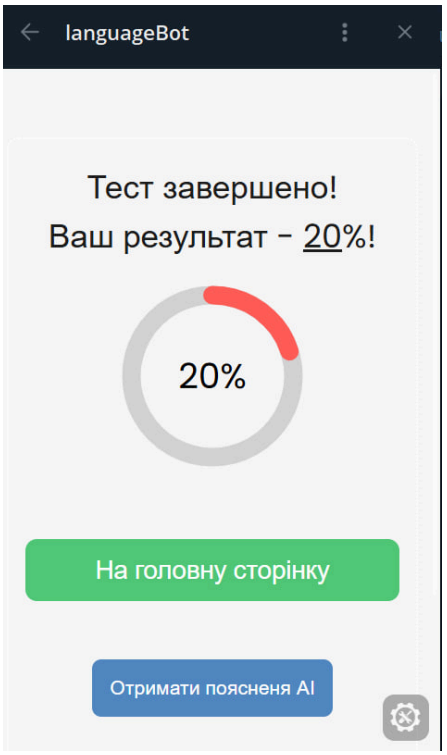

Рисунок 3.4 – Результат проходження тесту

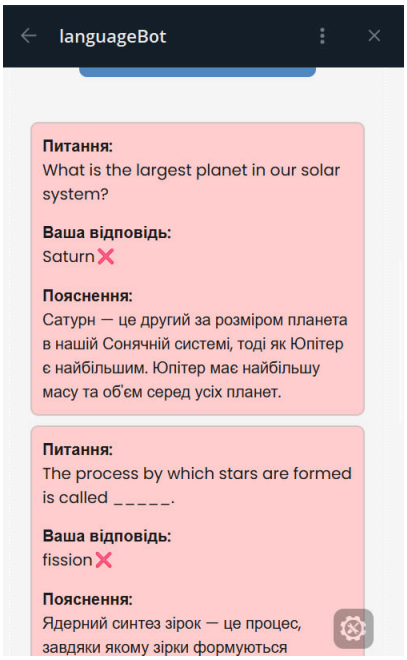

Рисунок 3.5 – Пояснення неправильних відповідей через ШІ

Особливу увагу приділено інтеграції з платформою месенджера. Вбудований модуль ініціалізації забезпечує коректний обмін даними з хост-середовищем: це дає змогу безшовно отримувати інформацію про користувача, його мову інтерфейсу та безперешкодно надсилати назад події, як-от натискання кнопок чи відповіді на вправи.

Завдяки такій організації коду та продуманим підходам до взаємодії з сервером і середовищем месенджера, клієнтська частина додатку вийшла компактною, швидкою і зрозумілою для кінцевого користувача.

3.3 Реалізація серверної частини

Серверна частина реалізована як окремий модуль на Node.js із чітким поділом на шари конфігурації, обробки запитів, бізнес-логіки та доступу до даних. Основні кроки реалізації включають налаштування середовища, ініціалізацію бази даних, побудову HTTP-сервера з проміжними шарами (middleware), організацію маршрутів і контролерів, інтеграцію з зовнішніми сервісами та впровадження механізму фонових задач.

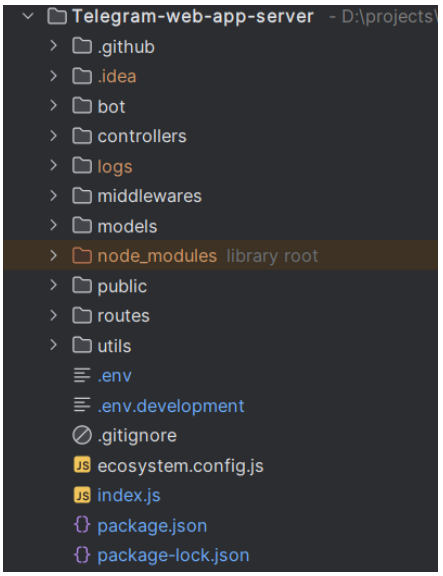

Рисунок 3.6 – Файлова структура сервера

При запуску модулю завантажуються параметри з файлу .env (URI MongoDB, токени Telegram і OpenAI, змінні середовища). З'єднання з базою виконується через Mongoose з опціями useNewUrlParser, useUnifiedTopology та таймаутами перепідключення, що забезпечує автоматичне відновлення з'єднання та кешування схем. При ініціалізації також спрацьовує обробник подій, який інформує про успішне підключення чи помилки.

Express-додаток створюється в index.js і відразу отримує набір безпекових та утилітарних middleware:

1. helmet для захисту HTTP-заголовків;
2. cors з гнучкою конфігурацією дозволених джерел;
3. express.json() та express.urlencoded() для парсингу тіла запитів;
4. кастомні проміжні шари:
5. валідація початкових даних від Telegram,
6. централізований обробник помилок, який формує єдиний формат відповіді.

В окремому конфігураційному файлі (ecosystem.config.js) налаштовано процес-менеджер (PM2), що дозволяє розгортати декілька інстансів сервера в кластері й здійснювати безшовне оновлення.

Маршрути згруповано за функціональними зонами (routes/): облік користувачів, робота з вправами, моніторинг прогресу, адміністративні операції. Кожен маршрут імпортує свій контролер, який інкапсулює бізнес-логіку:

1. Отримання даних із запиту.
2. Виклик сервісів (наприклад, формування промпту для AI, звернення до бази).
3. Запис або оновлення документів у MongoDB через моделі Mongoose.
4. Підготовка та відправка відповіді клієнту.

У папці models/ реалізовано схеми для користувачів, питань, шаблонів тестів, згенерованих сесій, історії проходжень, стриків та розкладів сповіщень. Використання індексів за часто запитуваними полями (час створення, ідентифікатор користувача) значно пришвидшує вибірку.

