Міністерство освіти і науки України
Одеський національний технологічний університет
Кафедра комп'ютерної інженерії

ПОЯСНЮВАЛЬНА ЗАПИСКА
ДО КВАЛІФІКАЦІЙНОЇ РОБОТИ

на тему *Розробка багатофункціонального телеграм-бота*
(назва кваліфікаційної роботи згідно наказу ОНТУ)

Здобувача *Кулика О. А.*
(прізвище, ініціали)

4 курсу *541а* групи

Керівники: *д.т.н., проф. Артеменко С.В.*
(посада, прізвище та ініціали)
ст. викл. Сіренко О.І.
(посада, прізвище та ініціали)

Консультанти: _____
(посада, прізвище та ініціали)
Phd, ст.викл. Богданов О.О.
(посада, прізвище та ініціали)

Кваліфікаційна робота допускається до захисту

Рішення кафедри від *05.06* 2024 р., протокол № *8*

Завідувач кафедри комп. інженерії _____ Сергій АРТЕМЕНКО
(назва кафедри) (підпис) (Ім'я ПРІЗВИЩЕ)

Одеса – 2024 рік

ОДЕСЬКИЙ НАЦІОНАЛЬНИЙ ТЕХНОЛОГІЧНИЙ УНІВЕРСИТЕТ

Факультет *комп'ютерної інженерії, програмування та кіберзахисту*
Кафедра *комп'ютерної інженерії*
Ступінь вищої освіти *бакалавр*
Спеціальність *123 «Комп'ютерна інженерія»*
Освітня програма *Мережеві технології та інтернет речей*

ЗАТВЕРДЖУЮ
Зав. кафедри *комп'ютерної інженерії*
Сергій АРТЕМЕНКО
« *30* » *серпня* 2023 року

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ ЗДОБУВАЧА

Кулика Олексія Андрійовича

1. Тема роботи *Розробка багатофункціонального телеграм-бота*

Затверджена наказом університету від « *30* » *серпня* 2023 р., наказ № *442-03*
2 Термін здачі здобувачем закінченої роботи *28 травня 2024 р.*
3. Вихідні дані роботи
1. Мова програмування Python. 2. Редактор коду Visual Studio Code.
3. Текстовий редактор Microsoft Word. 4. Редактор презентацій Microsoft PowerPoint

4. Перелік питань, які потрібно розробити
1. Вступ. 2. Збір та аналіз інформації. 3. Проектування бота.
4. Розробка бота. 5. Економічні розрахунки.
6. Загальні висновки. 7. Охорона праці.

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)
Слайд 1. Титульний. Слайд 2. Актуальність. Слайд 3. Об'єкт, предмет. Слайд 4. Задачі.
Слайд 5. Аналоги. Слайд 6. Проектування. Слайд 7. Діаграма станів.
Слайд 8. Діаграма послідовності. Слайд 9. Розробка. Слайд 10. Результат.
Слайд 11. Економічні розрахунки. Слайд 12. Загальні висновки.

6. Консультанти по роботі, із зазначенням розділів роботи, що стосуються їх

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв
Економіка	*Phd, ст. викл. Богданов О.О.*		
Охорона праці	*д.т.н., проф. Артеменко С.В.*		
Нормоконтроль	*ст. викл. Сіренко О.І.*		

7. Дата видачі завдання *30.08.2023*

Керівники *Сергій АРТЕМЕНКО*

Олександр СІРЕНКО

Завдання прийняв до виконання *Олексій КУЛИК*

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів кваліфікаційної роботи	Термін виконання етапів роботи	Примітка
1.	*Дослідженні предметної області*	*26.10.2023*	
2.	*Дослідження існуючих аналогів*	*30.11.2023*	
3.	*Дослідження методів реалізації телеграм-бота*	*28.01.2023*	
4.	*Проектування*	*15.02.2024*	
5.	*Розробка демонстраційної версії бота*	*27.03.2024*	
6.	*Підготовка техніко-економічної частини*	*15.04.2024*	
7.	*Підготовка розділу охорони праці*	*15.04.2024*	
8.	*Оформлення пояснювальної записки*	*27.05.2024*	
9.	*Оформлення графічної частини та лістингу*	*27.05.2024*	

Здобувач-дипломник _____ *Олексій КУЛИК*

Керівники роботи _____ *Сергій АРТЕМЕНКО*

_____ *Олександр СІРЕНКО*

Несу відповідальність за ідентичність електронного та друкованого варіантів кваліфікаційної роботи, даю згоду на обробку персональних даних та не заперечую проти розміщення кваліфікаційної роботи на офіційних web-ресурсах ОНТУ.

Підтверджую, що в кваліфікаційній роботі відсутні порушення норм академічної доброчесності.

Здобувач-дипломник *Олексій КУЛИК* _____

АНОТАЦІЯ

Кваліфікаційна робота присвячена розробці багатофункціонального *телеграм-бота* з використанням мови програмування *Python* та бібліотеки *Aiogram*. У роботі розглянуто основні принципи створення телеграм-ботів, їх архітектуру та функціональні можливості.

У першому розділі проаналізовано предметну область та існуючі аналоги телеграм-ботів.

У другому розділі представлено проектування бота, включаючи вибір технологічного стеку, проектування бази даних та інтерфейсу користувача.

У третьому розділі описано практичну реалізацію бота, включаючи використання асинхронного програмування, взаємодію з зовнішніми *API* та обробку запитів користувачів.

У четвертому розділі проведено економічне обґрунтування проекту, включаючи розрахунок трудомісткості та вартості розробки. У п'ятому розділі розглянуто питання охорони праці при роботі з комп'ютерною технікою.

Результатом роботи є створення багатофункціонального телеграм-бота, який надає користувачам широкий спектр можливостей, таких як завантаження відео з *TikTok,* генерація хештегів, пошук аніме за скриншотами, керування списками завдань, отримання новин та спілкування зі штучним інтелектом *ChatGPT.*

Ключові слова: *телеграм-бот, Python, Aiogram,* асинхронне програмування.

ABSTRACT

This qualification work is dedicated to the development of a multifunctional Telegram-bot using the Python programming language and the Aiogram library. The paper discusses the basic principles of creating Telegram-bots, their architecture and functionality.

The first chapter analyzes the subject area and existing analogues of Telegram-bots.

The second chapter presents the design of the bot, including the choice of technology stack, database design, and user interface.

The third chapter describes the practical implementation of the bot, including the use of asynchronous programming, interaction with external APIs, and user request handling.

The fourth chapter provides an economic justification for the project, including the calculation of labor intensity and development cost. The fifth chapter discusses occupational safety issues when working with computer equipment.

The result of the work is the creation of a multifunctional Telegram-bot that provides users with a wide range of features, such as downloading videos from TikTok, generating hashtags, searching for anime by screenshots, managing to-do lists, getting news, and communicating with the artificial intelligence ChatGPT.

Keywords: *Telegram-bot, Python, Aiogram, asynchronous programming.*

ЗМІСТ

стор.

					КРБ.КІ.1.442-03.2.2		
Змн.	Арк.	№ докум.	Підпис	Дата			
Розробив		Олексій КУЛИК			Розробка багатофункціонального телеграм-бота	Літ.	Арк. 6 / Аркушів 97
Перевірив		Олександр СІРЕНКО					
Рецензент		Володимир ПОПОВ					
Нормоконтроль		Олександр СІРЕНКО					
Затвердив		Сергій АРТЕМЕНКО					гр. 541, ОНТУ

ВСТУП

На сьогоднішній день ринок телеграм-ботів представлений безліччю різноманітних рішень, розроблених як для загального користування, так і для конкретних бізнес-потреб. Серед найбільш популярних рішень можна виділити застосунки для отримання новин, консультації з різних питань, замовлення їжі, придбання товарів та послуг, ігрові боти та багато іншого.

Типові завдання, що стоять перед розробниками телеграм-ботів, включають створення зручного інтерфейсу взаємодії з користувачем, обробку природної мови для розуміння запитів, інтеграцію із зовнішніми сервісами і *API*, а також реалізацію штучного інтелекту для поліпшення роботи бота і підвищення його функціональності.

Незважаючи на значний прогрес у галузі розробки телеграм-ботів, існує низка проблем та викликів. Однією з таких проблем є обмежена здатність ботів обробляти та розуміти контекст повідомлень. Вони працюють за жорстко заданими інструкціями, що може знижувати їхню здатність адаптуватися до різноманітних запитів користувачів. Також багато існуючих ботів мають обмежений спектр функцій, незручні у використанні чи навіть небезпечні для користувача. Крім того, іншими важливим викликом є необхідність впровадження механізмів автентифікації, оптимізації продуктивності та масштабування ботів при великих обсягах запитів.

У світлі активного розвитку інформаційних технологій та збільшення попиту на автоматизовані рішення, створення телеграм-бота залишається актуальним і перспективним завданням. Отже, об'єднання можливостей мови програмування *Python* у галузі аналізу даних, машинного навчання та розробки веб-додатків робить його ідеальним інструментом для створення складних, багатофункціональних та зручних телеграм-ботів, здатних ефективно вирішувати різноманітні завдання.

Зазначимо мету, об’єкт, предмет та задачі, які необхідно вирішити в ході

кваліфікаційної роботи.

Метою роботи є розробка багатофункціонального телеграм-бота, який має забезпечувати ефективну взаємодію з користувачами та вирішувати широкий спектр завдань, використовуючи можливості мови програмування *Python* для забезпечення надійності, зручності використання та багатофункціональності.

Об’єктом дослідження є багатофункціональний телеграм-бот.

Предметом дослідження є методи розробки багатофункціонального телеграм-бота.

Основними задачами, які необхідно вирішити в ході роботи є:

1. Аналіз предметної області.
2. Проектування архітектури бота.
3. Розробка бота.
4. Тестування розробленого бота.
5. Розробка рекомендацій щодо подальшого розвитку та вдосконалення бота.

Наукова новизна кваліфікаційної роботи полягає у використанні асинхронного програмування при розробці бота. Такий підхід дозволить йому ефективно обробляти багато запитів від користувачів одночасно, без блокування потоку виконання. Це забезпечує високу швидкодію та ефективність роботи, особливо в умовах великої кількості користувачів та інтенсивної взаємодії з ботом.

РОЗДІЛ 1
АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ДОСЛІДЖЕННЯ ІСНУЮЧИХ АНАЛОГІВ

1.1 Загальні положення предметної області

Месенджери є важливою частиною сучасної комунікаційної інфраструктури, яка забезпечує користувачів можливістю спілкування миттєво та ефективно через інтернет. Дуже важко уявити наше життя без них.

Вони мають кілька ключових характеристик:

1. Миттєвий зв'язок: однією з основних переваг месенджерів є можливість миттєвого обміну повідомленнями у реальному часі. Це дозволяє користувачам швидко та ефективно спілкуватися навіть на великі відстані.

2. Мультимедійні можливості: у сучасних месенджерах користувачі можуть надсилати не лише текстові повідомлення, а й голосові, відео- та медіа-повідомлення, а також обмінюватися файлами та зображеннями. Це розширює можливості комунікації та робить її різноманітнішою.

3. Групові чати: багато месенджерів надають можливість створення групових чатів, де кілька користувачів можуть спілкуватися одночасно. Це особливо зручно для організації робочих комунікацій чи спілкування з великою групою друзів.

4. Безпека: зі збільшенням важливості онлайн-приватності, месенджери ставлять перед собою завдання забезпечення безпеки та конфіденційності даних користувача. Багато месенджерів впроваджують шифрування кінцевого до кінцевого (*End-to-End Encryption*), щоб захистити вміст повідомлень від несанкціонованого доступу.

End-to-End Encryption (*E2EE*) – це метод шифрування даних, який забезпечує конфіденційність та безпеку передачі інформації від відправника до одержувача, тому ніякі проміжні вузли (включаючи постачальників послуг) не можуть прочитати вміст повідомлень у відкритому вигляді.

Принцип роботи *E2EE* (рис. 1.1) ґрунтується на використанні симетричного шифрування та обміні ключами між відправником та одержувачем без участі сторонніх вузлів. При використанні *E2EE* дані шифруються на пристрої відправника за допомогою унікального ключа шифрування, який відомий лише відправнику та одержувачу. Зашифровані дані потім передаються через мережу і розшифровуються тільки пристроєм одержувача з використанням того ж ключа шифрування. Проміжні вузли, через які проходять зашифровані дані, не мають доступу до ключа шифрування та не можуть розшифрувати вміст повідомлень.

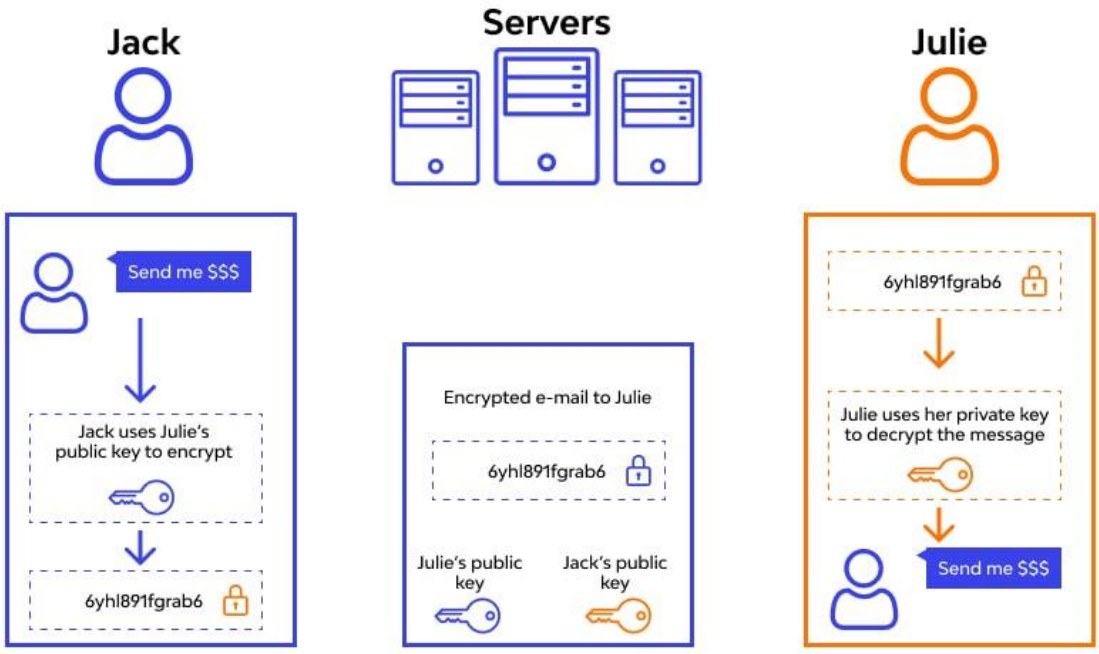


Рис. 1.1 – Принцип роботи шифрування *E2EE*

Переваги використання *End-to-End Encryption*:

1. Конфіденційність даних: *E2EE* забезпечує захист конфіденційності даних, оскільки тільки відправник та одержувач можуть прочитати вміст повідомлень.

