МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Сумський державний університет

Факультет електроніки та інформаційних технологій
Кафедра комп'ютерних наук

«До захисту допущено»
В.о. завідувача кафедри
Ігор ШЕЛЕХОВ
(підпис)

КВАЛІФІКАЦІЙНА РОБОТА

НА ЗДОБУТТЯ ОСВІТНЬОГО СТУПЕНЯ БАКАЛАВР

зі спеціальності 122 - Комп'ютерні науки,

освітньо-професійної програми «Інформатика»

на тему: «Віртуальний помічник аналітика даних»

здобувача групи ІН - 03 Коваленко Михайла Руслановича

Кваліфікаційна робота містить результати власних досліджень. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело.
Коваленко Михайло
(підпис)

Керівник,
Старший викладач комп'ютерних наук,
кандидат технічних наук
Ігор Шелехов
(підпис)

СУМИ – 2024

Сумський державний університет
Факультет електроніки та інформаційних технологій
Кафедра комп'ютерних наук

«Затверджую»
В.о. завідувача кафедри
_____ Ігор ШЕЛЕХОВ
(підпис)

ЗАВДАННЯ НА КВАЛІФІКАЦІЙНУ РОБОТУ

на здобуття освітнього ступеня бакалавр

зі спеціальності 122 - Комп'ютерних наук, освітньо-професійної програми «Інформатика»
здобувача групи ІН-03 Коваленко Михайла Руслановича

1. Тема роботи: «Віртуальний помічник аналітика даних»
затверджую наказом по СумДУ від «22» квітня 2024 р. № 0414-IV

2. Термін здачі здобувачем кваліфікаційної роботи до 1 червня 2024 року

3. Вхідні дані до кваліфікаційної роботи

4. Зміст розрахунково-пояснювальної записки (перелік питань, що їх належить розробити)

1) Аналіз проблеми предметної області, постановка й формування завдань дослідження.
2) Огляд технологій, що використовуються для розробки інформаційної системи у вигляді телеграм боту.
3) Розробка віртуального помічника для аналітики даних.
4) Аналіз результатів.

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

6. Консультанти до проекту (роботи), із значенням розділів проекту, що стосується їх

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв

7. Дата видачі завдання «06» травня 2024 р.

Завдання прийняв до виконання _____ (підпис) Керівник _____ (підпис)

КАЛЕНДАРНИЙ ПЛАН

№ п/п	Назва етапів кваліфікаційної роботи	Термін виконання	Примітка
1	Аналіз проблеми предметної області, постановка й формування завдань дослідження	06.05.24 – 10.05.24	
2	Огляд технологій, що використовуються для розробки інформаційної системи у вигляді телеграм боту	11.05.24 – 17.05.24	
3	Розробка віртуального помічника для аналітики даних	18.05.24 – 20.05.24	
4	Аналіз отриманих результатів	21.05.24 – 28.05.24	
5	Оформлення пояснювальної записки до кваліфікаційної роботи	27.05.24 – 31.05.24	

Здобувач вищої освіти _____ Керівник _____

АНОТАЦІЯ

Записка: 45 стр., 13 рис., 1 додаток, 9 використаних джерел.

Обґрунтування актуальності теми роботи – У сучасному світі обсяг даних, які необхідно обробляти та аналізувати, зростає з кожним днем. Це вимагає розробки ефективних інструментів для підтримки процесу прийняття рішень на основі даних. Віртуальні помічники, які використовують технології штучного інтелекту та машинного навчання, стають незамінними інструментами для аналітиків даних. Вони здатні автоматизувати рутинні задачі, надавати оперативну інформацію, здійснювати комплексний аналіз даних та допомагати у виявленні прихованих тенденцій та закономірностей. Використання віртуальних помічників дозволяє значно підвищити ефективність та точність аналітичних процесів, що робить тему дослідження актуальною та важливою для подальшого розвитку галузі.

Об’єкт дослідження — Об’єктом дослідження є процес розробки віртуального помічника аналітика даних, розроблений на основі Telegram-бота, який використовує API ChatGPT для обробки та аналізу даних.

Мета роботи — Метою роботи є розробка та впровадження віртуального помічника, який здатний здійснювати комплексний аналіз даних, надавати рекомендації та забезпечувати підтримку прийняття рішень для аналітиків даних. Віртуальний помічник повинен автоматизувати рутинні процеси та забезпечити високу точність та швидкість обробки даних.

Методи дослідження — алгоритми створення інформаційних систем з використанням певного стеку технологій.

Результати — розроблено інформаційну систему віртуального помічника для аналітики даних у вигляді телеграм боту, у користувача є можливість отримувати аналіз даних та поради. Інформаційна система створена на базі мови програмування Python та СУБД MongoDB.

ІНФОРМАЦІЙНА СИСТЕМА, ТЕЛЕГРАМ БОТ, АНАЛІТИКА ДАНИХ, PYTHON, MONGODB

ЗМІСТ

ВСТУП

Актуальність. Тема роботи є актуальною через важливість аналізу даних у сучасному світі, що допомагає приймати обґрунтовані рішення на основі великих обсягів інформації. Розробка віртуального помічника аналітика даних значно полегшує процес збору, обробки та аналізу даних, забезпечуючи користувачів швидким і точним інструментом для виконання складних аналітичних завдань. Завдяки інтеграції з API ChatGPT, такий помічник може надавати більш гнучкі та персоналізовані рішення.

Об’єкт дослідження. Віртуальний помічник для аналітики даних у вигляді телеграм-бота.

Предмет дослідження. Методологія створення та використання віртуального помічника для аналітики даних з інтеграцією API ChatGPT.

Гіпотеза. Користувачі Telegram частіше використовуватимуть віртуального помічника для аналітики даних, який надає точну та персоналізовану інформацію, допомагаючи вирішувати аналітичні завдання швидше та ефективніше.

Новизна. Розробка віртуального помічника для аналітики даних у вигляді телеграм-бота спрощує процес аналізу даних, забезпечуючи користувачів швидким і точним інструментом. Інтеграція з API ChatGPT дозволяє підвищити рівень взаємодії з користувачами, надаючи більш точні та персоналізовані рекомендації.

Структура. Робота включає вступ, аналітичний огляд, постановку задачі, вибір методів розв’язання, інформаційне та програмне забезпечення, візуалізацію роботи додатку, висновки та список використаних джерел.

1 АНАЛІТИЧНИЙ ОГЛЯД

1.1 Аналіз принципів побудови віртуальних помічників для аналізу даних

Віртуальні помічники, призначені для аналізу даних, автоматизують процеси обробки та аналізу інформації. Це дозволяє користувачам швидко отримувати цінну інформацію та ухвалювати обґрунтовані рішення, спираючись на великі обсяги даних. Для успішної розробки таких помічників необхідно дотримуватися певних принципів та використовувати найкращі практики.

Основні принципи та аналіз побудови віртуальних помічників для аналізу даних:

- Збір даних**:** Використання інформації з різних джерел, таких як бази даних і API, є ключовим для точного аналізу. Важливо забезпечити високу якість та актуальність зібраних даних.
- Алгоритми: Застосування алгоритмів обробки природної мови (NLP) допомагає в розумінні запитів користувачів та генерації відповідей. Використання методів машинного навчання дозволяє аналізувати дані та виявляти тенденції.
- Інтерфейс**:** Створення інтуїтивно зрозумілого та зручного інтерфейсу для взаємодії з помічником є надзвичайно важливим. Це забезпечує легкий доступ до функцій та можливостей системи.
- Безпека**:** Захист даних користувачів та дотримання політик конфіденційності мають бути пріоритетними. Важливо забезпечити безпечне зберігання та передачу даних, щоб запобігти несанкціонованому доступу.

Дотримуючись цих принципів, віртуальні помічники можуть надавати точну та своєчасну інформацію, що підвищує комфорт і безпеку користувачів.

1.2 Інструменти для розробки

Для створення Telegram-ботів існує кілька платформ та інструментів, що спрощують процес розробки, налаштування та управління ботами. Ось деякі з них:

BotFather (рис. 1.1) - це офіційний бот від Telegram, який використовується для створення та управління іншими ботами в мережі Telegram.

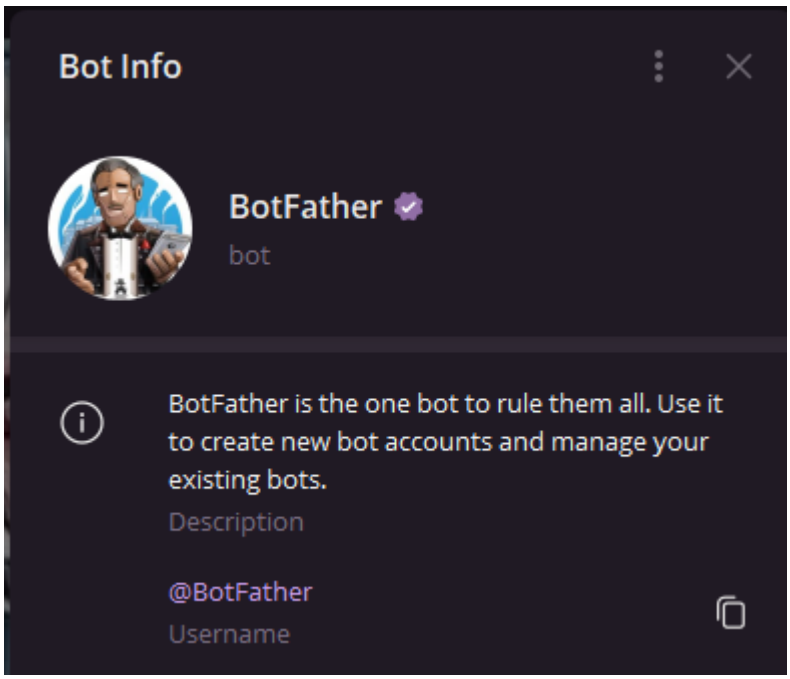

Рисунок 1.1 – BotFather офіційний бот від Telegram для створення ботів

Основні функції BotFather:

- Створення нового бота: BotFather дозволяє користувачам легко створювати нового бота, видаючи унікальний токен, який використовується для API Telegram. Цей токен є ключем до інтеграції вашого бота з Telegram API.
- Конфігурація ботів: За допомогою BotFather можна налаштувати такі параметри бота, як ім'я, опис, аватар, а також команди, які бот використовуватиме для взаємодії з користувачами.

