Міністерство освіти і науки України
Тернопільський національний технічний університет імені Івана Пулюя

Факультет комп'ютерно-інформаційних систем і програмної інженерії
(повна назва факультету)
Кафедра програмної інженерії
(повна назва кафедри)

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття освітнього ступеня

бакалавр
(назва освітнього ступеня)

на тему: Розробка інтелектуального чат-бота у Telegram з використанням передових технологій обробки природної мови та штучного інтелекту

Виконав: студент IV курсу, групи СП-42
спеціальності 121 Інженерія програмного забезпечення
(шифр і назва спеціальності)

Мазурчак Д. С.
(підпис) (прізвище та ініціали)

Керівник Цебрій О. Р.
(підпис) (прізвище та ініціали)

Нормоконтроль Цебрій О. Р.
(підпис) (прізвище та ініціали)

Завідувач кафедри Петрик М. Р.
(підпис) (прізвище та ініціали)

Рецензент
(підпис) (прізвище та ініціали)

Тернопіль
2024

Міністерство освіти і науки України
Тернопільський національний технічний університет імені Івана Пулюя

Факультет Комп'ютерно інформаційних систем і програмної інженерії
(повна назва факультету)

Кафедра Програмної інженерії
(повна назва кафедри)

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Петрик М. Р
(підпис) (прізвище та ініціали)
«__» _____ 2024 р.

З А В Д А Н Н Я
НА КВАЛІФІКАЦІЙНУ РОБОТУ

на здобуття освітнього ступеня бакалавр
(назва освітнього ступеня)

за спеціальністю 121 Інженерія програмного забезпечення
(шифр і назва спеціальності)

студенту Мазурчак Дмитро Сергійович
(прізвище, ім'я, по батькові)

1. Тема роботи Розробка інтелектуального чат-бота у Telegram з використанням передових технологій обробки природної мови та штучного інтелекту.

Керівник роботи к.т.н., доц. Каф ПІ Цебрій О. Р.
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

Затверджені наказом ректора від «___» _____ 2024 року № _____

2. Термін подання студентом завершеної роботи _____

3. Вихідні дані до роботи _____

4. Зміст роботи (перелік питань, які потрібно розробити):

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень, слайдів):

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Безпека життєдіяльності, основи охорони праці			

7. Дата видачі завдання _____

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів роботи	Термін виконання етапів роботи	Примітка

Студент _____ Мазурчак Д. С.
(підпис) (прізвище та ініціали)

Керівник роботи _____ Цебрій О. Р.
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Кваліфікаційна робота освітнього рівня «Бакалавр» // Мазурчак Дмитро Сергійович // Тернопільський національний технічний університет імені Івана Пулюя, факультет комп'ютерно інформаційних систем і програмної інженерії, кафедра програмної інженерії, група СП-42 // Тернопіль, 2024 // С. – 71, рис. – 28, табл. – 1, кресл. – 0, додат. – 2, бібліогр. – 19.

Ключові слова: чат-бот, Telegram, обробка природної мови, штучний інтелект, Python, Visual Studio Code, Git, aiogram, asyncio, Google Cloud Platform, Cloud Function.

Кваліфікаційна робота досліджує процеси аналізу, проєктування, розробку та тестування інтелектуального чат-бота для платформи Telegram з використанням передових технологій обробки природної мови та штучного інтелекту.

В першому розділі кваліфікаційної роботи досліджено предметну область, конкурентні рішення, проаналізовано вимоги і засоби та інструменти розробки. Розділ дозволяє ознайомитись з темою та актуальністю проблематики цієї теми.

В другому розділі кваліфікаційної роботи описано процеси вибору архітектури рішення, проєктування та програмної реалізації. Розділ дозволяє ознайомитись з процесами безпосередньої розробки на основі даних із першого розділу.

В третьому розділі кваліфікаційної роботи описано тестування та верифікацію розробленого програмного рішення. Розділ дозволяє ознайомитись з інструментами та підходами до тестування і сформулювати кінцеву оцінку задовільності виконаної роботи.

ANNOTATION

Qualification work of the educational level "Bachelor" / Mazurchak Dmytro Serhiiovych // Ternopil National Technical University named after Ivan Pului, Faculty of Computer Information Systems and Software Engineering, Department of Software Engineering, group SP-42 // Ternopil, 2024 // P. - 71, Fig. - 28, Table - 1, Drawings - 0, Supplementary - 2, Bibliography - 19.

Keywords: chatbot, Telegram, natural language processing, artificial intelligence, Python, Visual Studio Code, Git, aiogram, asyncio, Google Cloud Platform, Cloud Function.

The thesis explores the processes of analyzing, designing, developing, and testing an intelligent chatbot for the Telegram platform using advanced natural language processing and artificial intelligence technologies.

The first chapter of the qualification work explores the subject area, competitive solutions, analyzes requirements, and development tools and techniques. The section allows you to familiarize yourself with the topic and the relevance of the issues of this topic.

The second section of the qualification work describes the processes of choosing a solution architecture, design, and program implementation. The section allows you to familiarize yourself with the processes of direct development based on the data from the first section.

The third section of the qualification paper describes the testing and verification of the developed software solution. The section allows you to familiarize yourself with the tools and approaches to testing and formulate a final assessment of the satisfactory performance of the work performed.

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

Telegram – месенджер з відкритим API, що дозволяє розробникам створювати ботів та інтеграції.

Чат-бот – це програма, яка імітує спілкування з користувачами через текстові або голосові повідомлення.

Обробка природної мови (NLP, Natural Language Processing) – це галузь штучного інтелекту, що займається взаємодією між комп'ютерами та людськими (природними) мовами.

Штучний інтелект (AI, Artificial Intelligence) – це галузь комп'ютерної науки, яка спрямована на створення інтелектуальних агентів, що здатні виконувати завдання, які зазвичай потребують людського інтелекту.

Python – це високорівнева мова програмування загального призначення з динамічною типізацією.

Visual Studio Code (VS Code) – це безкоштовний редактор коду з відкритим вихідним кодом, розроблений Microsoft, який підтримує різні мови програмування.

Git – це розподілена система керування версіями, що дозволяє зберігати історію змін коду.

aiogram – це асинхронний Python фреймворк для створення Telegram-ботів.

asyncio – це стандартна бібліотека Python для написання асинхронного коду.

Google Cloud Platform (GCP) – це набір хмарних сервісів, що пропонуються Google для обчислень, зберігання даних та розробки додатків.

Cloud Function – це безсерверний обчислювальний сервіс на платформі Google Cloud, який дозволяє виконувати код у відповідь на події без необхідності керування серверами.

JSON (JavaScript Object Notation) – це текстовий формат обміну даними, що є зрозумілим для людини і легким для обробки машиною.

API (Application Programming Interface) – це набір правил і протоколів для взаємодії між різними програмними додатками.

HTTP (HyperText Transfer Protocol) – це протокол передачі гіпертексту, що використовується для передачі даних у Всесвітній павутині (WWW).

Webhook – це спосіб взаємодії веб-додатків у реальному часі шляхом надсилання HTTP-запитів при певних подіях.

Token – це ключ, який використовується для автентифікації та надання доступу до API.

ЗМІСТ

ВСТУП

У сучасному світі, що швидко розвивається, роль інтелектуальних чат-ботів у комунікації з користувачами стає все більш суттєвою. Ці технології використовують передові методи обробки природної мови та штучного інтелекту для надання користувачам зручного та ефективного способу взаємодії з різноманітними сервісами та послугами.

Актуальність даної теми полягає у необхідності вдосконалення систем взаємодії людини із комп'ютером, а саме, у вдосконаленні інтерфейсів, які використовують природну мову для комунікації. Розробка інтелектуального чат-бота у Telegram з використанням передових технологій обробки природної мови та штучного інтелекту стане вагомим внеском у цю сферу та дозволить дослідити можливості цих технологій у практичному застосуванні.

Метою дослідження є розробка та імплементація інтелектуального чат-бота у Telegram, який забезпечить користувачам інтуїтивно зрозуміле та ефективне спілкування з системою за допомогою природної мови. Для досягнення цієї мети необхідно вирішити такі завдання:

- Проаналізувати предметну область, зокрема розвиток чат-ботів та використання передових технологій обробки природної мови та штучного інтелекту.
- Обрати та обґрунтувати найбільш підходящі технології для розробки інтелектуального чат-бота.
- Розробити архітектуру системи та базу даних, а також реалізувати алгоритми обробки природної мови.
- Інтегрувати чат-бота з Telegram API та розробити користувацький інтерфейс.
- Провести тестування розробленої системи та впровадити її для практичного використання.

Об'єктом дослідження є процес розробки інтелектуального чат-бота у Telegram, а предметом - передові технології обробки природної мови та штучного інтелекту, використані для створення цієї системи.

Для досягнення поставленої мети та вирішення завдань дослідження будуть застосовані методи аналізу літературних джерел, аналізу програмного забезпечення, проектування систем, програмування, тестування та аналізу результатів.

Структура роботи включатиме аналіз предметної області та постановку задачі, проєктування та розробку чат-бота, тестування та впровадження системи, а також аналіз аспектів безпеки та охорони праці, що відносяться до розробки програмного забезпечення. Висновки роботи будуть містити узагальнення отриманих результатів, досягнення поставлених цілей та рекомендації для подальших досліджень у цій галузі.

РОЗДІЛ 1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ЗАДАЧІ

1.1 Постановка задачі

Месенджери, як засіб комунікації, виникли на зорі розвитку інтернету та цифрових технологій. Історія месенджерів починається з перших текстових повідомлень, відправлених через комп'ютерні мережі у 1960-х роках. Ці ранні форми обміну повідомленнями були обмежені технологічними можливостями того часу, але заклали основу для подальшого розвитку. Однією з перших систем, що отримала широку популярність, була система електронної пошти, розроблена у 1970-х роках.

Після цього у 1980-х роках з’явились перші програми для обміну миттєвими повідомленнями, такі як ICQ та AIM, які значно спростили та прискорили процес комунікації. З розвитком інтернету та мобільних технологій у 2000-х роках, месенджери еволюціонували у більш складні системи з розширеним функціоналом. З’явилися такі популярні месенджери, як WhatsApp, Viber, Telegram. Останій в свою чергу, став одним з найпопулярніших месенджерів завдяки своїй відкритій платформі, яка дозволяє розробникам створювати ботів та інші інтеграції (рис. 1.1).

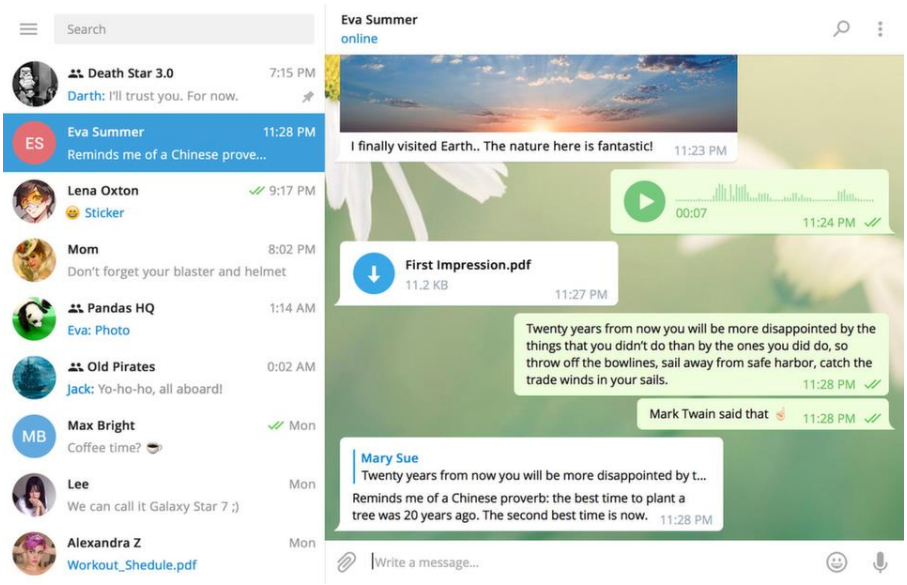

Рисунок 1.1 – Інтерфейс месенджера Telegram

Паралельно з розвитком месенджерів, активно розвивалася галузь штучного інтелекту (ШІ) [1]. Основи штучного інтелекту були закладені ще у 1950-х роках, коли перші вчені та інженери почали досліджувати можливості створення машин, які могли б імітувати людський інтелект. Одним з піонерів у цій галузі був Алан Тюрінг, який запропонував так званий "Тюрінг-тест" для визначення здатності машини до мислення (рис. 1.2).

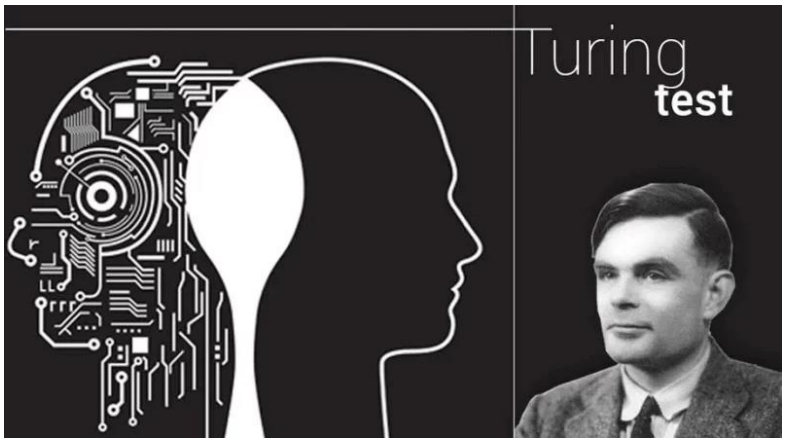

Рисунок 1.2 – Портрет Алана Тюрінга

Протягом наступних десятиліть, розвиток обчислювальної техніки та алгоритмів дозволив створювати все більш складні системи штучного інтелекту. У 1990-х та 2000-х роках значного розвитку набули методи машинного навчання, зокрема, нейронні мережі та алгоритми глибокого навчання. Ці методи дозволили значно покращити здатність комп'ютерів до розпізнавання образів, обробки природної мови та прийняття рішень. Наприклад, алгоритми обробки природної мови (NLP) [4] дозволили створювати системи, які можуть розуміти та генерувати текстові повідомлення на природній мові, що стало основою для розвитку інтелектуальних чат-ботів.

