МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

МАТЕМАТИЧНИЙ ФАКУЛЬТЕТ
Кафедра комп'ютерних наук

КВАЛІФІКАЦІЙНА РОБОТА БАКАЛАВРА

на тему: **«РЕАЛІЗАЦІЯ ЧАТ-БОТА ДЛЯ АВТОМАТИЗАЦІЇ СЛУЖБИ ПІДТРИМКИ КОРИСТУВАЧІВ»**

Виконала: студентка 4 курсу, групи 6.1221
спеціальності 122 Комп'ютерні науки
(шифр і назва спеціальності)
освітньої програми Комп'ютерні науки
(назва освітньої програми)
А.О. Ковердюк
(ініціали та прізвище)
Керівник доцент кафедри комп'ютерних наук, к.т.н. Добровольський Г.А.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)
Рецензент доцент кафедри програмної інженерії, к.ф.-м.н. Мильцев О. М.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)

Запоріжжя 2025

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

Факультет математичний

Кафедра комп'ютерних наук

Рівень вищої освіти бакалавр

Спеціальність 122 Комп'ютерні науки
(шифр і назва)

Освітня програма Комп'ютерні науки

ЗАТВЕРДЖУЮ
Завідувач кафедри комп'ютерних наук,
д.т.н., професор

_____ Шило Г.М.
(підпис)

« ____ » _____ 2024 р.

З А В Д А Н Н Я

НА КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТЦІ

Ковердюк Анастасії Олександрівні
(прізвище, ім'я та по-батькові)

1. Тема роботи Реалізація чат-бота для автоматизації служби підтримки користувачів

керівник роботи Добровольський Геннадій Анатолійович, к.т.н
(прізвище, ім'я та по-батькові, науковий ступінь, вчене звання)

затверджені наказом ЗНУ від « 20 » грудня 2024 року № 2001-с

2. Строк подання студентом роботи 13.06.2025

3. Вихідні дані до роботи 1. Постановка задачі.
2. Перелік літератури.
3. Обраний технологічний стек та інструментарій.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)
1. Постановка задачі.
2. Основні теоретичні відомості.
3. Розробка та реалізація інтелектуального чат-бота.
4. Експериментальне дослідження роботи чат-бота.

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)
Презентація

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв

7. Дата видачі завдання 25.12.2024

К А Л Е Н Д А Р Н И Й П Л А Н

№	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1.	Розробка плану роботи.	03.03.2025	
2.	Збір вихідних даних.	07.03.2025	
3.	Обробка методичних та теоретичних	10.03.2025	
	джерел.		
4.	Розробка першого та другого розділу.	01.04.2025	
5.	Розробка третього розділу.	10.05.2025	
6.	Оформлення та нормоконтроль	10.06.2025	
	кваліфікаційної роботи бакалавра.		
7.	Захист кваліфікаційної роботи.	18.06.2025	

Студент _____ А.О. Ковердюк
(підпис) (ініціали та прізвище)

Керівник роботи _____ Г.А. Добровольський
(підпис) (ініціали та прізвище)

Нормоконтроль пройдено

Нормоконтролер _____ О.Г. Спиця
(підпис) (ініціали та прізвище)

РЕФЕРАТ

Кваліфікаційна робота бакалавра «Реалізація чат-бота для автоматизації служби підтримки користувачів»: 47 с., 12 рис., 3 табл., 9 джерел.

АВТОМАТИЗАЦІЯ, ВЕЛИКІ МОВНІ МОДЕЛІ, ПРОМПТ-ІНЖИНІРИНГ, СЛУЖБА ПІДТРИМКИ, ЧАТ-БОТ, AIOGRAM, API, GEMINI, MOODLE, NLP, PYTHON, RAILWAY, TELEGRAM.

Об'єкт дослідження – процес автоматизації надання консультаційних послуг користувачам інформаційних систем за допомогою інтелектуальних програмних агентів.

Предмет дослідження – методи, алгоритми та програмні засоби для реалізації мультимодального чат-бота на базі великої мовної моделі.

Мета роботи – розробка та реалізація програмного прототипу інтелектуального мультимодального чат-бота для автоматизації служби підтримки користувачів LMS Moodle.

Методи дослідження: системний аналіз, порівняльний аналіз, об'єктно-орієнтоване програмування, методи інтеграції API та емпіричного тестування.

У кваліфікаційній роботі вирішено актуальну проблему автоматизації служб підтримки шляхом розробки інтелектуального чат-бота. На основі аналізу сучасних технологій штучного інтелекту було обрано стек: модель Google Gemini 2.5 Flash, месенджер Telegram, мова Python з фреймворком aiogram та хмарна платформа Railway.

Було спроєктовано та реалізовано програмний прототип, здатний обробляти текстові, голосові та графічні запити. Особливу увагу приділено системі промпт-інжинірингу для точного керування поведінкою AI. Проведене функціональне тестування підтвердило повну працездатність системи та її відповідність поставленим вимогам, продемонструвавши високий потенціал рішення для практичного застосування.

SUMMARY

Bachelor's Qualifying Theses «Implementation of Chat-Bot for Users' Support Service»: 47 pages, 12 figures, 3 tables, 9 references.

AUTOMATION, LARGE LANGUAGE MODELS, PROMPT ENGINEERING, SUPPORT SERVICE, CHATBOT, AIOGRAM, API, GEMINI, MOODLE, NLP, PYTHON, RAILWAY, TELEGRAM.

The object of the study is the process of automating the provision of consulting services to users of information systems using intelligent software agents.

The subject of the study is methods, algorithms, and software tools for implementing a multimodal chatbot based on a large language model.

The aim of the study is to develop and implement a software prototype of an intelligent multimodal chatbot for automating the LMS Moodle user support service.

Methods of research: system analysis, comparative analysis, object-oriented programming, API integration methods, and empirical testing.

The thesis solves the pressing problem of support service automation by developing an intelligent chatbot. Based on an analysis of modern artificial intelligence technologies, the following stack was selected: Google Gemini 2.5 Flash model, Telegram messenger, Python language with the aiogram framework, and the Railway cloud platform.

A software prototype capable of processing text, voice, and graphic queries was designed and implemented. Particular attention was paid to the prompt engineering system for precise control of AI behaviour. Functional testing confirmed the full operability of the system and its compliance with the requirements, demonstrating the high potential of the solution for practical application.

ЗМІСТ

Завдання на кваліфікаційну роботу……..2
Реферат 4
Summary 5
Скорочення та умовні познаки 8
Вступ 9
1 Теоретичні основи автоматизації служб підтримки за допомогою інтелектуальних чат-ботів 11
1.1 Автоматизація служб підтримки як сучасний тренд у цифровій трансформації 11
1.2 Чат-боти: визначення, класифікація та еволюція 12
1.3 Технології, що лежать в основі інтелектуальних чат-ботів 14
1.4 Порівняльний аналіз платформ та моделей для створення AI-ботів. 15
1.5 Аналіз платформ для розповсюдження та розгортання 18
1.5.1 Вибір каналу розповсюдження 18
1.5.2 Вибір платформи для розгортання 20
1.6 Висновки до розділу 1 22
2 Проєктування та реалізація чат-бота 23
2.1 Формулювання вимог до системи 23
2.2 Проєктування архітектури системи 24
2.3 Реалізація ключових програмних модулів 26
2.3.1 Структура проєкту та конфігурація 26
2.3.2 Обробники повідомлень (Handlers) 27
2.3.3 Реалізація мультимодальної логіки 28
2.3.4 Головна функція запуску 29
2.4 Розробка системи промпт-інжинірингу 30
2.4.1 Визначення ролі та бази знань 30
2.4.2 Керування форматом відповіді 31

2.4.3 Структура фінального промпта 32
2.5 Розгортання та забезпечення роботи 24/7 33
2.5.1 Структура фінального промпта 33
2.5.2 Процес розгортання на Railway 34
2.6 Висновки до розділу 2 35
3 Тестування та аналіз результатів 36
3.1 Розробка плану та методики тестування 36
3.2 Проведення тестування та аналіз результатів 37
3.3 Аналіз результатів та можливі шляхи покращення 42
3.4 Висновки до розділу 3 44
Висновки 45
Перелік посилань 47

СКОРОЧЕННЯ ТА УМОВНІ ПОЗНАКИ

NLP	Natural Language Processing
LLM	Large Language Model
API	Application Programming Interface
SDK	Software Development Kit
AI	Artificial Intelligence
ШІ	Штучний Інтелект
LMS	Learning Management System

ВСТУП

В епоху тотальної цифровізації суспільства та переходу освітніх і бізнес-процесів в онлайн-середовище критично зростає навантаження на служби технічної підтримки. Користувачі систем управління навчанням (LMS), корпоративних порталів та інших складних веб-платформ потребують швидкої та кваліфікованої допомоги 24/7. Традиційні методи підтримки, що базуються на роботі операторів-людей, стикаються з низкою викликів: обмежений робочий час, висока вартість утримання штату, людський фактор та нездатність одночасно обробляти велику кількість типових запитів.

Саме тому автоматизація служби підтримки за допомогою інтелектуальних чат-ботів є надзвичайно актуальним завданням. Сучасні досягнення в галузі штучного інтелекту, зокрема розвиток великих мовних моделей (LLM), відкрили нові горизонти для створення чат-ботів, здатних не просто реагувати на ключові слова, а розуміти природну мову, аналізувати контекст і навіть інтерпретувати зображення та голос. Такий підхід дозволяє кардинально підвищити якість та доступність підтримки, знизивши при цьому операційні витрати.

Реалізація чат-бота для користувачів такої поширеної платформи, як Moodle, є яскравим прикладом практичного застосування цієї технології. Можливість отримувати миттєві відповіді на запити, надіслані у вигляді тексту, голосового повідомлення або скріншота з проблемою, – це значний крок до створення по-справжньому ефективного та дружнього до користувача цифрового середовища.

Метою кваліфікаційної роботи є розробка та реалізація програмного прототипу інтелектуального чат-бота для месенджера Telegram, призначеного для автоматизації служби підтримки користувачів системи управління навчанням Moodle.