- Оновлення налаштувань бота: BotFather надає можливість оновлювати налаштування бота в будь-який час, що дозволяє розробникам швидко вносити зміни без необхідності перезапуску або повторного розгортання бота.
- Отримання інформації про ботів: Розробники можуть запитувати статистику і загальну інформацію про свої боти через BotFather, що включає кількість активних користувачів, трафік та інші важливі метрики.

Переваги:

- Простота використання: BotFather має інтуїтивно зрозумілий інтерфейс, який дозволяє швидко і легко створювати та налаштовувати ботів без потреби у глибоких технічних знаннях.
- Офіційна підтримка: Як офіційний інструмент Telegram, BotFather забезпечує стабільну та надійну роботу з Telegram API.
- Широкий функціонал: BotFather надає всі необхідні інструменти для управління ботами, включаючи створення токенів, налаштування команд, оновлення налаштувань та отримання статистики.

Недоліки:

- Обмежений інтерфейс: Взаємодія через текстові команди може бути менш зручною порівняно з графічними інтерфейсами.
- Обмеження у функціоналі: BotFather надає базові можливості для налаштування ботів, але для більш складних завдань можуть знадобитися додаткові інструменти або бібліотеки.
- Відсутність інтеграції з іншими сервісами: BotFather працює виключно в екосистемі Telegram, тому для інтеграції з іншими сервісами можуть знадобитися додаткові налаштування та інструменти.

Telegram Bot API (рис. 1.2) - це API, який використовується для створення та управління ботами в мережі Telegram. Він відіграє ключову роль у розгортанні та адмініструванні Telegram ботів, що можуть бути розроблені за допомогою Python.

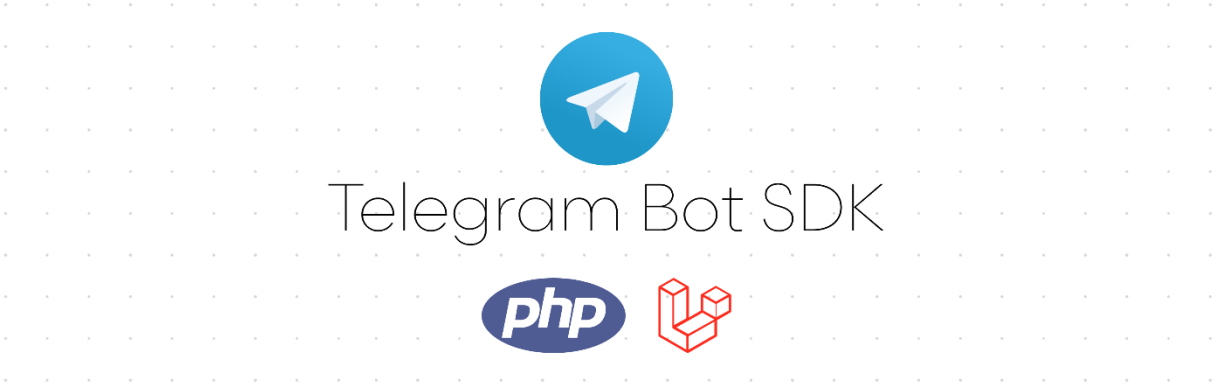

Рисунок 1.2 – Telegram Bot API

Основні функції Telegram Bot API:

- Надсилання та отримання повідомлень: Telegram Bot API дозволяє ботам надсилати та отримувати текстові повідомлення, фото, відео, документи та інші медіафайли.
- Обробка команд: API підтримує обробку команд користувачів, дозволяючи ботам реагувати на конкретні команди, такі як /start, /help та інші.
- Вебхуки: Telegram Bot API підтримує використання вебхуків для миттєвого отримання оновлень про нові повідомлення та події від користувачів.
- Керування чатами: API дозволяє ботам адмініструвати чати, додавати або видаляти учасників, змінювати налаштування групових чатів та багато іншого.
- Інтеграція з іншими сервісами: Telegram Bot API дозволяє інтегрувати ботів з іншими вебсервісами та базами даних для більш комплексної обробки даних та виконання завдань.

Переваги:

- Широкий функціонал: Telegram Bot API надає великий набір інструментів для створення функціональних ботів, включаючи роботу з медіафайлами, обробку команд та адміністрування чатів.

- Миттєві оновлення: Завдяки підтримці вебхуків, боти можуть миттєво реагувати на нові повідомлення та події, забезпечуючи швидку взаємодію з користувачами.
- Підтримка різних мов програмування: Хоча в прикладах часто використовується Python, Telegram Bot API підтримує розробку на багатьох інших мовах програмування, що дозволяє розробникам використовувати звичні інструменти.

Недоліки:

- Складність налаштування вебхуків: Налаштування вебхуків може бути складним для новачків, що вимагає додаткових знань про сервери та мережеві протоколи.
- Обмеження на кількість запитів: Telegram Bot API має обмеження на кількість запитів, які бот може виконати за певний період часу, що може стати проблемою для ботів з великою кількістю користувачів.
- Безпека даних: Необхідність зберігати токени доступу та інші конфіденційні дані може становити загрозу безпеці, якщо вони не будуть належним чином захищені.

Telegram Bot API є потужним інструментом для розробки ботів, який надає широкий спектр можливостей для взаємодії з користувачами та інтеграції з іншими сервісами. Завдяки своїм численним перевагам, він є незамінним для створення високоефективних та функціональних Telegram-ботів, попри деякі недоліки.

Dialogflow (рис. 1.3) - це платформа для створення розумних чат-ботів та голосових асистентів, розроблена компанією Google. Вона дозволяє розробникам легко інтегрувати свої додатки з різними платформами, включаючи Telegram, Slack, Facebook Messenger та інші. Dialogflow використовує передові

технології обробки природної мови (NLP) для забезпечення розуміння та обробки запитів користувачів.

Рисунок 1.3 – Dialogflow інтерфейс

Основні функції Dialogflow :

- Обробка природної мови (NLP): Dialogflow використовує передові алгоритми NLP для розуміння та обробки запитів користувачів, що дозволяє ботам надавати точні та релевантні відповіді.
- Підтримка множинних платформ: Платформа дозволяє інтегрувати чат-ботів з різними платформами, такими як Telegram, Facebook Messenger, Slack, веб-сайти та інші.
- Налаштування діалогів: Dialogflow надає зручний інтерфейс для налаштування сценаріїв діалогів, що дозволяє створювати складні діалоги з користувачами без необхідності писати код.
- Підтримка голосових асистентів: Платформа підтримує інтеграцію з голосовими асистентами, такими як Google Assistant та Amazon Alexa, що дозволяє створювати голосові інтерфейси для додатків.

- Аналітика та моніторинг: Dialogflow надає інструменти для аналітики та моніторингу, що дозволяє відстежувати ефективність ботів, аналізувати поведінку користувачів та покращувати взаємодію.

Переваги:

- Інтеграція з Google Cloud: Dialogflow легко інтегрується з іншими сервісами Google Cloud, що дозволяє використовувати потужні інструменти для зберігання та обробки даних.
- Простота налаштування: Завдяки інтуїтивному інтерфейсу, розробники можуть швидко створювати та налаштовувати ботів без необхідності глибоких знань у програмуванні.
- Потужні можливості NLP: Використання передових алгоритмів NLP забезпечує високу точність розпізнавання та обробки запитів користувачів

Недоліки:

- Залежність від Google: Використання Dialogflow передбачає залежність від екосистеми Google, що може бути небажаним для деяких організацій через питання конфіденційності та безпеки.
- Обмеження безкоштовного тарифу: Безкоштовний тариф має обмеження на кількість запитів та функціональні можливості, що може бути недостатньо для великих проєктів.
- Складність налаштування складних сценаріїв: Для налаштування складних діалогових сценаріїв може знадобитися більше часу та зусиль, особливо якщо сценарії включають багато умов та варіантів відповідей.

Dialogflow пропонує розумні рішення для створення чат-ботів та голосових асистентів з можливістю інтеграції з різними платформами та сервісами. Незважаючи на деякі обмеження, його переваги роблять його

привабливим вибором для розробників, які потребують ефективного інструменту для створення ботів.