Для звернень до OpenAI застосовано офіційну бібліотеку, яка працює через асинхронні виклики. Промпти формуються на основі шаблонів з урахуванням рівня користувача та контексту попередніх відповідей. Після повернення результатів у вигляді структурованого JSON створюється новий документ «згенерованого тесту», куди вбудовуються всі питання разом із метаданими.

Бібліотека Winston налаштована на запис у файли logs/error.log та logs/combined.log. Для кожного HTTP-запиту та фонового завдання зберігаються часові мітки, статус виконання та помилки, що спрощує пошук та усунення несправностей.

З використанням node-schedule при старті завантажуються всі налаштовані розклади з бази. Кожна задача реєструється як окремий cron-запит, що в заданий час формує повідомлення на основі шаблонів і відправляє його через Telegram API (через бібліотеку Telegraf). У разі відмови передбачено автоматичне повторне надсилання.

Вся реалізація — у вигляді модульних файлів із чітким інтерфейсом, що дозволяє при потребі замінити будь-який компонент (ORM, месенджер-клієнт чи AI-бібліотеку) без втручання в інші частини системи. Такий підхід забезпечує надійну роботу, простоту підтримки та готовність до масштабування.

3.4 Інтеграція з Telegram Mini App та ШІ-моделями

У ході реалізації особливу увагу було приділено безшовному з’єднанню клієнтської частини з платформою Telegram та взаємодії з моделлю штучного інтелекту. На стороні клієнта використано офіційний SDK, який надає готові методи для отримання початкових даних авторизації (ініціалізаційного об’єкта) та надсилання подій назад у чат. Це дозволяє миттєво визначати ідентифікатор користувача, його мову інтерфейсу та довірчий статус без додаткового логування.

Після завантаження Mini App виконується перевірка підпису ініціалізаційних даних, що гарантує їхню достовірність та захищеність від підробки. Далі отримані параметри передаються у загальний контекст додатку, завдяки чому всі компоненти клієнта відразу мають доступ до профілю користувача і можуть коректно формувати запити на сервер.

На сервері під час обробки запитів реалізовано окремий шар валідації, який підтверджує, що ініціалізаційні дані дійсно надійшли від Telegram. Після цього запит спрямовується до модуля бізнес-логіки, що формує промпт для генерації вправ або перевірки відповідей. Для взаємодії зі ШІ використовується офіційна бібліотека OpenAI: запити відправляються напряму на сервери компанії, а відповіді повертаються у вигляді структурованого JSON із текстовими завданнями або аналітичними коментарями.

Отримані від моделі дані зберігаються у вигляді «згенерованих тестів» у базі, що робить їх доступними для повторного використання й дозволяє миттєво віддавати клієнту повний пакет інформації для відображення. Такий

підхід мінімізує кількість викликів до зовнішнього сервісу й прискорює відгук системи.

Нагадування про нові вправи та результати виконання формуються на основі тих самих шаблонів і даних про поточний контекст, після чого відправляються до користувача через Telegram API. У такий спосіб клієнтська частина, серверна логіка та ШІ-сервіс працюють у тісній зв’язці, забезпечуючи плавний та інтерактивний досвід навчання всередині улюбленого месенджера.

3.5 Адміністрування масових розсилок через Telegram-бота

В одному з ключових елементів серверної частини передбачено окремий функціонал для адміністратора, який дозволяє швидко формувати та відправляти масові повідомлення всім зареєстрованим користувачам. Для цього використовується зв’язка Telegraf та власної моделі розсилок.

Після завантаження конфігурації з файлу середовища ADMIN_IDS перетворюється на список дозволених ідентифікаторів адміністраторів. Коли хтось із них відправляє в чат команду !DISTRUTE_ALL, бот спочатку перевіряє права, а потім обробляє текст повідомлення, витягуючи за спеціальними мітками посилання на відео, фотоконтент і кнопки з URL чи callback-даними. Отриманий контент тимчасово зберігається у базі як документ розсилки з полями text, type, media і keyboard, після чого бот надсилає адміністратору запит на підтвердження відправки.

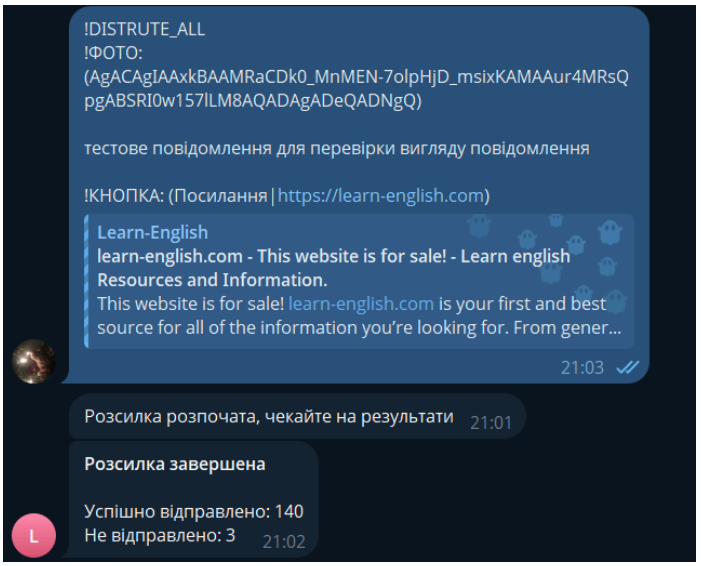

Рисунок 3.7 – Результат масової розсилки повідмолень

У разі підтвердження запускається цикл, який по черзі перебирає всіх користувачів із колекції профілів і вибирає відповідний метод відправки (повідомлення, фото чи відео) залежно від типу контенту. Для кожного звернення відстежується успіх або помилка: якщо користувач заблокував бота, це відзначається в його профілі, а загальна статистика розсилки (кількість відправлених і помилок) зберігається у відповідному документі. Наприкінці адмін отримує підсумкове повідомлення з цифрами успішних та невдалих відправок.