2. Безпека передачі даних: зашифровані дані неможливо прочитати без знання ключа шифрування, що робить їх безпечними для передачі по відкритих мережах, таких як Інтернет.

3. Захист від проміжних атак: оскільки дані шифруються на пристроях відправника та одержувача, а не на серверах або проміжних вузлах, *E2EE*

захищає інформацію від проміжних атак або перехоплення.

4. Довіра до сервісу: використання *E2EE* підвищує довіру користувачів до сервісу, оскільки демонструє прагнення до захисту особистої інформації та конфіденційності користувачів.

Однак, незважаючи на свої переваги, *E2EE* також має деякі обмеження та недоліки, включаючи складність відновлення даних при втраті ключа шифрування, відсутність можливості перегляду зашифрованих даних сторонніми вузлами (що може призвести до проблем у боротьбі з кіберзлочинністю) та деякі обмеження у функціональності додатків, пов'язані із необхідністю забезпечення сумісності з *E2EE*.

Архітектура месенджерів складається з кількох ключових компонентів, які взаємодіють між собою для забезпечення функціональності та продуктивності платформи. Її основні компоненти це:

1. Клієнтська програма: клієнтська програма є програмою, встановленою на пристрої користувача. Вона забезпечує інтерфейс для взаємодії з месенджером, надсилання та отримання повідомлень, управління контактами та інших функцій.

2. Серверна інфраструктура: месенджери зазвичай мають серверну інфраструктуру, яка відповідає за обробку та маршрутизацію повідомлень між користувачами, а також за зберігання даних про користувачів, їх контакти, історію повідомлень та інші параметри.

3. Протоколи зв'язку: для обміну даними між клієнтськими програмами та серверами месенджера використовуються різні протоколи зв'язку. Ці протоколи забезпечують безпечну та ефективну передачу повідомлень між пристроями користувачів та серверами. Найбільш поширені з них:

a) *HTTP* (*Hypertext Transfer Protocol*): *HTTP* – це стандартний протокол передачі даних у мережі Інтернет. Він часто використовується для надсилання запитів від клієнтських додатків до серверів месенджерів та отримання відповідей від серверів;

b) *HTTPS* (*Hypertext Transfer Protocol Secure*): *HTTPS* – це захищена

версія протоколу *HTTP*, яка використовує шифрування для забезпечення безпечної передачі даних. Їх порівняння продемонстровано на рис. 1.2. Багато месенджерів використовують його для захисту конфіденційності та цілісності даних;

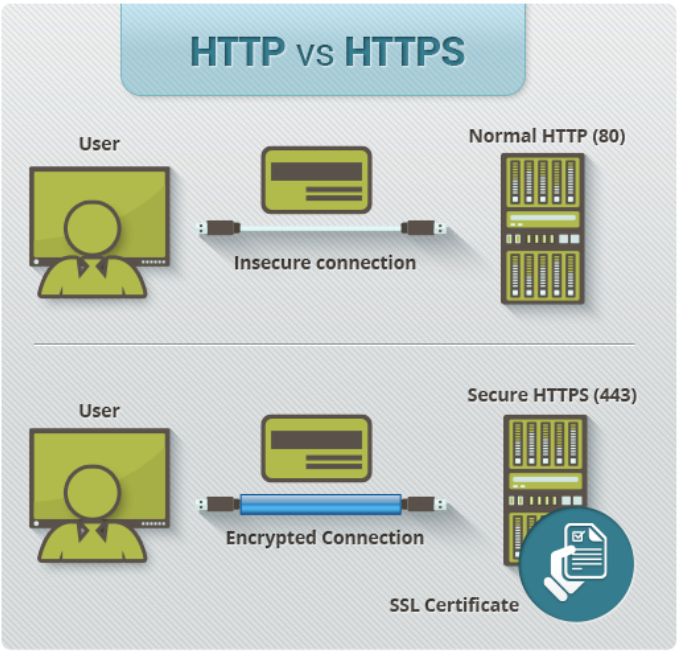


Рис. 1.2 – Різниця протоколів *HTTP* та *HTTPS*

c) *MTProto* (*Mobile Protocol*): *MTProto* – це пропрієтарний протокол, розроблений спеціально для використання в месенджері телеграм. Схему його роботи продемонстровано на рис. 1.3. Він оптимізований для обміну повідомленнями в реальному часі та забезпечує конфіденційність та безпеку повідомлень за допомогою шифрування кінцевого до кінцевого;
d) *XMPP* (*Extensible Messaging and Presence Protocol*): *XMPP* – це протокол, розроблений обмінюватись повідомленнями у часі. Схему його роботи продемонстровано на рис. 1.4. Він широко використовується для створення месенджерів та забезпечує можливість обміну повідомленнями між різними серверами;

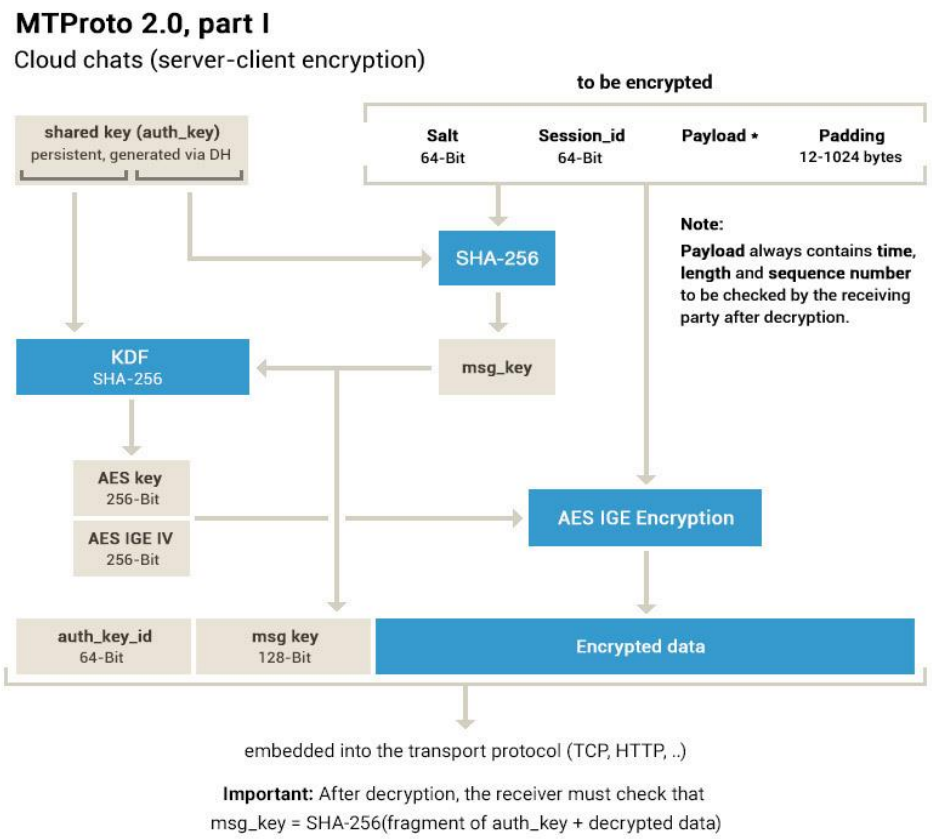


Рис. 1.3 – Протокол *MTProto*

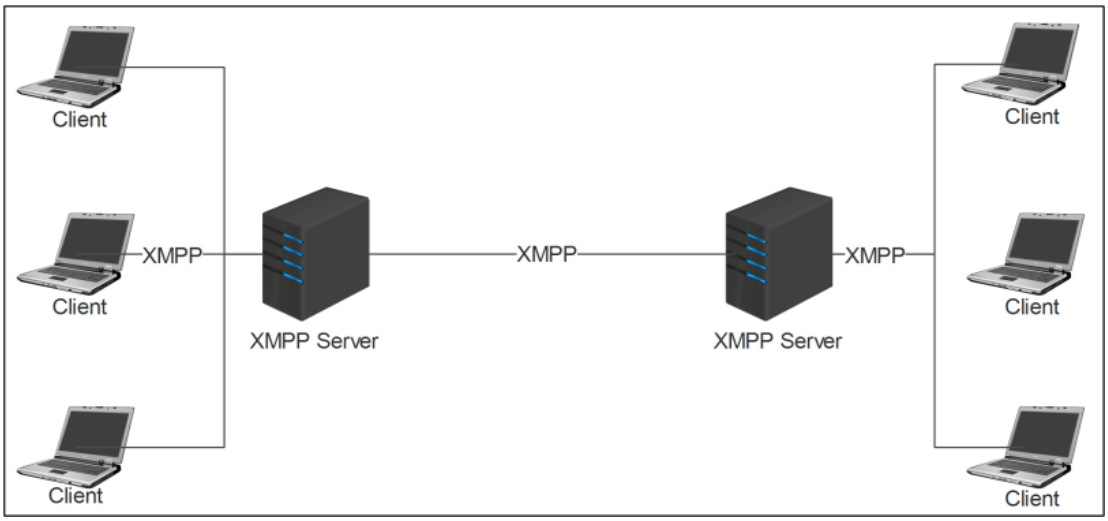


Рис. 1.4 – Протокол *XMPP*

е) *WebSocket* – це протокол, який забезпечує двосторонній зв'язок між клієнтськими програмами та серверами через єдине *TCP*-з'єднання. Схему його роботи продемонстровано на рис. 1.5. Він може використовуватися месенджерами для обміну повідомленнями в реальному часі без постійного оновлення сторінки;

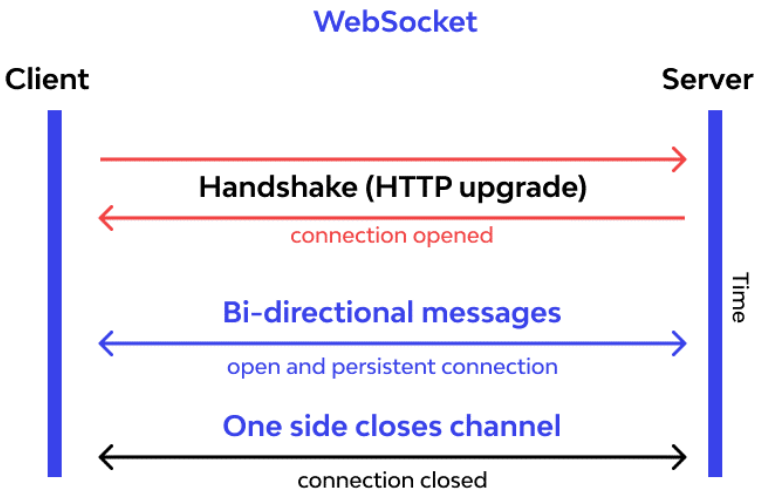


Рис. 1.5 – Протокол *WebSocket*

4. Шифрування: шифрування повідомлень є важливим аспектом месенджерів, оскільки воно забезпечує конфіденційність та безпеку листування. Багато месенджерів використовують шифрування кінцевого до кінцевого (*End-to-End Encryption*), щоб захистити дані від несанкціонованого доступу навіть у разі компрометації серверів.

Розглянемо мессенджер телеграм, що буде використовуватися у розробці нашого бота.

Телеграм – це месенджер, який надає можливість обміну повідомленнями, фотографіями, відеозаписами та іншими файлами, а також голосовий та відеозв'язок. Він був створений Павлом Дуровим та його братом Миколою Дуровим. Він вперше був запущений у серпні 2013 року.

Розглянемо статистику використання месенджера:

1. Кількість користувачів: станом на січень 2022 року кількість активних користувачів телеграм перевищувала 700 мільйонів по всьому світу.

2. Зростання популярності: за останні роки телеграм продовжував набирати популярності завдяки своїм функціям безпеки, конфіденційності та широким можливостям для спілкування та спільної роботи.

3. Поширення країнами: телеграм популярний у багатьох країнах світу, і його використання може різнитися залежно від регіону. Наприклад, у деяких країнах він може бути особливо популярним через свою конфіденційність та

можливість обходу цензури.

4. Бізнес-використання: телеграм також використовується для бізнес-цілей, у тому числі для створення каналів та груп для комунікації з клієнтами, а також для автоматизації деяких процесів за допомогою роботів.

Розглянемо архітектура месенджера:

1. Клієнтські програми телеграм – телеграм підтримує клієнтські програми для різних платформ, включаючи мобільні пристрої (*iOS*, *Android*), настільні комп'ютери (*Windows*, *macOS*, *Linux*) та веб-додаток. Ці програми забезпечують зручний і потужний інтерфейс користувача для обміну повідомленнями та інших функцій.

2. Сервери телеграм – телеграм використовує власні сервери для обробки повідомлень, зберігання даних користувачів та забезпечення доступності сервісу. Вони розподілені по всьому світу, що забезпечує швидкий доступ до сервісу для користувачів із різних регіонів.

3. *API* телеграм – він надає *API* для розробників, що дозволяє їм створювати власні програми та інтегрувати їх із платформою телеграм. Однією з таких програм є бот, розробка якого є темою цієї роботи. *API* забезпечує доступ до різних функцій месенджера, таких як надсилання повідомлень, керування чатами, робота з медіа-контентом і т.д.