Чат-боти, як застосування штучного інтелекту в месенджерах, почали набирати популярність у 2010-х роках. Це програмні агенти, які можуть взаємодіяти з користувачами через текстові або голосові повідомлення, імітуючи спілкування з реальною людиною. Використання методів обробки природної мови

дозволило значно підвищити якість та ефективність взаємодії з чат-ботами. Сучасні чат-боти можуть виконувати різноманітні завдання, від надання інформації та технічної підтримки до здійснення покупок та бронювання послуг.

Розвиток месенджерів та штучного інтелекту привів до необхідності створення інтегрованих систем, які поєднують можливості обох технологій. Одним з таких прикладів є інтелектуальні чат-боти у месенджері Telegram. Telegram надає розробникам можливість створювати ботів за допомогою свого відкритого API, що дозволяє інтегрувати функції штучного інтелекту, такі як обробка природної мови та машинне навчання, для створення більш інтуїтивних та ефективних інструментів взаємодії.

Завданням даного дослідження є розробка інтелектуального чат-бота для Telegram з використанням передових технологій обробки природної мови та штучного інтелекту. Це включає аналіз та вибір найбільш ефективних технологій та методів, проектування архітектури системи, розробку алгоритмів та інтеграцію з Telegram API [2]. Крім того, важливою частиною задачі є тестування розробленої системи та її впровадження для реального використання.

Таким чином, основними аспектами постановки задачі є дослідження історичного розвитку месенджерів та штучного інтелекту, аналіз сучасних технологій та методів обробки природної мови, а також розробка та імплементація інтелектуального чат-бота для Telegram, який забезпечить ефективну та інтуїтивну взаємодію з користувачами.

1.2 Аналіз та актуальність предметної області

З появою інтернету та цифрових технологій месенджери [3] стали невід’ємною частиною нашого повсякденного життя. Спрощуючи комунікацію, вони забезпечують швидке та зручне спілкування між користувачами з усього світу.

Однак із зростанням популярності месенджерів з'явилась потреба у розвитку інтелектуальних інструментів, які забезпечували більш ефективну та продуктивну комунікацію. Тут на сцену виходять інтелектуальні чат-боти.

За останні кілька років спостерігається стрімкий розвиток месенджерів та збільшення їх функціональності. Крім базових функцій обміну текстовими повідомленнями, сучасні месенджери дозволяють користувачам обмінюватись голосовими та відео дзвінками, створювати групові чати, відправляти геолокацію та медіафайли, використовувати емодзі та стікери та навіть здійснювати покупки через платформу месенджера (рис. 1.3).

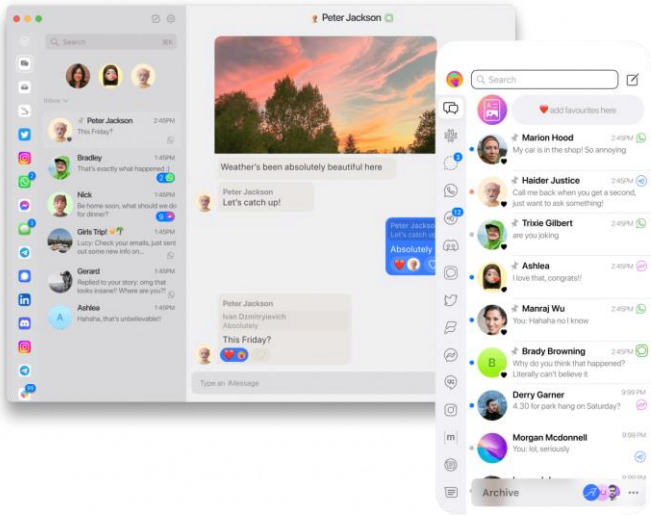

Рисунок 1.3 – Функціональні можливості сучасних месенджерів

Зростання популярності месенджерів призвело до появи широкого спектру застосувань, від особистого спілкування до комунікації у бізнесі та навіть урядових установах. Особливо велике значення має комунікація в онлайн-торгівлі, де месенджери часто використовуються для обслуговування клієнтів та проведення транзакцій.

У контексті постійного зростання обсягу інформації та потреби у її швидкому та зручному обробленні, використання інтелектуальних чат-ботів набуває великого значення. Інтелектуальні чат-боти можуть автоматизувати багато процесів, що раніше вимагали людського втручання, такі як відповіді на запитання користувачів,

надання інформації про товари та послуги, бронювання або покупка, розв'язання проблем клієнтів та багато інших.

Застосування інтелектуальних чат-ботів у бізнесі також може покращити взаємодію з клієнтами, забезпечуючи швидку та персоналізовану підтримку (рис. 1.4). Крім того, вони можуть забезпечити значні економічні вигоди, зменшуючи витрати на підтримку клієнтів та підвищуючи ефективність роботи персоналу.

Рисунок 1.4 – Використання інтелектуальних чат-ботів у бізнесі

Отже, розвиток месенджерів та штучного інтелекту, а також зростаюча потреба у швидкій та ефективній комунікації роблять актуальним використання інтелектуальних чат-ботів. Вони є потужним інструментом для автоматизації комунікації та покращення взаємодії з користувачами, який має широкі застосування в різних сферах життя та бізнесу.

1.3 Огляд конкурентних рішень

На ринку месенджерів [3] та інтелектуальних чат-ботів існує велика кількість конкурентних рішень, кожне з яких має свої унікальні характеристики та переваги. Для детального аналізу розглянемо деякі з найпопулярніших платформ та їхніх чат-ботів, таких як Facebook Messenger, WhatsApp, Viber та Slack.

Facebook Messenger є однією з найбільш популярних платформ для обміну миттєвими повідомленнями. Завдяки інтеграції з Facebook, він має величезну базу користувачів і пропонує широкий спектр функцій для розробки чат-ботів. Messenger API дозволяє створювати ботів, які можуть здійснювати автоматизовану підтримку клієнтів, відповідати на запитання користувачів, проводити опитування та навіть приймати платежі (рис. 1.5).

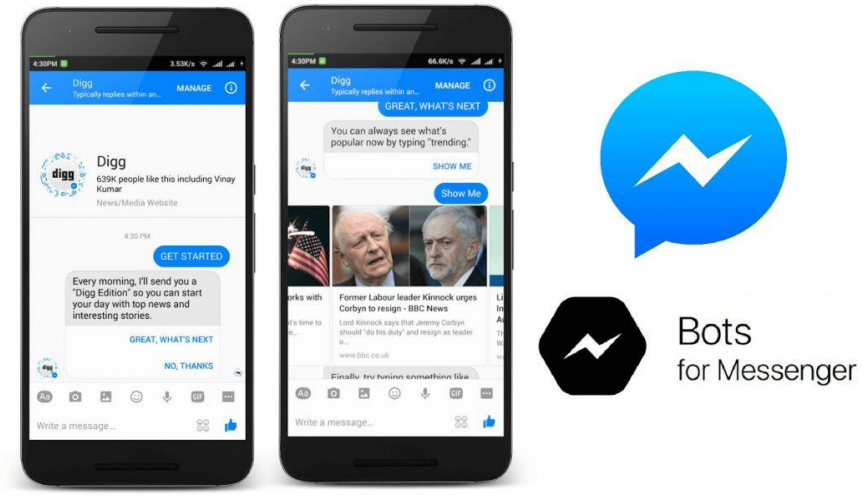

Рисунок 1.5 – Інтерфейс Facebook Messenger із ботами

Чат-боти у Facebook Messenger використовують методи обробки природної мови для розуміння запитів користувачів та надання відповідних відповідей. Крім того, платформа підтримує мультимедійні повідомлення, що дозволяє ботам надсилати користувачам зображення, відео та аудіофайли.

WhatsApp, як і Facebook Messenger, є одним з найпопулярніших месенджерів у світі. Він також пропонує можливості для створення чат-ботів через WhatsApp Business API. Ця платформа дозволяє підприємствам автоматизувати обробку запитів клієнтів, надсилати повідомлення про статус замовлень, надавати технічну підтримку та інші послуги.

Особливістю WhatsApp є його висока популярність у країнах, що розвиваються, а також висока ступінь захищеності повідомлень, завдяки наскрізному шифруванню. Чат-боти в WhatsApp можуть взаємодіяти з користувачами через текстові повідомлення, що робить їх зручними для широкого кола користувачів.

Viber є ще одним популярним месенджером, який пропонує можливості для створення чат-ботів. Viber Bot API дозволяє розробникам створювати ботів, які можуть здійснювати різноманітні функції, від автоматичної відповіді на запитання користувачів до надання інформації про продукти та послуги (рис. 1.6).

Рисунок 1.6 – Інтерфейс Viber з чат-ботами

Viber підтримує текстові, голосові та відео повідомлення, а також мультимедійний контент. Це дозволяє ботам надавати користувачам більш інтерактивний та зручний досвід взаємодії.

Slack є популярною платформою для командної комунікації та співпраці, яка також підтримує інтеграцію чат-ботів. Slack API дозволяє розробникам створювати ботів, які можуть автоматизувати різні завдання, такі як планування зустрічей, надсилання повідомлень про оновлення проектів та багато іншого.

Чат-боти у Slack використовуються переважно у корпоративному середовищі, де вони можуть значно підвищити продуктивність та ефективність роботи команд. Вони інтегруються з іншими інструментами та сервісами, що дозволяє створювати комплексні рішення для автоматизації бізнес-процесів.

Аналіз конкурентних рішень показує, що кожна з платформ має свої унікальні переваги та недоліки. Вибір платформи для розробки інтелектуального чат-бота залежить від конкретних вимог та цілей проекту. Telegram, завдяки своїй відкритій платформі та широким можливостям інтеграції, є привабливим варіантом для розробників, які прагнуть створювати складні та функціональні чат-боти.

1.4 Огляд сучасних технологій обробки природної мови

Обробка природної мови (NLP) [4] та штучний інтелект (AI) є двома ключовими технологіями, які забезпечують функціонування сучасних чат-ботів. У цьому розділі ми розглянемо основні концепції, інструменти та платформи, які використовуються для створення інтелектуальних систем, здатних розуміти та генерувати людську мову.

NLP займається взаємодією між комп'ютерами та людською мовою, дозволяючи машинам розуміти, інтерпретувати та відповідати на текстові та голосові запити користувачів. Основні завдання NLP включають розпізнавання мовлення, морфологічний аналіз, синтаксичний аналіз, семантичний аналіз та генерацію природної мови (рис. 1.7).

Одним із найпопулярніших інструментів для NLP є бібліотека NLTK (Natural Language Toolkit) для Python, яка містить численні інструменти та ресурси для обробки тексту. Іншим відомим інструментом є бібліотека SpaCy, яка забезпечує швидкий та ефективний аналіз тексту.

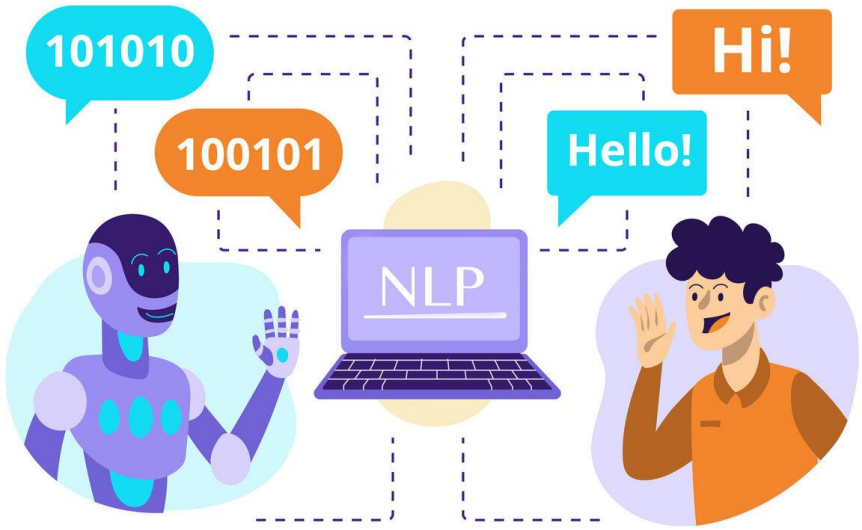

Рисунок 1.7 – Приклад використання NLP для аналізу тексту

AI займається розробкою систем, які можуть виконувати завдання, що зазвичай потребують людського інтелекту, такі як розпізнавання образів, прийняття рішень та навчання. AI тісно пов'язаний з машинним навчанням (ML), яке використовує алгоритми для аналізу даних, навчання моделей та прогнозування.

Одним із важливих аспектів AI є використання нейронних мереж, зокрема глибоких нейронних мереж, для побудови складних моделей обробки даних. Популярні фреймворки для машинного навчання - це TensorFlow та PyTorch, які надають потужні інструменти для розробки та тренування моделей AI.

Інтеграція NLP та AI в чат-боти дозволяє створювати інтелектуальні системи, які можуть розуміти та реагувати на запити користувачів у реальному часі. Однією з найбільш популярних архітектур для таких моделей є використання трансформерів, таких як BERT (Bidirectional Encoder Representations from Transformers) та GPT (Generative Pre-trained Transformer).

Моделі трансформерів здатні розуміти контекст та значення слів у тексті, що робить їх надзвичайно ефективними для задач NLP. Наприклад, BERT використовується для роботи з текстовими даними, такими як класифікація тексту, розпізнавання сутностей та заповнення пропусків у тексті, тоді як GPT може виводити розумні та послідовні тексти на основі початкового запиту.