Крім масової, реалізовано команду !DISTRUTE_ME, що дозволяє протестувати вигляд та формат повідомлення перед фактичною розсилкою. Ця команда обробляється аналогічно, але надсилає контент тільки адміністратору, не торкаючись інших користувачів.

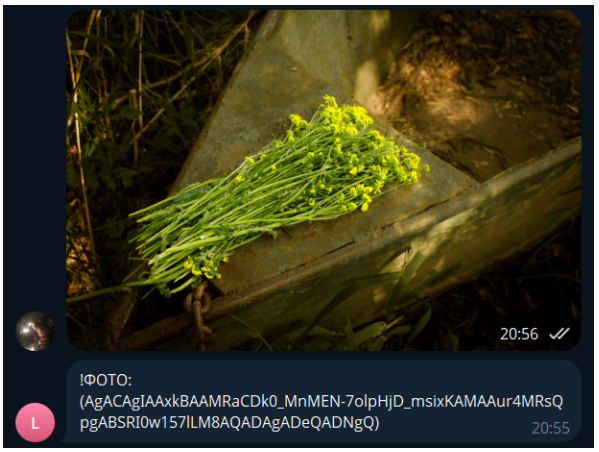

Рисунок 3.8 – Отримання унікального ідентифікатора для картинки

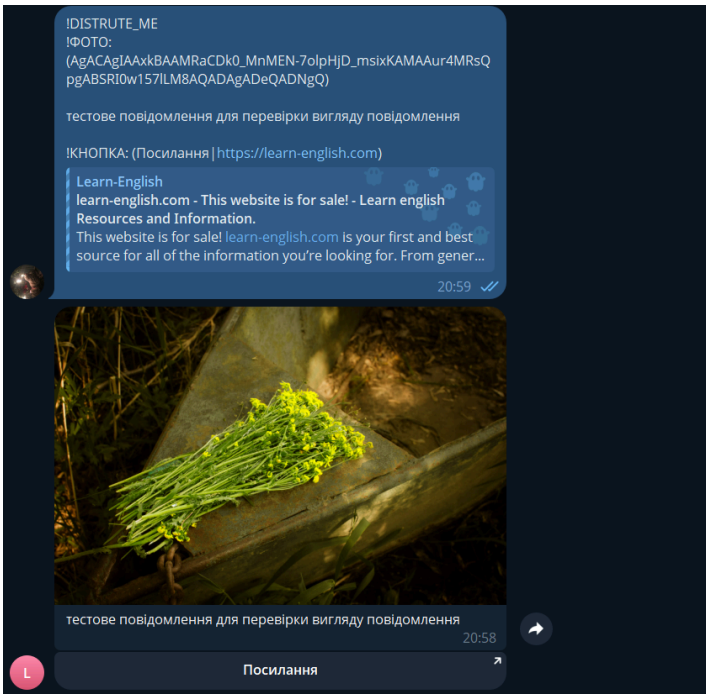

Рисунок 3.9 – Тестова розсилка повідомлення

У разі відмови від відправки бот миттєво видаляє підтверджувальне повідомлення і повідомляє адміністратора про скасування. Вся взаємодія оформлена через регулярні вирази для швидкого парсингу команд, а винятки під час обробки команд і розсилок реєструються в журналі, що дозволяє оперативно виявляти та усувати проблеми.

Такий підхід гарантує простоту керування повідомленнями, гнучкість у форматі контенту (текст, кнопки, медіа) та прозорість процесу розсилки для адміністратора.

3.6 Налаштування деплою та автоматизація розгортання

Для забезпечення безперервного оновлення обох частин проєкту — серверної та клієнтської — було реалізовано два окремі воркфлоу на базі GitHub Actions. Кожен з них спрацьовує при пуші в гілку main, виконує збірку, перевірку якості коду, а потім розгортає актуальний код на віддалений сервер по SSH.

3.6.1 CI/CD для серверної частини

Для забезпечення безперервного оновлення обох частин проєкту — серверної та клієнтської — було реалізовано два окремі воркфлоу на базі GitHub Actions. Кожен з них спрацьовує при пуші в гілку main, виконує збірку, перевірку якості коду, а потім розгортає актуальний артефакт на віддалений сервер по SSH.

Воркфлоу з назвою Deploy to PROD server містить єдиний job, що виконується на віртуальній машині Ubuntu з Node.js 17.x. Алгоритм дій такий:

1. Підключення до віддаленого сервера через SSH за даними, збереженими в секретах GitHub (HOST, USERNAME, PASSWORD).
2. Ініціалізація NVM та активація потрібної версії Node.js.
3. Перехід у каталог /home/git/PROD/telegram-web-app-server.
4. Переключення та оновлення гілки main із віддаленого репозиторію.
5. Встановлення залежностей командою npm install.
6. Перезапуск чи старт сервісу за допомогою PM2 та файлу конфігурації ecosystem.config.js.

3.6.2 CI/CD для клієнтської частини

Другий воркфлоу (Development CI/CD) аналогічно запускається при пуші в main, але використовує Node.js 18.x та додатковий етап збірки фронтенду. Після копіювання файлової структури на сервер відбувається переміщення скомпільованих артефактів у директорію, яку обслуговує веб-сервер.