OpenAI GPT-4 API (рис. 1.4) - це інтерфейс програмування додатків, розроблений компанією OpenAI, який дозволяє розробникам інтегрувати потужні можливості обробки природної мови моделі GPT-4 у свої додатки. Цей API надає можливість створювати розумні чат-боти, автоматизувати завдання з обробки тексту та забезпечувати високу якість взаємодії з користувачами.

Рисунок 1.4 – OpenAI GPT-4 API

Основні функції OpenAI GPT-4 API:

- Обробка природної мови (NLP): GPT-4 використовує передові алгоритми обробки природної мови для розуміння та генерації тексту, що дозволяє

створювати чат-ботів, які можуть відповідати на запити користувачів з високою точністю.

- Генерація тексту: API дозволяє генерувати текст на основі заданих підказок, що корисно для створення контенту, написання статей, генерації відповідей у чатах та інших завдань.
- Аналіз тексту: GPT-4 може аналізувати текст, виділяти ключові моменти, здійснювати резюмування та класифікацію тексту.
- Інтеграція з іншими сервісами: API дозволяє інтегрувати можливості GPT-4 з іншими вебсервісами та додатками, забезпечуючи комплексну обробку даних та виконання завдань.
- Підтримка множинних мов: GPT-4 підтримує обробку та генерацію тексту на багатьох мовах, що дозволяє створювати багатомовні додатки та сервіси

Переваги:

- Висока точність і якість: GPT-4 забезпечує високу точність та якість обробки тексту завдяки передовим алгоритмам обробки природної мови.
- Гнучкість у використанні: API дозволяє налаштовувати параметри генерації тексту, що забезпечує гнучкість у використанні та адаптації під конкретні завдання.
- Широкі можливості застосування: GPT-4 може використовуватися для різних завдань, включаючи створення чат-ботів, генерацію контенту, автоматизацію обробки тексту та багато іншого.

Недоліки:

- Висока вартість: Використання GPT-4 API може бути дорогим, особливо для великих проєктів з великою кількістю запитів.
- Залежність від інтернету: API працює через інтернет, що може бути проблемою в умовах поганого з'єднання або при необхідності обробки конфіденційних даних локально.

- Обмеження на використання: OpenAI встановлює певні обмеження на використання API, включаючи політики щодо етики та безпеки, що може обмежити деякі застосування.

OpenAI GPT-4 API є потужним інструментом для створення чат-ботів та автоматизації обробки тексту, що забезпечує широкий спектр можливостей для інтеграції з різними додатками та сервісами. Його переваги роблять його популярним вибором серед розробників, які шукають ефективні рішення для обробки природної мови, незважаючи на певні обмеження.

Landbot (рис. 1.5) - це платформа для створення інтерактивних чат-ботів, яка дозволяє легко автоматизувати взаємодію з клієнтами та підтримувати маркетингові кампанії. Вона орієнтована на створення ботів для вебсайтів, але також підтримує інтеграцію з такими платформами, як WhatsApp, Facebook Messenger та інші. Landbot надає інтуїтивно зрозумілий інтерфейс для створення ботів без необхідності програмування, що робить його привабливим рішенням для бізнесів різного розміру.

Рисунок 1.5 – Landbot інтерфейс

Основні функції Landbot:

- Створення чат-ботів: Landbot пропонує зручний інтерфейс для створення ботів за допомогою перетягування блоків. Користувачі можуть налаштовувати сценарії діалогів, автоматичні відповіді та інтеграцію з різними платформами.

- Мультиплатформенна підтримка: Платформа підтримує створення ботів для вебсайтів, WhatsApp, Facebook Messenger та інших платформ, забезпечуючи широкий охват аудиторії.
- Інтеграція з CRM та іншими інструментами: Landbot легко інтегрується з популярними CRM системами, такими як Salesforce та HubSpot, а також з іншими маркетинговими та аналітичними інструментами.
- Аналітика та звіти: Платформа надає потужні інструменти для аналітики та звітів, що дозволяють відстежувати ефективність ботів, аналізувати поведінку користувачів та оптимізувати маркетингові стратегії.
- Персоналізація: Landbot дозволяє налаштовувати персоналізовані повідомлення та сценарії для покращення взаємодії з користувачами.

Переваги:

- Інтуїтивно зрозумілий інтерфейс: Landbot пропонує зручний інтерфейс для створення ботів без необхідності програмування, що робить його доступним для широкого кола користувачів.
- Широкі можливості інтеграції: Платформа підтримує інтеграцію з різними CRM системами та маркетинговими інструментами, що забезпечує комплексний підхід до управління клієнтами.
- Потужні аналітичні інструменти: Landbot надає розширені можливості для аналітики, що дозволяє бізнесам відстежувати ефективність ботів та оптимізувати взаємодію з клієнтами.

Недоліки:

- Обмеження безкоштовного тарифу: Безкоштовний тариф має обмеження на кількість функцій та інтеграцій, що може не відповідати потребам великих проєктів.
- Висока вартість преміум-планів: Використання повного функціоналу Landbot може бути дорогим, особливо для малого бізнесу або стартапів з обмеженим бюджетом.
- Залежність від інтернету: Платформа працює через інтернет, що може бути проблемою в умовах поганого з'єднання або при необхідності обробки конфіденційних даних локально.

Landbot є ефективним рішенням для створення інтерактивних чат-ботів, пропонуючи численні можливості для інтеграції з різними платформами та сервісами. Попри певні обмеження, його переваги роблять його привабливим для бізнесів, які шукають потужні інструменти для взаємодії з клієнтами.

Ці інструменти надають можливості для створення ботів на різних рівнях складності, від простих текстових ботів до більш складних інтерактивних рішень.

1.3 Постановка задачі

Розробка віртуального помічника для аналітики даних вимагає чіткого розуміння кінцевих цілей та визначення конкретних завдань, які повинен виконувати бот. Цей розділ окреслює основні завдання та етапи розробки такого бота, що виконується на мові програмування Python із використанням фреймворків та інструментів, описаних у попередньому розділі.

Ключові задачі для віртуального помічника:

1. Вибір інструментів розробки:

Огляд та вибір технологічного стеку: Вибір Python як основної платформи для розробки бота, що забезпечить кросплатформенність та високу продуктивність.

Використання Telegram Bot API: Цей API буде використаний для створення бота, який може взаємодіяти з користувачами через платформу Telegram.

2. Розробка бази даних:

Створення моделей даних: Визначення структур даних, необхідних для зберігання інформації, яку буде обробляти бот.

Вибір системи управління базами даних (СУБД): Вибір MongoDB або іншої СУБД, яка інтегрується з Python, для зберігання і обробки даних.

3. Розробка алгоритму роботи бота:

Реалізація алгоритмів збору та аналізу даних: Інтеграція бота з зовнішніми джерелами даних та використання алгоритмів машинного навчання для аналізу цих даних.

Створення логіки взаємодії з користувачем: Розробка механізмів прийому запитів від користувачів та надання відповідей на основі аналізу даних.

4. Інтерфейс користувача:

Розробка користувацького інтерфейсу: Визначення способів взаємодії користувача з ботом, включаючи команди, кнопки та інші елементи інтерфейсу.

5. Тестування та деплоймент:

Тестування функціональності та безпеки: Здійснення різних видів тестування, включаючи модульне тестування, інтеграційне тестування та стрес-тестування.

Деплоймент бота: Розгортання бота на сервері або хмарному сервісі для забезпечення його доступності користувачам.

Ці інструменти надають можливості для створення ботів на різних рівнях складності, від простих текстових ботів до більш складних інтерактивних рішень. Вони допомагають автоматизувати процеси взаємодії з клієнтами, підвищують ефективність маркетингових кампаній та забезпечують зручний інтерфейс для управління ботами. Незалежно від вибраного інструменту, кожен

з них може стати потужним помічником у досягненні бізнес-цілей та покращенні користувацького досвіду.

.

2 ВИБІР МЕТОДУ РОЗВ’ЯЗАННЯ ЗАДАЧІ

2.1 Проектування бази даних

Проектування бази даних є важливою складовою при створенні систем, що залежать від ефективного управління та аналізу даних. У цьому розділі детально розглянемо ключові аспекти, які необхідно враховувати під час проектування бази даних для телеграм-бота, що використовує ChatGPT.

Аналіз вимог

Перед початком проектування необхідно чітко визначити та проаналізувати вимоги до бази даних:

Визначення даних: Які дані будуть збиратися, зберігатися та оброблятися? Це можуть бути дані з зовнішніх джерел, користувацькі дані, історія чатів, налаштування користувачів тощо.

Вимоги до обробки: Які операції будуть виконуватися з даними? Чи будуть дані просто зберігатися, чи потрібна складна обробка (наприклад, пошук, аналіз)?

Безпека даних: Які вимоги до безпеки і конфіденційності? Це включає контроль доступу, шифрування та інші механізми захисту даних.

Моделювання даних

Після збору вимог наступним кроком є створення моделі даних:

ER-діаграми: Використання сутнісно-зв'язкових діаграм для візуалізації сутностей бази даних та зв'язків між ними. Це допомагає виявити можливі зв'язки та залежності, які повинні бути враховані під час проектування.

Схеми бази даних: Детальне проектування схеми бази даних, включаючи колекції, поля, типи даних, ключі та інші атрибути.