Для досягнення поставленої мети необхідно вирішити такі задачі:

- проаналізувати науково-технічну літературу та наявні програмні рішення у сфері автоматизації служб підтримки та створення чат-ботів;
- обґрунтувати вибір інструментальних засобів для реалізації проєкту, зокрема мови програмування, фреймворку та API для інтеграції штучного інтелекту;
- сформулювати функціональні та нефункціональні вимоги до чат-бота;
- спроєктувати архітектуру програмного засобу, що забезпечує обробку текстових, голосових та графічних запитів;
- реалізувати програмний код чат-бота, включаючи логіку взаємодії з API Telegram та Google Gemini;
- розробити систему промптів для керування поведінкою мовної моделі та форматуванням її відповідей;
- провести тестування розробленого прототипу для перевірки його функціональності та відповідності поставленим вимогам.

Наукова новизна одержаних результатів полягає в розробці архітектури та програмного прототипу чат-бота, що комплексно поєднує обробку трьох типів даних (текст, голос, зображення) для надання підтримки в нішевій предметній області (LMS Moodle) шляхом інтеграції з сучасною мультимодальною мовною моделлю Google Gemini. Новизна також полягає у формуванні специфічних інструкцій (промптів), що керують поведінкою та форматом відповіді ШІ відповідно до технічних обмежень платформи Telegram.

Розроблений чат-бот може бути впроваджений в освітніх закладах та організаціях, що використовують Moodle, для зниження навантаження на службу підтримки та підвищення задоволеності користувачів. Програмний код проєкту є готовим прототипом, який може слугувати основою для подальшого розвитку та створення комерційних продуктів. Результати роботи можуть бути використані як навчальний матеріал для студентів, що вивчають розробку програмного забезпечення та інтеграцію систем штучного інтелекту.

1 ТЕОРЕТИЧНІ ОСНОВИ АВТОМАТИЗАЦІЇ СЛУЖБ ПІДТРИМКИ ЗА ДОПОМОГОЮ ІНТЕЛЕКТУАЛЬНИХ ЧАТ-БОТІВ

1.1 Автоматизація служб підтримки як сучасний тренд у цифровій трансформації

У сучасному світі, що характеризується стрімкою цифровою трансформацією, компанії та освітні установи активно впроваджують складні інформаційні системи для оптимізації своєї діяльності. Системи управління навчанням (LMS), корпоративні CRM- та ERP-системи, онлайн-банкінг та інші платформи стали невід'ємною частиною повсякденних операцій. Разом із цим зростає і потреба користувачів у якісній та своєчасній технічній підтримці, що стає критичним фактором для забезпечення їхньої лояльності та ефективної роботи.

Традиційна модель служби підтримки, що покладається виключно на операторів-людей, стикається з низкою системних викликів, які обмежують її ефективність:

- людські ресурси обмежені, що унеможливлює надання підтримки в режимі 24/7 без значного збільшення штату та витрат;
- фонд оплати праці кваліфікованих операторів, витрати на їхнє навчання, облаштування робочих місць та адміністрування становлять значну частину бюджету будь-якої організації;
- у пікові періоди кількість звернень може різко зростати, що призводить до довгих черг на відповідь та зниження якості обслуговування;
- ефективність оператора може залежати від його емоційного стану, втоми чи рівня кваліфікації;
- значна частина робочого часу витрачається на відповіді на однотипні, повторювані запитання, що призводить до професійного вигорання [1, с. 16].

Відповіддю на ці виклики стала автоматизація служб підтримки, що є одним із ключових напрямів цифрової трансформації бізнесу та сервісів. Впровадження автоматизованих систем, зокрема інтелектуальних чат-ботів, дозволяє досягти значних переваг. По-перше, вони забезпечують миттєву обробку запитів у будь-який час доби та дня тижня. По-друге, чат-боти здатні одночасно вести діалог із тисячами користувачів, ефективно вирішуючи проблему масштабованості. По-третє, вони кардинально знижують вартість обробки одного запиту, вивільняючи людські ресурси для вирішення складних, нетипових завдань, що потребують експертного втручання.

Таким чином, автоматизація служби підтримки – це не просто спосіб заощадити кошти, а стратегічна інвестиція в покращення клієнтського досвіду, підвищення операційної ефективності та здобуття конкурентної переваги в сучасному цифровому ландшафті.

1.2 Чат-боти: визначення, класифікація та еволюція

Чат-бот (англ. chatbot) – це комп’ютерна програма, призначена для імітації людської розмови через текстові або голосові повідомлення [1, с. 309]. Основна мета чат-бота – надавати користувачам інформацію, виконувати певні завдання або просто вести діалог у максимально природній та зручній для людини формі. Історія розвитку цих програмних агентів дозволяє виділити кілька ключових етапів їхньої еволюції, що безпосередньо впливає на їхню класифікацію.

Перші прототипи чат-ботів, як-от програма ELIZA, створена Джозефом Вейценбаумом у 1966 році, працювали на основі простих алгоритмів розпізнавання ключових слів та використання шаблонних відповідей [1, с. 3]. Цей підхід ліг в основу першого і найпростішого класу чат-ботів.

На сьогодні виділяють дві основні категорії чат-ботів за принципом їхньої роботи:

– декларативні (скриптові, або засновані на правилах);

– інтелектуальні (AI-based, або засновані на штучному інтелекті).

Декларативні чат-боти функціонують за заздалегідь прописаним сценарієм (скриптом). Вони розпізнають обмежений набір команд або ключових слів і надають шаблонні відповіді або пропонують користувачеві варіанти дій у вигляді кнопок. Логіка їхньої роботи жорстко визначена розробником.

– переваги: передбачуваність, висока швидкість роботи, простота розробки для вузьких завдань, повний контроль над діалогом;
– недоліки: нездатність розуміти запити, не передбачені сценарієм; відсутність гнучкості; не можуть навчатися в процесі взаємодії.

Інтелектуальний клас чат-ботів використовує технології машинного навчання (ML) та обробки природної мови (NLP) для розуміння запитів користувачів. Вони не обмежуються ключовими словами, а здатні аналізувати контекст, розпізнавати наміри (intents) та сутності (entities) в реченнях.

– переваги: здатність вести більш природний діалог; розуміння широкого спектра формулювань одного й того ж запиту; можливість самонавчання та покращення з часом;
– недоліки: вища складність розробки та навчання; потребують великих обсягів даних для тренування; менш передбачувані у своїх відповідях.

Сучасний етап еволюції чат-ботів пов'язаний із появою та стрімким розвитком великих мовних моделей (LLM), таких як GPT, Gemini та Claude. Ці моделі вивели інтелектуальні чат-боти на якісно новий рівень, наділивши їх здатністю генерувати зв’язні, контекстуально доречні та стилістично багаті відповіді, а також обробляти не лише текст, а й інші модальності даних – зображення та аудіо. Саме цей підклас інтелектуальних чат-ботів є предметом дослідження даної кваліфікаційної роботи.

1.3 Технології, що лежать в основі інтелектуальних чат-ботів

Функціонування інтелектуальних чат-ботів, здатних до ведення осмисленого діалогу, забезпечується комплексом технологій, що належать до сфери штучного інтелекту. Ключовою дисципліною є обробка природної мови (Natural Language Processing, NLP) – галузь інформатики та лінгвістики, що вивчає проблеми комп'ютерного аналізу та синтезу природної мови [1, с. 1]. У контексті чат-ботів NLP можна умовно розділити на два основні напрями: розуміння та генерування мови.

Розуміння природної мови (Natural Language Understanding, NLU) – це підгалузь NLP, відповідальна за те, щоб комп'ютер «зрозумів» зміст повідомлення користувача. NLU вирішує кілька ключових завдань:

- розпізнавання наміру (Intent Recognition): визначення головної мети або цілі, яку переслідує користувач;
- видобування сутностей (Entity Extraction): ідентифікація та вилучення з тексту ключових фрагментів даних, необхідних для виконання запиту;
- аналіз тональності (Sentiment Analysis): визначення емоційного забарвлення повідомлення (позитивне, негативне, нейтральне), що дозволяє боту адаптувати свою відповідь.

Генерування природної мови (Natural Language Generation, NLG) – це процес створення людиночитаного тексту на основі структурованих даних. Після того як система зрозуміла запит користувача і отримала необхідну інформацію з бази даних чи через API, NLG-модуль «упаковує» ці дані в граматично правильне та стилістично доречне речення. Наприклад, отримавши дані {місто: "Київ", дата: "20.10.2024", температура: "15°C", опади: "немає"}, NLG-система може згенерувати відповідь: «У неділю, 20 жовтня, в Києві очікується сонячна погода без опадів, температура повітря складе близько 15 градусів тепла».

Великі мовні моделі (Large Language Models, LLM) – це сучасний етап розвитку NLP-технологій, що являє собою глибокі нейронні мережі, навчені

на гігантських обсягах текстових даних. На відміну від традиційних NLP-систем, які потребували окремого налаштування для кожного завдання (розпізнавання намірів, сутностей тощо), LLM володіють узагальненим «розумінням» мови. Вони здатні виконувати широкий спектр завдань без спеціального донавчання (так зване «zero-shot» або «few-shot» learning) [9].

Ключова перевага LLM полягає в тому, що вони поєднують в собі NLU та NLG в єдиній, безшовній архітектурі. Модель отримує на вхід текстовий запит (промпт) і генерує відповідь, спираючись на вивчені патерни та знання про світ. Сучасні мультимодальні LLM, такі як Google Gemini, можуть обробляти не лише текст, а й зображення та аудіо, що відкриває принципово нові можливості для створення чат-ботів [2]. Саме використання такої моделі дозволяє реалізувати проєкт, де бот розуміє скріншоти та голосові повідомлення, що було б значно складніше реалізувати за допомогою класичних NLP-підходів.

1.4 Порівняльний аналіз платформ та моделей для створення AI-ботів

Вибір великої мовної моделі (LLM) є визначальним етапом при проєктуванні інтелектуального чат-бота, оскільки від її можливостей, вартості та доступності залежить функціональність, економічна доцільність та стабільність майбутнього продукту. Сучасний ринок LLM представлений кількома ключовими гравцями та типами моделей. Для обґрунтованого вибору було проведено порівняльний аналіз їхніх флагманських моделей за рядом критичних параметрів.

Критерії для порівняння:

– мультимодальність: здатність моделі обробляти різні типи даних (текст, зображення, аудіо). Це є ключовою вимогою для даного проєкту;

- продуктивність та якість: точність, релевантність та швидкість генерації відповідей;
- вартість використання (API Pricing): цінова політика за обробку даних;
- доступність та ліміти API: наявність безкоштовного доступу для розробки та обмеження на кількість запитів;
- зручність інтеграції: якість технічної документації та наявність клієнтських бібліотек (SDK).