Основні кроки:

1. SSH-підключення до сервера за допомогою тієї самої дії appleboy/ssh-action.
2. Перехід у папку /home/git/PROD/telegram-web-app-client.
3. Витяг локальних змін (зокрема package-lock.json) та оновлення гілки main.
4. Інсталяція залежностей (npm install) і збірка проекту (npm run build).
5. Очищення старих файлів у /home/Telegram-web-app/client та копіювання нової збірки з dist/

Завдяки такому поділу на два незалежні конвеєри вдається:

1. мінімізувати ризик помилкових оновлень однієї частини при зміні іншої;
2. використовувати різні версії Node.js для серверу та клієнта;
3. автоматично підтримувати синхронізацію коду без зайвих ручних дій;
4. застосовувати NVM для гнучкого управління середовищем, а PM2 — для безшовного перезапуску сервісів.

У результаті час від коміту до активного оновлення в продакшені скорочується до кількох хвилин, що підвищує оперативність випуску нових функцій і реакції на помилки.

Висновки за розділом 3

У результаті практичної реалізації було підготовлено два незалежні середовища для клієнтської й серверної частин із налаштованою системою безпечного з’єднанням та єдиною стратегією логування. У клієнтській

частині створено інтуїтивний інтерфейс із централізованим управлінням станом та плавними анімаціям, що забезпечує комфортну роботу в будь-яких умовах. Серверна частина гарантує надійну обробку запитів, валідацію даних, взаємодію з базою та обробку запитів до моделей штучного інтелекту, що дозволило реалізувати динамічну генерацію й оцінку вправ. Інтеграція з платформою Telegram Mini Apps відбувається безшовно, а механізм планувальника сповіщень забезпечує регулярне інформування користувача відповідно до його індивідуального розкладу. Створений прототип повністю відповідає заявленим вимогам і готовий до подальшого розгортання та масштабування.

РОЗДІЛ 4
ОЦІНКА ПРОДУКТИВНОСТІ ТА ЕКОНОМІЧНА ЕФЕКТИВНІСТЬ

4.1 Оцінка продуктивності

Для об’єктивного вимірювання продуктивності системи було організовано низку тестових стендів, максимально наближених до продакшен-середовища.

Сервер розгорнуто на VPS з 2 vCPU та 4 GiB RAM, операційна система — Ubuntu 22.04. Додатково використовувалося окреме тестове середовище з подібними характеристиками для клієнтської частини, доступом через мобільний 3G-емулятор та десктопні браузери останніх версій Chrome/Firefox. Для бази даних застосовано MongoDB Atlas (кластер M2).

API-ендпоінти без викликів до AI-модулю тестувалися із застосуванням autocannon, сценарії навантажень включали:

1. 100 одночасних підключень (concurrent connections) з рівномірним інтервалом запитів;
2. серії батарейних тестів (burst) пікованої частоти до 200 запитів/с. Клієнтські метрики збиралися з допомогою Lighthouse: показники First Contentful Paint, Time to Interactive і Speed Index.

Результати навантаження API.

Без AI-обробки середній час відповіді — 170 мс (p95 ≈ 260 мс), CPU-середнє навантаження — 55 % при 100 concurrent. Під час пікових батерійних тестів (200 запитів/с) p95 не перевищував 320 мс, а вільна пам’ять утримувалася на рівні не нижче 20 % від загального обсягу.

Виклики до OpenAI (генерація та оцінка вправ) демонстрували середній latency 380 мс (внутрішня обробка — 150 мс, мережа та відповідь AI — 230 мс).

Продуктивність клієнта.

При емуляції слабкого з’єднання (3G) перший meaningful paint фіксувався на 0,85 с, Time to Interactive — на 1,3 с. Повторні переходи між екраном вправ і

статистики відбувалися за 20–30 мс на оновлення компонентів. Рендер основного списку запитань (до 10 одиниць) займав до 15 мс, що гарантує плавність UX.

4.2 Економічна ефективність

Для перевірки життєздатності рішення було розраховано основні витрати й потенційні джерела доходу, виходячи з моделі малого стартапу на 1 000 активних користувачів. Більшість систем планується орендувати у західних компаній, де вартість оцінюється у Доларах США, тому у підрахунках ми все зведемо саме до цією валюти.

Основні підрахунки:

1. Інфраструктурні витрати:
    1. VPS (2 vCPU, 4 GiB RAM): ≈ $20/міс.
    2. MongoDB Atlas (кластер M2): ≈ $9/міс.
    3. Домени та SSL: $10/рік.
2. Змінні витрати на AI: При середньому тарифі $0,005 за 1 000 токенів та 2 000 токенів на запит, за 500 запитів/день отримаємо: Токени/день: 500 × 2 000 = 1 000 000 токенів → $5/день → $150/міс.
3. Обслуговування та підтримка: Година роботи адміністратора (0,2 від повної ставки) оцінюється в 250 грн/год. ($6/год). При 40 год/міс. це $240.

Модель доходу

Припустимо підписку $5/міс. за розширений функціонал. Для покриття витрат на AI та інфраструктуру (≈ $180/міс.) достатньо 36 активних підписників. При 1 000 користувачів з конверсією в 10 % (100 платних) чистий прибуток складе ≈ $320/міс.

Висновки за розділом 4

Проведені навантажувальні тести підтвердили, що серверна частина обробляє запити до 300 мс навіть під високою активністю, а клієнтський

інтерфейс готовий до взаємодії менш ніж за 1,5 с, що відповідає заявленим нефункціональним вимогам та забезпечує комфортну роботу сотень одночасних користувачів.

Фінансовий аналіз показав, що при поточних тарифах на хостинг і API-моделі витрати складають близько 180 USD на місяць, тоді як модель підписки з ціною 5 USD на користувача дозволяє покривати ці витрати та отримувати стабільний прибуток уже за 10 % конверсії серед активних користувачів.