Структура бази даних для телеграм-бота

Цей бот використовує MongoDB як базу даних. Основні колекції включають:

users: зберігає інформацію про користувачів Telegram.

chats: зберігає історію чатів та повідомлень.

settings: зберігає налаштування бота та користувача.

Приклад взаємодії з базою даних у коді:

```
from pymongo import MongoClient

client = MongoClient('localhost', 27017)
db = client['chatgpt_telegram_bot']
users_collection = db['users']
chats_collection = db['chats']
settings_collection = db['settings']
```

Рисунок 2.1 – Приклад взаємодії з базою даних у коді

Нормалізація

Нормалізація бази даних важлива для забезпечення її ефективності та уникнення редундантності даних:

Перша нормальна форма (1NF): Усунення повторюваних груп в одній таблиці.

Друга нормальна форма (2NF): Усунення залежностей, які залежать від частини первинного ключа.

Третя нормальна форма (3NF): Усунення залежностей між неключовими полями.

Вибір СУБД

Вибір системи управління базами даних залежить від технічних вимог, розміру проекту та доступних ресурсів:

NoSQL СУБД: MongoDB, яка використовується у цьому проекті, є прикладом NoSQL бази даних, яка добре підходить для зберігання неструктурованих даних та має високу масштабованість.

Заходи безпеки

Розробка політик безпеки та механізмів для захисту даних є обов'язковою частиною проектування:

Шифрування: Захист даних за допомогою шифрування під час зберігання та передачі.

Контроль доступу: Впровадження систем контролю доступу для забезпечення того, що тільки авторизовані користувачі мають доступ до чутливих даних.

Аудит та моніторинг: Ведення журналів доступу до даних та моніторинг неавторизованих спроб доступу.

Комплексний підхід до проектування бази даних забезпечує не тільки її ефективність і швидкість роботи, але й високий рівень безпеки та масштабованість системи.

2.2 Огляд інструментів для розробки

1. PyCharm (рис. 2.2) - інтегроване середовище розробки, яке забезпечує зручне написання коду на Python та пропонує безліч корисних інструментів.

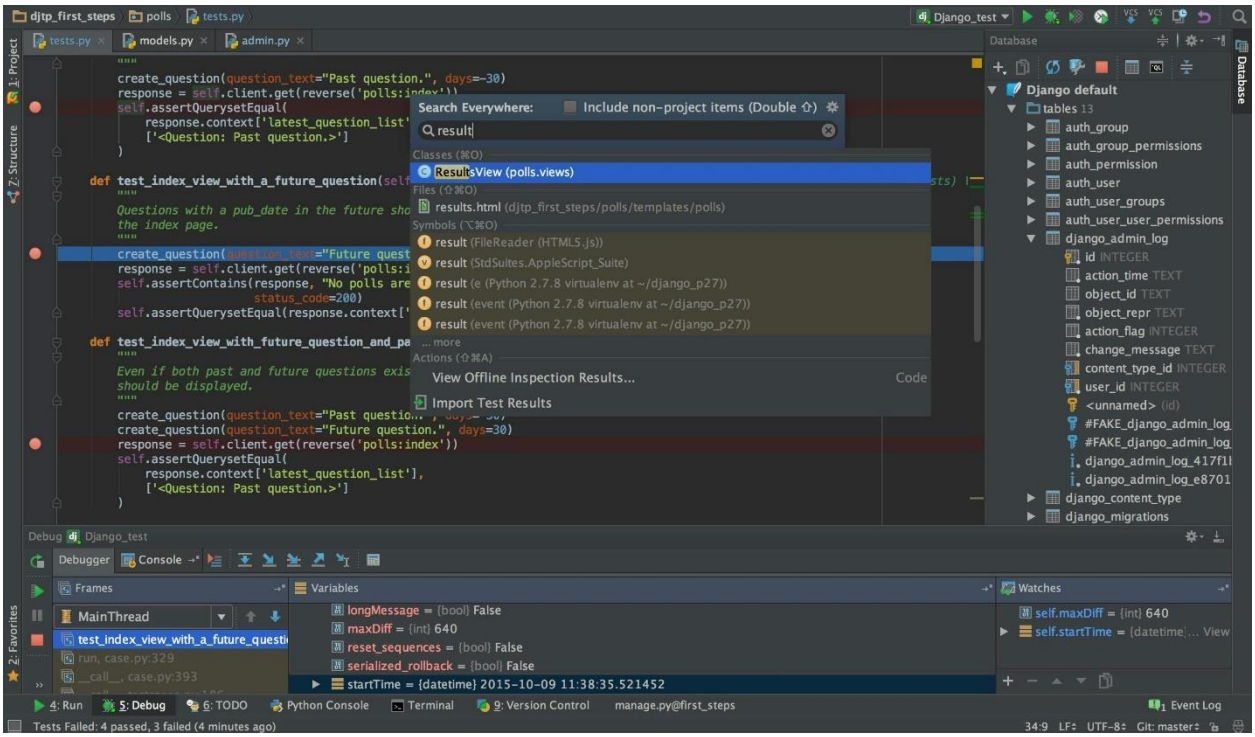

Рисунок 2.2 – Інтерфейс PyCharm

Можливості PyCharm:

- Аналіз коду;
- Інтеграція з Git;
- Редагування та автодоповнення коду;
- Відладка та тестування;
- Розширюваність;
- Наукові обчислення.

До переваг можна віднести:

- Простий у використанні;
- Активна спільнота;
- Безкоштовний та платний варіанти;
- Регулярні оновлення;
- Потужні функції.

Недоліків не так і багато, але вони є:

- Відсутність деяких функцій;
- Складність деяких функцій;
- Високі системні вимоги[9].

2. MongoDB (рис. 2.3) - зручний інструмент для роботи з NoSQL базою даних, який має інтуїтивно зрозумілий інтерфейс користувача.

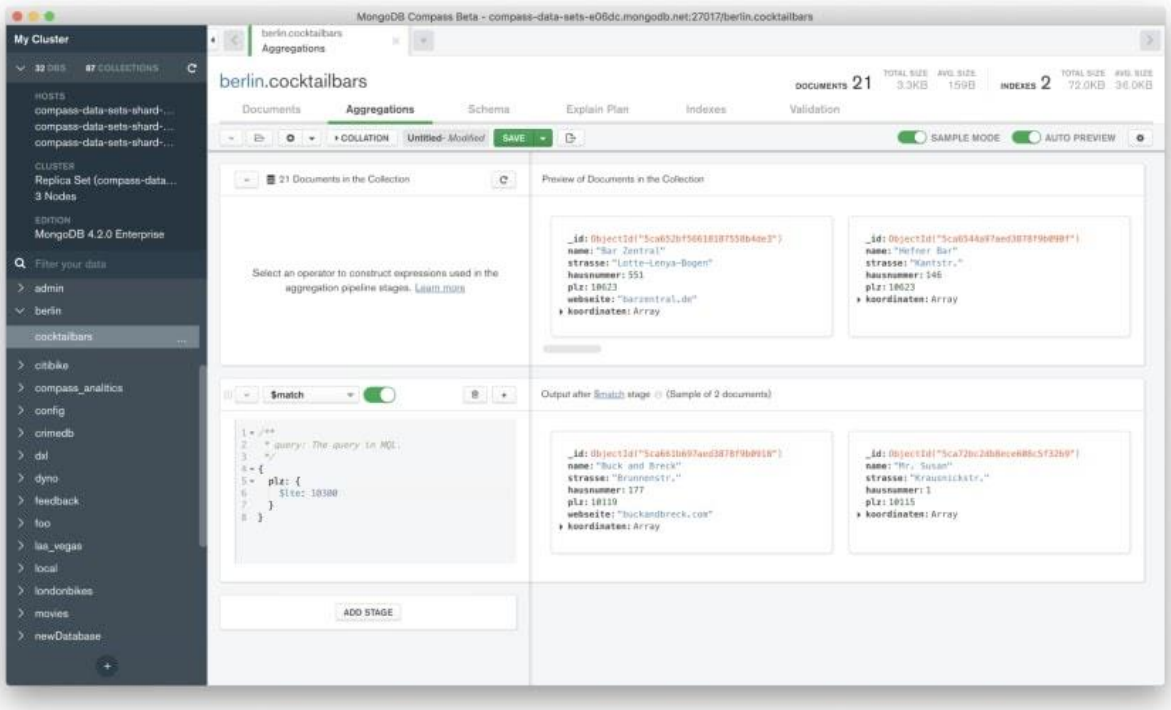

Рисунок 2.3 – Інтерфейс MongoDB

Можливості MongoDB:

- Візуалізація даних;
- Імпорт та експорт даних;
- Підтримка віддалених серверів;
- Виконання запитів до бази даних;
- Створення та керування об'єктами бази даних.

Переваги:

- Легкість у використанні;

- Активна спільнота;
- Регулярні оновлення;
- Безкоштовний та відкритий доступ;
- Потужні функції.

Недоліки:

- Обмежені можливості порівняно з комерційними інструментами;
- Можливість збоїв та проблем з продуктивністю.

2.3 Клієнтська частина система

Для взаємодії користувача з нашим ботом ми будемо використовувати комбінацію рукописного вводу та команд, які пропонує сам Telegram.