4. Протокол *MTProto:* для обміну даними між клієнтськими програмами та серверами телеграм використовується протокол *MTProto*, який забезпечує безпечну та ефективну передачу повідомлень.

Таким чином, архітектура телеграм поєднує у собі високу продуктивність, масштабованість та безпеку, що робить його одним з найпопулярніших месенджерів у світі.

1.2 Телеграм-боти

Телеграм боти є програмними агентами, які автоматизують різні завдання та функції в месенджера. Вони здатні виконувати різні дії, включаючи надсилання повідомлень, обробку запитів користувачів, взаємодію з іншими

сервісами та багато іншого. Завдання таких ботів можуть змінюватись від простих повідомлень та нагадувань до складних командних функцій та інтеграції із зовнішніми сервісами.

Архітектура ботів включає кілька ключових компонентів:

1. Телеграм *API:* боти взаємодіють з телеграм через телеграм-бот *API,* який надає набір методів для надсилання та отримання повідомлень, керування чатами, оновленнями та іншими функціями.

2. Серверний компонент: боти зазвичай розміщуються на серверах, де вони запускаються та працюють. Цей серверний компонент відповідає за прийом запитів від телеграм, обробку команд та повідомлень, а також надсилання відповідей.

3. Логіка обробки: це частина програмного коду, яка визначає логіку роботи бота. Вона може включати обробку вхідних повідомлень, виконання певних дій у відповідь на команди користувачів і взаємодію з іншими сервісами або базами даних.

4. Сховище даних (опційно): деякі боти можуть використовувати бази даних або інші сховища даних для збереження інформації про користувачів, історію повідомлень, налаштувань та інших параметрів.

Принцип роботи телеграм-ботів ґрунтується на взаємодії з *API*. Вони можуть працювати в режимі довгого опитування (*long polling*), коли вони постійно надсилають запити до *API* телеграм для перевірки наявності нових повідомлень або оновлень. Коли з'являється нове повідомлення або оновлення, бот отримує його та обробляє відповідно до логіки своєї програми.

Боти також можуть використовувати веб-хуки (*webhooks*), при яких вони реєструються на певну *URL*-адресу, і *API* телеграм автоматично надсилає повідомлення про нові повідомлення або оновлення на цю адресу.

Можливості телеграм-ботів:

1. Надсилання повідомлень: боти можуть надсилати текстові повідомлення, а також медіа-контент, такий як зображення, відео та аудіофайли.

2. Обробка команд: боти можуть реагувати на команди, введені користувачами в чаті, та виконувати відповідні дії, такі як відображення інформації, виконання розрахунків, керування іншими сервісами тощо.

3. Інтерактивні елементи: боти можуть надавати інтерактивні елементи, такі як кнопки, меню та інлайн-клавіатури, щоб користувачі могли взаємодіяти з ними зручніше.

4. Інтеграція із зовнішніми сервісами: боти можуть інтегруватися з різними зовнішніми сервісами, такими як поштові служби, соціальні мережі, онлайн бази даних та інші, щоб надавати додаткові функції та можливості.

5. Повідомлення та нагадування: боти можуть надсилати повідомлення та нагадування користувачам про важливі події, завдання або терміни.

6. Аналітика та звіти: деякі боти можуть збирати статистику про діяльність користувачів та генерувати аналітичні звіти на основі цих даних.

Це лише деякі з можливостей телеграм-ботів, їхня функціональність може бути значно розширена за допомогою кастомізації та інтеграції з іншими сервісами та платформами.

1.3 Аналіз існуючих аналогів

Так як за станом на 12 квітня 2024 року не було знайдено аналогів багатофункціональних ботів із схожим функціоналом, було прийнято рішення провести аналіз однофункціональних ботів, можливості яких частково збігаються з моїм ботом. Ось найпопулярніші з них із кожної категорії:

1. *TikTok Download Bot:* цей бот пропонує зручний спосіб завантаження відео з соціальної мережі *TikTok* через телеграм. Приклад його роботи продемонстровано на рис. 1.6. Функціонал: завантаження відео. Користувачі можуть надіслати посилання на відео з *TikTok*, і бот надасть їм посилання для завантаження відео. Переваги даного бота:

a) зручний, зрозумілий інтерфейс з інструкцією щодо використання;
b) швидкість роботи, середній час відповіді займає декілька секунд;
c) є можливість зворотного зв’язку.

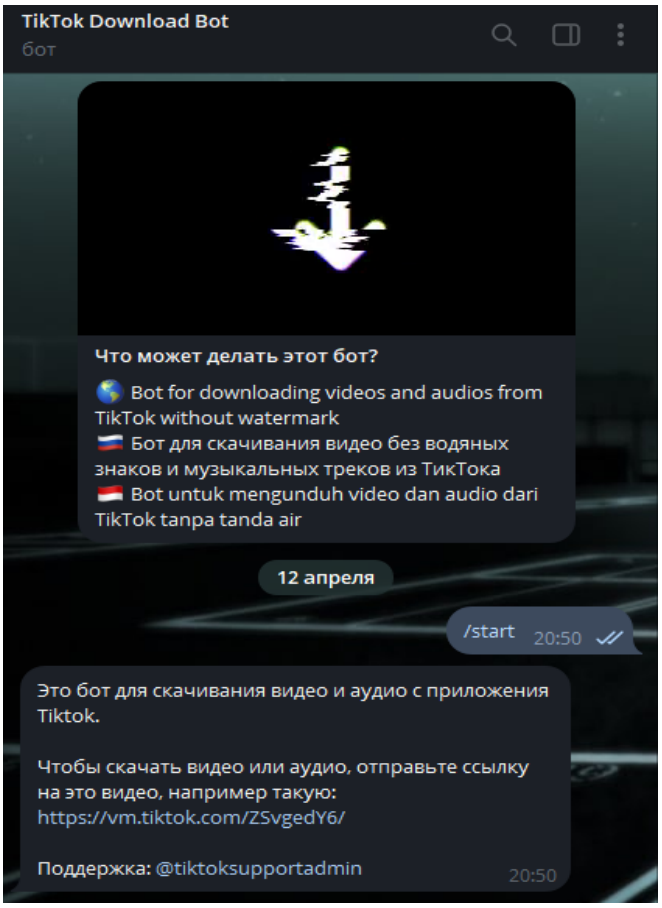


Рис. 1.6 – Скриншот роботи *TikTok Download Bot*

Недоліки:

a) недоступність вихідного коду;

b) через це користувачі не можуть перевірити безпеку і надійність бота, що може спричинити побоювання щодо обробки та зберігання їх персональних даних.

2. *Taskobot*: це бот для управління завданнями та списками справ. Він дозволяє створювати, редагувати та видаляти завдання, встановлювати їм терміни виконання та отримувати повідомлення про майбутні події. Завдяки чому користувачі можуть ефективно організовувати свій час і керувати своїми справами, не залишаючи телеграм. Приклад його роботи продемонстровано на рис. 1.7. Функціонал: створення завдань, керування списками справ, редагування та видалення завдань, система надсилання повідомленнь. Переваги даного бота:

a) зручний, зрозумілий інтерфейс керування завданнями;

b) можливість створювати сумісні задачі для декількох користувачів;

c) швидкість роботи. середній час відповіді займає декілька секунд.

Недоліки:

a) недоступний вихідний код;

b) відсутня можливість зворотного зв’язку.

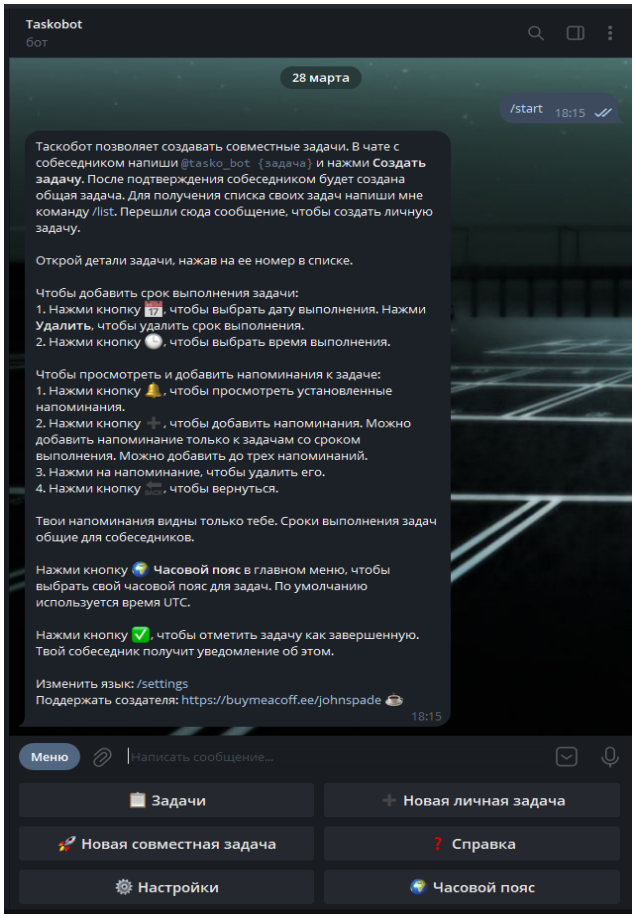


Рис. 1.7 – Скриншот роботи *Taskobot*

3. *WAIT: What Anime Is This*: цей бот дозволяє користувачам дізнатися, з якого аніме-серіалу їх скриншот. Приклад його роботи продемонстровано на рис. 1.8. Функціонал: пошук назви аніме за завантаженим скриншотом, надання необхідної інформації щодо певного моменту. Переваги даного бота:

a) швидкість роботи, середній час відповіді займає декілька секунд;

b) використання скриншотів для пошуку робить процес більш інтуїтивним та зручним;

c) відкритий вихідний код.

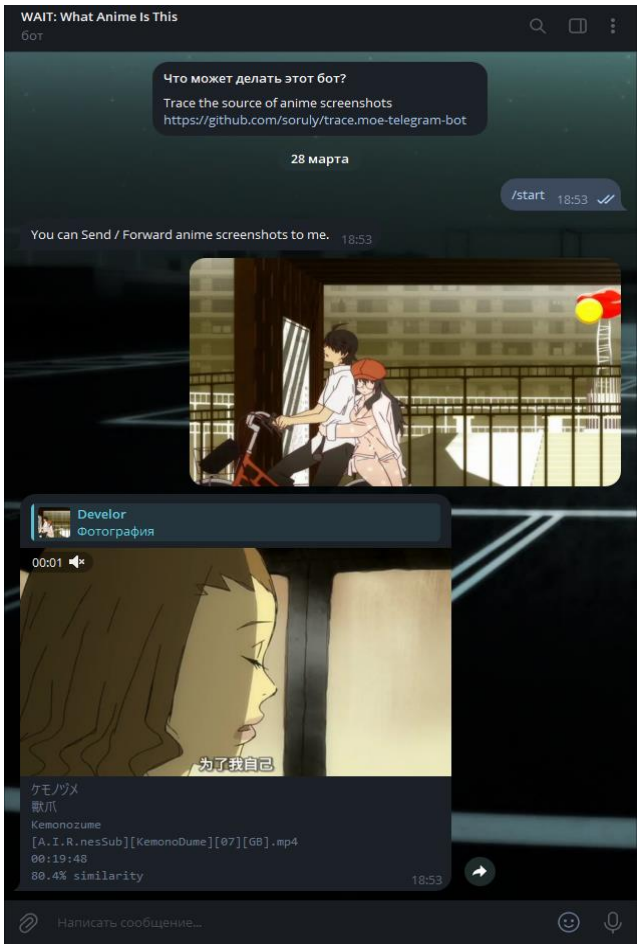


Рис. 1.8 – Скриншот роботи *WAIT: What Anime Is This*

Недоліки:

a) обмежений набір джерел пошуку інформації, що може призвести до некоректних відповідей, таким чином, лише з четвертого разу бот надав правильну відповідь;

b) неможливо отримати додаткову інформацію, таку як список епізодів або рекомендації;

c) відсутня можливість зворотного зв’язку.

4. *ChatGPT on Telegram Bot:* цей бот дозволяє користувачам взаємодіяти з моделлю штучного інтелекту *GPT* від *OpenAI*. Він надає можливість ставити йому запитання та отримувати відповіді від останньої версії моделі *GPT*. Приклад його роботи продемонстровано на рис. 1.9. Функціонал: природне спілкуванні, різноманітні теми, гнучкість у спілкуванні, можливість зворотного зв’язку. Із недоліків можна зазначити лише стислість відповідей. Переваги:

a) можливість використання штучного інтелекту без необхідності залишати месенджер;

b) *user-friendly* інтерфейс з відео-інструкціями щодо роботи бота;

c) відкритий вихідний код.

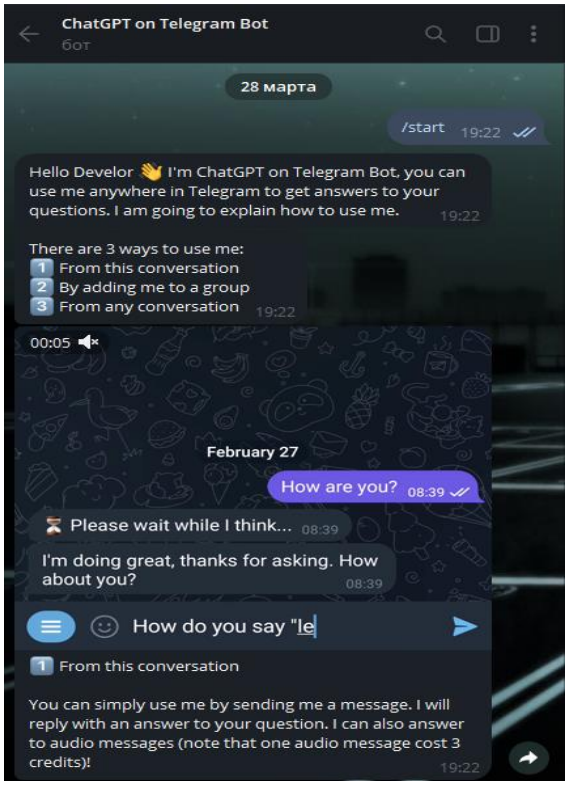


Рис. 1.9 – Скриншот роботи *ChatGPT on Telegram bot*

1.4 Огляд можливих вдосконалень бота на основі аналізу існуючих аналогів

Сформуємо таблицю з порівянням ботів, використовуючи результати, які ми отримали провівши аналіз ринку існуючих аналогів у частині 1.3. Також зробимо висновки щодо можливостей їх вдосконалення.