Отже, поєднання високої продуктивності та економічної ефективності робить розроблене рішення життєздатним для впровадження в навчальні практики та масштабування на широку аудиторію.

ВИСНОВКИ

У ході виконання дипломної роботи було обґрунтовано необхідність створення універсального автоматизованого крос-платформеного міні-додатку для вивчення іноземних мов у середовищі Telegram. Аналітичний розділ засвідчив, що наявні рішення не задовольняли одночасно потребу в динамічній генерації вправ, гнучкому адмініструванні контенту та користувачів, а також у регулярних нагадуваннях. На основі цього було сформульовано функціональні вимоги і визначено ключові сценарії використання.

У розділі проєктування була розроблена концептуальна модель із чітким розподілом на інтерфейсну частину, бізнес-логіку, зовнішні сервіси та планувальник сповіщень. Запропонована архітектура забезпечила модульність, масштабованість і можливість швидкої заміни окремих компонентів без впливу на загальну стабільність системи. Опис моделі даних і механізмів оптимізації зберігання підтвердив доцільність використання документно-орієнтованої бази даних для гетерогенних структур.

Практична реалізація підкреслила ефективність обраного підходу: клієнтську частину було створено інтуїтивно зрозумілою, з підтримкою офлайн-доступу до вправ та плавних анімацій, а серверну — налаштовано на надійну обробку запитів, інтеграцію з AI-сервісом та асинхронне надсилання нагадувань за індивідуальним графіком. Інтеграцію з платформою Telegram Mini Apps виконано безшовно, що спростило процес розгортання та використання рішення кінцевими користувачами.

Розроблений прототип повністю відповів поставленим завданням, продемонстрував необхідний рівень продуктивності та безпеки, а також легко масштабується для роботи з більшою кількістю користувачів і розширенням функціоналу. Практична цінність виконаної роботи полягає в готовності рішення до впровадження в освітні проєкти, а наукова новизна — у поєднанні

крос-платформеного підходу з автоматизованим механізмом генерації й оцінювання вправ на основі сучасних моделей ШІ.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Node.js [Електронний ресурс]. Посилання: https://nodejs.org/. Дата звернення: 16.04.2025
2. Express.js [Електронний ресурс]. Посилання: https://expressjs.com/. Дата звернення: 18.04.2025
3. CORS [Електронний ресурс]. Посилання: https://www.npmjs.com/package/cors. Дата звернення: 18.04.2025
4. Telegraf [Електронний ресурс]. Посилання: https://telegraf.js.org/. Дата звернення: 19.04.2025
5. Winston [Електронний ресурс]. Посилання: https://github.com/winstonjs/winston. Дата звернення: 19.04.2025
6. Telegram Mini Apps [Електронний ресурс]. Посилання: https://core.telegram.org/bots/webapps. Дата звернення: 20.04.2025
7. Mongoose [Електронний ресурс]. Посилання: https://mongoosejs.com/. Дата звернення: 21.04.2025
8. Node-schedule [Електронний ресурс]. Посилання: https://www.npmjs.com/package/node-schedule. Дата звернення: 21.04.2025
9. OpenAI API [Електронний ресурс]. Посилання: https://platform.openai.com/docs. Дата звернення: 23.04.2025
10. React [Електронний ресурс]. Посилання: https://reactjs.org/docs/getting-started.html. Дата звернення: 24.04.2025
11. Redux Toolkit [Електронний ресурс]. Посилання: https://redux-toolkit.js.org/. Дата звернення: 26.04.2025
12. Vite [Електронний ресурс]. Посилання: https://vitejs.dev/. Дата звернення: 28.04.2025
13. Framer Motion [Електронний ресурс]. Посилання: https://www.framer.com/motion/. Дата звернення: 29.04.2025

ДОДАТКИ

Додаток А

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Харківський національний університет імені В. Н. Каразіна

Навчально-науковий інститут комп'ютерних наук та штучного інтелекту
Кафедра комп'ютерних систем та робототехніки
Рівень вищої освіти (освітньо-кваліфікаційний рівень) **Бакалавр**
Галузь знань: 15 – Автоматизація та приладобудування
Спеціальність: 151 – Автоматизація та комп'ютерно-інтегровані технології
Освітня програма «Автоматизація та комп'ютерно-інтегровані технології»

ЗАТВЕРДЖУЮ
В.о. завідувача кафедри комп'ютерних
систем та робототехніки
к. ф.-м. н., доц. ХРУСЛОВ М. М.
«02» жовтня 2024року

З А В Д А Н Н Я
НА КВАЛІФІКАЦІЙНУ РОБОТУ

КУЛАЖЕНКО Даниїла Дмитровича
(прізвище, ім'я, по батькові студента)

1. Тема роботи **УНІВЕРСАЛЬНИЙ АВТОМАТИЗОВАНИЙ КРОС-ПЛАТФОРМНИЙ ЗАСТОСУНОК ДЛЯ ВИВЧЕННЯ МОВ ІЗ ШІ-МОДЕЛЯМИ.**

керівник роботи **Хруслов Максим Михайлович, кандидат фізико-математичних наук, доцент**
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом по університету ***16 квітня 2025 року № 4101-5/962***

2. Строк подання студентом роботи ***30 травня 2025 року***

3. Перелік питань, які потрібно розробити:
1. Аналіз сучасних підходів до розробки автоматизованих крос-платформних застосунків для вивчення іноземних мов та можливостей інтеграції моделей штучного інтелекту.
2. Розгляд концептуальних моделей крос-платформного застосунку для вивчення іноземної мови та необхідних інструментів задіяння моделей штучного інтелекту.
3. Дослідження методів тестування розробленої системи, для оцінки точності та надійності результатів.
4. Визначення можливих обмежень і проблем при використанні створеної моделі, таких як якість генерованих штучним інтелектом навчальних матеріалів.