Розглянемо основних представників ринку:

- OpenAI GPT-4o: флагманська мультимодальна модель від OpenAI, що є галузевим стандартом продуктивності;
- Google Gemini 2.5 Flash: остання розробка від Google, оптимізована для швидкості та низької вартості, з нативною мультимодальністю;
- DeepSeek-V2-Chat: високоефективна модель від DeepSeek AI. Її найсильнішою стороною є розуміння та генерація програмного коду, а також надзвичайно низька вартість;
- Anthropic Claude 3 Sonnet: модель, що робить акцент на безпеці та здатності обробляти дуже великі обсяги тексту.

Результати порівняльного аналізу ключових комерційних моделей зведено в таблицю 1.1.

Таблиця 1.1 – Порівняльний аналіз флагманських LLM

Критерій	OpenAI GPT-4o	Google Gemini 2.5 Flash	DeepSeek-V2-Chat	Anthropic Claude 3 Sonnet
Мультимодальність	так (текст, аудіо, зображення)	так (текст, аудіо, зображення)	ні (тільки текст)	так (текст, зображення)
Продуктивність	дуже висока	висока, оптимізована для швидкості	висока, найкраща у кодуванні	висока
Вартість (вхід), $/1M токенів	$5.00	$0.35	~$0.14	$3.00

Продовження таблиці 1.1

Вартість (вихід), $/1M токенів	$15.00	$0.70	~$0.28	$15.00
Безкоштовний доступ API	обмежений	так, щедрий free tier	так, значний обсяг безкоштовних токенів	обмежений
Ліміти Free Tier (RPM)	низькі	60 запитів на хвилину	помірні	низькі
Зручність інтеграції	відмінна документація, sdk для python	відмінна документація, sdk для python	добра документація, sdk для python	дуже добра документація, sdk для python

Аналіз даних, представлених у таблиці 1.1, дозволяє зробити однозначний висновок на користь моделі Google Gemini 2.5 Flash (а саме, версії gemini-2.5-flash-preview-05-20, що використовувалася в проєкті) для реалізації даної кваліфікаційної роботи.

Модель DeepSeek-V2-Chat заслуговує на окрему увагу. Вона демонструє видатні показники у роботі з програмним кодом та пропонує безпрецедентно низьку вартість, що робить її ідеальним кандидатом для завдань, пов'язаних із кодуванням. Однак її ключовим недоліком у контексті даного проєкту є відсутність мультимодальності в її загальнодоступному API для чат-взаємодії.

Оскільки технічне завдання прямо передбачає обробку зображень (скріншотів) та аудіо (голосових повідомлень), використання виключно текстової моделі, навіть такої економічно вигідної, є неможливим для досягнення поставленої мети.

Модель OpenAI GPT-4o є потужним конкурентом, але значно програє за вартістю та має менш доступний безкоштовний рівень для розробки. Модель Anthropic Claude 3 Sonnet також є дорожчою і не підтримує обробку аудіо через офіційний API на момент дослідження.

Таким чином, Google Gemini 2.5 Flash є єдиною моделлю серед розглянутих, яка оптимально поєднує в собі всі критично важливі для проєкту вимоги:

- повна мультимодальність: нативна підтримка тексту, аудіо та зображень;
- економічна доцільність: найнижча вартість у порівнянні з іншими мультимодальними моделями;
- доступність для розробки: наявність щедрого безкоштовного рівня з високим лімітом у 60 запитів на хвилину, що дозволяє проводити розробку та тестування без фінансових витрат;
- висока швидкість: версія «Flash» забезпечує кращий користувацький досвід в інтерактивних застосунках, що є пріоритетом для чат-бота.

1.5 Аналіз платформ для розповсюдження та розгортання

Після вибору технологій розробки, необхідно визначити два середовища: середовище взаємодії з користувачем (канал розповсюдження) та середовище виконання програмного коду (платформа розгортання).

1.5.1 Вибір каналу розповсюдження

Вибір каналу комунікації є не менш важливим, ніж вибір технології штучного інтелекту (ШІ), оскільки від нього залежить доступність, зручність використання та функціональні можливості чат-бота для кінцевого користувача. Ринок пропонує безліч платформ для розгортання ботів, від веб-віджетів на сайтах до інтеграцій у корпоративні месенджери. Для проєкту, орієнтованого на широку аудиторію користувачів, найбільш доцільним є використання популярних мобільних месенджерів.

Було проведено аналіз трьох провідних платформ: Telegram, Viber та Facebook (табл. 1.2).

Таблиця 1.2 – Порівняльний аналіз платформ-месенджерів

Критерій	Telegram	Viber	Facebook
Документація API	дуже детальна, відкрита, якісні приклади	задовільна, вимагає реєстрації бізнес-акаунту	детальна, але складна, орієнтована на екосистему meta
Функціональність API	широка: кнопки, клавіатури, медіа, форматування	обмежена порівняно з telegram	широка, але з багатьма політиками та обмеженнями
Вартість	повністю безкоштовно для розробників	може стягуватися плата за бізнес-повідомлення	безкоштовно, але з можливими витратами на рекламу для просування
Простота розробки	висока, наявність зрілих бібліотек (aiogram)	середня	середня, вимагає схвалення бота (review process)
Популярність в Україні	найвища	висока	середня
Мультимедійні можливості	відмінна підтримка фото, аудіо, відео, файлів	добра підтримка	добра підтримка

На основі проведеного аналізу, для реалізації проєкту було обрано месенджер Telegram. Цей вибір зумовлений низкою ключових переваг:

- Telegram Bot API є надзвичайно потужним, гнучким та чудово документованим [4];
- наявність високорівневих асинхронних бібліотек, таких як aiogram для Python, значно спрощує та прискорює процес створення бота, дозволяючи зосередитись на бізнес-логіці, а не на низькорівневій взаємодії з API [5];
- створення та підтримка бота в Telegram є повністю безкоштовними, що робить його ідеальною платформою для освітніх, дослідницьких та некомерційних проєктів;
- Telegram є найпопулярнішим месенджером в Україні, що гарантує максимальне охоплення потенційної аудиторії користувачів без необхідності встановлювати додаткові застосунки.

Також Telegram Bot API надає повний контроль над повідомленнями, інтерактивними елементами (кнопки, клавіатури) та підтримує багате форматування тексту, що було використано в проєкті.

Хоча Viber та Facebook також є життєздатними платформами, їхні API є більш обмежувальними, а процес розробки та схвалення може бути складнішим, що робить Telegram беззаперечним лідером для даного типу проєкту.

1.5.2 Вибір платформи для розгортання

Для забезпечення доступності чат-бота в режимі 24/7 та виконання нефункціональних вимог, необхідно було обрати надійну платформу для розміщення програмного коду. Запуск застосунку на локальному комп'ютері не розглядається як життєздатний варіант для робочого сервісу. Найбільш сучасним підходом є використання хмарних платформ типу «Платформа як послуга» (PaaS), які абстрагують розробника від адміністрування серверів.

Було проведено порівняльний аналіз трьох популярних PaaS-платформ: Heroku (традиційний лідер), Render та Railway (сучасний підхід) у таблиці 1.3.

Таблиця 1.3 – Порівняльний аналіз PaaS-платформ для розгортання

Критерій	Heroku	Render	Railway.app
Цінова політика	скасовано безкоштовний тариф, платний з $5/міс.	є обмежений безкоштовний тариф	модель «pay-as-you-go» з щедрим стартовим кредитом
Простота налаштування	висока, але вимагає Heroku CLI для деяких завдань	висока, інтуїтивний вебінтерфейс	найвища, мінімалістичний інтерфейс, автоматичне визначення
Автоматичне розгортання	так, через інтеграцію з GitHub	так, через інтеграцію з GitHub	так, миттєве розгортання після кожного коміту

Продовження таблиці 1.3

Підтримка Procfile	так (стандарт)	не вимагає, налаштовується через вебінтерфейс	так (підтримується для сумісності)
Гнучкість	обмежена стандартними «dynos»	гнучка конфігурація сервісів (веб, воркер, БД)	дуже гнучка, будь-який сервіс з Dockerfile або Nixpacks

На основі аналізу, наведеного в таблиці 1.3, для реалізації проєкту було обрано платформу Railway.app. Цей вибір обґрунтовано наступними перевагами:

- сучасна модель ціноутворення: на відміну від Heroku, що повністю відмовився від безкоштовних планів, та Render з його обмеженим тарифом, Railway пропонує модель оплати за фактично використані ресурси (Pay-as-you-go) та надає значний стартовий кредит (наприклад, $5), якого більш ніж достатньо для розробки, тестування та навіть довготривалої роботи малонавантаженого проєкту, як-от даний чат-бот. Це робить платформу економічно найвигіднішою [7];
- максимальна простота розгортання: Railway дотримується філософії «просто працює». Достатньо підключити репозиторій, і платформа автоматично, за допомогою системи Nixpacks, визначає мову програмування, встановлює залежності та запускає проєкт. Це значно знижує поріг входу та прискорює процес розгортання;
- висока гнучкість та швидкість: Railway миттєво реагує на зміни в репозиторії, автоматично запускаючи новий деплой, що ідеально підходить для ітеративної розробки. Підтримка Procfile забезпечує легку міграцію з інших платформ, як-от Heroku.

1.6 Висновки до розділу 1

У першому розділі кваліфікаційної роботи було проведено комплексний теоретичний аналіз предметної області автоматизації служб підтримки за допомогою інтелектуальних чат-ботів.

Було визначено, що традиційні методи підтримки є неефективними для обробки великої кількості запитів, що зумовлює актуальність впровадження автоматизованих рішень. Проаналізовано еволюцію чат-ботів від простих скриптових систем до сучасних мультимодальних агентів на базі великих мовних моделей (LLM).