Geek's Friend – це бот, розробка якого є метою цього дипломного проектування. Його фунцкіонал буде покривати усі недоліки, які були помічені у інших ботах. За рахунок того, що він буде багатофункціональним, користувач зможе використовувати його замість чотирьох інших ботів. У таблиці 1.1 представлено порівняння функціоналу та характеристик існуючих аналогів.

Таблиця 1.1

Порівняння функціоналу та характеристик існуючих аналогів

Функціонал чи характеристика	*TikTok Download Bot*	*Taskobot*	*WAIT: What Anime Is This*	*ChatGPT on Telegram Bot*	*Geek's Friend*
Завантаження відео з соц. мережі *TikTok*	+	-	-	-	+
Генерація хештегів для просунення публікації у соц. мережі *TikTok*	-	-	-	+	+
Керування списками справ	-	+	-	-	+
Пошук назви аніме за скриншотом	-	-	+	-	+
Надання останніх аніме-новин	-	-	-	-	+
Спілкування с моделлю штучного інтелекту *ChatGPT*	-	-	-	+	+
Відкритий вихідний код	-	-	+	+	+
Стабільна робота бота	+	+	+	+	+
Можливість зворотного зв'язку	+	-	-	+	+
Наявність опису можливостей бота з інструкціями щодо використання	+	+	+	+	+

На основі аналізу та порівняння існуючих аналогів можна зробити кілька пропозицій щодо покращення і розширення функціоналу ботів, а також помітки

про те, на що слід звернути увагу під час розробки свого бота:

1. U*ser-friendly* інтерфейс. Це означає, що інтерфейс боту має бути інтуїтивно зрозумілим для більшості користувачів. Цього можна досягти за допомогою впровадження наступних елементів:

- меню – бот повинен мати зрозуміле меню, в якому описаний весь його функціонал і як його використовувати;
- кнопки – окрім команд, бот повинен мати зручні кнопки, при натисканні яких будуть виконуватися відповідні функції. Це робить взаємодію з ботом більш простою та швидкою;
- можливість зворотного зв'язку – користувач повинен мати можливість залишити відгук чи надіслати баг репорт щодо роботи бота.

2. Надійний хостинг. Забезпечення надійного та стабільного хостингу для бота, що гарантує його доступність та продуктивність для користувачів у будь-який час.

3. Відкритий вихідний код. Потрібно розглянути можливість зробити вихідний код бота відкритим для спільноти, що дозволить забезпечити більшу відкритість, довіру та безпеку для користувачів.

4. Доступність додаткових ресурсів. Включення посилань на додаткові ресурси або джерела інформації може полегшити користувачам отримання більше детальної або контекстної інформації на певну тему.

Висновок до першого розділу

Визначено, що телеграм-бот є досить перспективним проектом, особливо з урахуванням постійного зростання користувачів месенджера, наявності попиту та розширених можливостей, що надає сама платформа.

Проаналізовано, що тематика розроблюваного бота є досить популярною, а його функції знаходяться у великому попиті. За станом на 12 квітня 2024 року, він не має аналогів, які б змогли конкурувати з ним за кількістю доступних функцій.

РОЗДІЛ 2
ПРОЕКТУВАННЯ ТЕЛЕГРАМ-БОТА

2.1 Вступ у проектування бота

Метою розробки багатофункціонального телеграм-бота є створення зручного інструменту для користувачів, що дозволяє виконувати різні задачі та отримувати необхідну інформацію без необхідності перемикатися між різними програмами чи ботами.

Функціонал, який він повинен мати:

1. Повідомлення щодо стану бота: бот буде надсилати адміністратору повідомлення коли він вмикається та вимикається. Це дозволить контролювати роботу бота та, у разі непередбаченного вимикання, швидко про це дізнатися і усунути проблему.

2. Видалення вебхуків: за замовченням, коли бот вимкнений, користувач все ще може надсилати йому повідомлення та команди. Тому, коли бот буде запущений, він відповість на всі ці повідомлення, тим самим заспамить чат з користувачем. Реалізувавши функціонал видалення вебхуків у коді, ми вирішиму цю проблему, і бот не буде реагувати на повідомлення, надіслані йому коли він був вимкнений.

3. Логування: після увімкнення логування, розробник буде отримувати у консоль детальну інформацію щодо всіх дій бота. Це дуже зручно і знадобиться при розробці і тестуванні. Приклад логування продемонстровано на рис. 2.1.

```
INFO:aiogram.dispatcher:Start polling
INFO:aiogram.dispatcher:Run polling for bot @GeeksFriendBot id=6930630618 - "Geek's Friend"
INFO:aiogram.event:Update id=746608690 is handled. Duration 297 ms by bot id=6930630618
INFO:aiogram.event:Update id=746608691 is handled. Duration 452 ms by bot id=6930630618
INFO:aiogram.event:Update id=746608692 is handled. Duration 312 ms by bot id=6930630618
INFO:aiogram.event:Update id=746608693 is handled. Duration 203 ms by bot id=6930630618
```

Рис. 2.1 – Приклад логування

4. Реакція на невідому команду чи повідомлення: на будь-яку команду чи

повідомлення, які не передбачені розробником, бот буде відповідати заготовленим повідомленням, яке повідомляє користувача, що бот його не розуміє. Це підвищить якість спілкування з користувачем.

5. *Finite-State Machine (FSM):* машина станів є зручним способом організації логіки роботи та управління діалогами з користувачем. Приклад логіки найпростішої машини станів наведено на рис. 2.2.

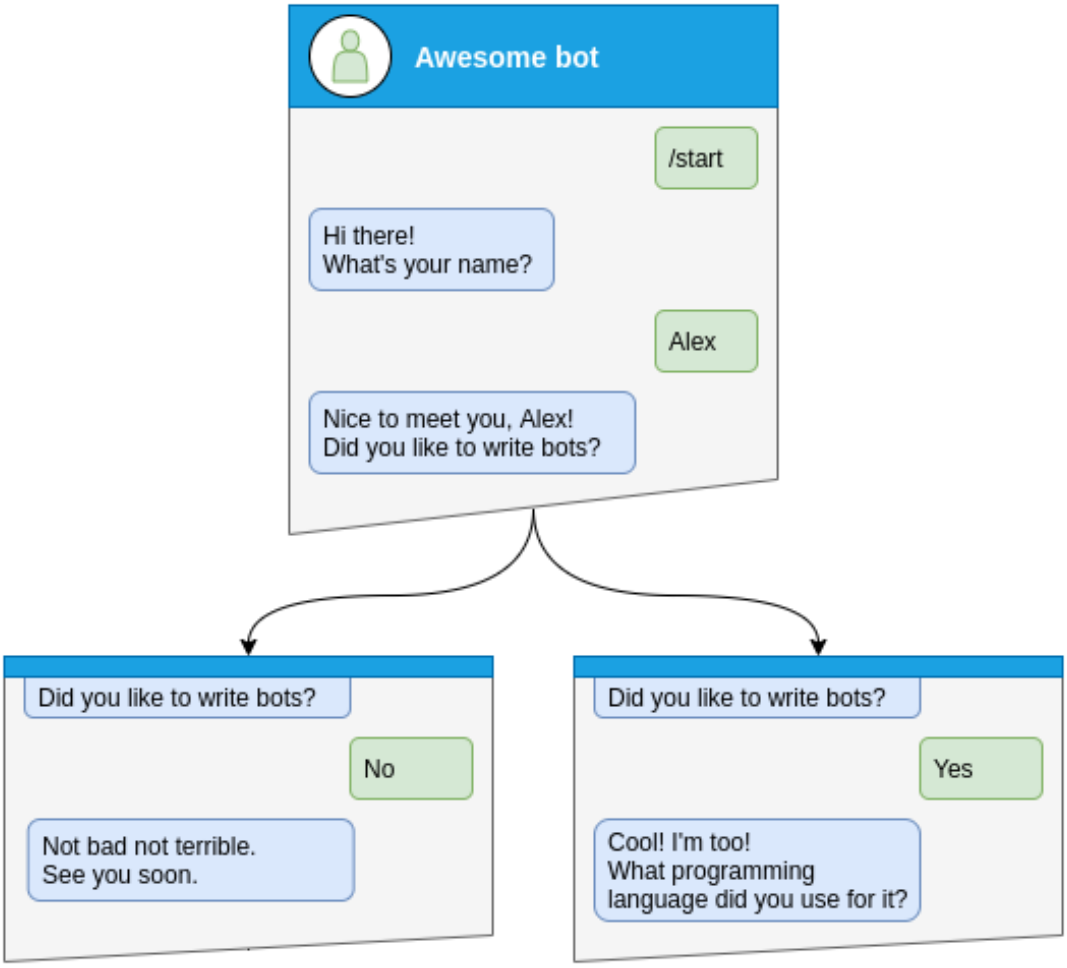


Рис. 2.2 – Приклад логіки найпростішої машини станів

Розглянемо функціонал машини станів:

1. Стан бота: кожен стан представляє певний стан взаємодії бота з користувачем, наприклад, це може бути «очікування команди», «очікування введення даних», «виконання операції» і т.д.

2. Події – є дії або вхідні сигнали, які ініціюють зміну стану бота. Наприклад, це може бути отримання повідомлення від користувача,

натискання кнопки, завершення операції тощо.

3. Переходи між станами: машина станів визначає, які переходи можуть відбуватися між різними станами бота у відповідь певні події. Наприклад, при отриманні команди від користувача бот може перейти зі стану «очікування команди» у стан «виконання операції».

4. Для кожного стану та події визначаються відповідні дії, які мають бути виконані, наприклад, під час переходу у стан «виконання операції» бот може розпочати виконання відповідного запиту чи дії.

На рис. 2.2 наведено приклад логіки найпростішої машини станів. Повна схема приведена у додатку А.

У нашому випадку, машина станів буде використана з метою запобігання помилок та забезпечення коректної взаємодії бота з користувачем. Розглянемо приклад її роботи.

Користувач відправляє боту команду «*/chatgpt*», яка активує функіонал діалогу зі штучним інтелектом *ChatGPT*. Але користувач передумав та захотів використати іншу команду. Наприклад, «*/feedback*» (активація функціоналу надсилання фідбеку адміністратору бота). У цьому випадку після відправки фідбеку він буде надісланий як запит до *ChatGPT*, і бот надішле відповідь штучного інтелекту.

Цю проблему ми можемо вирішити, використовуючи машину станів наступним чином. Закріпимо за кожною командою свій стан. Наприклад, у команди «*/chatgpt*» стан буде «*talking_chatgpt*», а у «*/feedback*» – «*sending_feedback*». Тепер після надсилання будь-якої з цих команд стан бота буде змінюватися і він буде «забувати» про виконання попередньої команди після відправки нової.

Для кращого розуміння побудуємо діаграму станів бота (рис 2.3), яка ілюструє різні стани, в яких може перебувати бот залежно від команд, одержуваних від користувачів.

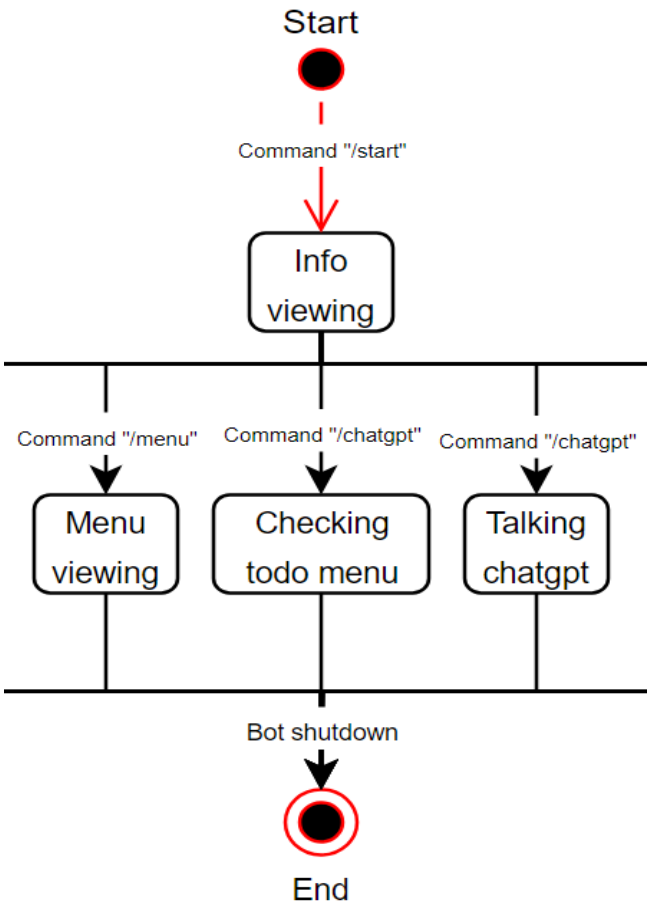


Рис. 2.3 – Діаграма станів бота

Повна діаграма наведена у додатку Б.

Розглянемо функціонал, який повинен мати наш бот:

1. Клавіатури та кнопки: бот буде мати клавіатури з кнопками, що значно полегшить взаємодію з користувачем, при натисканні на кнопку автоматично відправляється відповідна команда.

2. Завантаження відео з *TikTok* без водяного знаку: користувачі зможуть завантажувати відео з платформи *TikTok* без водяних знаків лише надіславши посилання на публікацію, це забезпечить зручність та збереження якості контенту.