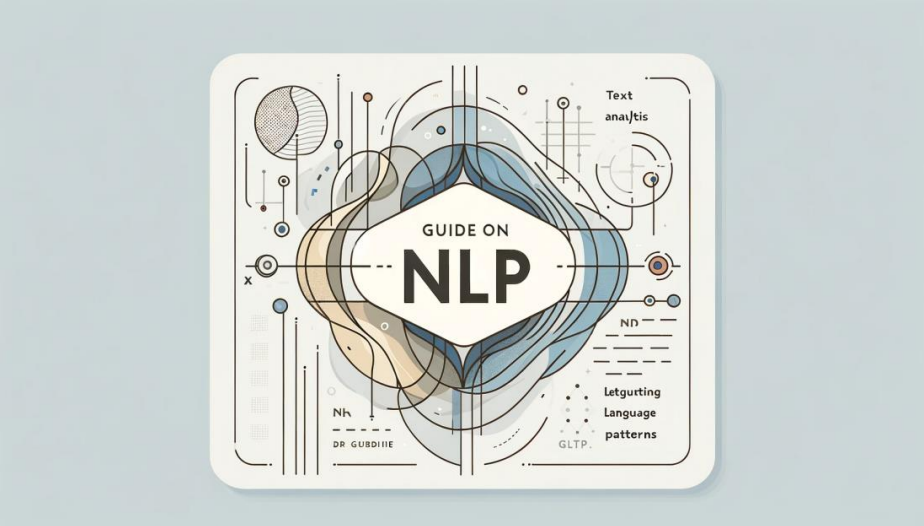

Рисунок 1.8 – Посібник з обробки природної мови NLP

Сучасні технології NLP та AI широко використовуються у різних сферах. Наприклад, в сфері обслуговування клієнтів інтелектуальні чат-боти забезпечують цілодобову підтримку користувачів, відповідаючи на запити та допомагаючи вирішувати проблеми. У медицині ці технології застосовуються для аналізу медичних записів та діагностики захворювань. У маркетингу вони використовуються для аналізу відгуків клієнтів та автоматизації комунікації з потенційними покупцями.

РОЗДІЛ 2. ПРОЄКТУВАННЯ ТА РОЗРОБКА ЧАТ-БОТА

2.1 Архітектура рішення

Архітектура рішення для розробки інтелектуального чат-бота у Telegram охоплює кілька ключових компонентів, включаючи серверну частину, обробку природної мови (NLP) [4], інтеграцію з Telegram API [2], базу даних та інтерфейс користувача. Ця архітектура забезпечує надійність, масштабованість та зручність у підтримці.

Основні компоненти архітектури

- Серверна частина. Серверна частина відповідає за обробку запитів від користувачів, взаємодію з зовнішніми сервісами та управління базою даних. Вибір технологій для серверної частини базується на Python, зокрема, фреймворках aiogram [9] та asyncio, які забезпечують асинхронну обробку запитів та високу продуктивність.
- Обробка природної мови (NLP). Для обробки природної мови використовуються сучасні бібліотеки та моделі NLP, такі як SpaCy або NLTK. Ці інструменти дозволяють аналізувати текст, витягувати ключові слова, розпізнавати наміри користувачів та генерувати відповідні відповіді.
- Інтеграція з Telegram API. Інтеграція з Telegram API дозволяє чат-боту взаємодіяти з користувачами через Telegram. Використовуючи Telegram Bot API, бот отримує повідомлення від користувачів, обробляє їх та відправляє відповіді. Фреймворк aiogram полегшує цю інтеграцію, забезпечуючи зручний інтерфейс для роботи з Telegram API.
- База даних. Для зберігання даних використовується реляційна база даних, наприклад, PostgreSQL. База даних зберігає інформацію про користувачів, історію повідомлень та інші необхідні дані. ORM (Object-Relational Mapping) бібліотеки, такі як SQLAlchemy, використовуються для взаємодії з базою даних, що спрощує управління даними.

- Інтерфейс користувача. Інтерфейс користувача представлений у вигляді текстових повідомлень у Telegram. Чат-бот надає користувачам можливість взаємодіяти з системою за допомогою команд, кнопок та інших елементів інтерфейсу, що забезпечує зручність використання.

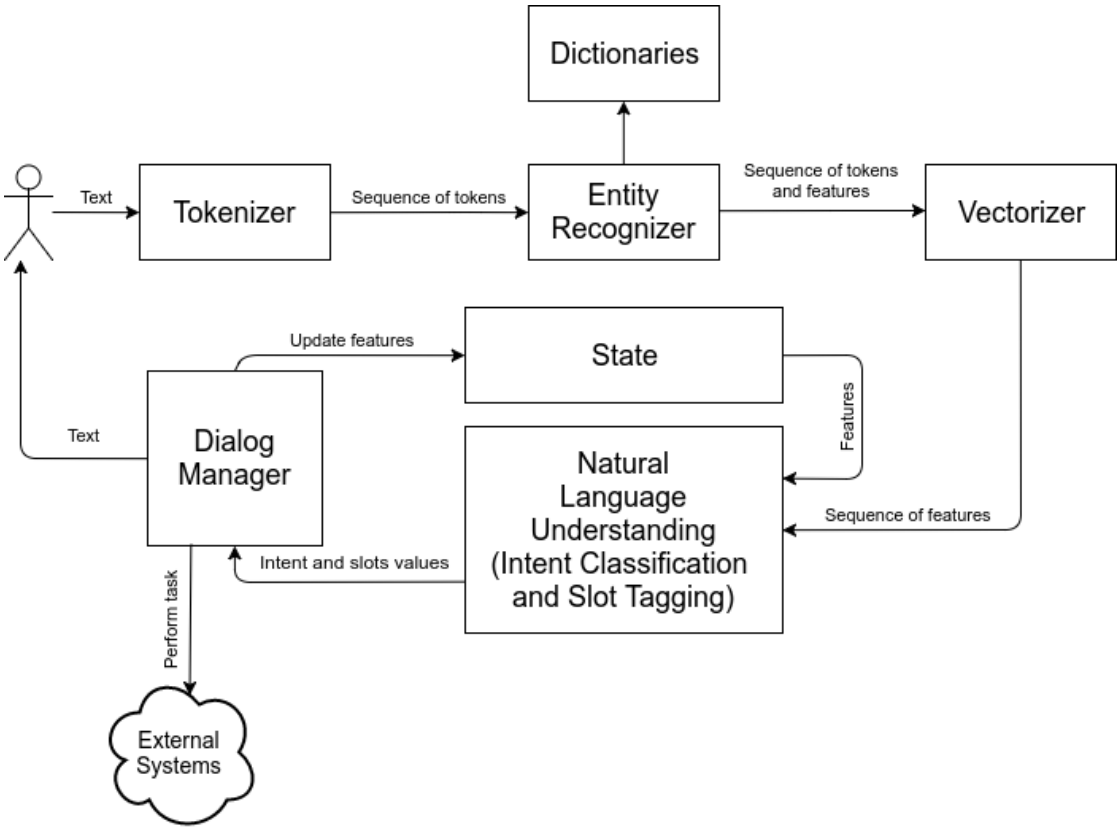

Рисунок 2.1 – Архітектура рішення чат-бота

Потік даних в системі

1. Отримання повідомлення. Користувач відправляє повідомлення чат-боту через Telegram. Повідомлення передається до серверної частини через Telegram API.
2. Обробка повідомлення. Серверна частина отримує повідомлення та передає його до модуля NLP для аналізу. NLP модуль витягує ключову інформацію та визначає намір користувача.

3. Генерація відповіді. На основі аналізу повідомлення та даних з бази даних генерується відповідь. Серверна частина формує текст відповіді та передає його назад до Telegram API.
4. Відправка відповіді. Telegram API відправляє сформовану відповідь користувачу. Користувач отримує повідомлення у своєму чаті з ботом.

Для забезпечення безпеки даних користувачів використовуються методи шифрування та автентифікації. Шифрування забезпечує захист даних при передачі між компонентами системи, а автентифікація гарантує, що доступ до даних мають лише авторизовані користувачі.

Масштабованість системи досягається через використання асинхронних процесів та мікросервісної архітектури. Це дозволяє легко розширювати систему, додаючи нові компоненти або збільшуючи потужність існуючих.

Архітектура рішення для інтелектуального чат-бота у Telegram забезпечує ефективну та надійну обробку запитів користувачів. Використання сучасних технологій обробки природної мови та інструментів для інтеграції з Telegram API дозволяє створювати функціональні та зручні у використанні системи. Завдяки добре продуманій архітектурі, чат-бот може забезпечити високу якість взаємодії з користувачами та бути легко масштабованим для майбутнього розвитку.

2.2 Вибір технологій

Вибір технологій для розробки інтелектуального чат-бота у Telegram був здійснений з урахуванням низки факторів, таких як продуктивність, зручність використання, підтримка сучасних методів обробки природної мови (NLP) [4] та інтеграція з Telegram API. У цьому розділі розглянемо деталі вибраних технологій та обґрунтуємо їх використання.

Python [5] був обраний в якості основної мови програмування для розробки чат-боту через його простоту, зручність в сприйнятті коду та багато доступних бібліотек для роботи з даними та обробки природної мови. Це сучасний асинхронний фреймворк для Python, який дозволяє створювати ботів для Telegram з високою продуктивністю. Aiogram [9] забезпечує простий та зрозумілий інтерфейс для взаємодії з Telegram API [2], підтримує асинхронну обробку запитів, що значно підвищує ефективність роботи бота.

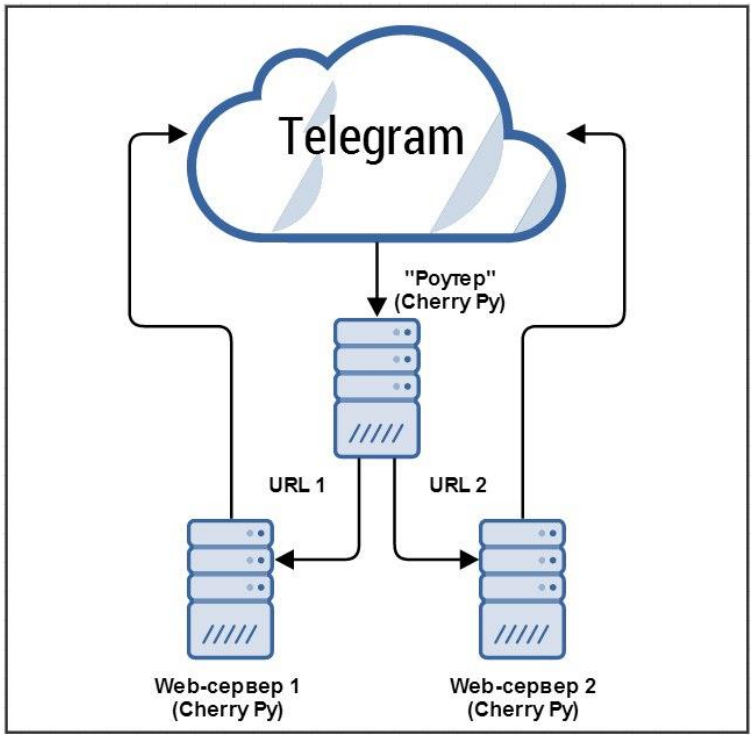

Рисунок 2.3 – Структура взаємодії бота з Telegram API за допомогою aiogram

Для управління асинхронними операціями було використано бібліотеку asyncio. Він дозволяє виконувати багато задач одночасно, що є дуже важливим для обробки великої кількості запитів у реальному часі. Asyncio використовується разом з aiogram для роботи з Telegram API в асинхронному середовищі. Asyncio надає зручний та високопродуктивний механізм для обробки асинхронних операцій.

Для зберігання даних чат-бота використовується вбудована база даних SQLite. Вона є легкою та не потребує окремого серверного середовища, що робить

її ідеальним вибором для невеликих та середніх проектів. SQLite [6] забезпечує швидкий доступ до даних та простоту налаштування.

Для виконання HTTP-запитів використовується бібліотека aiohttp, яка підтримує асинхронні запити. Це дозволяє ботам ефективно взаємодіяти з зовнішніми API та сервісами, не блокуючи основний потік виконання.

Бібліотека dotenv використовується для завантаження конфігураційних параметрів з файлу .env. Це дозволяє зберігати конфіденційні дані, такі як токени доступу та інші параметри налаштувань, у захищеному місці, що покращує безпеку проекту.

Visual Studio Code [7] був обраний в якості основного інструмента для розробки завдяки своїм потужними функціональними можливостями та підтримкою розширень для Python. VS Code надає зручне середовище для написання, налагодження та тестування коду, підтримуючи численні розширення для підвищення продуктивності.

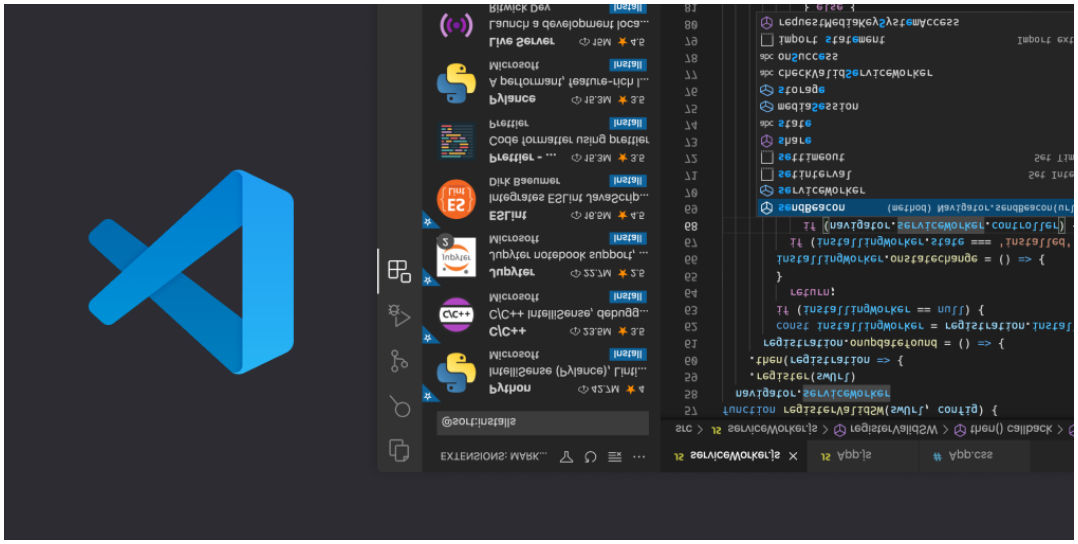

Рисунок 2.4 – Інтерфейс Visual Studio Code

Для управління версіями коду використовується система контролю версій Git. Git дозволяє відслідковувати зміни в коді, співпрацювати з іншими розробниками та керувати різними версіями проекту. Використання GitHub або іншої платформи для хостингу репозиторіїв забезпечує надійне зберігання та доступ до коду.