4. План роботи

№ з/п	Назви етапів роботи	Термін виконання етапів роботи
1.	Затвердження теми роботи	05.09.2024 - 02.10.2024
2.	Літературний огляд з проблематики розробки автоматизованих крос-платформних застосунків	03.10.2024 - 15.10.2024
3.	Огляд методів розробки застосунків для мульти-платформного користування з можливістю інтеграції моделей штучного інтелекту	16.10.2024 - 30.10.2024
4.	Розробка концепції моделі автоматизованої лінгвістичної системи спрямованої на вивчення мов та впровадження моделі штучного інтелекту	01.11.2024 - 01.12.2024
5.	Тестування системи	12.12.2024 - 20.12.2024
6.	Аналіз результатів тестування та визначення ефективності	21.12.2024- 31.12.2024
7.	Підготовка рекомендацій та вдосконалення моделі	01.01.2025- 01.02.2025
8.	Підготовка і оформлення звітних матеріалів. Написання статті за матеріалами кваліфікаційної роботи.	02.02.2025- 31.03.2025-
9.	Підготовка і оформлення звітних матеріалів та додатків кваліфікаційної роботи. Оформлення списку літератури	01.04.2025- 15.04.2025-
10.	Оформлення пояснювальної записки кваліфікаційної роботи відповідно вимогам до звітів про НДР.	16.04.2025- 30.04.2025-
11.	Оформлення звіту про переддипломну практику	01.05.2025- 29.05.2025-
12.	Представлення кваліфікаційної роботи керівнику та рецензенту	30.05.2025-

5. Дата видачі завдання ***02 жовтня 2024 року.***

Студент Д.Д. Кулаженко _____
ініціали, прізвище підпис

Керівник роботи М.М. Хруслов _____
ініціали, прізвище підпис

Додаток Б

Затверджую

_____

«_____»_____2024 р.

Технічне завдання

на розробку системи

«Універсальний автоматизований крос-платформний застосунок для вивчення мов із ші-моделями».

Назва розділу	Назва і зміст підрозділу
1. Введення	1.1. Назва проекту: Розробка автоматизованого крос-платформеного застосунку для вивчення іноземної мови. Впровадження моделей штучного інтелекту. 1.2. Галузь застосування: Освіта, Корпоративне навчання персоналу
2. Підстава для розробки	2.1. Освітній курс за спеціальністю 151 – Автоматизація та комп’ютерно-інтегровані технології. 2.2. Завдання на дипломну роботу бакалавра, затверджено наказом ХНУ імені В. Н. Каразіна № 4101-5/962 від 16.04.2025 р.
3. Призначення розробки	3.1. Мета: Підвищення швидкості та якості вивчення іноземних мов за допомогою AI 3.2. Призначення: Інтерактивна платформа для самостійного та групового навчання, Адаптивні вправи та тестування з персональними рекомендаціями.

	3.3. Початкові дані для розробки: Корпуси текстів і аудіо з різних мовних рівнів, Словникові бази та мовні моделі, Вимоги та сценарії користувачів
4. Технічні вимоги до програмного виробу	4.1. Функціональні характеристики: • Крос-платформний UI (Android, iOS, Windows, macOS) • Модулі уроків, вправ, тестів, словника, словникових карток • AI-аналіз помилок та адаптивний підбір завдань 4.2. Надійність: • Доступність сервісу ≥99 % • Захист персональних даних (шифрування, SSL) • Регулярне резервне копіювання 4.3. Умови експлуатації: • Інтернет-з’єднання (опціонально офлайн-режим) • Мін. вимоги до пристроїв: 2 GB RAM, 100 MB вільного диску 4.4. Технічні засоби: • Користувацькі: смартфон, планшет, ПК • Серверні: хмарна платформа з GPU для AI-моделей 4.5. Сумісність: • Підтримка останніх двох версій ОС • Інтеграція з API словників і платформами LMS 4.6. Маркування та упаковка: • Цифровий дистрибутив (APK, IPA, MSI/DMG) • Електронна ліцензія та сертифікат відповідності 4.7. Транспортування та зберігання: • Розповсюдження через офіційні маркетплейси • Зберігання вихідних файлів у SCM (Git) 4.8. Спеціальні вимоги:

<table>
<tr><td></td><td>• Відповідність GDPR і локальним законам
• Локалізація інтерфейсу під кілька мов
• Мінімальне споживання трафіку й енергії</td></tr>
<tr><td>5. Вимоги до програмної документації.</td><td>• Функціональні та технічні специфікації
• Інсталяційний посібник і користувацький гід
• Документація API і опис схеми БД</td></tr>
<tr><td>6. Техніко-економічні показники</td><td>• Окупність інвестицій (ROI)
• Загальна вартість володіння (TCO)
• Ключові метрики: кількість активних користувачів, середній час сесії</td></tr>
<tr><td>7. Стадії і етапи розробки</td><td>
<table>
<tr><th>Дата</th><th>Назва етапу</th></tr>
<tr><td>05.09.2024 - 02.10.2024</td><td>Затвердження теми роботи</td></tr>
<tr><td>03.10.2024 - 15.10.2024</td><td>Літературний огляд з проблематики розробки автоматизованих крос-платформних застосунків</td></tr>
<tr><td>16.10.2024 - 30.10.2024</td><td>Огляд методів розробки застосунків для мульти-платформного користування з можливістю інтеграції моделей штучного інтелекту</td></tr>
</table>
</td></tr>
</table>