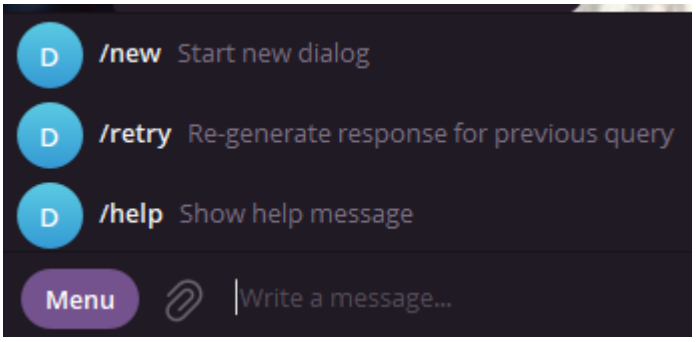

Рисунок 2.4 – Приклад меню з командами

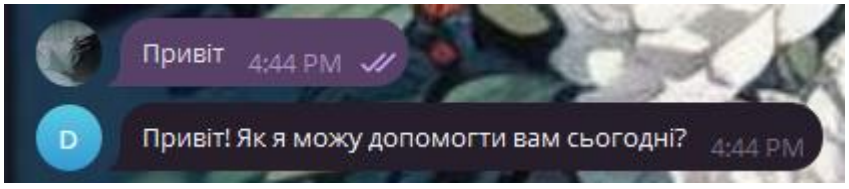

Рисунок 2.5 – Приклад взаємодії за допомогою вводу користувача

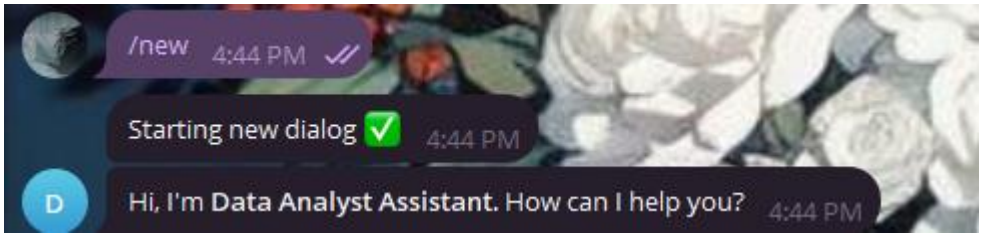

Рисунок 2.6 – Приклад взаємодії за допомогою команд Telegram

Рисунок 2.7 – Приклад взаємодії за допомогою вводу користувача

3 ІНФОРМАЦІЙНЕ ТА ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ

3.1 Розробка дизайну

Use Case (сценарій використання) — це перелік дій, які користувач може виконувати для роботи з програмою або додатком для досягнення поставленої мети. Тестування варіантів використання виконується для виявлення логічних прогалин або помилок у веб-програмі, які важко виявити під час тестування певних модулів або частин веб-програми. У основі своїй варіанти використання описують, що саме система робить, а не як.

Розроблено Use Case діаграму (рис. 3.1) для нашої інформаційної системи для аналітики данних у вигляді телеграм боту.

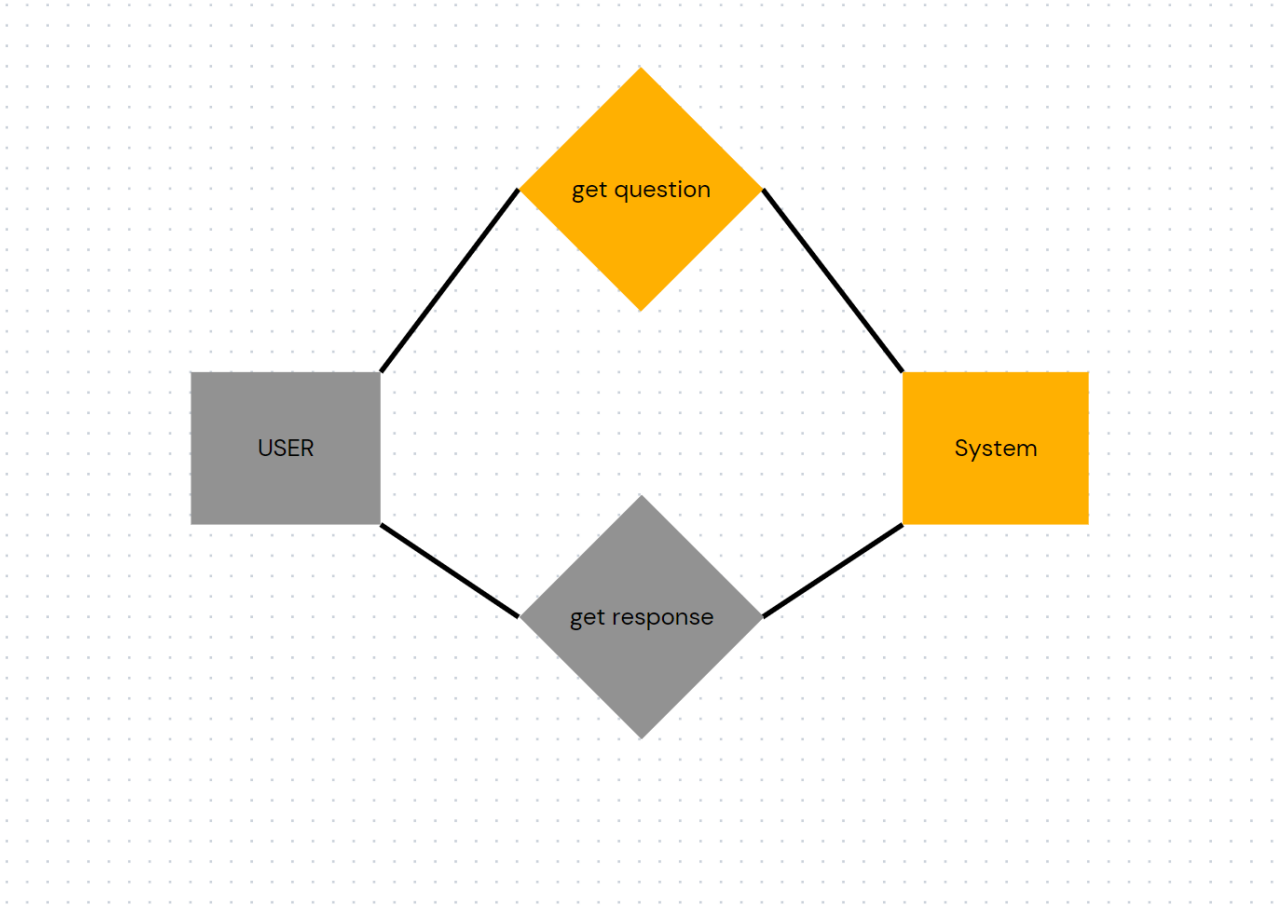

Рисунок 3.1 – Use Case діаграма

З нашої діаграми варіантів використання визначимо умови для взаємодії з ботом:

1. Реєстрація нового користувача: Користувач надсилає команду /start, після чого бот реєструє його у системі та ініціалізує новий діалог.

2. Отримання допомоги: Користувач надсилає команду /help, і бот надає перелік доступних команд і можливостей.

3. Початок нового діалогу: Користувач надсилає команду /new, що дозволяє розпочати новий діалог, видаляючи контекст попереднього.

4. Повторна генерація відповіді: Користувач надсилає команду /retry, щоб бот повторно згенерував останню відповідь у разі потреби іншого результату.

3.2 Програмна реалізація

Опис основних файлів: bot.py, config.py, database.py, openai_utils.py, які створені для роботи інформаційної системи аналізу даних у вигляді телеграм-бота. (Додаток):

1) bot.py – основний додаток телеграм-бота.

Імпортуються необхідні модулі, такі як asyncio для асинхронного програмування, telegram для взаємодії з API Telegram, а також додаткові модулі, що забезпечують функціональність бота (наприклад, openai для роботи з OpenAI API, config для налаштувань, database для взаємодії з базою даних, openai_utils для обробки запитів до OpenAI).

```python
import io
import logging
import asyncio
import traceback
import html
import json
from datetime import datetime
import openai

import telegram
from telegram import (
    Update,
    User,
    InlineKeyboardButton,
    InlineKeyboardMarkup,
    BotCommand
)
from telegram.ext import (
    Application,
    ApplicationBuilder,
    CallbackContext,
    CommandHandler,
    MessageHandler,
    CallbackQueryHandler,
    AIORateLimiter,
    filters
)
from telegram.constants import ParseMode, ChatAction

import config
import database
import openai_utils

import base64
```

Створюється об'єкт Database для роботи з базою даних та налаштовується логування.

```python
# setup
db = database.Database()
logger = logging.getLogger(__name__)
```

У функції start_handle відбувається реєстрація користувача, якщо він не існує, та ініціалізація нового діалогу.

```python
async def start_handle(update: Update, context: CallbackContext):
    await register
```

користувача, якщо він не існує, та ініціалізація нового діалогу.

```python
async def start_handle(update: Update, context: CallbackContext):
    await register_user_if_not_exists(update, context, update.message.from_user)
    user_id = update.message.from_user.id

    db.set_user_attribute(user_id, key: "last_interaction", datetime.now())
    db.start_new_dialog(user_id)

    reply_text = "Hi! I'm <b>ChatGPT</b> bot implemented with OpenAI API \n\n"
    reply_text += HELP_MESSAGE

    await update.message.reply_text(reply_text, parse_mode=ParseMode.HTML)
    await show_chat_modes_handle(update, context)
```