3. Генерація хештегів для публікацій *TikTok: б*от буде надавати можливість автоматично генерувати хештеги для публікацій у *TikTok*, що допомагає підвищити видимість контенту та залучити аудиторію.

4. Пошук аніме-серіалу за скриншотом: користувачі зможуть відправляти скриншоти з будь-яких аніме-серіалів та отримувати інформацію про них, таку

як назву, жанр, опис тощо.

5. Керування списками завдань (*ToDo list*): бот буде надавати можливість створювати, керувати та відстежувати списки завдань, забезпечуючи ефективну організацію робочого чи особистого часу користувача.

6. Надсилання останніх аніме-новин: користувачі зможуть отримувати свіжі новини, оновлення та анонси зі світу аніме прямо в телеграм, без необхідності шукати інформацію самостійно.

7. Спілкування із *ChatGPT:* користувачі будуть мати можливість спілкуватися зі штучним інтелектом *ChatGPT* через діалог з ботом, не залишаючи телеграм.

8. Надсилання фідбеку розробнику бота: користувачі зможуть надсилати фідбек адміністратору щодо роботи бота.

9. Можливість фінансової підтримки: користувачі зможуть надавати фінансову підтримку розробнику чи адміністратору бота.

2.2 Проектування архітектури бота

При проектуванні архітектури телеграм-бота необхідно враховувати його масштабованість, гнучкість і продуктивність. Для досягнення цих цілей вибирається наступний технологічний стек та архітектурні рішення.

По-перше, бібліотека для роботи з телеграм *API.* Було обрано *Aiogram* через такі переваги:

1. Асинхронне програмування: *Aiogram* забезпечує підтримку асинхронного програмування, що дозволяє обробляти безліч запитів паралельно без блокування основного потоку виконання. Це підвищує продуктивність та чуйність робота.

2. Простота використання: бібліотека *Aiogram* має інтуїтивно зрозумілий та зручний інтерфейс, що робить розробку та підтримку бота більш простою та приємною. Її *API* легко розуміти і використовувати, навіть для розробників-початківців.

3. Обширна документація та спільнота розробників: *Aiogram* має велику

документацію та активну спільноту розробників, що забезпечує доступ до корисних ресурсів та підтримки у разі виникнення питань або проблем.

Оберемо бібліотеку для використання асинхронного програмування. Було обрано *Asyncio* з наступних причин:

1. Висока продуктивність: асинхронне програмування дозволяє ефективно використовувати ресурси та обробляти безліч запитів паралельно, що підвищує продуктивність та чуйність бота.

2. Ефективне використання ресурсів: *Asyncio* дозволяє уникнути блокування потоку виконання при очікуванні відповіді від зовнішніх сервісів, що економить ресурси та покращує масштабованість програми.

3. Простота розробки та підтримки: *Asyncio* забезпечує зручний та інтуїтивно зрозумілий інтерфейс для написання асинхронного коду, що спрощує розробку та підтримку бота.

Оберемо бібліотеку для парсингу даних з вебсайту *Reddit*. Було обрано *asyncpraw*. Це потужний інструмент для взаємодії з *API Reddit*. Переваги використання *asyncpraw*:

1. Асинхронне програмування та висока продуктивність: *Asyncpraw* дозволяє виконувати асинхронні запити до *API Reddit*, що робить її більш ефективною. Це особливо корисно під час роботи з великим обсягом даних або при паралельній обробці кількох запитів.

2. Простота використання: бібліотека має інтуїтивно зрозумілий синтаксис та добре документована, що спрощує її використання для розробників.

3. Підтримка асинхронних контекстів: *Asyncpraw* добре інтегрується з іншими асинхронними бібліотеками та фреймворками *Python*, такими як *asyncio, aiohttp* та іншими. Це забезпечує можливість ефективної роботи з *Reddit API* в рамках асинхронної програми.

4. Широкі можливості: Бібліотека надає широкий набір можливостей для роботи з *Reddit API*, включаючи одержання публікацій з різних субреддитів, їх пошук за ключовими словами, отримання коментарів, інформації про

користувачів і багато іншого. Це дозволяє розробникам створювати потужні та гнучкі додатки для аналізу та обробки даних із *Reddit*.

5. Обширна документація та спільнота розробників: *Asyncpraw* має велику документацію та активну спільноту розробників, що забезпечує доступ до корисних ресурсів та підтримки у разі виникнення питань або проблем.

2.3 Проектування серверної частини бота

Серверна частина телеграм-бота забезпечує взаємодію між пристроями користувачів, телеграм та зовнішніми *API* (*OpenAI, SauceNAO, Reddit, TikTok*). У цьому розділі ми розглянемо *API*, протоколи, команди та потоки даних, які використовуються для зв'язку між цими компонентами. Почнемо з *API* та протоколів:

1. Телеграм-бот *API*

Телеграм надає *API* для створення та управління ботами, що дозволяє відправляти та отримувати повідомлення, керувати файлами та медіа, використовувати *inline* та *reply* клавіатури, обробляти команди від користувача та багато іншого. Протокол: *HTTPS.* Основні команди:

- *getUpdates* – отримання оновлень (повідомлень, команд) від користувачів;
- *sendMessage* – надсилання текстових повідомлень користувачам;
- *sendPhoto, sendVideo, sendDocument* – надсилання медіафайлів користувачам;
- *editMessageText, editMessageCaption* – редагування раніше надісланих повідомлень;
- *answerCallbackQuery* – відповідь на запити *inline* клавіатур;
- *setWebhook* – налаштування веб-хука для автоматичного отримання оновлень;
- *deleteWebhook* – видалення веб-хука.

2. *OpenAI API*

API OpenAI використовується для інтеграції можливостей *ChatGPT*.

Протокол: *HTTPS*. Основні команди:

- *completions* – генерація текстів на основі заданого контексту.
- *chat* – ведення діалогу з користувачем.

3. *SauceNAO API*

API для пошуку інформації з зображень (аніме) на основі скриншотів, завантажених користувачем. Протокол: *HTTPS*. Основна команда:
search – пошук інформації щодо зображення.

4. *Reddit API*:

API для отримання даних з соц. мережі *Reddit*, таких як останні новини або популярні публікації. Протокол: *HTTPS*. Основні команди:

- *hot* – отримання гарячих постів із вибраного сабреддита.
- *new* – отримання нових постів.
- *top* – отримання найкращих постів за вибраний період.

5. *TikTok Video No Watermark API*:

API для завантаження відео із *TikTok* без водяного знаку. Протокол: *HTTPS*. Основні команди:

- *getVideoInfo* – отримання інформації про відео використовуючи його *URL*.
- *downloadVideo* – завантаження відео без водяного знака за наданим *URL*.

Тепер розглянемо взаємодію бота з телеграм.

Обробка вхідних даних:

1. Отримання даних: бот отримує дані від телеграм (повідомлення, команди, колбеки тощо) через *webhook* або періодично просить оновлення за допомогою *getUpdates*.

2. Формат даних: вхідні дані є *JSON*-об'єктами, що містять інформацію про повідомлення, команди, медіафайли та взаємодії з користувачем.

Приклад вхідних даних, які отримує бот, наведено на рис. 2.4.

Рис. 2.4 – Приклад вхідних даних

Приклад робочого процесу. Користувач на своєму пристрої надсилає боту команду «/*start*»:

1. Телеграм сервер отримує команду та пересилає її на сервер бота через *webhook* або *getUpdates*.

2. Сервер бота отримує команду від телеграм сервера.

3. Сервер бота перевіряє базу даних на наявність інформації про користувача. Якщо користувач новий, додає його в базу даних.

4. Сервер бота надсилає вітальне повідомлення користувачу з інформацією про доступні команди.

5. Телеграм сервер отримує відповідь від сервера бота і пересилає її користувачу.

На рис. 2.5 наведено діаграму послідовності дій опрацювання команди «/*start*».

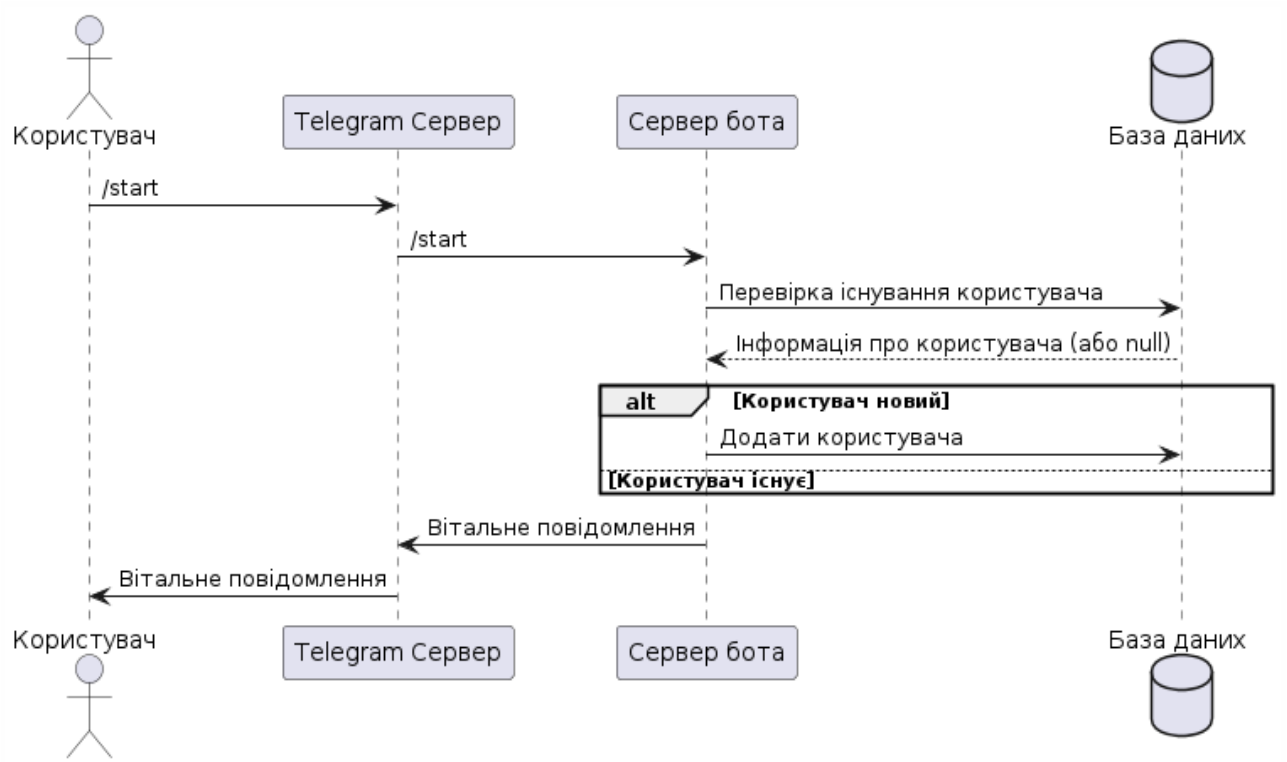


Рис. 2.5 – Діаграма послідовності дій опрацювання команди «/*start*»

Користувач відправляє команду «/*video*» та посилання для завантаження відео з *TikTok*:

1. Телеграм сервер отримує команду та пересилає її на сервер бота.

2. Сервер бота отримує команду та посилання на відео.

3. Сервер бота зберігає інформацію про запит в базу даних.

4. Сервер бота взаємодіє із зовнішнім *API TikTok Video No Watermark* для завантаження відео без водяного знака:

- відправляє запит до *api* з посиланням на відео;
- отримує відповідь від *api* з посиланням на відео без водяного знака.

5. Сервер бота отримує відео від *API* і надсилає його користувачу.

6. Телеграм сервер отримує відео від сервера бота і пересилає його користувачу.

На рис. 2.6 наведено діаграму послідовності дій опрацювання команди «/*video*».

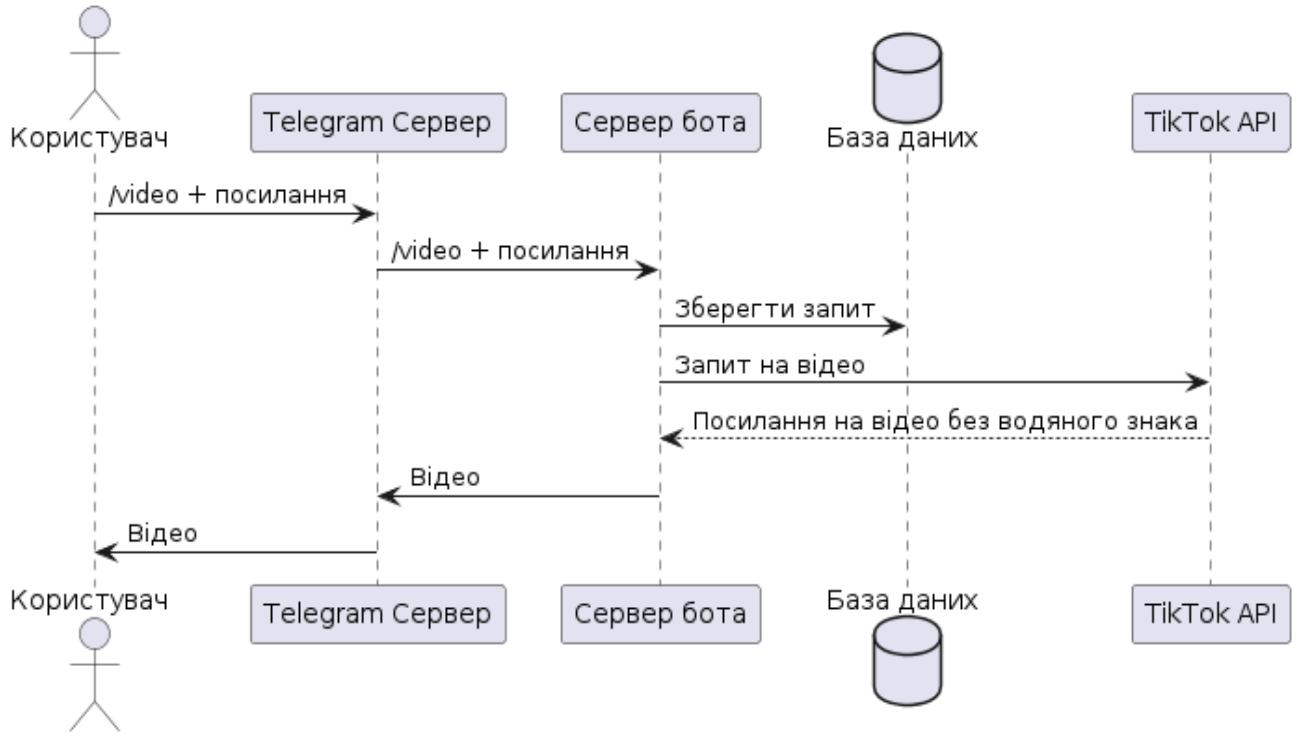


Рис. 2.6 – Діаграма послідовності дій опрацювання команди «/*video*»

Користувач відправляє команду «/*sauce*» та картинку для пошуку аніме по скриншоту:

1. Телеграм сервер отримує команду та пересилає її на сервер бота.
2. Сервер бота отримує зображення від користувача.
3. Сервер бота зберігає інформацію про запит в базу даних.
4. Сервер бота надсилає зображення до *SauceNAO API* для пошуку інформації про аніме:
   - відправляє зображення до *API*;
   - отримує відповідь від *API* з результатами пошуку.
5. Сервер бота отримує дані від *API* і формує відповідь для користувача.
6. Сервер бота надсилає результати користувачу.
7. Телеграм сервер отримує результати від сервера бота і пересилає їх користувачу.