	01.11.2024 - 01.12.2024	Розробка концепції моделі автоматизованої лінгвістичної системи спрямованої на вивчення мов та впровалження моделі штучного інтелекту
	12.12.2024 - 20.12.2024	Тестування системи
	21.12.2024-31.12.2024	Аналіз результатів тестування та визначення ефективності
	01.01.2025-01.02.2025	Підготовка рекомендацій та вдосконалення моделі
	02.02.2025-31.03.2025	Підготовка і оформлення звітних матеріалів. Написання статті за матеріалами кваліфікаційної роботи.
	01.04.2025-15.04.2025	Підготовка і оформлення звітних матеріалів та додатків кваліфікаційної роботи. Оформлення списку літератури
	16.04.2025-30.04.2025	Оформлення

<table>
<tr><td></td><td><table>
<tr><td></td><td>пояснювальної записки кваліфікаційної роботи відповідно вимогам до звітів про НДР</td></tr>
<tr><td>01.05.2025-12.05.2025</td><td>Оформлення звіту про переддипломну практику</td></tr>
<tr><td>12.05.2025 – 31.05.2025</td><td>Представлення кваліфікаційної роботи керівнику та рецензенту</td></tr>
<tr><td>31.05.2025</td><td>Оформлення пояснювальної записки та підготовка презентації</td></tr>
</table></td></tr>
<tr><td>8. Порядок контролю і приймання</td><td>Проміжне тестування модулів (unit/integration)

Система приймальних випробувань (FAT/SAT)

Підписання актів виконаних робіт і передачі ПЗ користувачу</td></tr>
</table>

Виконавець
студент групи КУ-41
Кулаженко Д. Д.

Замовник
к. ф.-м. н., доц.
Хруслов М. М.

Додаток В

Програма і методика випробувань програмного виробу

«Універсальний автоматизований кросплатформний застосунок для вивчення мов із ШІ-моделями»

1. Об’єкт випробувань

1.1. Назва розробленого прототипу: «Універсальний автоматизований кросплатформний застосунок для вивчення мов із ші-моделями».

1.2. Галузь застосування: Інформаційні технології

1.3. Перераховані відомості запозичуються з відповідних розділів Технічного завдання.

2. Мета випробувань

Перевірка відповідності функціональні можливості системи заявленим функціональним можливостям в технічному завданні (Додаток Б до пояснювальної записки до кваліфікаційної роботи).

3. Загальні положення

3.1. Підстави для проведення випробувань

Підставою для проведення випробувань є наказ про призначення атестаційної комісії.

3.2. Місце і тривалість випробувань

Приймальні (приймально-здавальні) випробування проводяться на базі комп’ютерного класу кафедри в період роботи атестаційної комісії.

3.3. Обсяг випробувань

Приймальні випробування програмного виробу проводяться в обсязі відповідному цієї програми і методики випробувань.

3.4. Організації, які беруть участь у випробуваннях

Приймальні випробування проводяться атестаційною комісією напередодні засідання (або в процесі засідання) за участю Замовника, Виконавці та інших осіб, присутніх на засіданні.

4. Вимоги до програми або програмного виробу

Програмний продукт повинен задовольняти такі вимоги:

1. Платформа: будь-який ПК із доступом до Інтернет і встановленим Telegram-клієнтом (версія 8.0+).
2. Надійність: система має стабільно працювати при обробці запитів до AI-модулю та одночасному підключенні до 100 користувачів.
3. Сумісність: додаток має коректно працювати з браузерами Chrome/Firefox та Telegram Desktop/Mobile (iOS/Android).

4. Продуктивність: час обробки запиту до AI-модулю не повинен перевищувати 400 мс (за середнього навантаження).
5. Масштабованість: можливість додавання нових мов і розширення бази вправ без зміни архітектури.
6. Апаратура: ПК (Windows/Linux/macOS) з 4 ГБ ОЗП та сучасним процесором або віртуальна машина з подібними характеристиками.

Спеціальних вимог до маркування, транспортування і зберігання не пред'являється.

5. Вимоги до програмної документації

Документація до програмного забезпечення для продукту «Універсальний автоматизований кросплатформний застосунок для вивчення мов із ШІ-моделями» повинна містити:

1. Опис основних вимог та функціональності системи (подається як Додаток В пояснювальної записки до роботи).
2. Програма тестування та методологія розробленого програмного забезпечення (подається як Додаток В пояснювальної записки до роботи).
3. Опис архітектури розробленої системи (подається в розділах 2 та 3 пояснювальної записки до роботи).

6. Засоби і порядок випробувань

6.1. Інструменти випробування

– ПК або ноутбук із Telegram Desktop/Mobile.
– Емулятор мобільної мережі для тестів продуктивності (наприклад, Chrome DevTools).
– Інструмент для навантажувального тестування API (autocannon).
– Таблиці й протоколи перевірок (схеми в Excel або Google Sheets).

6.2. Порядок випробувань

6.2.1 Етап 1: Ознайомчий

- Перевірка комплектності документації.
- Перевірка наявності коду, ER-діаграм, інструкції користувача.
- Оцінка готовності середовища: під'єднання до Telegram, налаштування AI-ключа.

6.2.2. Етап 2: Функціональні випробування

Тест 1: Авторизація користувача

- Відкрити Telegram Mini App, передати ініціалізаційні дані.
- Перевірити, що система коректно розпізнає ID і зберігає профіль у БД.