Для обробки повідомлень користувачів та генерації відповідей використовується локальний сервер Ollama з моделлю Llama2. Ця модель забезпечує високу якість генерації тексту та підтримку складних діалогових сценаріїв. Використання локального сервера дозволяє контролювати процес обробки даних та зменшує залежність від зовнішніх сервісів.

Вибір наведених технологій обґрунтований їхньою відповідністю вимогам проекту та можливістю забезпечити ефективну роботу чат-бота. Використання Python [5] дозволяє швидко розробляти та тестувати код, а асинхронні бібліотеки, такі як aiogram та asyncio, забезпечують високу продуктивність та масштабованість. SQLite [6] забезпечує просте та швидке зберігання даних, тоді як aiohttp дозволяє ефективно працювати з зовнішніми API. Visual Studio Code [7] та Git - інструменти програмування та контролю версій, які надають зручне середовище розробки та управління кодом, сприяючи покращенню продуктивності та якості роботи. Використання локального сервера Ollama з моделлю Llama2 дозволяє забезпечити високу якість обробки повідомлень та контролювати процес генерації відповідей.

Таким чином, вибрані технології забезпечують надійну та ефективну роботу чат-бота, що відповідає вимогам сучасних користувачів та стандартам розробки програмного забезпечення.

2.3 Створення Telegram-бота

Для створення бота я використав офіційний Telegram-бот під назвою BotFather. Перш за все, я відкрив Telegram-додаток і ввів у пошук @BotFather. Після цього я вибрав BotFather зі списку результатів і натиснув кнопку Start, щоб розпочати взаємодію.

Наступним кроком було створення нового бота. Я ввів команду /newbot, після чого BotFather попросив мене надати ім'я для бота. Я вирішив назвати бота

BrainChat. Далі потрібно було вибрати унікальне користувацьке ім'я, яке закінчується на bot. Після декількох спроб я вибрав ім'я BrainChat1_bot, яке виявилося доступним.

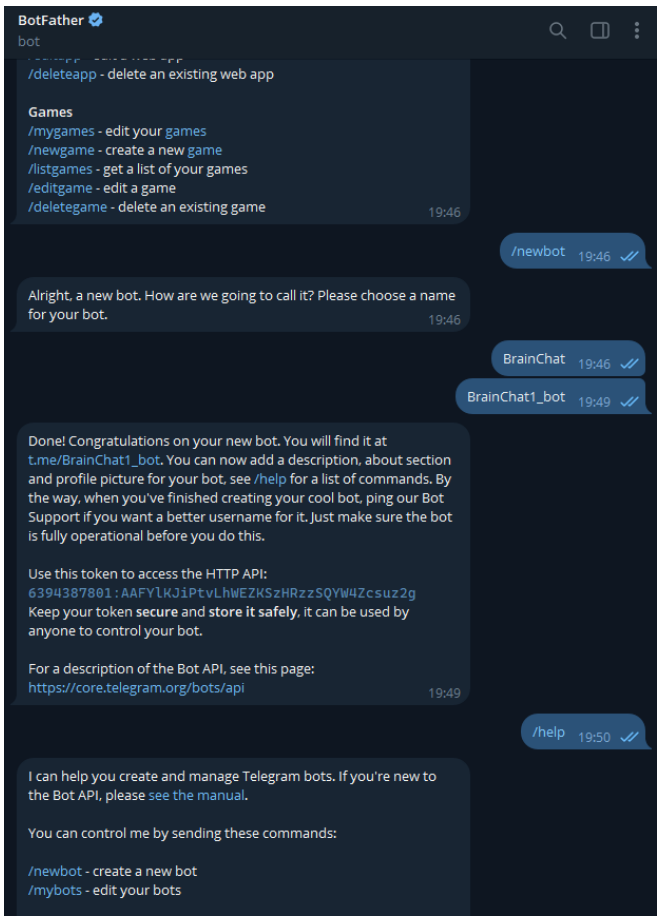

Рисунок 2.5 – Створення нового бота з BotFather

Після успішного створення бота BotFather надав мені токен доступу. Цей токен є унікальним ідентифікатором мого бота і необхідний для взаємодії з Telegram API [2]. Я зберіг цей токен у безпечному місці для подальшого використання.

Далі я налаштував бота за допомогою BotFather. Зокрема, я встановив опис бота, інформацію про нього за допомогою цих команд:

- /start - Початок роботи з ботом
- /newbot – Створення нового бота
- /help - Отримання допомоги

- /token – Отримання токена бота
- /setuserpic – Зміна зображення бота

Отримавши токен доступу, я інтегрував його у свій код для створення функціонального Telegram-бота. Я використав бібліотеку Aiogram [9] для створення асинхронного бота на Python. Нижче наведено приклад коду, який я використовував для інтеграції:

```
bot = Bot(token=token)
dp = Dispatcher()
builder = InlineKeyboardBuilder()
builder.row(
    types.InlineKeyboardButton(text="ℹ About", callback_data="info"),
    types.InlineKeyboardButton(text="⚙ Settings", callback_data="modelmanager"),
)
```

Рисунок 2.6 – Інтеграція токена

Процес створення та налаштування Telegram-бота за допомогою BotFather є простим і зручним. Використання цього офіційного бота дозволяє швидко створити базові параметри для нового бота та отримати токен доступу, необхідний для подальшої інтеграції з Telegram API [2]. Ці кроки є важливими для ефективного створення інтерактивного додатку, що забезпечує якісну взаємодію з користувачами.

2.4 Проєктування бази даних

Проєктування бази даних є ключовим етапом розробки інтелектуального чат-бота, оскільки забезпечує надійне зберігання та доступ до необхідної інформації. У цьому проекті використовується база даних SQLite3 [6], яка відома своєю легкістю та ефективністю для невеликих та середніх проектів. Основною таблицею бази даних є users, яка зберігає інформацію про користувачів та їхні взаємодії з ботом.

Основною таблицею бази даних є users, яка зберігає інформацію про користувачів та їхні взаємодії з ботом. Структура таблиці є наступною:

- id: Цілочисельний первинний ключ, що унікально ідентифікує кожного користувача.
- full_name: Строкове поле для зберігання повного імені користувача.
- input_data: Строкове поле для зберігання введених користувачем даних.
- output_data: Строкове поле для зберігання відповіді бота на введені дані.

```python
def log_to_db(user_id: int, full_name: str, input_data: str, output: str):
    db = sqlite3.connect('test.db')
    cursor = db.cursor()
    cursor.execute("""CREATE TABLE IF NOT EXISTS users(
        id INTEGER PRIMARY KEY,
        full_name TEXT,
        input_data TEXT,
        output_data TEXT
    )""")
    cursor.execute("INSERT INTO users (id, full_name, input_data, output_data) VALUES (?, ?, ?, ?)",
                   (user_id, str(full_name), str(input_data), str(output)))
    db.commit()
    db.close()
```

Рисунок 2.7 – Функція для збереження даних у базу даних

Таблиця users використовується для зберігання даних про взаємодії користувачів з ботом. Це включає ідентифікацію користувачів за допомогою унікального id, зберігання повного імені користувача для особистого звернення, логування введених користувачем даних (input_data), що може бути використано для аналізу або покращення роботи бота, та зберігання відповідей бота (output_data), що дозволяє відстежувати ефективність відповідей та проводити їх аналіз.

Для створення таблиці використовується наступний SQL-запит:

```sql
CREATE TABLE IF NOT EXISTS users(
    id INTEGER PRIMARY KEY,
    full_name TEXT,
    input_data TEXT,
    output_data TEXT
);
```

Рисунок 2.8 – SQL-запит для створення таблиці

Цей запит створює таблицю users, якщо вона не існує, з відповідними стовпцями. Основною операцією з базою даних є вставка нових записів, що здійснюється за допомогою наступного запиту:

```
INSERT INTO users (id, full_name, input_data, output_data) VALUES (?, ?, ?, ?);
```

Рисунок 2.9 – Запит для добавлення інформації про користувача

Даний запит додає новий запис у таблицю users, використовуючи надані значення для id, full_name, input_data та output_data. Використання SQLite [6] обґрунтоване декількома важливими факторами: легкість та простота, оскільки SQLite не потребує налаштування окремого серверного середовища, вбудованість, де база даних зберігається у вигляді одного файлу, що спрощує її інтеграцію та переносимість, швидкість, яка забезпечує швидкий доступ до даних, що є критичним для реального часу роботи чат-бота, та повна підтримка SQL, яка дозволяє використовувати потужні інструменти для маніпуляції даними.

Інтеграція бази даних з чат-ботом здійснюється за допомогою функції log_to_db, яка виконує підключення до бази даних, перевірку та створення таблиці users, якщо вона не існує, вставку нового запису з даними користувача та відповіддю бота, і закриття підключення до бази даних. Цей підхід забезпечує надійне зберігання даних про взаємодії користувачів з ботом, що дозволяє вести історію повідомлень та аналізувати ефективність роботи бота.

Проєктування бази даних є важливим кроком у розробці інтелектуального чат-бота. Використання SQLite забезпечує легкість, швидкість та простоту інтеграції, що є ключовими факторами для успішної реалізації проекту. Таблиця users забезпечує надійне зберігання даних про користувачів та їх взаємодії з ботом, що дозволяє покращувати роботу бота та забезпечувати високий рівень користувацького досвіду.

2.5 Розробка алгоритмів для обробки природної мови

Обробка природної мови (NLP) [4] є ключовою частиною будь-якого чат-бота, оскільки саме ця технологія дозволяє системі розуміти та генерувати людську мову. У даному проекті використовуються різноманітні алгоритми та методи для забезпечення якісної роботи чат-бота, написаного на Python [5] з використанням фреймворка Aiogram та LLM моделі Llama2-chat.

Чат-бот виконує кілька основних завдань, які пов’язані з обробкою природної мови:

- Попередня обробка тексту: видалення спеціальних символів, корекція форматування.
- Аналіз інтентів: розпізнавання намірів користувача на основі введеного тексту.
- Генерація відповідей: формування відповідей на основі запитів користувачів.

Попередня обробка тексту є важливим етапом, що дозволяє покращити точність розпізнавання та генерації тексту. У цьому проекті використовується функція для автоматичного екранування спеціальних символів Markdown, що забезпечує коректне відображення тексту.

```python
def md_autofixer(text: str) -> str:
    # In MarkdownV2, these characters must be escaped: _ * [ ] ( ) ~ ` > # + - = | { } . !
    escape_chars = r"_[]()~>#+-=|{}.!"
    # Use a backslash to escape special characters
    return "".join("\\" + char if char in escape_chars else char for char in text)
```

Рисунок 2.10 – Попередня обробка тексту

Для генерації відповідей використовується модель Llama2-chat, яка розгорнута на локальному сервері. Запити до моделі надсилаються через

асинхронний клієнт aiohttp. Взаємодія з моделлю здійснюється через функцію, яка обробляє запити та отримує результати у режимі реального часу.

```python
async def generate(payload: dict, modelname: str, prompt: str):
    # try:
    async with aiohttp.ClientSession() as session:
        url = f"http://{ollama_base_url}:{ollama_port}/api/chat"

        # Stream from API
        async with session.post(url, json=payload) as response:
            async for chunk in response.content:
                if chunk:
                    decoded_chunk = chunk.decode()
                    if decoded_chunk.strip():
                        yield json.loads(decoded_chunk)
```

Рисунок 2.11 – Код взаємодії з LLM моделлю

Коли користувач надсилає повідомлення, воно обробляється функцією, яка перевіряє тип повідомлення та, якщо необхідно, формує запит до моделі Llama2-chat через основну функцію генерації відповідей. Відповіді моделі поступово накопичуються і відправляються користувачу в режимі реального часу.

```python
# React on message | LLM will respond on user's message or mention in groups
@dp.message()
@perms_allowed
async def handle_message(message: types.Message):
    await get_bot_info()
    if message.chat.type == "private":
        await ollama_request(message)
    if is_mentioned_in_group_or_supergroup(message):
        # Remove the mention from the message
        if message.text is not None:
            text_without_mention = message.text.replace(mention, "").strip()
            prompt = text_without_mention
        else:
            text_without_mention = message.caption.replace(mention, "").strip()
            prompt = text_without_mention

        # Pass the modified text and bot instance to ollama_request
        await ollama_request(message, prompt)
```

Рисунок 2.12 – Код для обробки запитів користувача

Функція генерації відповіді здійснює основну роботу з формування відповіді. Вона формує запит до моделі, обробляє поточні відповіді та відправляє їх

користувачу. Після отримання повної відповіді вона записує діалог до бази даних для подальшого аналізу.

```python
async def ollama_request(message: types.Message, prompt: str = None):
    try:
        await bot.send_chat_action(message.chat.id, "typing")
        image_base64 = ""
        if message.content_type == "photo":
            image_buffer = io.BytesIO()
            await bot.download(message.photo[-1], destination=image_buffer)
            image_base64 = base64.b64encode(image_buffer.getvalue()).decode("utf-8")

        if prompt is None:
            prompt = message.text or message.caption

        full_response = ""
        sent_message = None
        last_sent_text = None

        async with ACTIVE_CHATS_LOCK:
            # Add prompt to active chats object
            if ACTIVE_CHATS.get(message.from_user.id) is None:
                ACTIVE_CHATS[message.from_user.id] = {
                    "model": modelname,
                    "messages": [
                        {
                            "role": "user",
                            "content": prompt,
                            "images": ([image_base64] if image_base64 else []),
                        }
                    ],
                    "stream": True,
                }
            else:
                ACTIVE_CHATS[message.from_user.id]["messages"].append(
                    {
                        "role": "user",
                        "content": prompt,
                        "images": ([image_base64] if image_base64 else []),
                    }
                )
        logging.info(
            f"[Request]: Processing '{prompt}' for {message.from_user.first_name} {message.from_user.last_name}"
        )
        payload = ACTIVE_CHATS.get(message.from_user.id)
        async for response_data in generate(payload, modelname, prompt):
            msg = response_data.get("message")
            if msg is None:
                continue
            chunk = msg.get("content", "")
            full_response += chunk
            full_response_stripped = full_response.strip()
            # avoid Bad Request: message text is empty
            if full_response_stripped == "":
                continue
```

Рисунок 2.13 – Код генерації відповіді

Першочерговою ідеєю було використання Google Cloud Functions [8] для взаємодії з LLM моделлю Vertex AI. Хоча через завершення пробного періоду в GCP цей метод більше не використовується, він був реалізований та протестований.