На рис. 2.7 наведено діаграму послідовності дій опрацювання команди «/*sauce*».

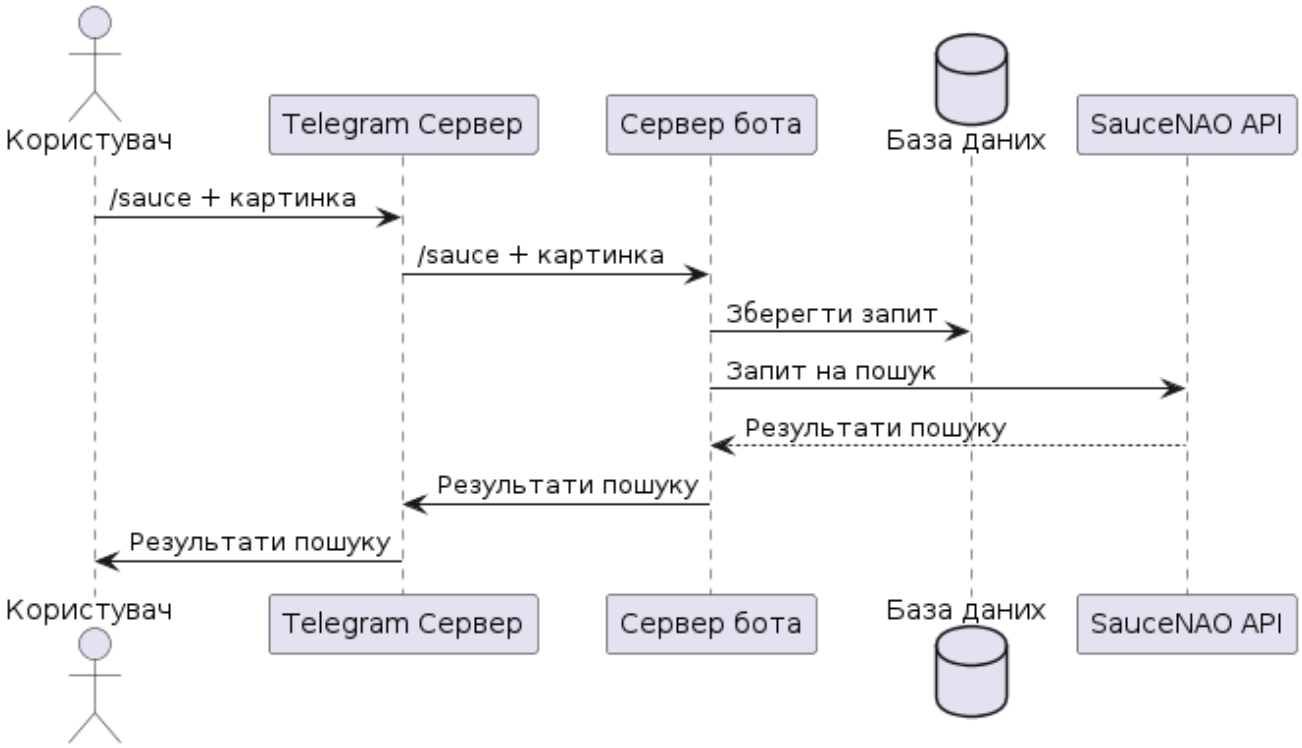


Рис. 2.7 – Діаграма послідовності дій опрацювання команди «/*sauce*»

Користувач відправляє команду «/*news*» для отримання останніх новин зі світу аніме:

1. Телеграм сервер отримує команду та пересилає її на сервер бота.
2. Сервер бота отримує команду від користувача.
3. Сервер бота зберігає інформацію про запит в базу даних.
4. Сервер бота взаємодіє із *Reddit API* для отримання останніх новин:
   - відправляє запит до *Reddit API* для отримання топових постів з певного субреддіту;
   - отримує відповідь від *API* з останніми новинами.
5. Сервер бота обробляє отримані новини та формує повідомлення для користувача.
6. Сервер бота надсилає користувачу повідомлення з останніми новинами.
7. Телеграм сервер отримує повідомлення від сервера бота і пересилає його користувачу.

На рис. 2.8 наведено діаграму послідовності дій опрацювання команди «/*news*».

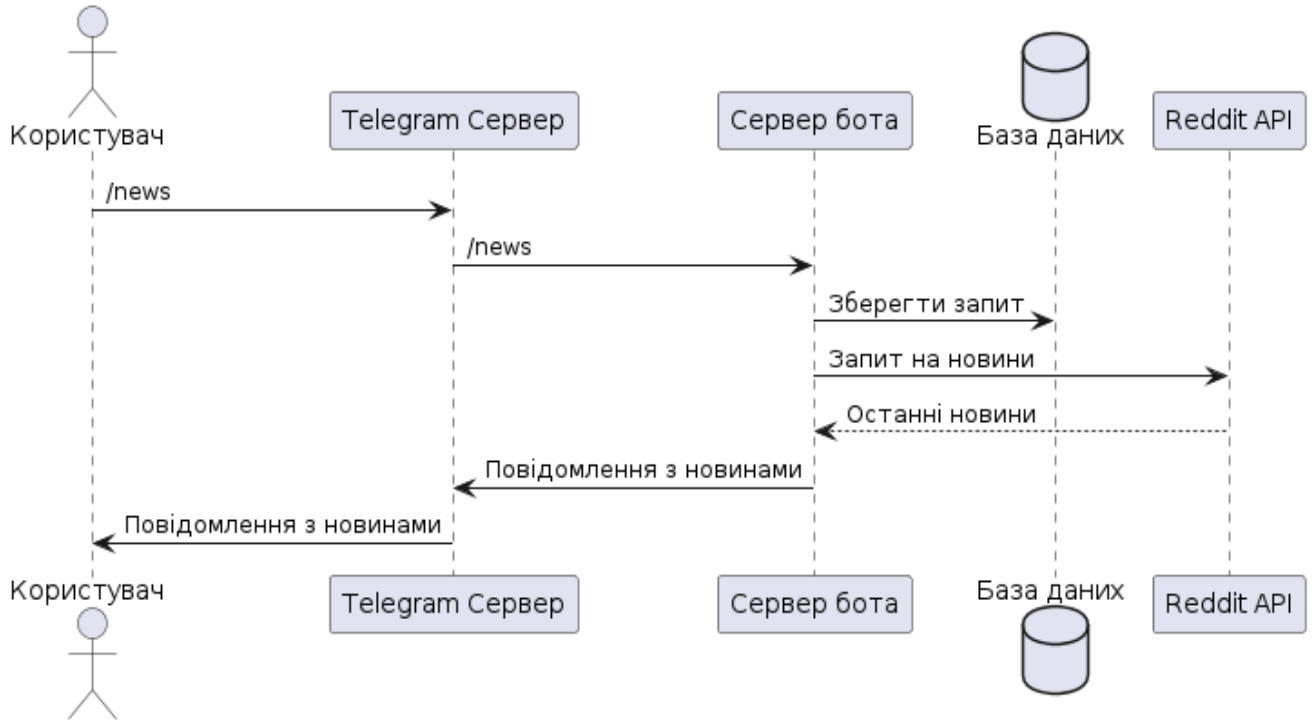


Рис. 2.8 – Діаграма послідовності дій опрацювання команди «/*news*»

Розглянемо детальніше взаємодію телеграм-бот *API, Aiogram* 3 та серверу телеграм.

Для взаємодії з телеграм сервером використовується телеграм-бот *API*. Це *API* надає різноманітні методи для відправки та отримання повідомлень, обробки команд і взаємодії з користувачами. Бібліотека *Aiogram 3* є зручною оболонкою для роботи з телеграм-бот *API*, яка дозволяє легко інтегруватися з телеграм і забезпечує асинхронну обробку запитів.

У цьому процесі відбувається інтерактивна взаємодія між користувачем, сервером телеграм, сервером бота, базою даних та зовнішніми *API*. Кожен запит проходить через сервер телеграм, який пересилає команди на сервер бота, де вони обробляються, взаємодіють із зовнішніми *API* для отримання необхідних даних або виконання завдань, та зберігають інформацію в базі даних для подальшої обробки.

Для кращого розуміння побудуємо схему (рис. 2.9):

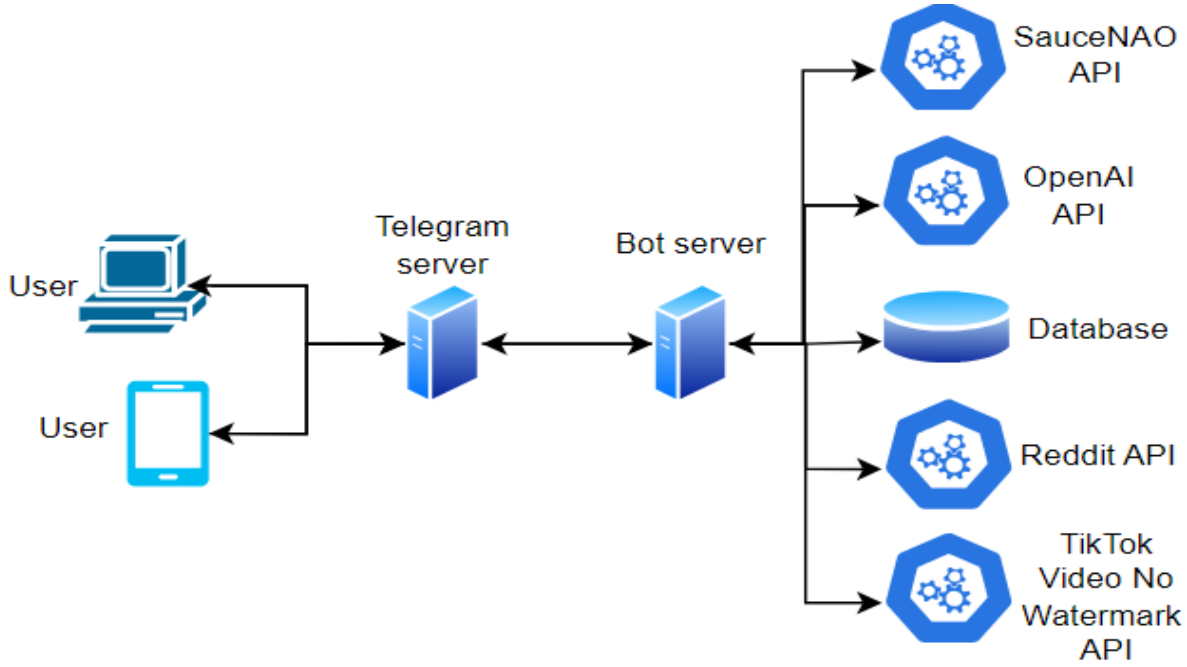


Рис. 2.9 – Загальна схема серверної частини бота

Отже, серверна частина бота включає інтеграцію з різними *API* для надання широкого спектру функціоналу користувачам. Взаємодія між користувачами, телеграм та зовнішніми сервісами здійснюється через безпечні *HTTPS* з'єднання та *API* виклики, забезпечуючи надійну та ефективну роботу бота.

2.4 Проектування інтерфейсу бота

По-перше, розробимо структуру команд бота. Складатися вона буде з наступних команд (кожна з яких у першу чергу активує машину станів та змінює стан бота):

1. «*/start*» – частіше за все ця команда є першим повідомленням, яке надсилається користувачем автоматично після запуску бота. Бот відповість великим повідомленням із привітанням та описом своїх можливостей, а також внесе користувача до бази даних. Детальніше про роль бази даних розповідається у підрозділі 2.4.

2. «*/info*» – це скорочена версія команди «*/start*». У відповідь бот надасть коротше повідомлення, яке містить лише перелік своїх можливостей.

3. «*/commands*» – бот надішле користувачеві список усіх доступних

команд.

4. «*/video*» – активує функціонал завантаження відео із соц. мережі *TikTok*. Бот запропонує надіслати йому посилання на публікацію. У відповідь відправить відео без водяного знаку, яке можна завантажити.

5. «*/hashtags*» – активує функціонал генерації хештегів для публікацій у соц. мережі *TikTok*. Бот запропонує надіслати йому тему публікації і у відповідь надасть список хештегів.

6. «*/chatgpt*» – активує функціонал спілкування зі штучним інтелектом *ChatGPT*. У відповідь повідомить про те, що *ChatGPT* підключено і наступні повідомлення будуть вважатися запитами до нього.