Проведено детальний порівняльний аналіз ключових технологій та платформ. В результаті аналізу було обґрунтовано вибір наступного технологічного стеку для реалізації програмного прототипу:

- модель штучного інтелекту: Google Gemini 2.5 Flash – як найбільш економічно вигідне та функціонально відповідне рішення, що підтримує обробку текстових, голосових та графічних даних;
- платформа розповсюдження: месенджер Telegram – як найпопулярніша в Україні платформа з найпотужнішим та найзручнішим API для розробників;
- платформа розгортання: PaaS-сервіс Railway.app – як оптимальний вибір, що поєднує в собі простоту використання, гнучкість, високу швидкість розробки та привабливу економічну модель;
- мова програмування та фреймворк: Python та бібліотека aiogram – як галузевий стандарт для розробки асинхронних Telegram-ботів.

Таким чином, теоретичний аналіз дозволив сформувати міцний технологічний фундамент для подальшого проєктування та реалізації системи, описаних у наступних розділах роботи.

2 ПРОЄКТУВАННЯ ТА РЕАЛІЗАЦІЯ ЧАТ-БОТА

Цей розділ присвячено детальному опису процесу створення програмного прототипу чат-бота для служби підтримки користувачів Moodle. На основі теоретичного аналізу, проведеного в попередньому розділі, тут розглядаються етапи проєктування, розробки ключових модулів та розгортання застосунку.

2.1 Формулювання вимог до системи

Першим кроком у розробці будь-якого програмного продукту є визначення вимог. Це дозволяє чітко окреслити межі проєкту, визначити критерії успішності та сформувати основу для проєктування та тестування. Вимоги до чат-бота було розділено на дві категорії: функціональні та нефункціональні.

Функціональні вимоги (описують, що система повинна робити):

- F.1. Система повинна реагувати на команду /start, надсилаючи персоналізоване вітальне повідомлення;
- F.2. Система повинна приймати та обробляти текстові повідомлення від користувача, що стосуються функціонала LMS Moodle;
- F.3. Система повинна приймати та обробляти голосові повідомлення, транскрибуючи їх та використовуючи отриманий текст як запит;
- F.4. Система повинна приймати та обробляти зображення (скріншоти), аналізуючи їхній вміст у контексті запиту користувача, наданого в підписі до фото;
- F.5. Система повинна надавати відповіді українською мовою;

– F.6. Кожна відповідь, що базується на документації, повинна містити гіперпосилання на відповідну сторінку офіційної документації Moodle [3];
– F.7. Система повинна коректно обробляти запити, що не стосуються її предметної області (Moodle), інформуючи користувача про обмеження своєї компетенції.

Нефункціональні вимоги (описують, як система повинна працювати):

– NF.1. Система повинна бути доступною для користувачів у режимі 24/7;
– NF.2. Час відправки відповіді на складний запит (з обробкою аудіо чи фото) не повинен перевищувати 15 секунд для забезпечення комфортної взаємодії;
– NF.3. Взаємодія з ботом має бути інтуїтивно зрозумілою для будь-якого користувача месенджера Telegram без необхідності вивчення спеціальних команд;
– NF.4. Секретні ключі доступу (токен бота, ключ API Gemini) не повинні зберігатися у відкритому вигляді в коді. Вони мають передаватися в застосунок через змінні оточення;
– NF.5. Програмний код має бути добре структурованим, прокоментованим та відповідати стандартам написання коду на мові Python (PEP 8) для спрощення подальшої модифікації та підтримки.

2.2 Проєктування архітектури системи

Архітектура програмного забезпечення визначає його основні компоненти та принципи їхньої взаємодії. Для розробленого чат-бота було обрано архітектуру, що базується на взаємодії кількох незалежних сервісів через їхні програмні інтерфейси (API).

Ключовими компонентами системи є:

– клієнтський застосунок (Telegram App): мобільний або десктопний додаток, через який користувач взаємодіє з системою;
– сервер Telegram Bot API: проміжний сервіс від Telegram, що приймає повідомлення від користувачів та пересилає їх у вигляді об'єктів-оновлень (Updates) на сервер бота, а також приймає команди від бота для відправки повідомлень користувачу;
– бекенд-застосунок (Бот на Railway.app): ядро системи, розроблене на Python з використанням фреймворку aiogram. Цей компонент відповідає за прийом оновлень від Telegram, обробку бізнес-логіки, формування запитів до AI-моделі та відправку відповідей;
– сервіс Google Gemini API: зовнішній сервіс штучного інтелекту, що приймає від бекенд-застосунку запит (промпт) та дані (текст, аудіо, фото) і повертає згенеровану відповідь.

Схему взаємодії цих компонентів при обробці запиту користувача зображено на рисунку 2.1.

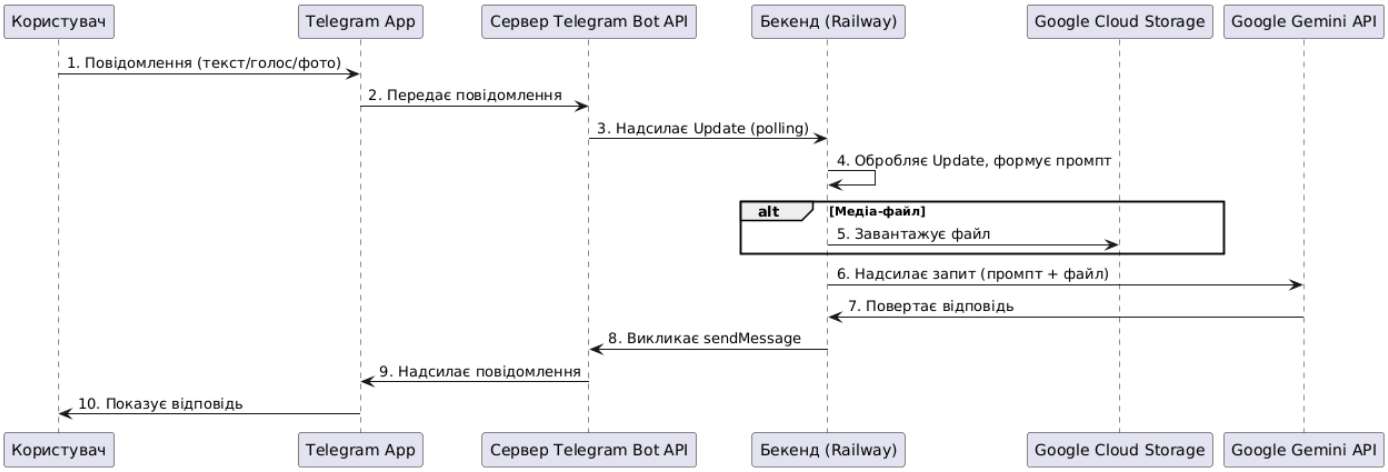


Рисунок 2.1 – Архітектурна схема взаємодії компонентів системи

Як видно з діаграми, кожен компонент системи виконує чітко визначену роль:

– користувач ініціює взаємодію;

- Telegram Bot API виступає як шлюз (gateway), що передає повідомлення між користувачем та бекендом;
- бекенд-застосунок (бот) є ядром системи, що реалізує всю бізнес-логіку: обробляє оновлення, керує файлами, формує промпти та взаємодіє з Gemini;
- Google Gemini API виступає як «мозок» системи, що відповідає за аналіз даних та генерацію відповіді.

Тобто, застосунок працює за асинхронним принципом, очікуючи на оновлення від Telegram. Такий підхід, реалізований за допомогою бібліотеки aiogram, дозволяє ефективно обробляти запити від багатьох користувачів одночасно, не блокуючи основний потік виконання під час очікування відповіді від зовнішніх сервісів, таких як Gemini API.

2.3 Реалізація ключових програмних модулів

Програмна реалізація чат-бота виконана на мові програмування Python з використанням асинхронного фреймворку aiogram [6]. Структура проєкту організована таким чином, щоб забезпечити читабельність, супроводжуваність та легкість подальшого розширення. У цьому підрозділі розглянуто основні компоненти коду та їхню роль у функціонуванні системи.

2.3.1 Структура проєкту та конфігурація

Основою проєкту є головний файл bot.py, що містить усю логіку роботи бота. Для керування залежностями використовується файл requirements.txt, а для безпечного зберігання конфігураційних даних – файл .env.

- файл requirements.txt: містить перелік усіх зовнішніх бібліотек Python, необхідних для роботи застосунку (aiogram, python-dotenv, google-generativeai). Це дозволяє автоматизувати процес

налаштування робочого середовища за допомогою команди pip install -r requirements.txt;

- файл .env: використовується для зберігання секретних ключів та налаштувань поза системою контролю версій. У проєкті він містить наступні змінні: TELEGRAM_BOT_TOKEN, GEMINI_API_KEY та GEMINI_MODEL;
- папки temp_audio та temp_images: тимчасові директорії, що створюються автоматично при запуску для збереження файлів, які надсилає користувач, перед їхньою обробкою та подальшим видаленням.

На початку виконання скрипту bot.py відбувається ініціалізація основних об'єктів: завантаження змінних оточення, налаштування логування, ініціалізація клієнта Google Gemini API та об'єктів Bot і Dispatcher з бібліотеки aiogram.

2.3.2 Обробники повідомлень (Handlers)

Логіка обробки вхідних повідомлень реалізована за допомогою механізму обробників (handlers) фреймворку aiogram. Кожен обробник – це асинхронна функція-декоратор, що відповідає за певний тип повідомлення.

Обробник, що реагує на стартову команду /start наведено на рисунку 2.2.

```
@router.message(CommandStart())
async def command_start_handler(message: Message) -> None:
    # ... код відправки вітального повідомлення ...
```

Рисунок 2.2 – Обробник команди /start

Ця функція реагує на команду /start, відправляючи користувачу персоналізоване вітальне повідомлення з коротким описом можливостей бота.

Основний обробник текстового повідомлення від користувача представлено на рисунку 2.3.

```
@router.message(F.text)
async def moodle_question_handler(message: Message) -> None:
    # ... код обробки тексту ...
```

Рисунок 2.3 – Обробник текстових повідомлень

Цей обробник активується на будь-яке текстове повідомлення. Він формує запит (промпт) до моделі Gemini, передає їй текст користувача, отримує згенеровану відповідь та надсилає її назад користувачу.