Ось приклад коду функції, яка приймає запити, обробляє їх з використанням Vertex AI та повертає відповіді.

```python
@functions_framework.http
def appeal_to_llm(request):
    # Handle preflight OPTIONS request for CORS
    if request.method == 'OPTIONS':
        headers = headers
        return ('', 204, headers)

    # Parse the JSON request body
    request_json = request.get_json()
    if not request_json:
        return jsonify({"error": "Request body is empty"}, 400)

    # Extract the history and input text from the request
    history = request_json.get('history')
    if history is None or len(history) == 0:
        return jsonify({"error": "History is empty"}, 400)

    user_prompt = history[-1].get('text')
    if user_prompt is None or user_prompt == "":
        return jsonify({"error": "user_prompt is not passed or empty"}, 400)

    # Generate a response using the language model
    llm_response = conversation.predict(
        input=user_prompt,
        context=context,
    )

    # Set the response headers for CORS
    headers = {
        'Access-Control-Allow-Origin': '*',
    }
    response = {
        "llm_response": llm_response,
        "web_link": web_link,
        "ui_link_name": article_name
    }
    return (jsonify(response), 200, headers)
```

Рисунок 2.14 – Код функції GCP

Ця функція обробляє HTTP-запити, зберігає історію діалогу, здійснює пошук подібних фрагментів у базі даних і генерує відповіді за допомогою LLM моделі Vertex AI. Незважаючи на те, що цей підхід більше не використовується, його розробка допомогла зрозуміти можливості інтеграції з хмарними сервісами для обробки природної мови.

Вибір алгоритмів та технологій для обробки природної мови в цьому проекті був обумовлений кількома факторами:

- Точність і ефективність: Модель Llama2-chat забезпечує високу якість генерації тексту, що дозволяє чат-боту давати релевантні та зрозумілі відповіді.
- Масштабованість: Використання асинхронного програмування з aiohttp дозволяє обробляти запити від багатьох користувачів одночасно, що підвищує ефективність системи.
- Простота інтеграції: Фреймворк Aiogram [9] забезпечує зручний інтерфейс для взаємодії з Telegram API, що спрощує розробку та підтримку чат-бота.
- Інтеграція з хмарними сервісами: Використання Google Cloud Functions [8] і Vertex AI було первісним підходом для обробки запитів, що демонструє можливості масштабованої хмарної інфраструктури для NLP завдань.

Таким чином, використання передових методів обробки природної мови та асинхронного програмування дозволяє створити ефективний і надійний чат-бот, який може задовольнити потреби користувачів у різних сценаріях використання.

2.6 Інтеграція компонентів системи

Інтеграція компонентів системи є ключовим етапом розробки та забезпечення безперебійної взаємодії між різними модулями. У даному проекті використовуються різні технології та сервіси, які працюють разом для надання користувачам інтерактивного досвіду спілкування з ботом, що використовує мовну модель LLama2. Основні компоненти системи включають:

1. Telegram бот (aiogram)
2. LLama2 модель через Ollama сервер
3. База даних SQLite

4. API для обробки запитів

Бот створюється та керується за допомогою бібліотеки aiogram [9], яка забезпечує асинхронне опрацювання повідомлень та команд. Основні функції бота включають:

- Прийом та обробка команд від користувачів (/start, /reset, /history)
- Відправлення запитів до моделі LLama2 для генерації відповідей
- Зберігання та відображення історії повідомлень
- Обробка зображень та інших медіа-файлів

Для генерації відповідей використовується мовна модель LLama2, що розгорнута на Ollama сервері. Модель отримує запити від бота та генерує відповідні відповіді.

База даних SQLite використовується для зберігання історії повідомлень та інших важливих даних. Це дозволяє зберігати контекст розмови та забезпечувати доступ до історичних даних.

Відправка запиту до моделі: Коли користувач надсилає повідомлення боту, воно передається до LLama2 моделі для генерації відповіді. Використовується функція generate, яка надсилає запит до Ollama сервера та отримує відповідь.

Обробка відповіді: Отримана відповідь надсилається назад користувачу через Telegram. У разі необхідності, відповідь редагується для відповідності форматуванню Markdown.

Збереження даних: Під час кожної взаємодії з користувачем, інформація про запит та відповідь зберігається в базі даних SQLite. Це дозволяє зберігати історію повідомлень та надавати користувачу доступ до них за допомогою команди /history.

Доступ до даних: За допомогою функцій, які взаємодіють з базою даних, бот може надавати користувачу доступ до історії повідомлень або інших збережених даних.

```
if "." in chunk or "\n" in chunk or "!" in chunk or "?" in chunk:
    if sent_message:
        if last_sent_text != full_response_stripped:
            await bot.edit_message_text(
                chat_id=message.chat.id,
                message_id=sent_message.message_id,
                text=full_response_stripped,
            )
            last_sent_text = full_response_stripped
    else:
        sent_message = await bot.send_message(
            chat_id=message.chat.id,
            text=full_response_stripped,
            reply_to_message_id=message.message_id,
        )
        last_sent_text = full_response_stripped

if response_data.get("done"):
    if full_response_stripped and last_sent_text != full_response_stripped:
        if sent_message:
            await bot.edit_message_text(
                chat_id=message.chat.id,
                message_id=sent_message.message_id,
                text=full_response_stripped,
            )
        else:
            sent_message = await bot.send_message(
                chat_id=message.chat.id, text=full_response_stripped
            )
    await bot.edit_message_text(
        chat_id=message.chat.id,
        message_id=sent_message.message_id,
        text=md_autofixer(
            full_response_stripped
            + f"\n\nCurrent Model: `{modelname}`**\n**Generated in {response_data.get('total_duration') / 1e9:.2f}s"
        ),
        parse_mode=ParseMode.MARKDOWN_V2,
    )
```

Рисунок 2.14 – Інтеграція Telegram бота з LLama2 моделлю

Після інтеграції компонентів система була ретельно протестована для забезпечення коректної роботи. Тестування включало:

- Перевірку коректності обробки команд та повідомлень.
- Тестування взаємодії з LLama2 моделлю та коректності генерації відповідей.
- Верифікацію збереження та доступу до даних у базі даних.

Інтеграція компонентів системи забезпечила створення функціонального та інтерактивного бота, який може відповідати на запити користувачів за допомогою потужної мовної моделі LLama2. Використання асинхронних функцій дозволило забезпечити високу швидкість обробки запитів та відповіді в реальному часі. Система готова до використання та подальшого розширення функціоналу.

РОЗДІЛ 3. ТЕСТУВАННЯ ТА ВПРОВАДЖЕННЯ ЧАТ-БОТА

3.1 Стратегія тестування

Тестування є ключовим етапом розробки програмного забезпечення, оскільки воно забезпечує правильну роботу додатка та відповідність його функціональним і нефункціональним вимогам. Для чат-бота, створеного з використанням моделі LLama2 для генерації відповідей, буде застосована стратегія тестування, що включає функціональне та інтеграційне тестування [10].

Основні цілі тестування чат-бота:

- Переконатися, що бот правильно реагує на команди користувачів і генерує коректні відповіді.
- Перевірити, що всі компоненти системи працюють злагоджено і взаємодіють без збоїв.
- Виявити та виправити помилки, що можуть вплинути на продуктивність або стабільність чат-бота.
- Оцінити зручність використання бота з точки зору кінцевих користувачів.

Для досягнення цих цілей буде проведено два основних типи тестування: функціональне тестування та інтеграційне тестування.

Функціональне тестування спрямоване на перевірку роботи конкретних функцій чат-бота відповідно до специфікацій. В рамках цього типу тестування будуть перевірені такі аспекти:

- Коректність виконання команд: Перевірка реакцій бота на стандартні команди, такі як /start, /reset, /history тощо.
- Генерація відповідей: Оцінка правильності відповідей, що генеруються моделлю LLama2 на запити користувачів.
- Обробка помилок: Перевірка здатності бота коректно обробляти помилки та непередбачувані ситуації, наприклад, відсутність інтернет-з'єднання або недоступність сервера LLama2.

- Логування: Перевірка, чи правильно логуються дії користувачів і відповіді бота в базу даних SQLite.

Інтеграційне тестування спрямоване на перевірку взаємодії між різними компонентами системи. В рамках цього типу тестування будуть перевірені такі аспекти:

- Взаємодія між Telegram API та ботом: Перевірка коректності обробки запитів від Telegram API та відправки відповідей користувачам.
- Інтеграція з моделлю LLama2: Оцінка правильності взаємодії між ботом і сервером, на якому розгорнута модель LLama2, включаючи передачу запитів і отримання відповідей.
- Робота з базою даних: Перевірка коректності збереження і отримання даних з бази даних SQLite, зокрема логів дій користувачів і згенерованих відповідей.
- Обробка паралельних запитів: Тестування здатності бота обробляти декілька запитів одночасно без втрати продуктивності або коректності відповідей.

Функціональне тестування

1. Підготовка тестових сценаріїв: Визначення команд і сценаріїв, які потрібно протестувати, створення тестових кейсів для кожної команди та сценарію.
2. Виконання тестування: Запуск тестів для кожної команди, перевірка відповідності отриманих результатів очікуваним.
3. Документування результатів: Запис результатів тестування, виявлення та документування помилок.
4. Виправлення помилок: Аналіз виявлених помилок, внесення виправлень у код, повторне тестування після внесення виправлень.

Інтеграційне тестування

1. Підготовка інтеграційних сценаріїв: Визначення основних інтеграційних точок між компонентами системи, створення тестових кейсів для кожної інтеграційної точки.

2. Виконання тестування: Запуск інтеграційних тестів, перевірка коректності взаємодії між компонентами.
3. Документування результатів: Запис результатів інтеграційного тестування, виявлення та документування проблем.
4. Виправлення проблем: Аналіз виявлених проблем, внесення виправлень у код, повторне інтеграційне тестування після внесення виправлень.

Отож, ці види тестування допоможуть виявити і виправити потенційні проблеми ще до того, як чат-бот буде представлений користувачам, навіть в ознайомчих цілях. Дотримання визначеної стратегії тестування забезпечить високу якість програмного забезпечення та задоволення користувачів.

3.2 Застосування функціонального та інтеграційного тестування

Кнопка Start використовується для запуску чат-бота. Цю кнопку слід протестувати, щоб переконатися, що:

- Натискання кнопки Start запускає чат-бота.
- Чат-бот вітає користувача та надає йому інструкції щодо того, як ним користуватися.
- Користувач може розпочати розмову з ботом після натискання кнопки Start.

Кнопка About використовується для надання користувачеві інформації про чат-бота. Цю кнопку слід протестувати, щоб переконатися, що:

- Натискання кнопки About надає користувачеві інформацію про чат-бота.
- Інформація точна та актуальна.
- Інформація легко читається та зрозуміла.

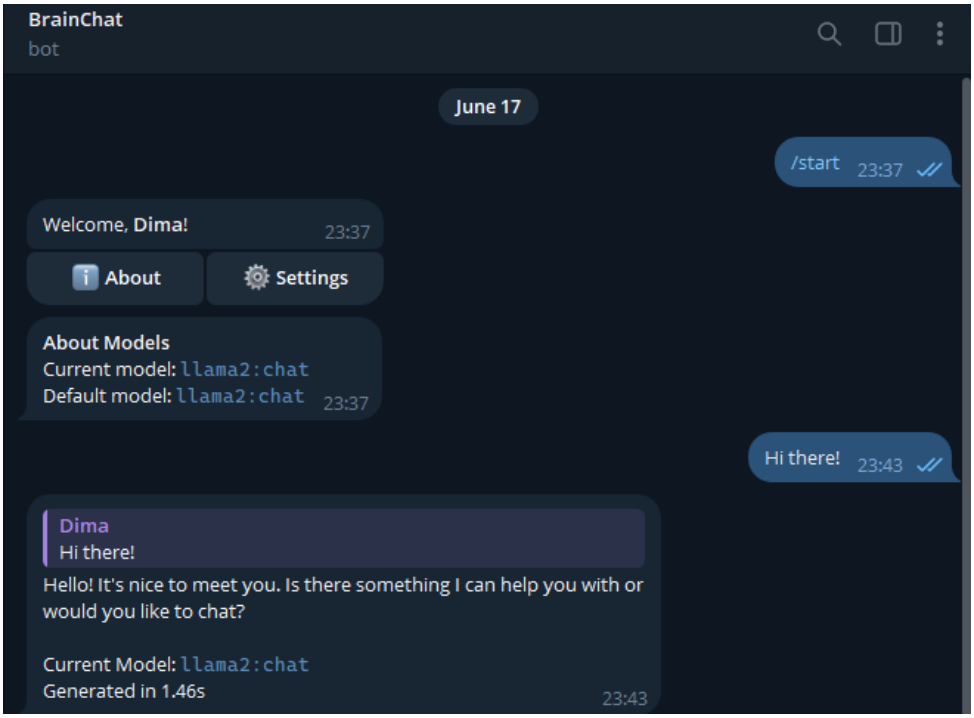

Рисунок 3.1 – Тестування кнопок start та about

Важливо протестувати його інструменти. Інструменти бота - це функції, які допомагають користувачам взаємодіяти з ботом, такі як історія чату та кнопка скидання.