7. «*/todo*»: активує функціонал управління списком завдань користувача. У відповідь бот надішле меню із відповідним функціоналом та інструкціями щодо його використання.

8. «*/newtask TASK_NAME # DATETIME*» – ця команда є частиною функціоналу керування списками задач. Вона буде використовуватися користувачем для створення нових задач. Кожна з яких заноситься у базу даних.

9. «*/mytasks*» – ця команда є частиною функціоналу керування списками задач. Вона буде використовуватися користувачем для перегляду його існуючих задач. Перед тим як надіслати список, бот буде порівнювати заданий користувачем час в кожній із задач та видаляти із списку прострочені.

10. «*/news*» – активує функціонал надсилання останніх новин зі світу аніме. У відповідь бот надішле 10 останніх новин з певного вебсайту.

11. «*/sauce*» – активує функціонал пошуку аніме-серіалу. У відповідь бот попросить надіслати йому скриншот. Після отримання якого надішле повідомлення з результатами пошуку, із красиво оформленим банером і короткою інформацією про аніме-серіал.

12. «/feedback» – активує функціонал зворотного зв’язку. Бот відповість, що наступне повідомлення користувача буде вважатися фідбеком і буде надіслано адміністратору чи його розробнику.

13. «*/coffee*» – активує функціонал за допомогою якого користувач зможе

матеріально підтримати розробку бота. У відповідь бот надішле повідомлення із посиланням на вебсайт, де це можна зробити.

Для зручної та ефективної взаємодії бота з користувачем використовуються клавіатури з кнопками. У телеграм-ботів є два види клавіатур, *Inline* та *Reply*. Відрізняються вони лише тим, що перша прикріплюється у кінці повідомлення бота, а друга, після певних дій, з'являється у якості меню під рядком введення повідомлення користувача. Приклади Inline та Reply наведені на рис. 2.10 та 2.11 відповідно.

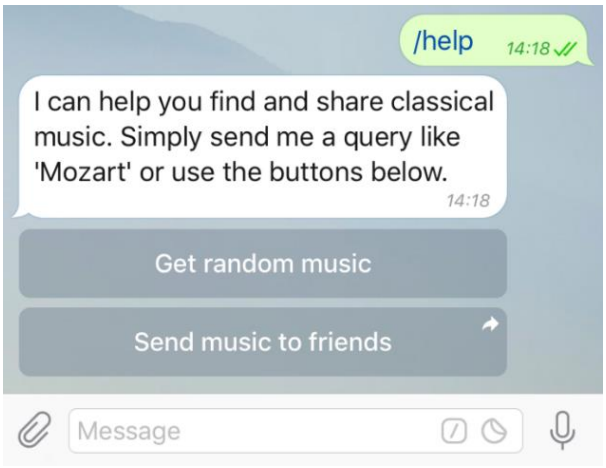


Рис. 2.10 – Приклад *Inline* клавіатури

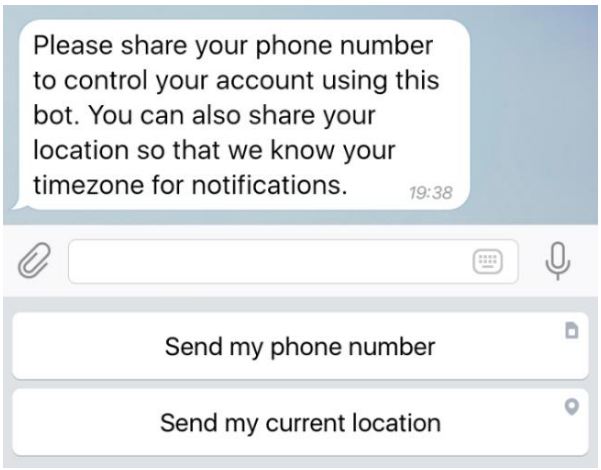


Рис. 2.11 – Приклад *Reply* клавіатури

У розробці бота було вирішено використовувати *Inline* клавіатури. Спроектуємо загальну структурну схему клавіатур бота (рис. 2.12), на якій чітко видно шлях з головного меню до виконання певної функції. Де *Start* –

головна клавіатура.

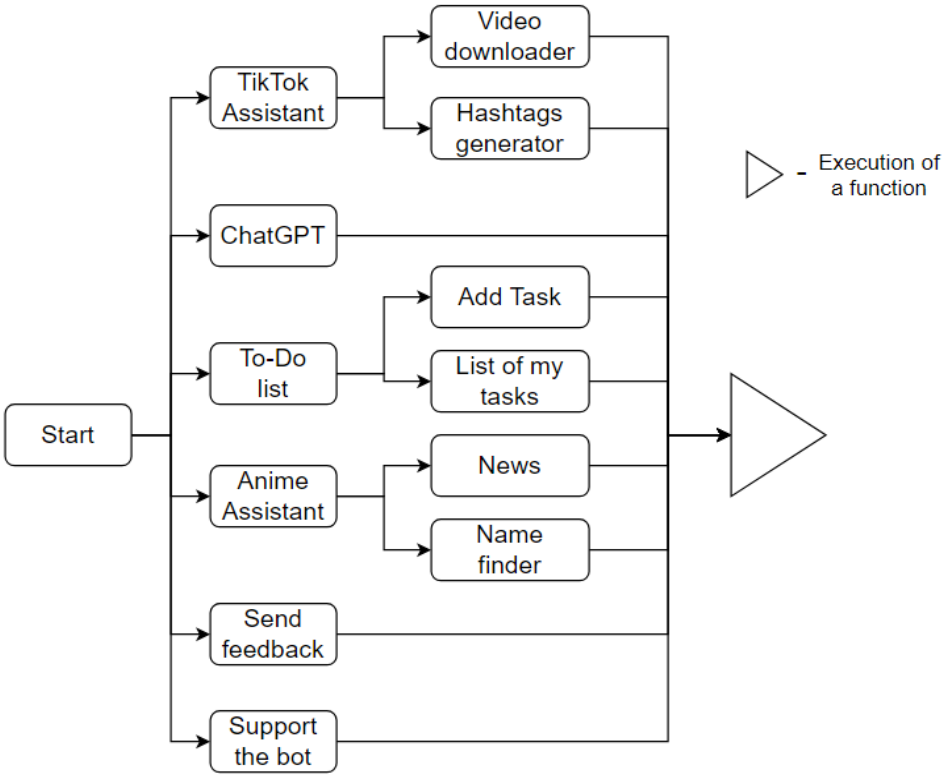


Рис. 2.12 – Загальна структурна схема клавіатур та функцій бота

Тепер розробимо макети повідомлень, які бот буде надсилати у якості відповідей на різні команди чи переходи через клавіатури:

1. Макет відповіді на команду «*/start*» (рис. 2.13)

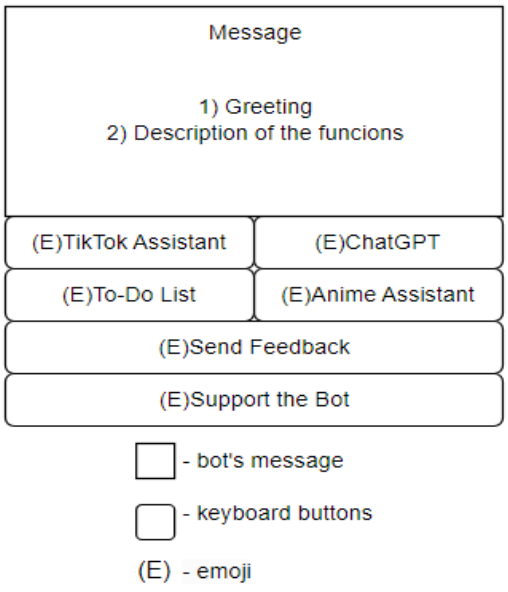


Рис. 2.13 – Макет відповіді бота на команду «*/start*»

2. Макет меню *TikTok Assistant* (рис. 2.14)

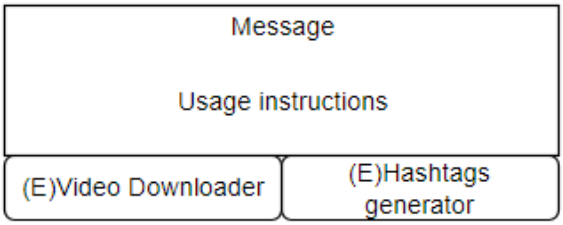


Рис. 2.14 – Макет меню *TikTok Asssistant*

3. Макет меню *To-Do List* (рис. 2.15)

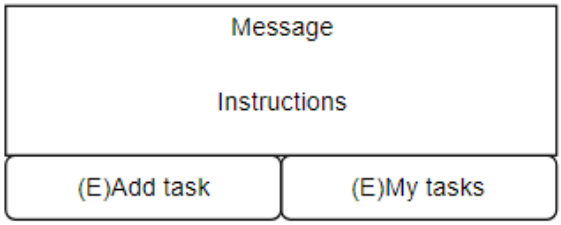


Рис. 2.15 – Макет меню *To-Do List*

4. Макет меню *Anime Assistant* (рис. 2.16)

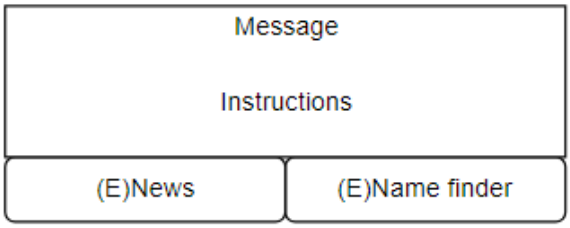


Рис. 2.16 – Макет меню *Anime Asssistant*

2.5 Проектування бази даних

У цьому розділі описується процес проектування бази даних, включаючи виділення сутностей, нормалізацію та створення основних таблиць бази даних. Для зберігання інформації використовується реляційна база даних *SQL*.

Виділення сутностей – першим кроком у проектуванні бази даних є

визначення основних сутностей, які необхідно зберігати. У випадку нашого телеграм-бота, ми виділяємо дві основні сутності:

1. Користувачі (*Users*) – містить інформацію про користувачів, які взаємодіють з ботом.

2. Завдання (*Tasks*) – містить інформацію про завдання, створені користувачами через функціонал *To-Do list*.

Нормалізація бази даних – це процес організації даних у таблицях таким чином, щоб мінімізувати дублювання та забезпечити цілісність даних. Ми виконуємо нормалізацію до третьої нормальної форми (*3NF*), яка передбачає наступні кроки:

1. Перша нормальна форма (*1NF*):
   - усі значення в таблиці є атомарними (неподільними);
   - у таблиці немає повторюваних груп або масивів.
2. Друга нормальна форма (*2NF*):
   - таблиця задовольняє *1NF*.
   - усі неключові атрибути залежать від усього первинного ключа.
3. Третя нормальна форма (*3NF*):
   - таблиця задовольняє *2NF*;
   - жоден неключовий атрибут не залежить транзитивно від первинного ключа.

Структура таблиць – на основі виділених сутностей та нормалізації, ми визначили структуру двох основних таблиць: «*users*» та «*tasks*».

Таблиця «*users*»: ця таблиця призначена для зберігання інформації про користувачів, які запустили бота. Вона містить такі поля:

- *Id* – унікальний ідентифікатор користувача (автоматично генерується);
- *user_id* – ідентифікатор користувача в телеграм;
- *user_name* – ім'я користувача в телеграм;
- *first_launch* – дата та час першого запуску бота користувачем.

Побудова таблиці (листінг 2.1):

Листінг 3.1. Запит створення таблиці *users*

```
CREATE TABLE users (
id INTEGER PRIMARY KEY AUTOINCREMENT,
user_id INTEGER (11) UNIQUE,
user_name TEXT UNIQUE,
first_launch DATETIME
);
```

Таблиця «*tasks*»: ця таблиця призначена для зберігання інформації про завдання, створені користувачами через функціонал *To-Do list*. Вона містить такі поля:

- *task_id* – унікальний ідентифікатор завдання (автоматично генерується);
- *user_id* – ідентифікатор користувача, якому належить завдання;
- *description* – опис завдання;
- *datetime* – дата та час створення завдання.

Побудова таблиці (листінг 3.2):

Листінг 3.2. Запит створення таблиці *tasks*

```
CREATE TABLE tasks (
task_id INTEGER PRIMARY KEY AUTOINCREMENT,
user_id INTEGER,
description TEXT,
datetime DATETIME,
FOREIGN KEY (user_id) REFERENCES users (user_id)
);
```

Ці таблиці забезпечують структуроване зберігання даних про користувачів та їхні завдання, а також забезпечують зв'язок між ними через зовнішній ключ «*user_id*» у таблиці «*tasks*», який посилається на таблицю «*users*». Така структура бази даних дозволяє ефективно управляти та аналізувати інформацію, зібрану ботом, та забезпечує надійність і цілісність даних.

На рис. 2.17 наведено схему бази даних нашого проекту.

Отже, у цьому підрозділі ми описали процес проектування бази даних для нашого бота. Було виділено основні сутності, проведено нормалізацію бази даних до третьої нормальної форми та створено структуру таблиць «*users*» і «*tasks*». Це забезпечує ефективне зберігання та обробку даних, необхідних для роботи бота. У наступному розділі буде розглянуто вибір СУБД і обґрунтовано

його використання.