Далі створюються обробники команд, такі як /help, /retry, /new, які відповідають за різні функції бота. Наприклад, обробник команди /help виводить користувачеві повідомлення з доступними командами

```python
async def help_handle(update: Update, context: CallbackContext):
    await register_user_if_not_exists(update, context, update.message.from_user)
    user_id = update.message.from_user.id
    db.set_user_attribute(user_id, key: "last_interaction", datetime.now())
    await update.message.reply_text(HELP_MESSAGE, parse_mode=ParseMode.HTML)
```

Запуск програми здійснюється за допомогою функції run_bot, яка налаштовує і запускає Telegram бота.

```python
def run_bot() -> None:
    application = (
        ApplicationBuilder()
        .token(config.telegram_token)
        .concurrent_updates(True)
        .rate_limiter(AIORateLimiter(max_retries=5))
        .http_version("1.1")
        .get_updates_http_version("1.1")
        .post_init(post_init)
        .build()
    )

    # add handlers
    user_filter = filters.ALL
    if len(config.allowed_telegram_usernames) > 0:
        usernames = [x for x in config.allowed_telegram_usernames if isinstance(x, str)]
        any_ids = [x for x in config.allowed_telegram_usernames if isinstance(x, int)]
        user_ids = [x for x in any_ids if x > 0]
        group_ids = [x for x in any_ids if x < 0]
        user_filter = filters.User(username=usernames) | filters.User(user_id=user_ids) | filters.Chat(chat_id=group_ids)

    application.add_handler(CommandHandler( command: "start", start_handle, filters=user_filter))
    application.add_handler(CommandHandler( command: "help", help_handle, filters=user_filter))
    application.add_handler(CommandHandler( command: "help_group_chat", help_group_chat_handle, filters=user_filter))

    application.add_handler(MessageHandler(filters.TEXT & ~filters.COMMAND & user_filter, message_handle))
    application.add_handler(MessageHandler(filters.PHOTO & ~filters.COMMAND & user_filter, message_handle))
    application.add_handler(MessageHandler(filters.VIDEO & ~filters.COMMAND & user_filter, unsupport_message_handle))
    application.add_handler(MessageHandler(filters.Document.ALL & ~filters.COMMAND & user_filter, unsupport_message_handle))
    application.add_handler(CommandHandler( command: "retry", retry_handle, filters=user_filter))
    application.add_handler(CommandHandler( command: "new", new_dialog_handle, filters=user_filter))
    application.add_handler(CommandHandler( command: "cancel", cancel_handle, filters=user_filter))

    application.add_handler(MessageHandler(filters.VOICE & user_filter, voice_message_handle))

    application.add_error_handler(error_handle)

    # start the bot
    application.run_polling()
```

Запуск програми.

2) config.py – файл конфігурацій.

Імпортуються необхідні модулі для роботи з конфігураційними файлами: yaml, dotenv та Path з бібліотеки pathlib.

```python
import yaml
import dotenv
from pathlib import Path
```

Створюється об'єкт ConfigParser для читання конфігураційного файлу config.yml, звідки витягуються необхідні параметри, такі як токен для Telegram бота, ключ API для OpenAI тощо.

```python
# load yaml config
with open(config_dir / "config.yml", 'r', encoding='utf-8') as f:
    config_yaml = yaml.safe_load(f)

# load .env config
config_env = dotenv.dotenv_values(config_dir / "config.env")

# config parameters
telegram_token = config_yaml["telegram_token"]
openai_api_key = config_yaml["openai_api_key"]
openai_api_base = config_yaml.get("openai_api_base", None)
allowed_telegram_usernames = config_yaml["allowed_telegram_usernames"]
new_dialog_timeout = config_yaml["new_dialog_timeout"]
enable_message_streaming = config_yaml.get("enable_message_streaming", True)
return_n_generated_images = config_yaml.get("return_n_generated_images", 1)
image_size = config_yaml.get("image_size", "512x512")
n_chat_modes_per_page = config_yaml.get("n_chat_modes_per_page", 5)
mongodb_uri = f"mongodb://mongo:{config_env['MONGODB_PORT']}"

# chat_modes
with open(config_dir / "chat_modes.yml", 'r', encoding='utf-8') as f:
    chat_modes = yaml.safe_load(f)
```

3) database.py – файл роботи з базою даних.

Імпортуються модулі pymongo для роботи з базою даних MongoDB, а також uuid для створення унікальних ідентифікаторів діалогів.

```python
import pymongo
import uuid
from datetime import datetime
```

Створюється клас Database, який містить методи для взаємодії з базою даних, такі як перевірка існування користувача, додавання нового користувача, запуск нового діалогу, отримання та встановлення атрибутів користувача.

```python
class Database:
    def __init__(self):
        self.client = pymongo.MongoClient(config.mongodb_uri)
        self.db = self.client["chatgpt_telegram_bot"]

        self.user_collection = self.db["user"]
        self.dialog_collection = self.db["dialog"]

    def check_if_user_exists(self, user_id: int, raise_exception: bool = False):
        if self.user_collection.count_documents({"_id": user_id}) > 0:
            return True
        else:
            if raise_exception:
                raise ValueError(f"User {user_id} does not exist")
            else:
                return False
```

```python
    def add_new_user(
        self,
        user_id: int,
        chat_id: int,
        username: str = "",
        first_name: str = "",
        last_name: str = "",
    ):
        user_dict = {
            "_id": user_id,
            "chat_id": chat_id,

            "username": username,
            "first_name": first_name,
            "last_name": last_name,
        }
```

4) openai_utils.py – файл для роботи з OpenAI API.

Імпортуються необхідні модулі для роботи з OpenAI API та обробки запитів. Використовується бібліотека openai для взаємодії з API, а також tiktoken для підрахунку токенів.

```python
import base64
from io import BytesIO
import config
import logging

import tiktoken
import openai
```

Створюється клас ChatGPT, який містить методи для відправки повідомлень, генерації зображень та транскрибування аудіо. Наприклад, метод send_message відповідає за відправку текстових повідомлень до моделі OpenAI та отримання відповіді.

```python
class ChatGPT:
    def __init__(self, model="gpt-3.5-turbo"):
        assert model in {"text-davinci-003", "gpt-3.5-turbo-16k", "gpt-3.5-turbo", "gpt-4", "gpt-4-1106-preview", "gpt-4-vision-preview"}, f"Unknown model: {model}"
        self.model = model

    async def send_message(self, message, dialog_messages=[], chat_mode="assistant"):
        if chat_mode not in config.chat_modes.keys():
            raise ValueError(f"Chat mode {chat_mode} is not supported")

        n_dialog_messages_before = len(dialog_messages)
        answer = None
        while answer is None:
            try:
                if self.model in {"gpt-3.5-turbo-16k", "gpt-3.5-turbo", "gpt-4", "gpt-4-1106-preview", "gpt-4-vision-preview"}:
                    messages = self._generate_prompt_messages(message, dialog_messages, chat_mode)

                    r = await openai.ChatCompletion.acreate(
                        model=self.model,
                        messages=messages,
                        **OPENAI_COMPLETION_OPTIONS
                    )
                    answer = r.choices[0].message["content"]
                elif self.model == "text-davinci-003":
                    prompt = self._generate_prompt(message, dialog_messages, chat_mode)
                    r = await openai.Completion.acreate(
                        engine=self.model,
                        prompt=prompt,
                        **OPENAI_COMPLETION_OPTIONS
                    )
                    answer = r.choices[0].text
                else:
                    raise ValueError(f"Unknown model: {self.model}")

                answer = self._postprocess_answer(answer)
                n_input_tokens, n_output_tokens = r.usage.prompt_tokens, r.usage.completion_tokens
            except openai.error.InvalidRequestError as e:  # too many tokens
                if len(dialog_messages) == 0:
                    raise ValueError("Dialog messages is reduced to zero, but still has too many tokens to make completion") from e
```

```python
                # forget first message in dialog_messages
                dialog_messages = dialog_messages[1:]

        n_first_dialog_messages_removed = n_dialog_messages_before - len(dialog_messages)

        return answer, (n_input_tokens, n_output_tokens), n_first_dialog_messages_removed
```

Таким чином, структура та функціонал телеграм бота описуються детально, включаючи основні файли та їхні ключові частини, що забезпечують роботу інформаційної системи аналізу даних.

3.3 Робота додатку

У процесі створення додатку отримана працездатна інформаційна система аналізу даних у вигляді телеграм бота.

Вводимо команду /start, щоб почати взаємодію з ботом. Бот привітає користувача та виведе список доступних команд (рис. 4.1).