Історія чату - це функція, яка дозволяє користувачам переглядати попередні розмови з ботом. Цю функцію слід протестувати, щоб переконатися, що:

- Вся розмова з ботом правильно записується в історію чату.
- Історія чату легко доступна для користувачів.
- Користувачі можуть легко переглядати та прокручувати історію чату.
- Користувачі можуть видаляти повідомлення з історії чату.

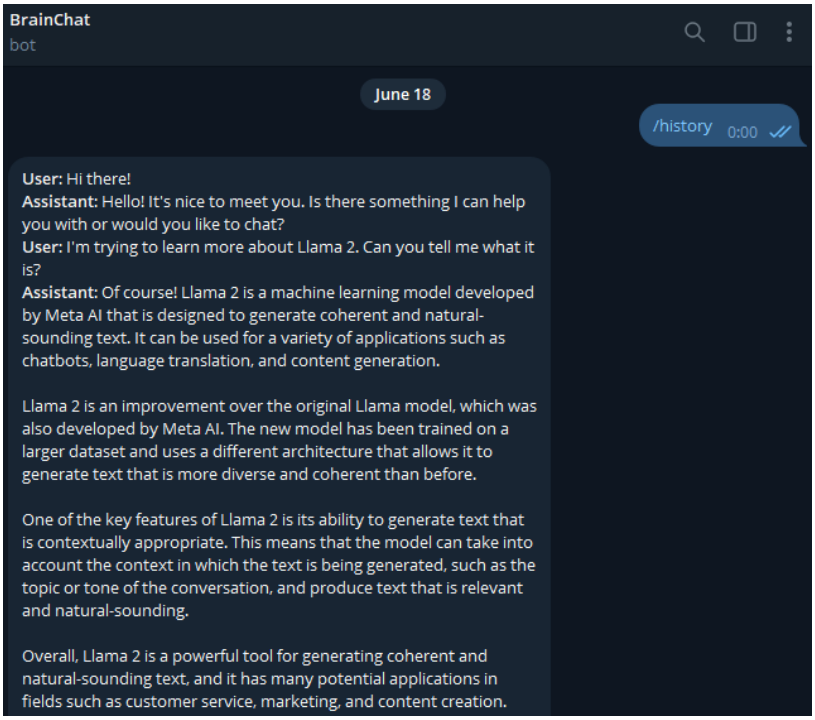

Рисунок 3.2 – Тестування кнопки history

Кнопка скидання - це функція, яка дозволяє користувачам скинути поточну розмову з ботом. Функція тестується для того щоб, кнопка скидання легко доступна для користувачів, натискання кнопки скидання скидає поточну розмову з ботом, після скидання розмови користувач може розпочати нову розмову з ботом.

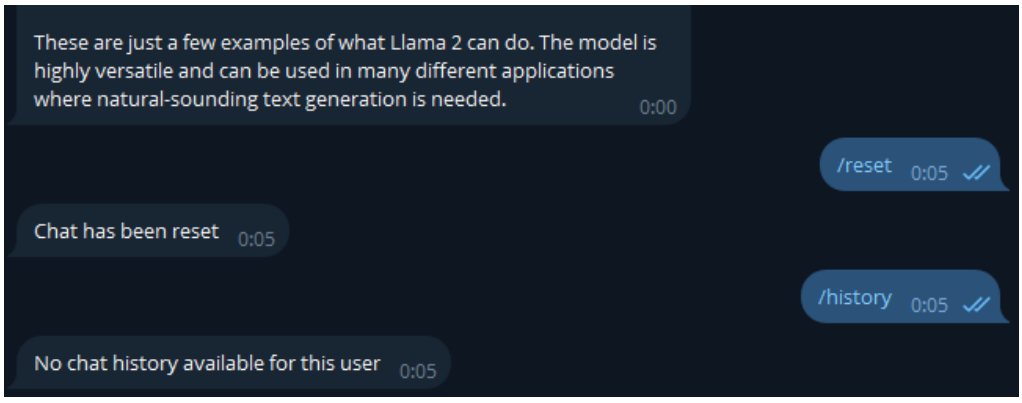

Рисунок 3.2 – Тестування кнопки reset

Чат-бот повинен правильно логувати дії користувачів і відповіді бота. Ці журнали можна використовувати для налагодження бота та відстеження його використання. Журнали повинні містити таку інформацію, як:

- Дата та час дії
- Ідентифікатор користувача
- Дія користувача
- Відповідь бота

Журнали повинні зберігатися в безпечному місці та доступні лише уповноваженим користувачам.

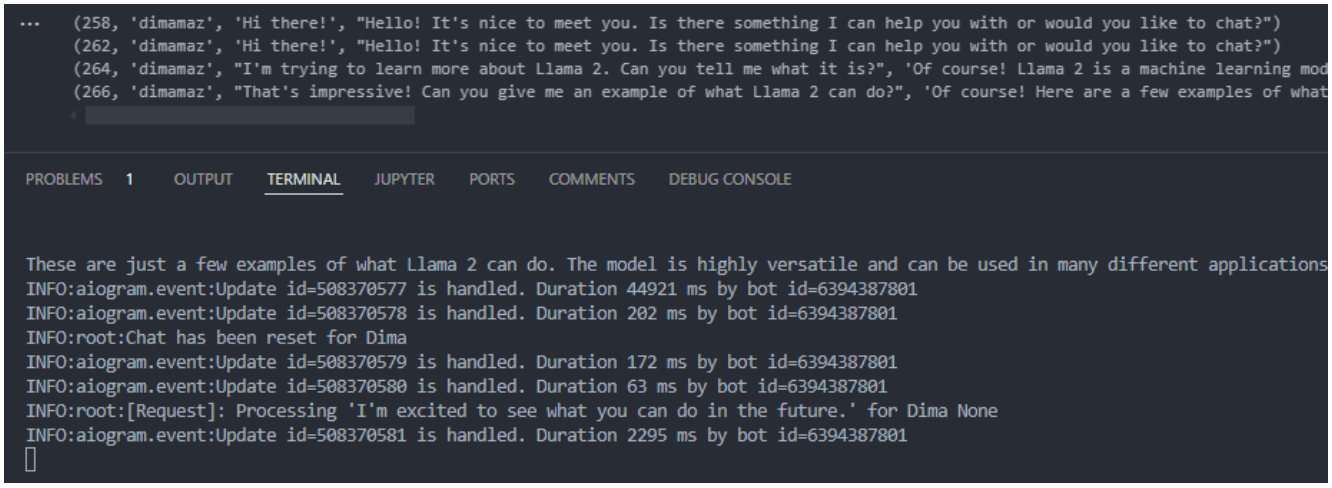

Рисунок 3.3 – Тестування логування дій в термінал та базу данних

Окрім вищеописаних тестів, я також провів додаткове тестування, щоб гарантувати всебічну та надійну роботу чат-бота. Це включало:

Тестування з різними типами користувачів: Я імітував поведінку користувачів з різним рівнем досвіду та знань, щоб переконатися, що бот зрозумілий та простий у використанні для всіх.

Тестування на різних пристроях: Я тестував бот на різних мобільних пристроях та комп'ютерах, щоб гарантувати його коректну роботу на різних платформах та екранах.

Функціональне тестування є важливою частиною розробки чат-бота. Це допомагає гарантувати, що бот відповідає очікуванням користувачів і що він правильно працює. Тестування кнопок Start та About, а також виконання

додаткових рекомендацій щодо тестування допоможе створити високоякісний чат-бот, який буде цінним для користувачів.

Для тестування коректності запитів до сервера LLM я написав код Python, який використовує бібліотеку requests для надсилання POST-запиту на сервер. Запит містить JSON-дані, які описують запит користувача. Сервер LLM обробляє запит і генерує відповідь, яка також повертається у форматі JSON [12].

```python
import requests
import json

url = "http://127.0.0.1:11434/api/chat"

headers = {
    'Content-Type': 'application/json'
}

# data = {"model":"llama2",
#         "prompt":"Why is the sky blue?",
#         "stream": False}

data = {
    "model": "llama2:chat",
    "messages": [
        {
        "role": "user",
        "content": "how are you?"
        }
    ],
    "stream": False
    }

response = requests.post(url, headers=headers, data=json.dumps(data))

if response.status_code == 200:
    response = response.text
    data = json.loads(response)
    print(data["message"]["content"])
    #data = json.loads(response)
    #print(data["response"])
else:
    print("Error:", response.status_code)
```

Рисунок 3.4 – Код з тестуванням серверу з LLM

Цей код надсилає запит до сервера LLM з наступними параметрами:

- model: це тип моделі, яка використовується.
- messages: масив повідомлень, які надсилає користувач. У цьому випадку масив містить одне повідомлення: "how are you?".
- stream: False - це вказує на те, що бот не повинен надсилати проміжні відповіді.
- Якщо запит успішний, код друкує відповідь сервера LLM.

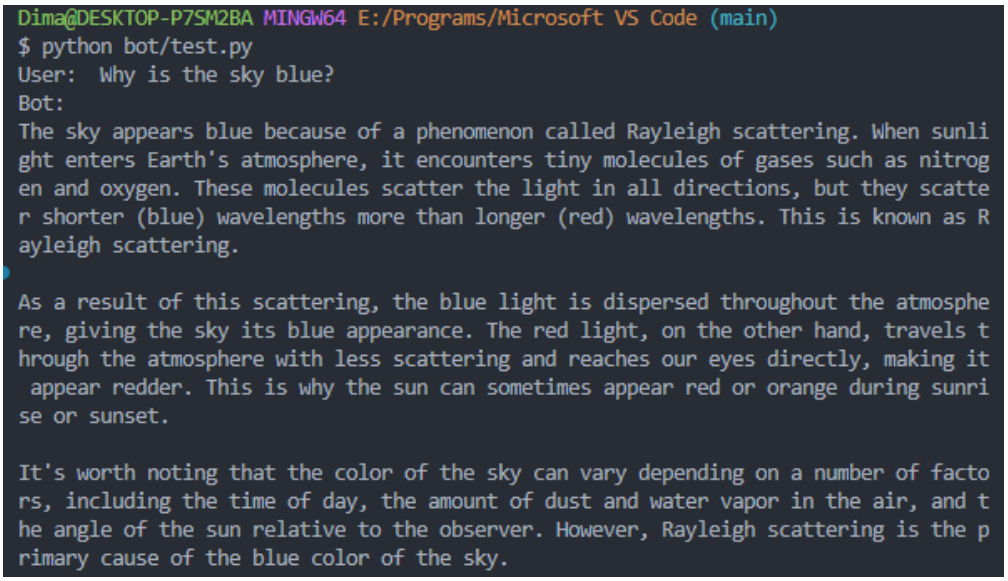

Рисунок 3.5 – Результат тестування запиту на сервер LLM

Окрім тесту, описаного вище, я також провів додаткові інтеграційні тести, щоб перевірити різні аспекти взаємодії бота з сервером LLM. Ці тести включали:

- Тестування різних типів запитів, таких як запити на інформацію, запити на виконання завдань та запити на розмову.
- Тестування обробки помилок сервером LLM.
- Тестування продуктивності сервера LLM.

Інтеграційне тестування показало, що бот правильно взаємодіє з сервером LLM. Бот може надсилати запити серверу, отримувати відповіді та обробляти їх. Сервер LLM може генерувати відповіді, які є релевантними та корисними.

РОЗДІЛ 4. БЕЗПЕКА ЖИТТЄДІЯЛЬНОСТІ ТА ОСНОВИ ОХОРОНИ ПРАЦІ

4.1 Значення адаптації в трудовому процесі.

Адаптація працівника в трудовому процесі є ключовим аспектом, що впливає на ефективність роботи, безпеку та загальний добробут співробітників. Вона включає в себе низку фізіологічних, психологічних і соціальних процесів, що дозволяють працівнику інтегруватися у нові умови праці та оптимально виконувати свої обов'язки.

Адаптація працівників поділяється на професійну, соціально-психологічну та фізіологічну.

- Професійна адаптація передбачає освоєння нових навичок, методів роботи та розуміння професійних стандартів.
- Соціально-психологічна адаптація включає інтеграцію працівника в колектив, встановлення взаємовідносин з колегами та керівництвом, що сприяє створенню позитивного психологічного клімату.
- Фізіологічна адаптація полягає у пристосуванні організму до фізичних умов праці, таких як рівень шуму, освітлення, температурний режим та інші фактори.

Адаптація має безпосередній вплив на безпеку та продуктивність праці. Добре адаптовані працівники демонструють вищу ефективність, меншу кількість помилок і аварійних ситуацій, а також кращий фізичний та психологічний стан. Зниження ризику виробничих травм досягається завдяки кращому розумінню обов'язків та правил безпеки. Підвищення продуктивності праці відбувається за рахунок освоєння нових методів роботи і інструментів, що сприяє швидкому виконанню завдань з меншими витратами енергії. Соціально-психологічна адаптація сприяє створенню позитивного психологічного клімату, що в свою чергу підвищує мотивацію і задоволення від роботи.

Процес адаптації умовно поділяється на підготовчий етап, початковий етап, активну адаптацію та завершальний етап. Підготовчий етап включає знайомство з підприємством, його історією, структурою, основними вимогами до роботи і правилами внутрішнього розпорядку. Початковий етап охоплює безпосередній вступ на робоче місце, знайомство з колективом, керівником та робочим місцем. Активна адаптація передбачає освоєння професійних навичок, вивчення специфіки роботи та встановлення взаємодії з колегами. Завершальний етап – це повне інтегрування в робочий процес, досягнення стабільної продуктивності та прийняття колективу.

Для успішної адаптації підприємства використовують такі методи:

- Менторство та наставництво – призначення досвідченого наставника, який допомагає новому працівникові освоїтися на робочому місці.
- Тренінги та навчання – проведення навчальних програм, що спрямовані на розвиток професійних навичок та знань.
- Зворотній зв'язок – регулярне надання зворотного зв'язку працівнику для корекції його дій та підвищення ефективності роботи.
- Психологічна підтримка – організація консультацій з психологами для вирішення можливих стресових ситуацій та покращення психоемоційного стану працівників.