2.3.3 Реалізація мультимодальної логіки

Найбільш складною частиною реалізації є обробка нетекстових даних. Цей процес вимагає взаємодії з файловою системою та виконання кількох послідовних кроків. Розглянемо його на прикладі обробника зображень (див. рис. 2.4).

```
@router.message(F.photo)
async def photo_message_handler(message: Message, bot: Bot) -> None:
    # ... код обробки фото ...
```

Рисунок 2.4 – Обробник зображень (фото)

Логіка роботи цього обробника реалізує алгоритм, описаний на діаграмі послідовності (рис. 2.1), і складається з таких етапів:

– інформування користувача: бот негайно надсилає повідомлення «Аналізую зображення...», щоб користувач знав, що його запит прийнято в роботу;

- завантаження файлу: застосунок отримує ідентифікатор файлу зображення найкращої якості (message.photo[-1]), завантажує його з серверів Telegram та зберігає у тимчасову директорію temp_images під унікальним ім'ям;
- запит до Gemini API:
  а) за допомогою функції asyncio.to_thread тимчасовий файл асинхронно завантажується на сервери Google;
  б) формується мультимодальний запит, що складається з текстового промпта (інструкції для моделі) та посилання на завантажений файл;
  в) виконується асинхронний запит model.generate_content_async().
- відправка відповіді та очищення:
  а) отримана від Gemini відповідь надсилається користувачу;
  б) блок finally гарантує, що тимчасовий файл буде видалено як з локального диску, так і з серверів Google, незалежно від успішності виконання попередніх кроків. Цей підхід забезпечує ефективне використання ресурсів.

Аналогічна логіка застосовується і для обробки голосових повідомлень, з тією різницею, що використовується інший фільтр (F.voice) та MIME-тип файлу (audio/ogg).

2.3.4 Головна функція запуску

Точкою входу в програму є асинхронна функція main(), яка виконує фінальні налаштування та запускає бота (див. рис. 2.5).

```
async def main() -> None:
    # Підключаємо роутер до диспетчера
    dp.include_router(router)
```

Рисунок 2.5 – Функція запуску бота

```
    # Видаляємо вебхук та накопичені оновлення
    await bot.delete_webhook(drop_pending_updates=True)
    # Запускаємо бота в режимі довгого опитування
    await dp.start_polling(bot)

if __name__ == "__main__":
    # Запускаємо асинхронну функцію main
    asyncio.run(main())
```

Рисунок 2.5, аркуш 2

Виклик dp.start_polling(bot) встановлює безперервне з'єднання з серверами Telegram для отримання оновлень, що дозволяє боту працювати в постійному режимі очікування.

2.4 Розробка системи промпт-інжинірингу

На відміну від традиційного програмування, де логіка жорстко задається кодом, при роботі з великими мовними моделями (LLM) поведінка системи значною мірою визначається вхідними інструкціями – промптами (англ. prompts). Процес створення, тестування та оптимізації цих інструкцій називається промпт-інжинірингом (prompt engineering) [8]. Для даного проєкту було розроблено систему промптів, що вирішує кілька ключових завдань: визначення ролі моделі, керування форматом відповіді та її поведінкою [9].

2.4.1 Визначення ролі та бази знань

Першочерговим завданням було чітко окреслити для моделі її роль та обмежити її базу знань. Це необхідно, щоб уникнути "галюцинацій"

(генерування неправдивої інформації) та відповідей, що не стосуються предметної області. Для цього було використано наступну базову інструкцію:

«Ти — експерт-помічник по системі управління навчанням Moodle. Твоя база знань — офіційна документація Moodle Docs (https://docs.moodle.org/)».

Ця інструкція одразу задає моделі контекст: вона не є універсальним асистентом, а спеціалізованим експертом, що має спиратися на конкретне джерело інформації.

2.4.2 Керування форматом відповіді

Як було виявлено на етапі тестування, API Telegram накладає суворі обмеження на HTML-розмітку, яку він може обробити. Модель Gemini, навчена на величезних масивах веб-даних, за замовчуванням може використовувати широкий спектр HTML-тегів (<p>, тощо), що призводить до помилок парсингу на стороні Telegram.

Для вирішення цієї проблеми було розроблено спеціальне правило форматування, яке жорстко обмежує набір дозволених тегів. Це правило було винесено в окрему змінну FORMATTING_RULE для перевикористання у всіх промптах (див. рис. 2.6).

```
FORMATTING_RULE = """
Для форматування тексту ти повинен використовувати тільки наступні HTML-теги:
<b>...</b> - для жирного тексту
<i>...</i> - для курсиву
<code>...</code> - для коду або назв кнопок
<a href='URL'>...</a> - для посилань
Категорично заборонено використовувати будь-які інші теги...
"""
```

Рисунок 2.6 – Правило форматування для промпта

Така чітка та категорична інструкція змушує модель генерувати відповідь у форматі, що є гарантовано сумісним з API Telegram.

2.4.3 Структура фінального промпта

Фінальний промпт для кожного типу запиту (текст, голос, фото) динамічно конструюється з кількох блоків:

- роль: визначення ролі експерта по Moodle;
- завдання: опис конкретного завдання (проаналізувати текст, розшифрувати аудіо тощо) та передача даних від користувача;
- правило поведінки: інструкція щодо відсутності привітань;
- правило форматування: інструкція щодо використання HTML-тегів.

Фінальний промпт для обробки текстового запиту, на рисунку 2.7, виглядає наступним чином.

```
prompt = f"""
Ти — експерт-помічник по системі управління навчанням Moodle. База знань — офіційна документація Moodle (https://moodle.org/).
Завдання: Проаналізуй запит "{message.text}", дай коротку відповідь українською.
В кінці відповіді намагайся надати релевантні посилання на офіційну англомовну документацію Moodle (https://docs.moodle.org/500/en/Main_page).
Вітайся з користувачем тільки коли він сам привітався.
Використовуй емоджи, якщо це доречно.
{FORMATTING_RULE}
"""
```

Рисунок 2.7 – Фінальна структура промпта

Такий модульний підхід до промпт-інжинірингу дозволив створити гнучку та керовану систему, де поведінка штучного інтелекту чітко контролюється розробником для досягнення необхідного результату та забезпечення стабільної роботи в середовищі Telegram.

2.5 Розгортання та забезпечення роботи 24/7

Для виконання нефункціональної вимоги щодо доступності системи в режимі 24/7 (NF.1), програмний прототип чат-бота було розгорнуто на хмарній PaaS-платформі Railway.app. Цей процес забезпечив автоматизацію розгортання та стабільне середовище для виконання коду. Процедура складалася з наступних етапів.

2.5.1 Структура фінального промпта

Перед завантаженням коду до системи контролю версій GitHub було виконано кілька підготовчих кроків:

- створення файлу requirements.txt: цей файл містить перелік усіх необхідних Python-бібліотек і дозволяє платформі Railway автоматично встановити їх за допомогою команди pip install -r requirements.txt;
- створення файлу Procfile: оскільки чат-бот є не веб-сервером, а фоновим процесом (worker), необхідно було явно вказати платформі, як саме запускати застосунок. Для цього було створено файл Procfile з єдиним рядком: «worker: python main.py». Ця інструкція повідомляє Railway, що потрібно запустити процес типу worker, виконавши команду python main.py;
- налаштування файлу .gitignore: для дотримання найкращих практик безпеки, файл .env, що містить секретні ключі, було додано до

.gitignore. Це запобігає випадковому потраплянню конфіденційних даних до репозиторію.

2.5.2 Процес розгортання на Railway

Після розміщення підготовленого коду в приватному репозиторії на GitHub, у вебінтерфейсі Railway було виконано наступні дії:

- створення нового проєкту: було створено новий проєкт та обрано опцію «Deploy from GitHub repo»;
- підключення репозиторію: Railway було надано доступ до приватного репозиторію з кодом чат-бота. Платформа автоматично проаналізувала код, виявила наявність файлу Procfile і коректно визначила тип сервісу;
- налаштування змінних оточення: для забезпечення безпеки (вимога NF.4), усі секретні ключі було введено у відповідному розділі налаштувань проєкту на Railway («Variables»). Було створено наступні змінні: TELEGRAM_BOT_TOKEN, GEMINI_API_KEY, GEMINI_MODEL;
- ці змінні автоматично передаються в робоче середовище застосунку під час його запуску, що є безпечним та рекомендованим способом керування конфігурацією;
- запуск розгортання: після налаштування змінних, Railway автоматично ініціював процес розгортання (deploy). Цей процес включав клонування репозиторію, встановлення залежностей з requirements.txt та запуск процесу worker згідно з інструкцією у Procfile.

Вбудована консоль логів («Logs») на платформі Railway дозволила в реальному часі відстежити процес запуску бота та переконатися у відсутності помилок. Після успішного розгортання чат-бот став повністю функціональним та доступним для користувачів у Telegram.

2.6 Висновки до розділу 2

У другому розділі кваліфікаційної роботи було детально описано процес перетворення теоретичних концепцій у робочий програмний продукт.

Було сформульовано чіткі функціональні та нефункціональні вимоги до системи. На їх основі спроєктовано архітектуру чат-бота, що візуалізовано за допомогою діаграм варіантів використання та послідовності.

Детально розглянуто реалізацію ключових програмних модулів на мові Python з використанням фреймворку aiogram. Описано логіку обробки текстових, голосових та графічних повідомлень, а також механізми взаємодії з Google Gemini API. Особливу увагу приділено системі промпт-інжинірингу, що дозволяє тонко керувати поведінкою та форматом відповідей штучного інтелекту.

На завершення було описано процес розгортання застосунку на хмарній платформі Railway, що забезпечило його стабільну та безперебійну роботу в режимі 24/7. Таким чином, у рамках другого розділу було успішно створено та розгорнуто програмний прототип, що повністю відповідає поставленим вимогам та готовий до етапу тестування.

3 ТЕСТУВАННЯ ТА АНАЛІЗ РЕЗУЛЬТАТІВ

Цей розділ присвячено верифікації працездатності розробленого чат-бота та аналізу якості його роботи. Тестування є невід'ємним етапом життєвого циклу програмного забезпечення, що дозволяє перевірити відповідність продукту раніше визначеним вимогам, виявити потенційні помилки та оцінити загальний користувацький досвід.

3.1 Розробка плану та методики тестування

Метою тестування є перевірка ключових функціональних та нефункціональних аспектів роботи чат-бота. Для цього було розроблено план, що охоплює кілька видів тестування, які виконувались вручну (manual testing).