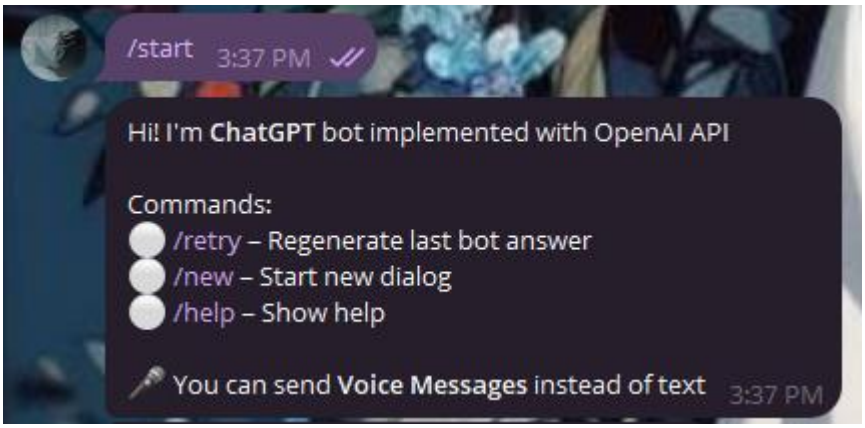

Рисунок 4.1 – Привітання бота та список команд

Вводимо команду /new, щоб розпочати новий діалог (рис. 4.2). Бот розпочне новий діалог з користувачем, видаляючи контекст попереднього діалогу.

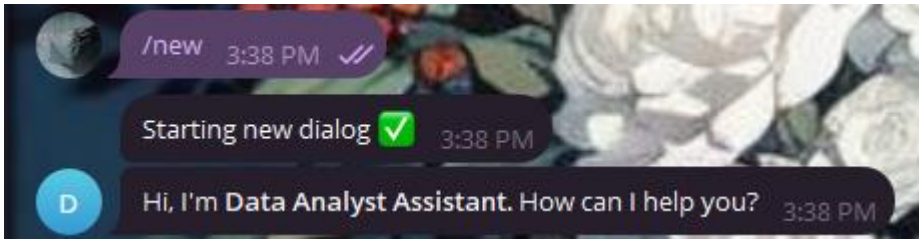

Рисунок 4.2 – Розпочати новий діалог

Використовуємо команду /retry для повторної генерації останньої відповіді бота, якщо потрібно отримати інший результат (рис. 4.3).

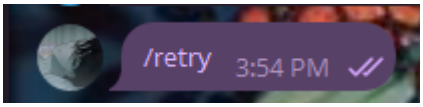

Рисунок 4.3 – Повторна генерація відповіді

Надсилаємо текстове повідомлення, щоб отримати відповідь від бота. Бот генерує відповідь на основі введеного тексту та моделі, яка використовується (рис. 4.4).

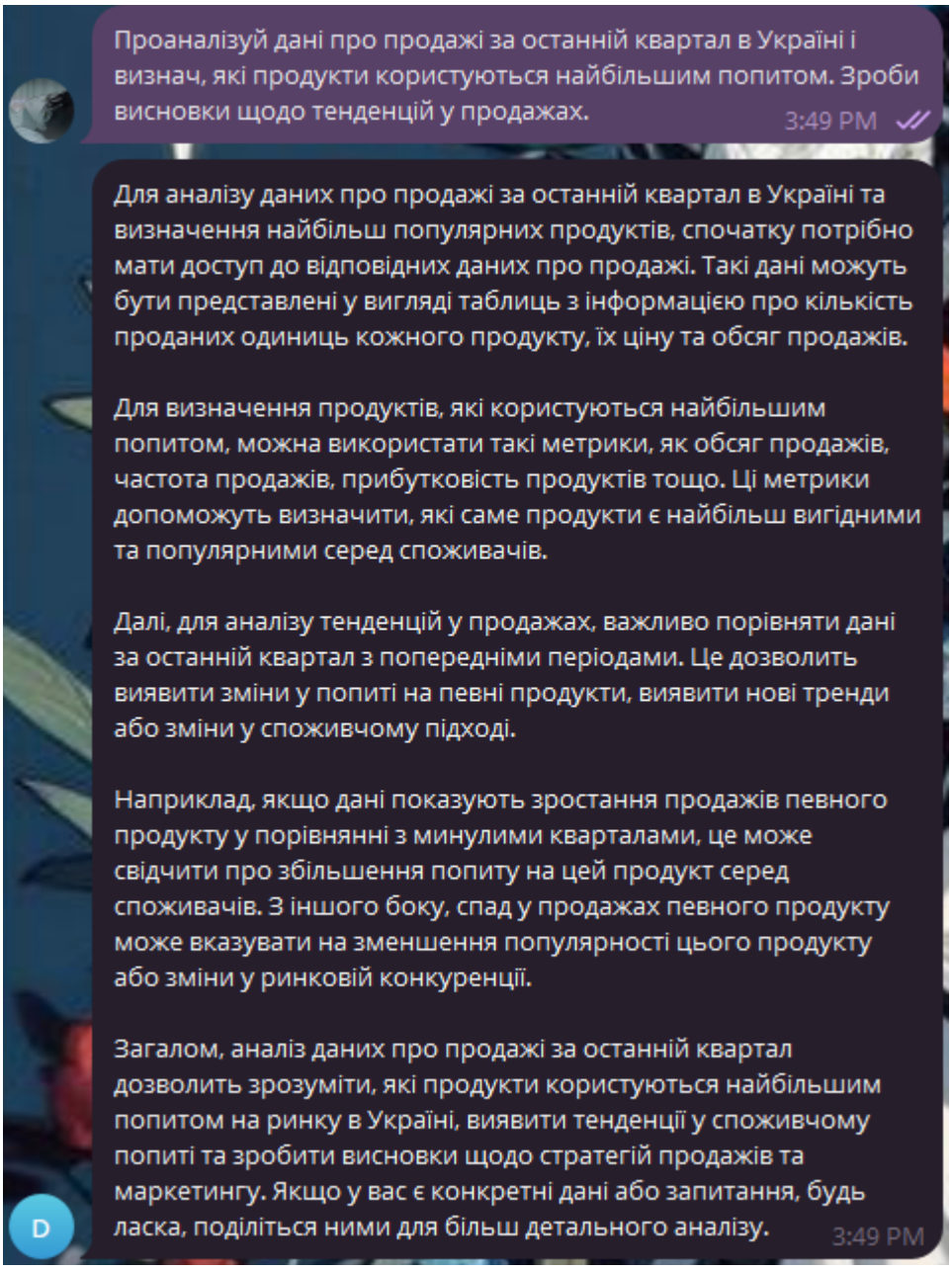

Рисунок 4.4 – Генерація відповіді бота

Надсилаємо голосове повідомлення, яке бот транскрибує та відповідає на основі транскрибованого тексту (рис. 4.5).

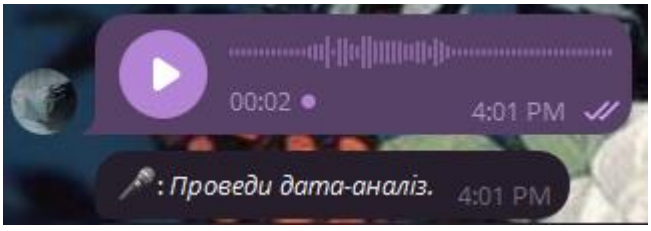

Рисунок 4.5 – Транскрипція та відповідь на голосове повідомлення

Таким чином, телеграм бот надає зручний інтерфейс для взаємодії з користувачем, дозволяючи отримувати аналітичні дані та відповіді на основі введених повідомлень.

ВИСНОВКИ

Під час виконання кваліфікаційної роботи створено інформаційну систему у вигляді телеграм-бота для аналітики даних за допомогою мови програмування Python.

- Обрані інструменти розробки: Python як мову програмування, MongoDB для управління базою даних та OpenAI API для генерації відповідей та аналізу даних.
- Розробка бази даних: Створено базу даних для зберігання інформації про місцезнаходження користувача.
- Розробка структури бота: Створено структуру телеграм-бота, яка включає основні файли bot.py, config.py, database.py, openai_utils.py для забезпечення функціональності бота.
- Розробка алгоритму роботи бота: Створено алгоритм, який дозволяє користувачам взаємодіяти з ботом за допомогою текстових та голосових повідомлень, отримувати аналітичні дані та зображення на основі запитів.
- Тестування бота: Проведено тестування телеграм-бота, який працює безперебійно, зручно для користувачів, відповідає всім поставленим завданням та забезпечує високу якість обробки запитів.

В результаті користувачі можуть зручно та безперебійно отримувати аналітичні дані через інформаційну систему у вигляді телеграм-бота. Проект у майбутньому буде розширюватися, включаючи додавання нових функцій та можливостей для покращення взаємодії з користувачами.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. "The Python Tutorial".
   URL:https://docs.python.org/3/tutorial/index.html
   (дата звернення: 10.06.2023).
2. "Building Telegram Bot with Python-Telegram-Bot: A Comprehensive Guide".
   URL : https://medium.com/@moraneus/building-telegram-bot-with-python-telegram-bot-a-comprehensive-guide-7e33f014dc79
   (дата звернення: 12.06.2023).
3. MongoDB. Все, що потрібно знати про бази даних для початківців.
   URL: https://dan-it.com.ua/uk/blog/vse-shho-potribno-znati-pro-bazi-danih-dlja-pochatkivciv-mysql-postgresql-mongodb/
   (дата звернення: 19.06.2023).
4. All you need to know about Chatbot Frameworks. Maruti Techlabs. URL: https://www.linkedin.com/pulse/all-you-need-know-chatbot-frameworks-mitul-makadia
   (дата звернення: 09.06.2023).
5. Чат-боти для бізнесу: що це таке і як їх можна використовувати. URL: https://ag.marketing/blog/chat-boti-dlya-biznesu/
   (дата звернення: 09.06.2023).
6. Введення в Dialogflow: налаштування та обмеження.
   URL: https://magdamagla.com/posts/dialogflow-settings/df-settings/
   (дата звернення: 12.04.2024).
7. Основи Pycharm.
   URL: https://w3schoolsua.github.io/hyperskill/pycharm-basics.html#gsc.tab=0
   (дата звернення: 12.04.2024).
8. Програмне забезпечення Docker. Docker overview
   URL: https://docs.docker.com/get-started/
   (дата звернення: 12.04.2024).