Крім методів підтримки, є й інші інструменти, що допомагають у процесі адаптації:

1. Оцінка компетенцій. Важливо визначити рівень професійних знань і навичок працівника на початку адаптації. Це допоможе визначити, які навчальні програми необхідні для розвитку його компетенцій.
2. Інформаційні ресурси. Надання доступу до необхідних інформаційних ресурсів, таких як інструкції, внутрішні правила та стандарти, що допомагають працівникові орієнтуватися у своїй роботі.

3. Соціальні заходи. Організація соціальних заходів, таких як тимбілдинги, корпоративні свята, які сприяють кращій інтеграції працівника в колектив і покращенню соціально-психологічного клімату.
4. Кар'єрне зростання. Розробка кар'єрних планів і можливостей для професійного зростання мотивує працівників і сприяє їх тривалій адаптації.
5. Моніторинг та оцінка адаптаційного процесу. Регулярний моніторинг і оцінка процесу адаптації дозволяють вчасно виявляти проблеми і коригувати стратегії адаптації.

Адаптація працівників у трудовому процесі є важливим елементом, що забезпечує не тільки ефективність роботи, але і безпеку, здоров'я та загальне благополуччя співробітників. Інтеграція нових працівників за допомогою чітко структурованих етапів та методів підтримки дозволяє створити гармонійне та продуктивне робоче середовище.

4.2 Вплив кольору на покращення умов праці та підвищення продуктивності виробництва.

Колір має значний вплив на фізіологічний та психологічний стан людини, що безпосередньо відображається на її працездатності та загальному самопочутті. Використання кольору як одного з елементів робочого середовища сприяє створенню комфортних умов праці.

Кольори можуть викликати різні реакції організму, такі як зміни у частоті серцевих скорочень, артеріальному тиску та активності мозку. Наприклад, теплі кольори (червоний, оранжевий, жовтий) стимулюють нервову систему, підвищують активність та можуть сприяти підвищенню концентрації уваги. Водночас, холодні кольори (синій, зелений) заспокоюють, знижують напругу та сприяють розслабленню.

Застосування кольорів в робочому середовищі має значний вплив на продуктивність праці. Дослідження показали, що правильне використання кольору може збільшити продуктивність на 10-20%. Наприклад, в офісних приміщеннях часто використовують нейтральні або прохолодні кольори, які допомагають зосередитися і знижують стомлюваність. У виробничих приміщеннях, де важлива швидкість і точність, доцільно використовувати яскраві кольори, що стимулюють активність.

Згідно з чинними стандартами та рекомендаціями, кольори в робочому середовищі повинні бути обрані з урахуванням психологічного впливу на працівників. Важливим є також дотримання санітарних норм та правил безпеки. Крім того, "Світло і освітлення. Освітлення робочих місць" визначає норми освітленості, які включають врахування кольору приміщень та обладнання для досягнення оптимального робочого середовища.

Таблиця 4.1 – Вплив кольорів на життєдіяльність людини

Колір	Артеріальний тиск	Пульс	Частота дихань	Емоційний стан	Час реакції	Працездатність
Червоний	Підвищується	Збільшується	Збільшується	Збуджує, стимулює	Зменшується	Спочатку підвищується ненадовго, а потім зменшується наполовину
Оранжевий	Мало підвищується	Трохи збільшується	Трохи збільшується	Збуджує, бадьорить	Трохи зменшується	Так само, але менш виражено
Жовтий	Не змінюється	Не змінюється	Не змінюється	Урівноважує	Не змінюється	Суттєво не змінюється
Зелений	Не змінюється	Не змінюється	Не змінюється	Урівноважує	Не змінюється	Трохи підвищується

Продовження таблиці 4.1

Колір	Артеріальний тиск	Пульс	Частота дихань	Емоційний стан	Час реакції	Працездатність
Блакитний	Дещо знижуєтьс я	Трохи сповільнює ться	Трохи сповільнює ться	Заспоко ює	Суттєво не змінюється	Трохи підвищуєть ся
Синій	Знижуєть ся	Сповільню-ється	Сповільню-ється	Заспоко ює, трохи загальмо вує	Трохи сповільнює ться	Підвищуєть ся на 3—9 %
Фіолето вий	Знижуєть ся	Сповільню-ється	Сповільню-ється	Пригнічу є	Виразно сповільнює ться	Стабільно знижується

Практичні рекомендації:

1. Офісні приміщення: Використання світло-блакитних, зелених та пастельних відтінків допомагає знизити рівень стресу та сприяє концентрації. Уникайте надто яскравих кольорів, які можуть викликати втому очей.
2. Виробничі приміщення: Червоні та оранжеві акценти стимулюють енергію та активність, однак їх слід використовувати дозовано. Зелений колір може бути використаний у зонах відпочинку для розслаблення працівників.
3. Зони відпочинку: Використання м'яких, заспокійливих кольорів, таких як бежевий, світло-зелений або блакитний, допомагає працівникам розслабитися та відновити сили.

Монітори є невід'ємною частиною сучасних робочих місць, особливо в офісах, де працівники проводять значний час перед екранами. Випромінювання моніторів, як правило, складається з електромагнітного випромінювання та синього світла, що може мати різний вплив на здоров'я працівників та їх продуктивність.

1. Електромагнітне випромінювання: Виникає від роботи електронних компонентів. Незважаючи на те, що рівень цього випромінювання є досить

низьким, тривале перебування перед монітором може призвести до накопичення його впливу.

2. Синє світло: Випромінюється світлодіодами, які використовуються в сучасних моніторах. Це світло має коротку довжину хвилі і високу енергію, що може викликати напругу очей та порушення сну.

Вплив на здоров'я та продуктивність

1. Зорове стомлення: Тривале використання монітора може викликати зорове стомлення, яке проявляється у вигляді болю в очах, розмитості зору, головного болю та сухості очей. Це знижує здатність до концентрації та продуктивність праці.
2. Циркадні ритми: Вплив синього світла на організм може порушувати природні циркадні ритми, що призводить до проблем зі сном. Порушення сну негативно впливає на загальний стан здоров'я, продуктивність та настрій працівників.
3. Електромагнітне випромінювання: Деякі дослідження вказують на можливі довгострокові наслідки впливу електромагнітного випромінювання на здоров'я, проте рівень випромінювання сучасних моніторів зазвичай знаходиться в межах безпечних норм.

Врахування впливу кольору на працездатність є важливим аспектом організації робочого середовища. Правильний вибір кольорів може суттєво покращити умови праці, зменшити стрес та підвищити продуктивність. Важливо дотримуватись чинних стандартів та рекомендацій, які базуються на наукових дослідженнях та досвіді використання кольорів у різних сферах діяльності.

ВИСНОВКИ

В результаті виконання цієї кваліфікаційної роботи було розроблено та імплементовано інтелектуального чат-бота у Telegram, який забезпечить користувачам інтуїтивно зрозуміле та ефективне спілкування з системою за допомогою природної мови. Для досягнення цієї мети були поставлені наступні задачі:

- Проаналізувати предметну область, включаючи розвиток чат-ботів та сучасні технології обробки природної мови та штучного інтелекту.
- Обрати та обґрунтувати технології, які найкраще підходять для розробки чат-бота.
- Розробити архітектуру системи, спроектувати базу даних та реалізувати алгоритми обробки природної мови.
- Інтегрувати чат-бота з Telegram API та розробити зручний користувацький інтерфейс.
- Провести тестування розробленої системи та впровадити її для практичного використання.

Для досягнення поставленої мети та вирішення задач дослідження були використані методи аналізу літературних джерел, аналізу програмного забезпечення, проєктування систем, програмування, тестування та аналізу результатів.

Структура даної роботи включала кілька розділів:

- Аналіз предметної області та задачі.
- Проєктування та розробка чат-бота.
- Тестування та впровадження чат-бота.
- Безпека життєдіяльності та основи охорони праці.

У першому розділі було проведено детальний аналіз розвитку чат-ботів та сучасних технологій обробки природної мови і штучного інтелекту. Було

розглянуто конкурентні рішення та сформульовано вимоги до інтелектуального чат-бота.

У другому розділі були обрані та обґрунтовані технології для розробки чат-бота, спроектована архітектура системи та база даних, розроблені алгоритми для обробки природної мови та здійснена інтеграція з Telegram API. Особлива увага була приділена розробці зручного користувацького інтерфейсу.

У третьому розділі була описана стратегія тестування, проведено функціональне та інтеграційне тестування системи. Результати тестування підтвердили відповідність чат-бота поставленим вимогам та його готовність до використання.

У четвертому розділі були розглянуті питання безпеки та охорони праці, які виникають при розробці програмного забезпечення, зокрема аналіз потенційних небезпек, організація безпечних умов праці та нормативні вимоги до робочого місця розробника.

Розробка інтелектуального чат-бота для Telegram виявилася успішною. Система забезпечує ефективне та інтуїтивно зрозуміле спілкування з користувачами за допомогою природної мови. Проведене тестування підтвердило надійність та відповідність розробленого чат-бота поставленим вимогам. Дана робота демонструє можливості сучасних технологій обробки природної мови та штучного інтелекту у створенні зручних та ефективних систем комунікації. Результати дослідження можуть бути використані для подальших розробок у цій галузі.

Таким чином, розробка інтелектуального чат-бота для Telegram є актуальною та перспективною задачею, яка сприяє розвитку нових форм взаємодії людини з комп'ютером. Впровадження таких систем дозволяє покращити користувацький досвід та автоматизувати багато процесів, що сприяє підвищенню ефективності роботи різних сервісів.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Еволюція штучного інтелекту (ШІ): Визначні моменти в історії та застосування [Електронний ресурс] // cases.media. – 2023. – Режим доступу до ресурсу: https://cases.media/en/article/evolyuciya-shtuchnogo-intelektu-shi-viznachni-momenti-v-istoriyi-ta-zastosuvannya.

2. Telegram APIs [Електронний ресурс] – Режим доступу до ресурсу: https://core.telegram.org.

3. Що треба знати месенджери [Електронний ресурс] – Режим доступу до ресурсу: https://mediamaker.me/yak-zahystyty-pryvatni-rozmovy-v-mesendzherah-vid-shpyguniv-hakeriv-i-vorogiv-8956/.

4. На старті в NLP [Електронний ресурс] – Режим доступу до ресурсу: https://dou.ua/forums/topic/44792/.

5. Python documentation [Електронний ресурс] – Режим доступу до ресурсу: https://docs.python.org/3/.

6. SQLite Documentation [Електронний ресурс] – Режим доступу до ресурсу: https://www.sqlite.org/docs.html.

7. Documentation for Visual Studio Code [Електронний ресурс] – Режим доступу до ресурсу: https://code.visualstudio.com/docs.

8. Cloud Functions documentation [Електронний ресурс] – Режим доступу до ресурсу: https://cloud.google.com/functions/docs.

9. Aiogram 3.8.0 documentation [Електронний ресурс] – Режим доступу до ресурсу: https://docs.aiogram.dev/en/latest/.

10. Тестова стратегія [Електронний ресурс] – Режим доступу до ресурсу: https://training.qatestlab.com/blog/technical-articles/difference-between-test-strategy-and-test-plan/.

11. Що таке функціональне тестування [Електронний ресурс] – Режим доступу до ресурсу: https://training.qatestlab.com/blog/technical-articles/difference-between-functional-and-non-functional-testing/.

12. Що таке інтеграційне тестування [Електронний ресурс] – Режим доступу до ресурсу: https://www.zaptest.com/uk/що-таке-інтеграційне-тестування-глиб.

13. Advancing Computational Architectures for Analyzing and Simulation of Systems of nanoporous particles filtration Software / Petryk, M., Brevus, V., Mykhalyk, D. Kyshkevych, O. – Ternopil: Ternopil Ivan Puluj National Technical University,, 2023.

14. Application of machine learning for modeling of 6061-T651 aluminum alloy stress–strain diagram / Yasniy, O. , Pastukh, O. , Didych, I. , Yatsyshyn, V. , Chykhira, I. – Тернопіль: Ternopil Ivan Puluj National Technical University,, 2023.

15. Modelling of non-isothermal adsorption of gases in nanoporous adsorbent based on Langmuir equilibrium / Petryk, M. , Boyko, I. , Fraissard, J. , Lebovka, N. – Тернопіль: Ternopil Ivan Puluj National Technical University,, 2023.

16. Стиценко Т.Є., Пронюк Г.В., Сердюк Н.М., Хондак І.І. «Безпека життєдіяльності»: навч. посібник / Т.Є Стиценко, Г.В. Пронюк, Н.М. Сердюк, І.І. Хондак. – Харків: ХНУРЕ, 2018. – 98 – 106 с.