План тестування:

- модульне тестування (Unit Testing): на етапі розробки кожен обробник (для тексту, голосу, фото) тестувався ізольовано для перевірки коректності його внутрішньої логіки;
- інтеграційне тестування (Integration Testing): перевірка коректної взаємодії всіх компонентів системи: клієнт Telegram -> бекенд на Railway -> Google Gemini API. Метою було переконатися, що дані успішно передаються по всьому ланцюжку;
- тестування функціональних вимог (Functional Testing): основний етап тестування, спрямований на перевірку відповідності системи функціональним вимогам, сформульованим у підрозділі 2.1. Для цього було розроблено набір тестових сценаріїв (тест-кейсів);
- тестування користувацького інтерфейсу (UI Testing): оцінка зручності взаємодії з ботом, читабельності та коректності форматування його відповідей у месенджері Telegram.

Методика функціонального тестування: для кожної ключової функції було розроблено тестовий сценарій, що включає наступні кроки:

- дія: дія, яку виконує користувач (наприклад, «Надіслати текстове повідомлення з питанням про Moodle»);
- очікуваний результат: поведінка системи, яка вважається коректною (наприклад, «Бот надає релевантну відповідь українською мовою з посиланням на документацію»);
- фактичний результат: реальна відповідь системи під час тестування;
- статус: «Успішно» або «Невдало».

Тестові сценарії охоплювали як позитивні випадки (очікувана поведінка користувача), так і негативні (запити не по темі, відправка непідтримуваних типів файлів тощо).

3.2 Проведення тестування та аналіз результатів

На цьому етапі було проведено тестування за розробленими сценаріями. Нижче наведено приклади виконання ключових тест-кейсів, що ілюструють роботу основних функцій чат-бота.

Тест-кейс 1: Обробка текстового запиту:

- дія: користувач надсилає текстове повідомлення: «Як створити тест у курсі?»;
- очікуваний результат: бот надає коротку інструкцію щодо створення тесту в Moodle, відповідь сформульована українською мовою, містить посилання на офіційну документацію та відформатована за допомогою HTML;
- фактичний результат: (наведено на рисунку 3.1);
- аналіз: відповідь є релевантною, чіткою та відповідає запиту. Форматування та посилання на джерело присутні.
- статус: успішно.

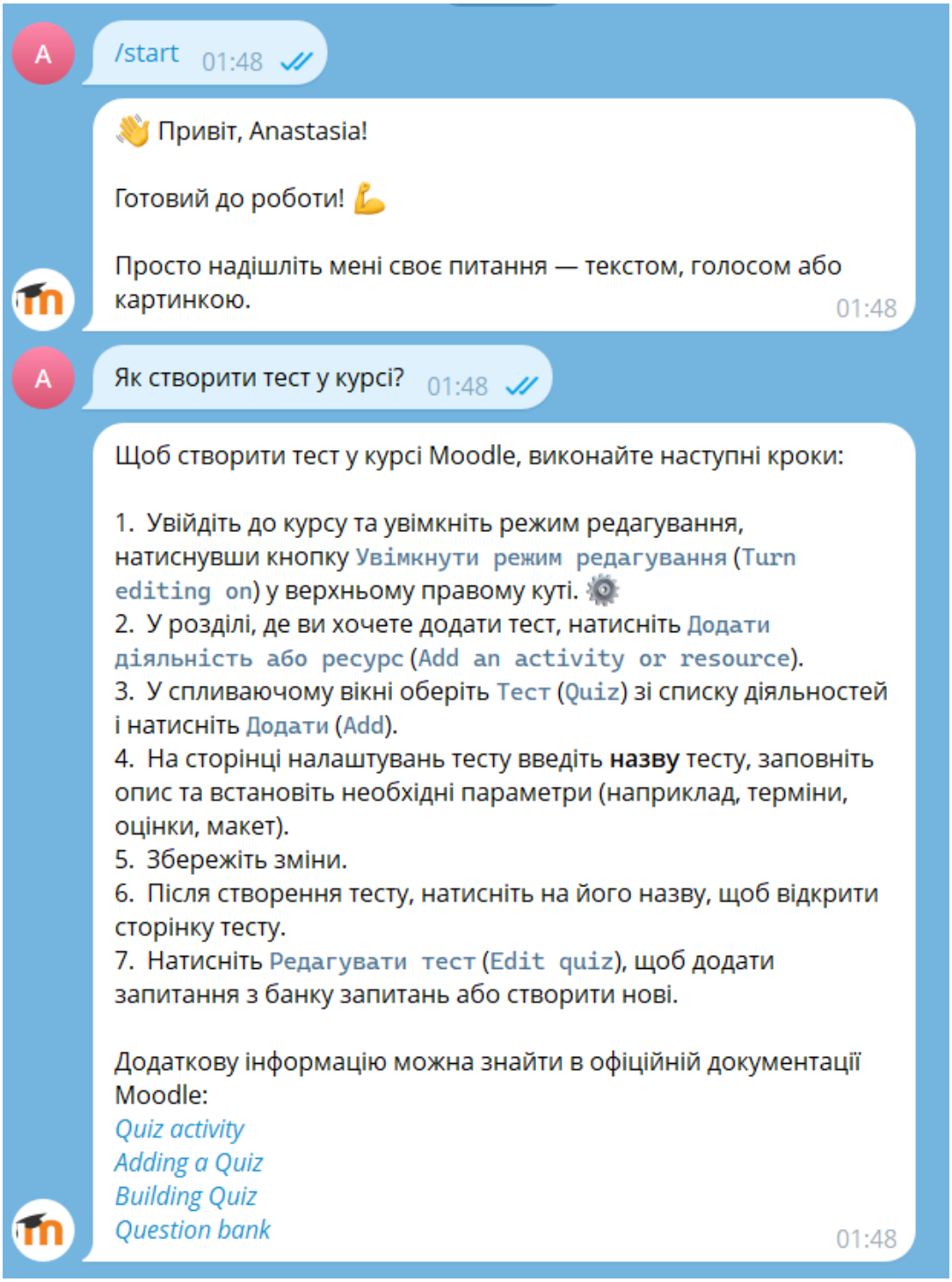


Рисунок 3.1 – Результат обробки текстового запиту

Тест-кейс 2: Обробка голосового запиту:

– дія: користувач надсилає голосове повідомлення, в якому запитує: «Розкажи, як записати студента на курс.»;
– очікуваний результат: бот коректно транскрибує аудіо, розуміє зміст запиту та надає інструкцію щодо зарахування студентів;

– фактичний результат: (наведено на рисунку 3.2);
– аналіз: система успішно перетворила мовлення в текст та надала коректну відповідь, що демонструє працездатність мультимодальної логіки;
– статус: успішно.