9. OpenAI documentation.
URL: https://platform.openai.com/docs/overview
(дата звернення: 13.04.2024).

ДОДАТОК

bot.py

```python
async def start_handle(update: Update, context: CallbackContext):
    await register_user_if_not_exists(update, context, update.message.from_user)
    user_id = update.message.from_user.id

    db.set_user_attribute(user_id, key: "last_interaction", datetime.now())
    db.start_new_dialog(user_id)

    reply_text = "Hi! I'm <b>ChatGPT</b> bot implemented with OpenAI API \n\n"
    reply_text += HELP_MESSAGE

    await update.message.reply_text(reply_text, parse_mode=ParseMode.HTML)
    await show_chat_modes_handle(update, context)


async def help_handle(update: Update, context: CallbackContext):
    await register_user_if_not_exists(update, context, update.message.from_user)
    user_id = update.message.from_user.id
    db.set_user_attribute(user_id, key: "last_interaction", datetime.now())
    await update.message.reply_text(HELP_MESSAGE, parse_mode=ParseMode.HTML)


async def help_group_chat_handle(update: Update, context: CallbackContext):
    await register_user_if_not_exists(update, context, update.message.from_user)
    user_id = update.message.from_user.id
    db.set_user_attribute(user_id, key: "last_interaction", datetime.now())

    text = HELP_GROUP_CHAT_MESSAGE.format(bot_username="@" + context.bot.username)

    await update.message.reply_text(text, parse_mode=ParseMode.HTML)
    await update.message.reply_video(config.help_group_chat_video_path)
```

```python
async def retry_handle(update: Update, context: CallbackContext):
    await register_user_if_not_exists(update, context, update.message.from_user)
    if await is_previous_message_not_answered_yet(update, context): return

    user_id = update.message.from_user.id
    db.set_user_attribute(user_id, key: "last_interaction", datetime.now())

    dialog_messages = db.get_dialog_messages(user_id, dialog_id=None)
    if len(dialog_messages) == 0:
        await update.message.reply_text("No message to retry 🤷‍♂️")
        return

    last_dialog_message = dialog_messages.pop()
    db.set_dialog_messages(user_id, dialog_messages, dialog_id=None)  # last message was removed from the context

    await message_handle(update, context, message=last_dialog_message["user"], use_new_dialog_timeout=False)
```

config.py

```
config_dir = Path(__file__).parent.parent.resolve() / "config"

with open(config_dir / "config.yml", 'r', encoding='utf-8') as f:
    config_yaml = yaml.safe_load(f)

config_env = dotenv.dotenv_values(config_dir / "config.env")

telegram_token = config_yaml["telegram_token"]
openai_api_key = config_yaml["openai_api_key"]
openai_api_base = config_yaml.get("openai_api_base", None)
allowed_telegram_usernames = config_yaml["allowed_telegram_usernames"]
new_dialog_timeout = config_yaml["new_dialog_timeout"]
enable_message_streaming = config_yaml.get("enable_message_streaming", True)
return_n_generated_images = config_yaml.get("return_n_generated_images", 1)
image_size = config_yaml.get("image_size", "512x512")
n_chat_modes_per_page = config_yaml.get("n_chat_modes_per_page", 5)
mongodb_uri = f"mongodb://mongo:{config_env['MONGODB_PORT']}"

with open(config_dir / "chat_modes.yml", 'r', encoding='utf-8') as f:
    chat_modes = yaml.safe_load(f)

with open(config_dir / "models.yml", 'r', encoding='utf-8') as f:
    models = yaml.safe_load(f)
```

Database.py

```python
class Database:
    def __init__(self):
        self.client = pymongo.MongoClient(config.mongodb_uri)
        self.db = self.client["chatgpt_telegram_bot"]

        self.user_collection = self.db["user"]
        self.dialog_collection = self.db["dialog"]

    def check_if_user_exists(self, user_id: int, raise_exception: bool = False):
        if self.user_collection.count_documents({"_id": user_id}) > 0:
            return True
        else:
            if raise_exception:
                raise ValueError(f"User {user_id} does not exist")
            else:
                return False

    def add_new_user(
        self,
        user_id: int,
        chat_id: int,
        username: str = "",
        first_name: str = "",
        last_name: str = "",
    ):
        user_dict = {
            "_id": user_id,
            "chat_id": chat_id,

            "username": username,
            "first_name": first_name,
            "last_name": last_name,
        }
```

database.py

```python
async def send_message(self, message, dialog_messages=[], chat_mode="assistant"):
    if chat_mode not in config.chat_modes.keys():
        raise ValueError(f"Chat mode {chat_mode} is not supported")

    n_dialog_messages_before = len(dialog_messages)
    answer = None
    while answer is None:
        try:
            if self.model in {"gpt-3.5-turbo-16k", "gpt-3.5-turbo", "gpt-4", "gpt-4-1106-preview", "gpt-4-vision-preview"}:
                messages = self._generate_prompt_messages(message, dialog_messages, chat_mode)

                r = await openai.ChatCompletion.acreate(
                    model=self.model,
                    messages=messages,
                    **OPENAI_COMPLETION_OPTIONS
                )
                answer = r.choices[0].message["content"]
            elif self.model == "text-davinci-003":
                prompt = self._generate_prompt(message, dialog_messages, chat_mode)
                r = await openai.Completion.acreate(
                    engine=self.model,
                    prompt=prompt,
                    **OPENAI_COMPLETION_OPTIONS
                )
                answer = r.choices[0].text
            else:
                raise ValueError(f"Unknown model: {self.model}")

            answer = self._postprocess_answer(answer)
            n_input_tokens, n_output_tokens = r.usage.prompt_tokens, r.usage.completion_tokens
        except openai.error.InvalidRequestError as e:
            if len(dialog_messages) == 0:
                raise ValueError("Dialog messages is reduced to zero, but still has too many tokens to make completion") from e

            dialog_messages = dialog_messages[1:]

    n_first_dialog_messages_removed = n_dialog_messages_before - len(dialog_messages)

    return answer, (n_input_tokens, n_output_tokens), n_first_dialog_messages_removed
```

config.yml

```yaml
telegram_token: ""
openai_api_key: ""
openai_api_base: null
allowed_telegram_usernames: []
new_dialog_timeout: 600
return_n_generated_images: 1
n_chat_modes_per_page: 5
image_size: "512x512"
enable_message_streaming: true
current_model: gpt-3.5-turbo-16k

chatgpt_price_per_1000_tokens: 0.002
gpt_price_per_1000_tokens: 0.02
whisper_price_per_1_min: 0.006
```

config.env

```
MONGODB_PATH=./mongodb
MONGODB_PORT=27017

MONGO_EXPRESS_PORT=8081
MONGO_EXPRESS_USERNAME=username
MONGO_EXPRESS_PASSWORD=password
```

models.yml

```
gpt-3.5-turbo-16k:
  type: chat_completion
  name: GPT-16K
  description: ChatGPT is that well-known model. It's <b>fast</b> and <b>cheap</b>. Ideal for everyday tasks. If there are some tasks it can't handle, try the <b>GPT-4</b>.

  price_per_1000_input_tokens: 0.003
  price_per_1000_output_tokens: 0.004

  scores:
    Smart: 3
    Fast: 5
    Cheap: 5
```

post_init→ bot.py

```
async def post_init(application: Application):
    await application.bot.set_my_commands([
        BotCommand( command: "/new",  description: "Start new dialog"),
        BotCommand( command: "/retry",  description: "Re-generate response for previous query"),
        BotCommand( command: "/help",  description: "Show help message"),
    ])
```

handlers → bot.py

```
application.add_handler(CommandHandler( command: "start", start_handle, filters=user_filter))
application.add_handler(CommandHandler( command: "help", help_handle, filters=user_filter))
application.add_handler(CommandHandler( command: "help_group_chat", help_group_chat_handle, filters=user_filter))

application.add_handler(MessageHandler(filters.TEXT & ~filters.COMMAND & user_filter, message_handle))
application.add_handler(MessageHandler(filters.PHOTO & ~filters.COMMAND & user_filter, message_handle))
application.add_handler(MessageHandler(filters.VIDEO & ~filters.COMMAND & user_filter, unsupport_message_handle))
application.add_handler(MessageHandler(filters.Document.ALL & ~filters.COMMAND & user_filter, unsupport_message_handle))
application.add_handler(CommandHandler( command: "retry", retry_handle, filters=user_filter))
application.add_handler(CommandHandler( command: "new", new_dialog_handle, filters=user_filter))
application.add_handler(CommandHandler( command: "cancel", cancel_handle, filters=user_filter))

application.add_handler(MessageHandler(filters.VOICE & user_filter, voice_message_handle))
```