17. Адаптація як пристосування працівника до умов праці в державному органі [Електронний ресурс] – Режим доступу до ресурсу: https://i.factor.ua/ukr/journals/ds/2018/july/issue-7/article-37777.html

18. Основи теорії кольору. Навчально-методичний посібник для здобувачів освітнього ступеня бакалавра спеціальності 186 “Видавництво та поліграфія” усіх форм навчання [Електронний ресурс]/ [упоряд. Т.І. Веретільник, Л.Д. Мисник, Капітан Р.Б., Мамонов Ю.П., Манзюра О.В.] ; М-во освіти і науки України, Черкас. держ. технол. ун-т. – Черкаси : ЧДТУ, 2020 – 76 – 83с

19. ДСТУ EN 12464-1:2016 Світло та освітлення. Освітлення робочих місць. Частина 1. Внутрішні робочі місця (EN 12464-1:2011, IDT)

ДОДАТКИ

Лістинг 1 – код functions.py

```
import logging
import os
import aiohttp
import sqlite3
import json
from aiogram import types
from asyncio import Lock
from functools import wraps
from dotenv import load_dotenv
# --- Environment
load_dotenv()
# --- Environment Checker
token = os.getenv("TOKEN")
allowed_ids = list(map(int, os.getenv("USER_IDS", "").split(",")))
admin_ids = list(map(int, os.getenv("ADMIN_IDS", "").split(",")))
ollama_base_url = os.getenv("OLLAMA_BASE_URL")
ollama_port = os.getenv("OLLAMA_PORT", "11434")
log_level_str = os.getenv("LOG_LEVEL", "INFO")

# --- Other
log_levels = list(logging._levelToName.values())
# ['CRITICAL', 'ERROR', 'WARNING', 'INFO', 'DEBUG', 'NOTSET']

# Set default level to be INFO
if log_level_str not in log_levels:
    log_level = logging.DEBUG
else:
    log_level = logging.getLevelName(log_level_str)

logging.basicConfig(level=log_level)

# Ollama API
# Model List
```

```
async def model_list():
    async with aiohttp.ClientSession() as session:
        url = f"http://{ollama_base_url}:{ollama_port}/api/tags"

        async with session.get(url) as response:
            if response.status == 200:
                data = await response.json()
                return data["models"]

            else:
                return []
async def generate(payload: dict, modelname: str, prompt: str):
    # try:
    async with aiohttp.ClientSession() as session:
        url = f"http://{ollama_base_url}:{ollama_port}/api/chat"

        # Stream from API
        async with session.post(url, json=payload) as response:
            async for chunk in response.content:
                if chunk:
                    decoded_chunk = chunk.decode()
                    if decoded_chunk.strip():
                        yield json.loads(decoded_chunk)

# Aiogram functions & wraps
def perms_allowed(func):
    @wraps(func)
        async def wrapper(message: types.Message = None, query:
types.CallbackQuery = None):
        user_id = message.from_user.id if message else query.from_user.id
        if user_id in admin_ids or user_id in allowed_ids:
            if message:
                return await func(message)
            elif query:
                return await func(query=query)

        else:
            if message:
```

```
                if message and message.chat.type in ["supergroup", "group"]:
                    return
                await message.answer("Access Denied")
            elif query:
                if message and message.chat.type in ["supergroup", "group"]:
                    return
                await query.answer("Access Denied")

    return wrapper

def perms_admins(func):
    @wraps(func)
        async def wrapper(message: types.Message = None, query:
types.CallbackQuery = None):
        user_id = message.from_user.id if message else query.from_user.id
        if user_id in admin_ids:
            if message:
                return await func(message)
            elif query:
                return await func(query=query)
        else:
            if message:
                if message and message.chat.type in ["supergroup", "group"]:
                    return
                await message.answer("Access Denied")
                logging.info(
                                    f"[MSG] {message.from_user.first_name}
{message.from_user.last_name}({message.from_user.id}) is not allowed to use
this bot."
                )
            elif query:
                if message and message.chat.type in ["supergroup", "group"]:
                    return
                await query.answer("Access Denied")
                logging.info(
                                    f"[QUERY] {message.from_user.first_name}
{message.from_user.last_name}({message.from_user.id}) is not allowed to use
this bot."
```

```
                )

    return wrapper

def md_autofixer(text: str) -> str:
    # In MarkdownV2, these characters must be escaped: _ * [ ] ( ) ~ ` > #
+ - = | { } . !
    escape_chars = r"_[]()~>#+-=|{}.!"
    # Use a backslash to escape special characters
    return "".join("\\" + char if char in escape_chars else char for char in
text)

def log_to_db(user_id: int, full_name: str, input_data: str, output: str):
    db = sqlite3.connect('logs.db')
    cursor = db.cursor()
    cursor.execute("""CREATE TABLE IF NOT EXISTS users(
        id INTEGER PRIMARY KEY,
        full_name TEXT,
        input_data TEXT,
        output_data TEXT
    )""")
    cursor.execute("INSERT INTO users (id, full_name, input_data, output_data)
VALUES (?, ?, ?, ?)",
                   (user_id, str(full_name), str(input_data), str(output)))
    db.commit()
    db.close()

# Context-Related
class contextLock:
    lock = Lock()

    async def __aenter__(self):
        await self.lock.acquire()

    async def __aexit__(self, exc_type, exc_value, exc_traceback):
        self.lock.release()
```

Лістинг 2 – код run.py

```
from aiogram import Bot, Dispatcher
from aiogram.enums import ParseMode
from aiogram.filters.command import Command, CommandStart
from aiogram.types import Message
from aiogram.utils.keyboard import InlineKeyboardBuilder
from func.functions import *

# Other
import asyncio
import traceback
import io
import base64

bot = Bot(token=token)
dp = Dispatcher()
builder = InlineKeyboardBuilder()
builder.row(
    types.InlineKeyboardButton(text="ⓘ About", callback_data="info"),
                    types.InlineKeyboardButton(text="⚙          Settings",
callback_data="modelmanager"),
)

commands = [
    types.BotCommand(command="start", description="Start"),
    types.BotCommand(command="reset", description="Reset Chat"),
    types.BotCommand(command="history", description="Look through messages"),
]

# Context variables for OllamaAPI
ACTIVE_CHATS = {}
ACTIVE_CHATS_LOCK = contextLock()
modelname = os.getenv("INITMODEL")
mention = None

# Telegram group types
CHAT_TYPE_GROUP = "group"
```

```
CHAT_TYPE_SUPERGROUP = "supergroup"



def is_mentioned_in_group_or_supergroup(message):
    return message.chat.type in [CHAT_TYPE_GROUP, CHAT_TYPE_SUPERGROUP] and
(
        (message.text is not None and message.text.startswith(mention))
                    or     (message.caption     is     not     None     and
message.caption.startswith(mention))
    )

async def get_bot_info():
    global mention
    if mention is None:
        get = await bot.get_me()
        mention = f"@{get.username}"
    return mention

# /start command
@dp.message(CommandStart())
async def command_start_handler(message: Message) -> None:
    start_message = f"Welcome, <b>{message.from_user.full_name}</b>!"
    await message.answer(
        start_message,
        parse_mode=ParseMode.HTML,
        reply_markup=builder.as_markup(),
        disable_web_page_preview=True,
    )

# /reset command, wipes context (history)
@dp.message(Command("reset"))
async def command_reset_handler(message: Message) -> None:
    if message.from_user.id in allowed_ids:
        if message.from_user.id in ACTIVE_CHATS:
            async with ACTIVE_CHATS_LOCK:
                ACTIVE_CHATS.pop(message.from_user.id)
```

```
                            logging.info(f"Chat has been reset for
{message.from_user.first_name}")
            await bot.send_message(
                chat_id=message.chat.id,
                text="Chat has been reset",
            )

# /history command | Displays dialogs between LLM and USER
@dp.message(Command("history"))
async def command_get_context_handler(message: Message) -> None:
    if message.from_user.id in allowed_ids:
        if message.from_user.id in ACTIVE_CHATS:
            messages = ACTIVE_CHATS.get(message.chat.id)["messages"]
            context = ""
            for msg in messages:
                                context += f"*{msg['role'].capitalize()}*:
{msg['content']}\n"
            await bot.send_message(
                chat_id=message.chat.id,
                text=context,
                parse_mode=ParseMode.MARKDOWN,
            )
        else:
            await bot.send_message(
                chat_id=message.chat.id,
                text="No chat history available for this user",
            )

@dp.callback_query(lambda query: query.data == "modelmanager")
async def modelmanager_callback_handler(query: types.CallbackQuery):
    models = await model_list()
    modelmanager_builder = InlineKeyboardBuilder()
    for model in models:
        modelname = model["name"]
        modelfamilies = ""
        if model["details"]["families"]:
            modelicon = {"llama": "", "clip": ""}
            try:
```

```
                modelfamilies = "".join(
                                    [modelicon[family]   for   family   in
model["details"]["families"]]
                )
            except KeyError as e:
                # Use a default value when the key is not found
                modelfamilies = f"✨"
        # Add a button for each model
        modelmanager_builder.row(
            types.InlineKeyboardButton(
                                    text=f"{modelname}    {modelfamilies}",
callback_data=f"model_{modelname}"
            )
        )
    await query.message.edit_text(
        f"{len(models)} models available.\n = Regular\n📸 = Multimodal",
        reply_markup=modelmanager_builder.as_markup(),
    )

@dp.callback_query(lambda query: query.data.startswith("model_"))
async def model_callback_handler(query: types.CallbackQuery):
    global modelname
    global modelfamily
    modelname = query.data.split("model_")[1]
    await query.answer(f"Chosen model: {modelname}")

@dp.callback_query(lambda query: query.data == "info")
@perms_admins
async def info_callback_handler(query: types.CallbackQuery):
    dotenv_model = os.getenv("INITMODEL")
    global modelname
    await bot.send_message(
        chat_id=query.message.chat.id,
                        text=f"<b>About     Models</b>\nCurrent     model:
<code>{modelname}</code>\nDefault model: <code>{dotenv_model}</code>",
        parse_mode=ParseMode.HTML,
        disable_web_page_preview=True,
    )
```

```
# React on message | LLM will respond on user's message or mention in groups
@dp.message()
@perms_allowed
async def handle_message(message: types.Message):
    await get_bot_info()
    if message.chat.type == "private":
        await ollama_request(message)
    if is_mentioned_in_group_or_supergroup(message):
        # Remove the mention from the message
        if message.text is not None:
            text_without_mention = message.text.replace(mention, "").strip()
            prompt = text_without_mention
        else:
                text_without_mention  =  message.caption.replace(mention,
"").strip()
            prompt = text_without_mention

        # Pass the modified text and bot instance to ollama_request
        await ollama_request(message, prompt)


...

async def ollama_request(message: types.Message, prompt: str = None):
    try:
        await bot.send_chat_action(message.chat.id, "typing")
        image_base64 = ""
        if message.content_type == "photo":
            image_buffer = io.BytesIO()
            await bot.download(message.photo[-1], destination=image_buffer)
                                                    image_base64        =
base64.b64encode(image_buffer.getvalue()).decode("utf-8")

        if prompt is None:
            prompt = message.text or message.caption

        full_response = ""
        sent_message = None
```

```
        last_sent_text = None

        async with ACTIVE_CHATS_LOCK:
            # Add prompt to active chats object
            if ACTIVE_CHATS.get(message.from_user.id) is None:
                ACTIVE_CHATS[message.from_user.id] = {
                    "model": modelname,
                    "messages": [
                        {
                            "role": "user",
                            "content": prompt,
                             "images": ([image_base64] if image_base64 else
[]),
                        }
                    ],
                    "stream": True,
                }
            else:
                ACTIVE_CHATS[message.from_user.id]["messages"].append(
                    {
                        "role": "user",
                        "content": prompt,
                        "images": ([image_base64] if image_base64 else []),
                    }
                )
        logging.info(
                            f"[Request]:    Processing    '{prompt}'    for
{message.from_user.first_name} {message.from_user.last_name}"
        )
        payload = ACTIVE_CHATS.get(message.from_user.id)
        async for response_data in generate(payload, modelname, prompt):
            msg = response_data.get("message")
            if msg is None:
                continue
            chunk = msg.get("content", "")
            full_response += chunk
            full_response_stripped = full_response.strip()
            # avoid Bad Request: message text is empty
```

```
            if full_response_stripped == "":
                continue

            if "." in chunk or "\n" in chunk or "!" in chunk or "?" in chunk:
                if sent_message:
                    if last_sent_text != full_response_stripped:
                        await bot.edit_message_text(
                            chat_id=message.chat.id,
                            message_id=sent_message.message_id,
                            text=full_response_stripped,
                        )
                        last_sent_text = full_response_stripped
                else:
                    sent_message = await bot.send_message(
                        chat_id=message.chat.id,
                        text=full_response_stripped,
                        reply_to_message_id=message.message_id,
                    )
                    last_sent_text = full_response_stripped

            if response_data.get("done"):
                        if  full_response_stripped  and  last_sent_text  !=
full_response_stripped:
                    if sent_message:
                        await bot.edit_message_text(
                            chat_id=message.chat.id,
                            message_id=sent_message.message_id,
                            text=full_response_stripped,
                        )
                    else:
                        sent_message = await bot.send_message(
                                                    chat_id=message.chat.id,
text=full_response_stripped
                        )
                await bot.edit_message_text(
                    chat_id=message.chat.id,
                    message_id=sent_message.message_id,
                    text=md_autofixer(
```

```
                        full_response_stripped
                        + f"\n\nCurrent Model: `{modelname}`**\n**Generated
in {response_data.get('total_duration') / 1e9:.2f}s"
                    ),
                    parse_mode=ParseMode.MARKDOWN_V2,
                )

                async with ACTIVE_CHATS_LOCK:
                    if ACTIVE_CHATS.get(message.from_user.id) is not None:
                        # Add response to active chats object
                     ACTIVE_CHATS[message.from_user.id]["messages"].append(
                                        {"role": "assistant", "content":
full_response_stripped}
                        )
                        logging.info(
                               f"[Response]: '{full_response_stripped}' for
{message.from_user.first_name} {message.from_user.last_name}"
                        )
                                    print("INFO: ", message.message_id,
message.from_user.username, prompt, full_response_stripped)
                        log_to_db(
                            message.message_id,
                            message.from_user.username,
                            prompt,
                            full_response_stripped,
                        )
                    else:
                        await bot.send_message(
                            chat_id=message.chat.id, text="Chat was reset"
                        )

                break
    except Exception as e:
        await bot.send_message(
            chat_id=message.chat.id,
            text=f"""Error occurred\n```\n{traceback.format_exc()}\n```""",
            parse_mode=ParseMode.MARKDOWN_V2,
        )
```

```
async def main():
    await bot.set_my_commands(commands)
    await dp.start_polling(bot, skip_update=True)

if __name__ == "__main__":
    asyncio.run(main())
```

Лістинг 3 – код test.py

```
import requests
import json

url = "http://127.0.0.1:11434/api/chat"

headers = {
    'Content-Type': 'application/json'
}

# data = {"model":"llama2",
#          "prompt":"Why is the sky blue?",
#          "stream": False}

data = {
    "model": "llama2:chat",
    "messages": [
        {
        "role": "user",
        "content": "Why is the sky blue?"
        }
    ],
    "stream": False
    }

response = requests.post(url, headers=headers, data=json.dumps(data))

if response.status_code == 200:
    response = response.text
    output = json.loads(response)
```

```
    print("User: ", data["messages"][0]["content"])
    print("Bot: ", output["message"]["content"])
    #data = json.loads(response)
    #print(data["response"])
else:
    print("Error:", response.status_code)
```

Додаток Б – Диск з роботою