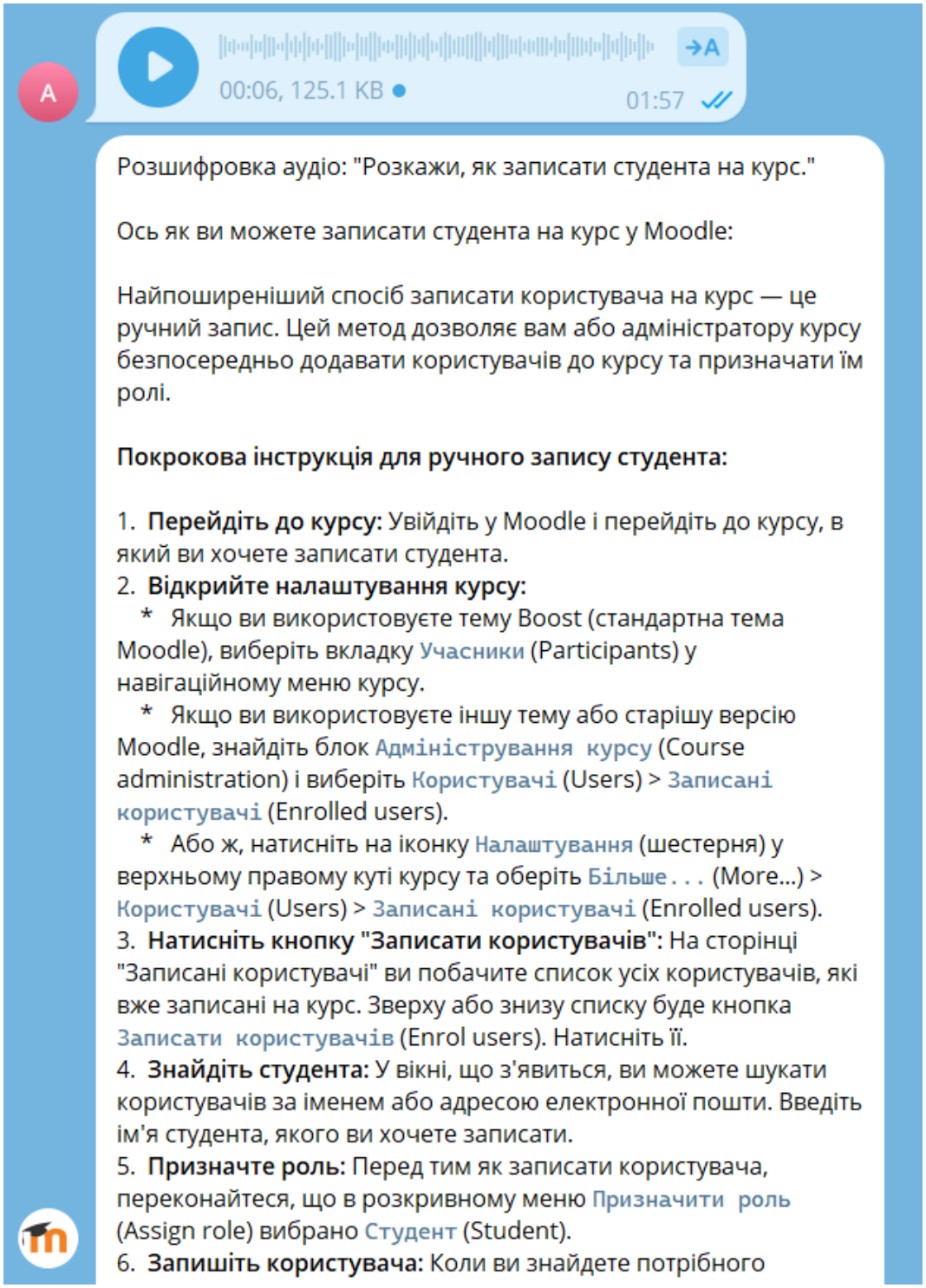


Рисунок 3.2 – Результат обробки голосового запиту

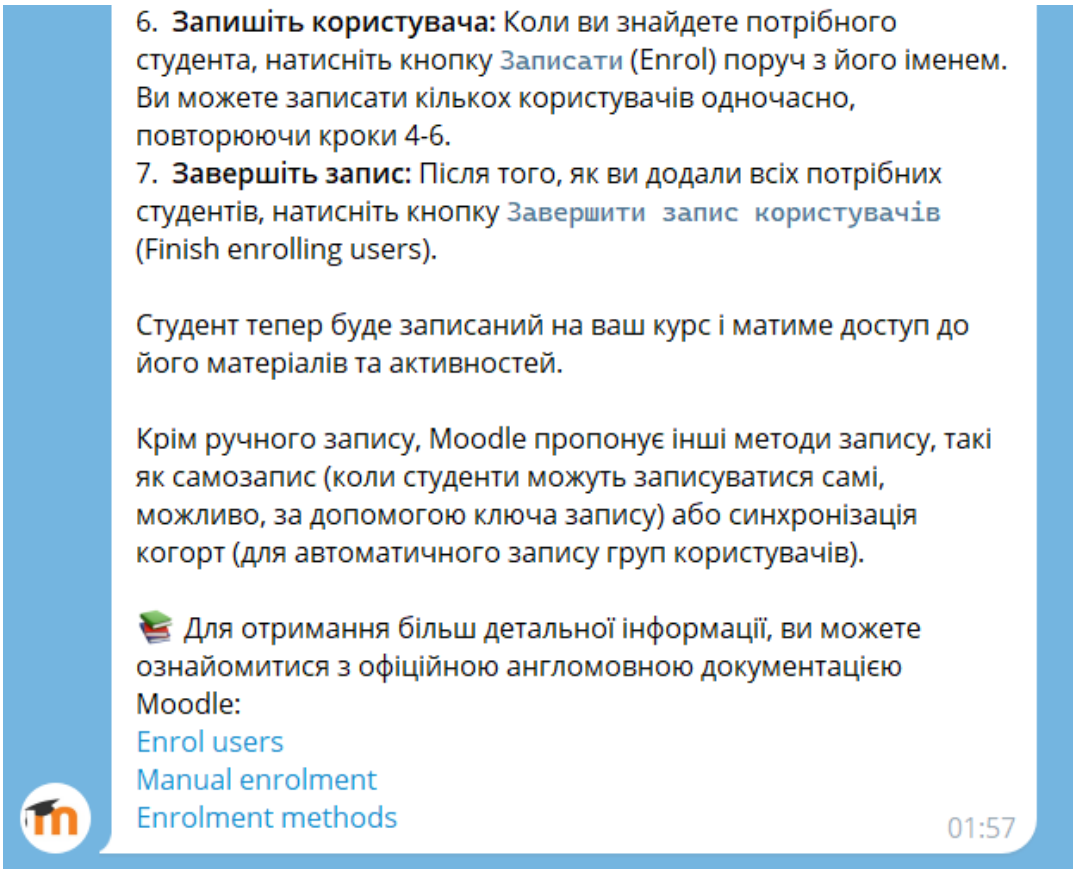


Рисунок 3.2, аркуш 2

Тест-кейс 3: Обробка запиту із зображенням:

– дія: користувач надсилає скріншот сторінки завдань курсу в Moodle з підписом: «Куди тут натискати, щоб здати роботу?»;
– очікуваний результат: бот аналізує як зображення, так і текстовий запит, і надає конкретну вказівку, вказуючи на потрібний елемент інтерфейсу;
– фактичний результат: (наведено на рисунках 3.3–3.4);
– аналіз: бот продемонстрував здатність до візуального аналізу в контексті запиту, що є найскладнішою реалізованою функцією. Відповідь є точною та корисною;
– статус: успішно.

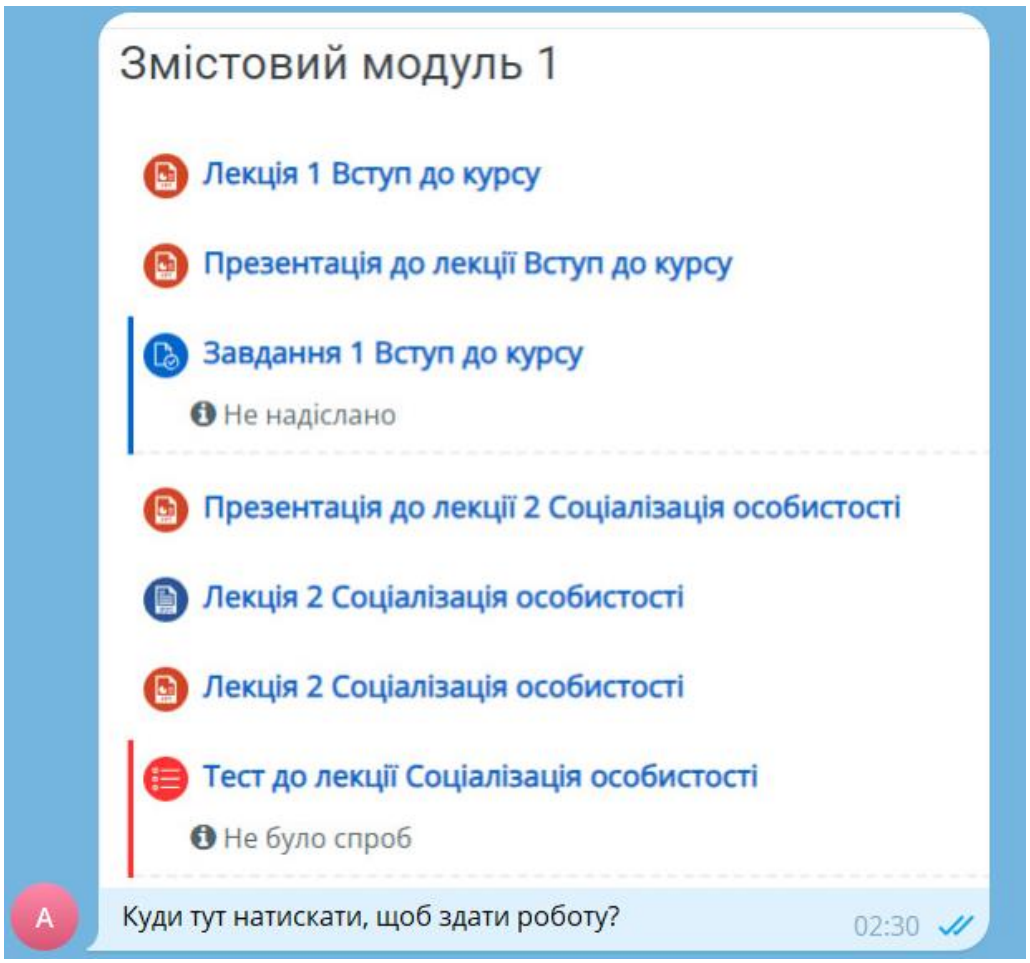


Рисунок 3.3 – Запит користувача із зображенням

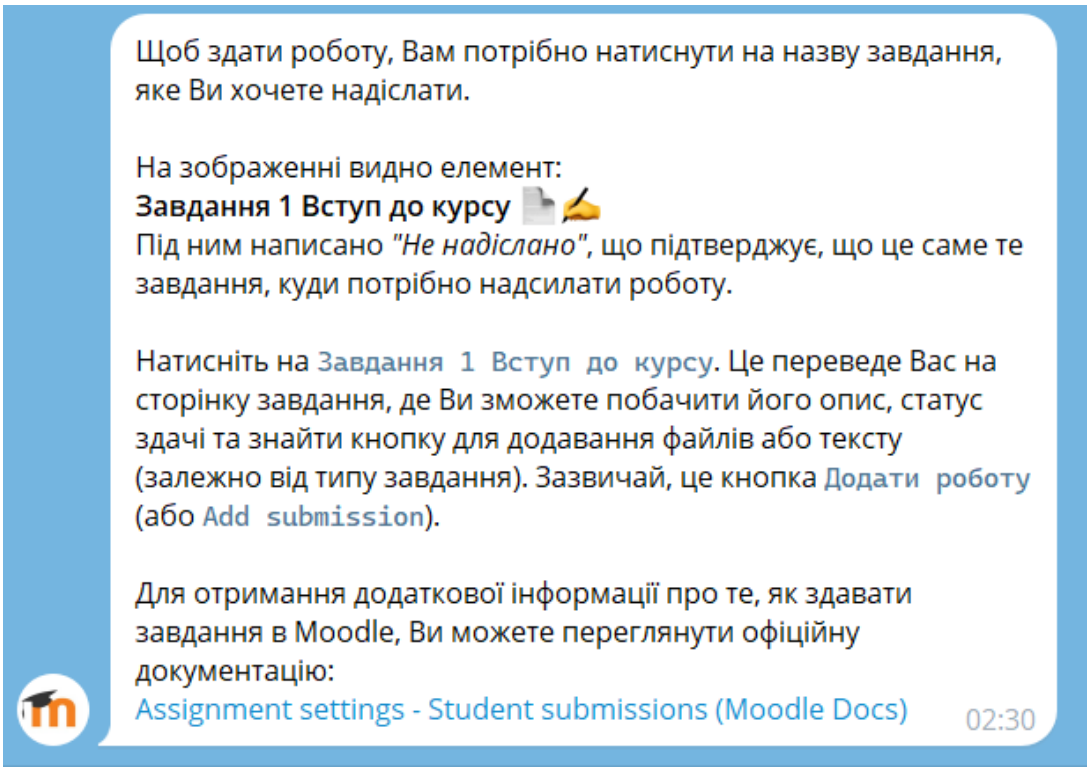


Рисунок 3.4 – Результат обробки запиту із зображенням

Тест-кейс 4: Обробка запиту не по темі (негативний сценарій):

– дія: користувач надсилає повідомлення: «Яка сьогодні погода?»
– очікуваний результат: бот інформує, що він є спеціалізованим помічником по Moodle і не може відповісти на це питання;
– фактичний результат: (наведено на рисунку 3.5);
– аналіз: система коректно дотримується інструкцій, заданих у промпті, та не виходить за межі своєї предметної області;
– статус: успішно.