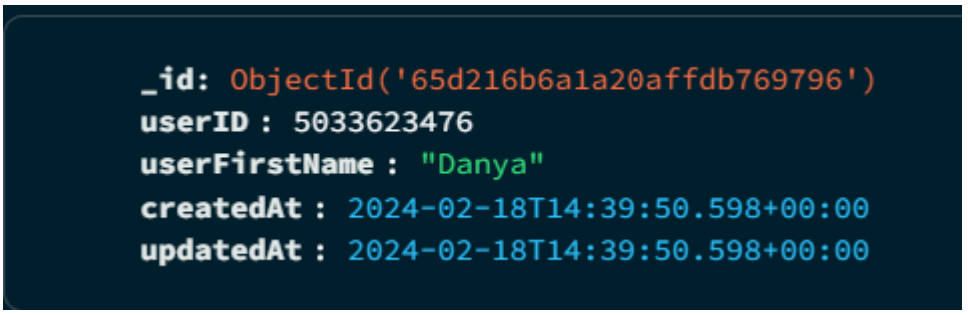

Рис. В.1. Після авторизації система зберігає профіль у БД.

Тест 2: Генерація вправи (коректне формування AI-запиту)

- Користувач обирає тему.
- Клік «Генерувати вправу» → перевірити відповідь від сервера.
- Оцінити, що у відповідь приходить текст вправи.

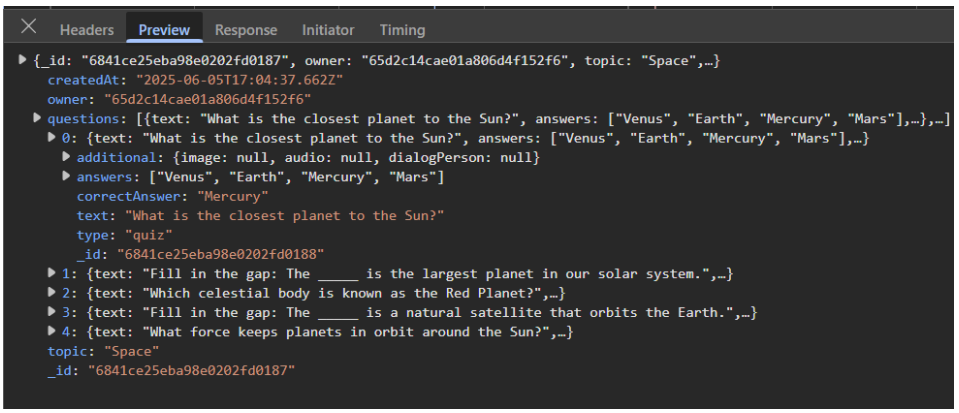

Рис. В.2. API відповідь на запит створення тесту через AI

Тест 3: Перевірка відповіді

- Користувач вводить некоректні відповіді .
- Система відправляє запит до AI-модуля для валідації.

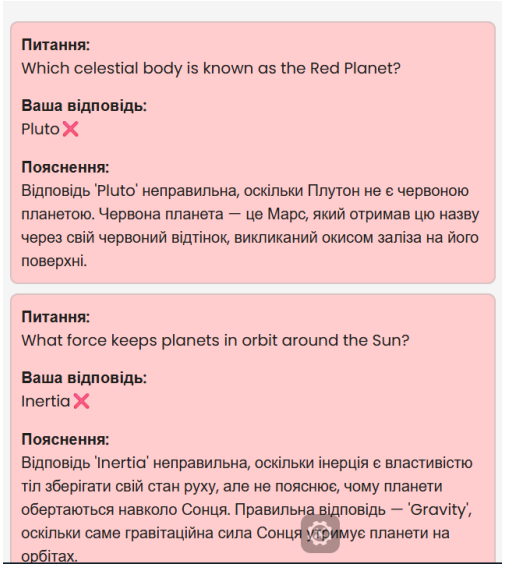

Рис. В.3. Програма знайшла неправильні відповіді і дав пояснення завдяки AI-модулю

Тест 4: Масове надсилання

- Адміністратор надсилає шаблонне повідомлення «!DISTRUTE_ALL Привіт!»
- Перевірити, що кожен зареєстрований користувач отримує текст у чат.

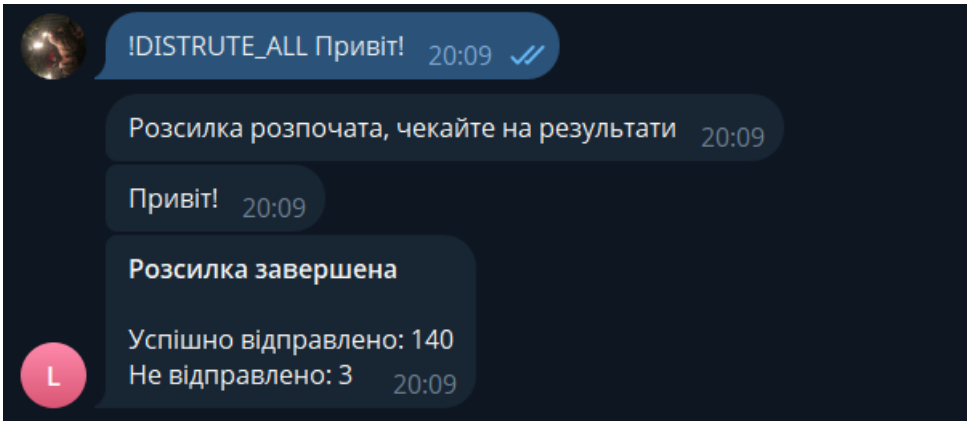

Рис. В.4. Статистика показує кількість успішних/невдалих надсилань, а в чаті користувачів відображено «Привіт!»

Висновки: усі функції працюють відповідно до ТЗ; навантаження витримано, система готова до впровадження.

Виконавець: студент групи КУ-41, Кулаженко Д.Д. _____