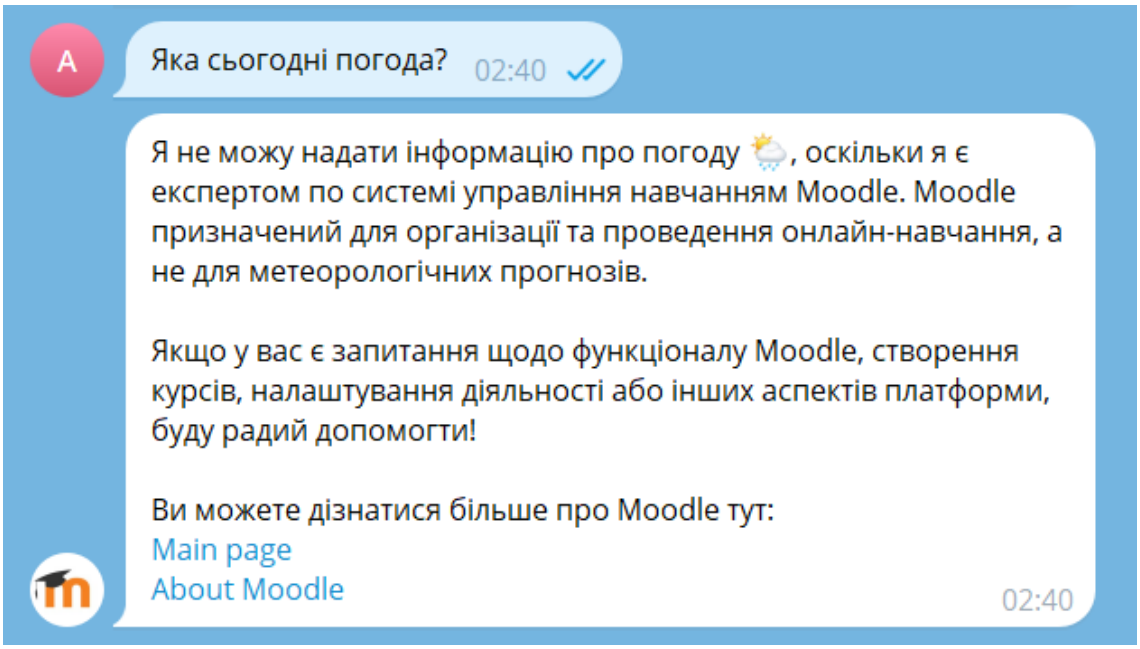


Рисунок 3.5 – Результат обробки запиту не по темі

Проведене тестування підтвердило, що всі основні функціональні вимоги, поставлені до системи, були реалізовані успішно.

3.3 Аналіз результатів та можливі шляхи покращення

Проведене тестування підтвердило, що розроблений чат-бот є працездатним програмним продуктом, який успішно виконує всі поставлені функціональні вимоги. Система коректно обробляє текстові, голосові та

графічні запити, надає релевантні відповіді українською мовою та дотримується заданих правил поведінки і форматування. Таким чином, основна мета кваліфікаційної роботи – створення прототипу для автоматизації служби підтримки – була досягнута.

Разом з тим, у ході тестування та аналізу було виявлено низку аспектів, які можуть бути покращені в подальших версіях продукту:

- час відповіді: хоча час відповіді в більшості випадків є прийнятним, обробка складних запитів із медіафайлами іноді може займати до 20 секунд. Це пов'язано з послідовними операціями завантаження файлів та запиту до зовнішнього API. Можливим шляхом оптимізації є використання більш швидких версій моделей (якщо такі з'являться) або паралелізація певних операцій;
- «галюцинації» моделі: як і будь-яка LLM, Gemini іноді може генерувати фактично неточну інформацію або «вигадувати» деталі, яких немає в документації. Хоча промпт-інжиніринг мінімізує цей ризик, повністю усунути його на даному етапі розвитку технології неможливо;
- обмежений контекст діалогу: поточна реалізація обробляє кожен запит ізольовано. Бот не «пам'ятає» попередні повідомлення користувача в рамках однієї сесії.

Можливі шляхи подальшого розвитку проєкту:

- впровадження історії діалогів: інтеграція механізму збереження контексту розмови (наприклад, за допомогою бази даних Redis), що дозволить боту вести більш складні та послідовні діалоги;
- розширення бази знань: підключення додаткових джерел інформації, таких як форуми спільноти Moodle або внутрішні бази знань конкретного навчального закладу;
- додавання інтерактивних елементів: використання вбудованих клавіатур (inline keyboards) для надання користувачам варіантів дій

або уточнення їхніх запитів, що зробить взаємодію більш структурованою;

– персоналізація: інтеграція з базою даних Moodle для надання персоналізованих відповідей (наприклад, «Ваш дедлайн по курсу 'Програмування' – завтра»).

Незважаючи на зазначені аспекти, розроблений прототип є міцним фундаментом для створення повноцінного комерційного продукту.

3.4 Висновки до розділу 3

У третьому розділі кваліфікаційної роботи було проведено експериментальне дослідження розробленого програмного прототипу інтелектуального чат-бота (з назвою Moodle Doc Bot).

Було розроблено методику та детальний план тестування, що охоплював перевірку всіх основних функціональних вимог системи. На основі цього плану було проведено серію тестів, що включали відправку текстових, голосових та графічних запитів, а також запитів, що не стосуються предметної області.

Результати тестування, задокументовані у вигляді скріншотів реальних діалогів, підтвердили повну працездатність системи. Було продемонстровано, що чат-бот успішно обробляє всі типи вхідних даних, надає релевантні та коректно відформатовані відповіді, а також адекватно реагує на непередбачувані запити. Аналіз результатів дозволив не тільки підтвердити відповідність системи початковим вимогам, але й визначити потенційні напрями для її подальшого вдосконалення, такі як впровадження історії діалогів та розширення бази знань.

Таким чином, етап тестування успішно завершено, а отримані результати доводять, що мета кваліфікаційної роботи досягнута, а створений прототип є життєздатним та ефективним рішенням для автоматизації служби підтримки.

ВИСНОВКИ

У ході виконання даної кваліфікаційної роботи було успішно досягнуто поставленої мети – розроблено та реалізовано програмний прототип інтелектуального чат-бота для месенджера Telegram (з назвою Moodle Doc Bot), призначеного для автоматизації служби підтримки користувачів системи управління навчанням Moodle

У першому, теоретичному, розділі було проведено комплексний аналіз предметної області. Досліджено проблематику традиційних служб підтримки, проаналізовано еволюцію та класифікацію чат-ботів. На основі порівняльного аналізу сучасних технологій штучного інтелекту та платформ розгортання було обґрунтовано вибір технологічного стеку: модель Google Gemini 2.5 Flash як ядро ШІ, месенджер Telegram як канал комунікації, платформа Railway.app для розгортання, та мова Python з фреймворком aiogram як інструмент розробки.

У другому, практичному, розділі було детально описано процес створення системи. Сформульовано функціональні та нефункціональні вимоги, спроєктовано архітектуру взаємодії компонентів, що візуалізовано за допомогою UML-діаграм. Розглянуто ключові аспекти програмної реалізації, включаючи обробку текстових, голосових та графічних повідомлень. Особливу увагу приділено системі промпт-інжинірингу, що дозволило тонко налаштувати поведінку та формат відповідей мовної моделі. На завершення описано процес розгортання застосунку в хмарному середовищі для забезпечення його безперебійної роботи.

У третьому розділі було проведено тестування розробленого прототипу. За допомогою набору тест-кейсів було підтверджено, що система відповідає всім заявленим функціональним вимогам, коректно обробляє різні типи запитів та надає релевантні відповіді. Проаналізовано результати тестування,

визначено сильні сторони реалізації та окреслено можливі напрями для подальшого вдосконалення продукту.

Таким чином, у рамках кваліфікаційної роботи було створено повноцінний, працездатний програмний прототип, що демонструє високий потенціал використання сучасних мультимодальних мовних моделей для автоматизації та покращення якості сервісних служб. Отримані результати мають практичне значення і можуть слугувати основою для подальших наукових досліджень та розробки комерційних продуктів.

ПЕРЕЛІК ПОСИЛАНЬ

1. Jurafsky D., Martin J. H. Speech and Language Processing. 3rd Edition. Prentice Hall, 2024. 585 p.

2. Gemini API docs. URL : https://ai.google.dev/gemini-api/docs (дата звернення : 15.04.2025).

3. Moodle 5.0 Documentation. Main Page. URL : https://docs.moodle.org/500/en/Main_page (дата звернення : 06.03.2025).

4. Telegram Bot API. URL : https://core.telegram.org/bots/api. (дата звернення : 14.04.2025).

5. aiogram 3 Documentation. URL : https://docs.aiogram.dev/uk-ua/dev-3.x/ (дата звернення : 14.04.2025).

6. Python 3.12.10 documentation. URL : https://docs.python.org/3.12/index.html (дата звернення : 05.03.2025).

7. Railway Documentation. URL : https://docs.railway.com/ (дата звернення : 08.05.2025).

8. Prompt Engineering Guide. URL : https://www.promptingguide.ai/ (дата звернення : 08.03.2025).

9. Lilian Weng. Prompt Engineering. URL : https://lilianweng.github.io/posts/2023-03-15-prompt-engineering/ (дата звернення : 09.03.2025).

Декларація
академічної доброчесності
здобувача ступеня вищої освіти ЗНУ

Я, Ковердюк Анастасія Олександрівна

студентка 4 курсу, денної форми навчання, математичного факультету, спеціальності 122 комп'ютері науки, адреса електронної пошти ccooshane1@gmail.com, – підтверджую, що написана мною кваліфікаційна робота бакалавра на тему «Реалізація чат-бота для автоматизації служби підтримки користувачів» відповідає вимогам академічної доброчесності та не містить порушень, що визначені у ст. 42 Закону України «Про освіту», зі змістом яких ознайомлена;

– заявляю, що надана мною для перевірки електронна версія роботи є ідентичною її друкованій версії;

– згодна на перевірку моєї роботи на відповідність критеріям академічної доброчесності у будь-який спосіб, у тому числі за допомогою інтернет-системи, а також на архівування моєї роботи в базі даних цієї системи.

Студент

		Ковердюк А.О.
(дата)	(підпис)	(прізвище, ініціали)

Науковий керівник

		Добровольський Г.А.
(дата)	(підпис)	(прізвище, ініціали)