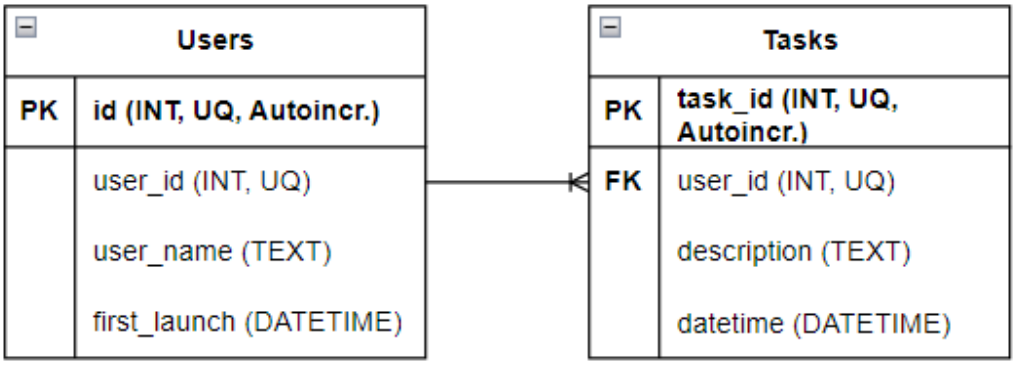


Рис. 2.17 – Схема бази даних

Висновок до другого розділу

Розглянуто проектування багатофункціонального телеграм-бота. У процесі проектування визначені цілі та завдання бота, розроблені структура діалогів і команд, а також логіка взаємодії з користувачем. Було обрано стек технологій, що включає бібліотеку *Aiogram* для асинхронної взаємодії з телеграм *API* і базу даних *SQLite*. Ці технології забезпечують високу продуктивність та зручність розробки.

Виконано проектування бази даних, що включає виділення основних сутностей, нормалізацію даних та створення таблиць для зберігання інформації про користувачів та їх завдання. Таблиці «*users*» та «*tasks*» були спроектовані з урахуванням вимог цілісності даних та зручності їх обробки. Це дозволяє ефективно управляти даними, забезпечуючи надійну роботу бота та зручність для користувачів.

РОЗДІЛ 3
ПРАКТИЧНА РЕАЛІЗАЦІЯ ТА РЕЗУЛЬТАТИ ДОСЛІДЖЕННЯ

3.1 Вибір і обґрунтування програмних засобів реалізації проекту

Оберемо програмні засоби та технології, які будуть використовуватися для розробки бота:

1. Мова програмування: *Python* версії 3.11 була обрана як основний інструмент для розробки телеграм-бота з кількох причин:

a) простота та читаність коду: *Python* відомий своєю простотою та лаконічністю, що робить розробку та підтримку коду більш ефективними. Це особливо важливо для проектів, де необхідно швидко прототипувати та впроваджувати новий функціонал.

b) велика екосистема бібліотек: *Python* має велику екосистему бібліотек і фреймворків, включаючи бібліотеки для роботи з асинхронним програмуванням та розробки телеграм-ботів. Це полегшує інтеграцію із зовнішніми сервісами та прискорює розробку.

c) широке співтовариство розробників. *Python* має велику та активну спільноту розробників, які створюють та підтримують безліч корисних інструментів та бібліотек. Це забезпечує доступ до великої бази знань та підтримку у разі виникнення проблем.

2. Зберігання даних: для ефективного зберігання та керування даними, пов'язаними з користувачами та їх завданнями, в буде використовуватися реляційна база даних *SQLiteStudio* версії 3.4.4. Вона була обрана з наступних причин:

a) простота у використанні та налаштуванні – *SQLiteStudio* має простий та інтуїтивно зрозумілий інтерфейс, що спрощує роботу з базою даних та інтеграцію її в проект. Це особливо важливо для невеликих проектів, де потрібна легка та швидка установка.

b) підтримка *SQL: SQLiteStudio* підтримує мову *SQL*, що забезпечує гнучкість та потужні можливості роботи з даними, такі як виконання запитів, створення індексів та керування транзакціями.

c) кросплатформенність та портативність: *SQLiteStudio* працює на різних операційних системах і легко переноситься, що робить її зручною для розробки та використання на різних платформах.

d) підтримка транзакцій: *SQLiteStudio* підтримує транзакції для забезпечення цілісності даних та захисту від втрати даних у разі збою.

3. Система контролю версій: на всіх етапах розробки використовувалася система контролю версій *Git*. *Git* дозволяє відстежувати зміни в коді, повертатися до попередніх версій, працювати над різними гілками розробки та співпрацювати з іншими розробниками. Використання *Git* забезпечує надійність, відстежуваність змін та можливість швидкого відновлення у разі помилок.

4. Середа розробки – у якості середи розробки було обрано *Visual Studio Code*. Він може бути корисним при розробці телеграм-бота завдяки наступним перевагам:

a) підтримка *Python: VS Code* має вбудовану підтримку *Python*, включаючи підсвічування синтаксису, автодоповнення, форматування коду та налагодження. Це спрощує процес написання та налагодження коду робота.

b) розширення для телеграм: *VS Code* пропонує розширення для роботи з телеграм *API* та *Aiogram*, які можуть надати додаткові інструменти та функції для розробки бота.

c) інтегрований термінал: *VS Code* має вбудований термінал, який дозволяє запускати бота, виконувати команди Git та керувати віртуальним оточенням *Python* безпосередньо з редактора.

d) легкість та швидкодія: *VS Code* є легковагим і швидким редактором, який споживає менше ресурсів, ніж багато інших *IDE*.

Це особливо важливо при роботі на слабких комп'ютерах або одночасному запуску декількох програм.

е) налаштування: *VS Code* можна налаштувати під свої уподобання, використовуючи теми, розширення та налаштування користувача. Це дозволяє створити комфортне та продуктивне середовище розробки.

3.2 Створення бота

У телеграм боти створюються за допомогою офіційного бота *BotFather*. Він допомагає створювати та керувати іншими ботами у цій соціальній мережі. Він надає простий інтерфейс для створення нового бота, отримання токена доступу, зміни налаштувань, видалення бота тощо. Профіль бота продемонстровано на рис. 3.1.

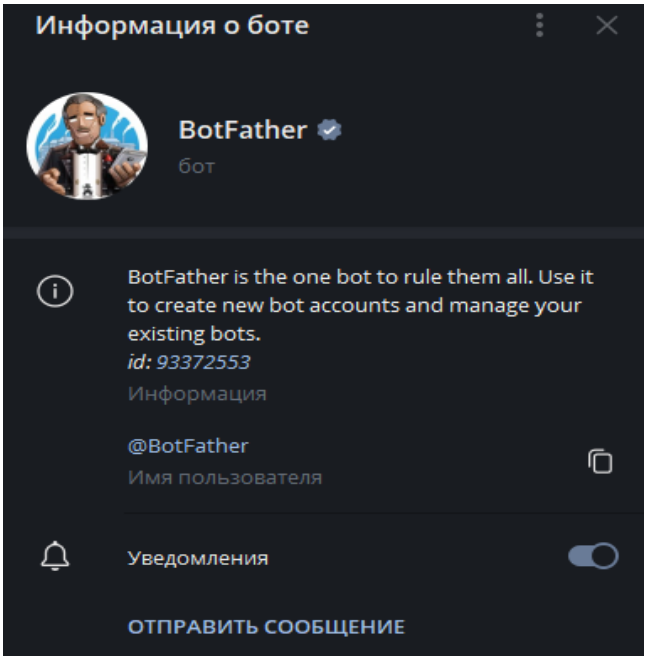


Рис. 3.1 – Профіль *BotFather*

Розглянемо процес створення бота в BotFather:

1. Знайти *BotFather* для цього потрібно відкрити телеграм і ввести «*@BotFather*» у рядку пошуку. Вибрати офіційний обліковий запис (з синьою галочкою). Натиснути «Запустити» (або надіслати команду «/*start*») для

активації *BotFather*.

2. Створення нового бота:

- надіслати команду *«/newbot»* і дотримуватися інструкцій;
- ввести ім'я, яке відображатиметься у чатах;
- ввести унікальне ім'я користувача, яке закінчується на «*bot*» (наприклад, «*my_new_bot*»).

3. Отримати токен – після успішного створення *BotFather* видасть токен доступу. Це довгий рядок символів, який потрібний для взаємодії з вашим ботом через *API* телеграм.

Перейдемо до створення бота *Geek's Friend*.

По-перше, переглянемо функціонал, який має *BotFather*, надіславши повідомлення «*menu*».

Результат виконання команди продемонстровано на рис. 3.2.

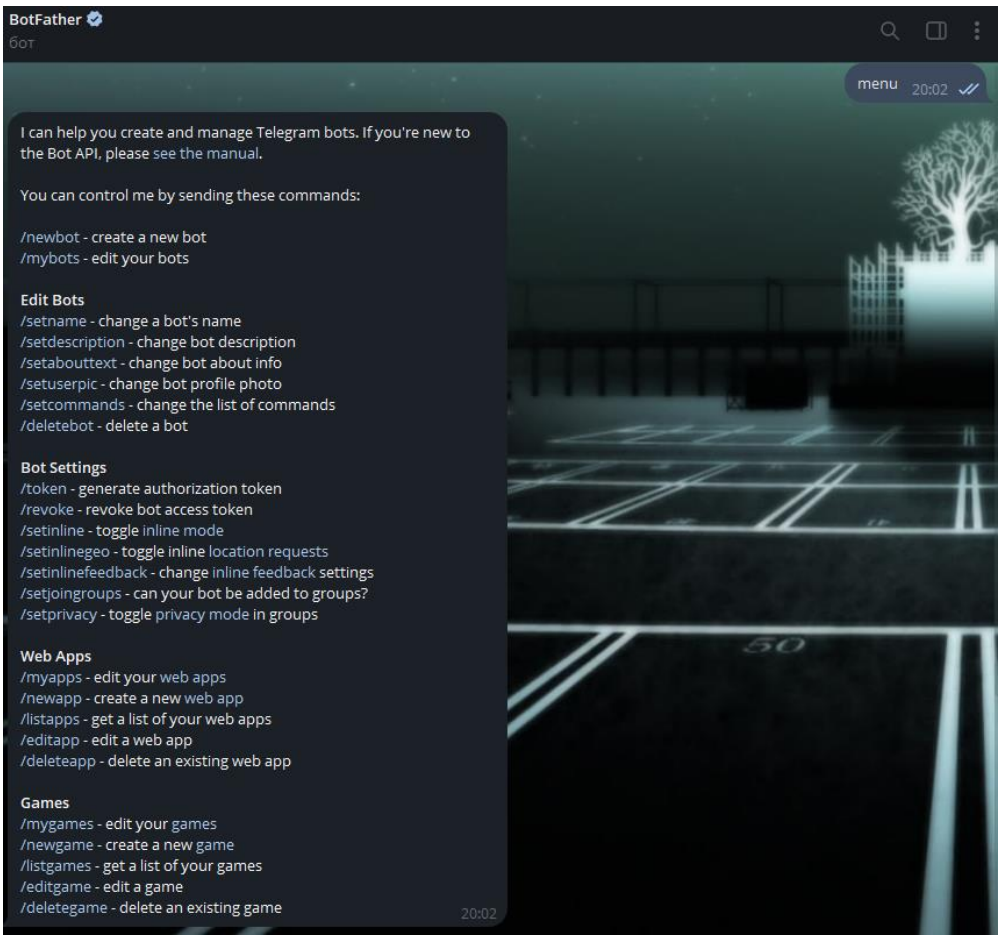


Рис. 3.2 – Функціонал *BotFather*

Створимо бота *Geek's Friend* з ім'ям користувача *GeeksFriendBot*.

Процес створення продемонстровано на рис. 3.3.

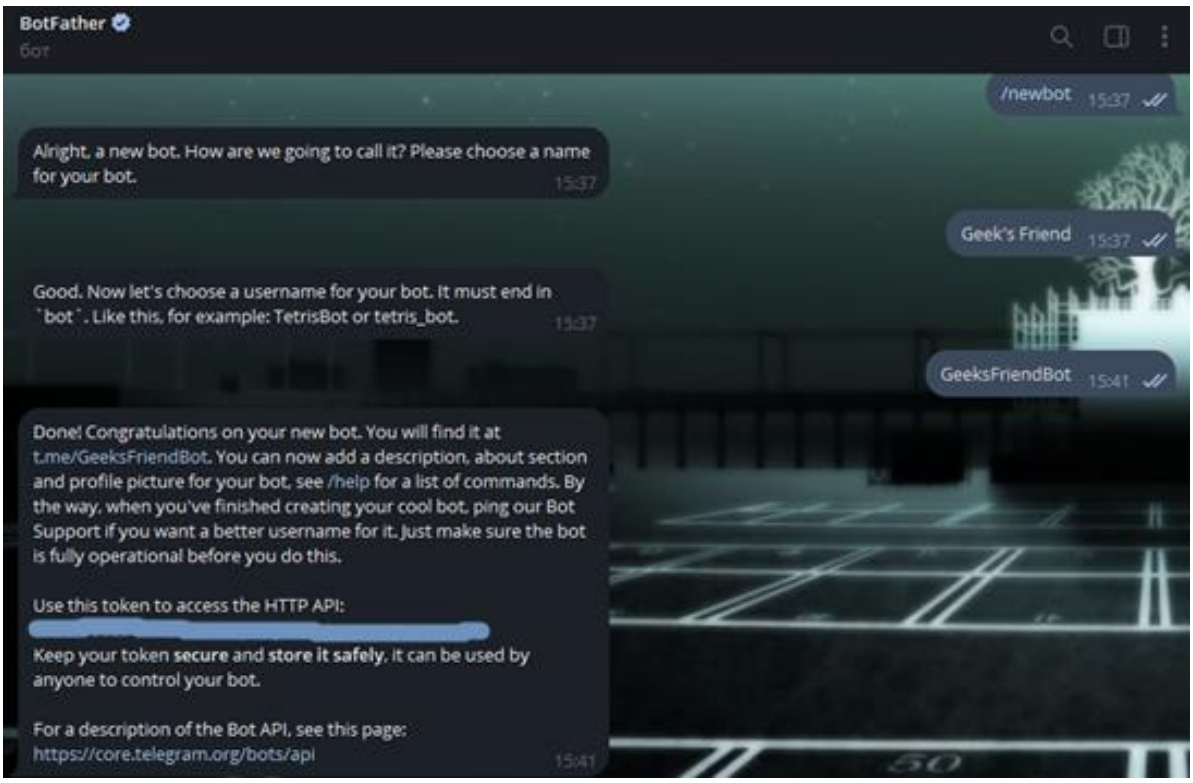


Рис. 3.3 – Створення бота *Geek's Friend*

Бот було успішно створено, ми отримали токен. Тепер, надіславши команду «/*mybots*», ми побачимо інформацію про нашого бота (рис.3.4):

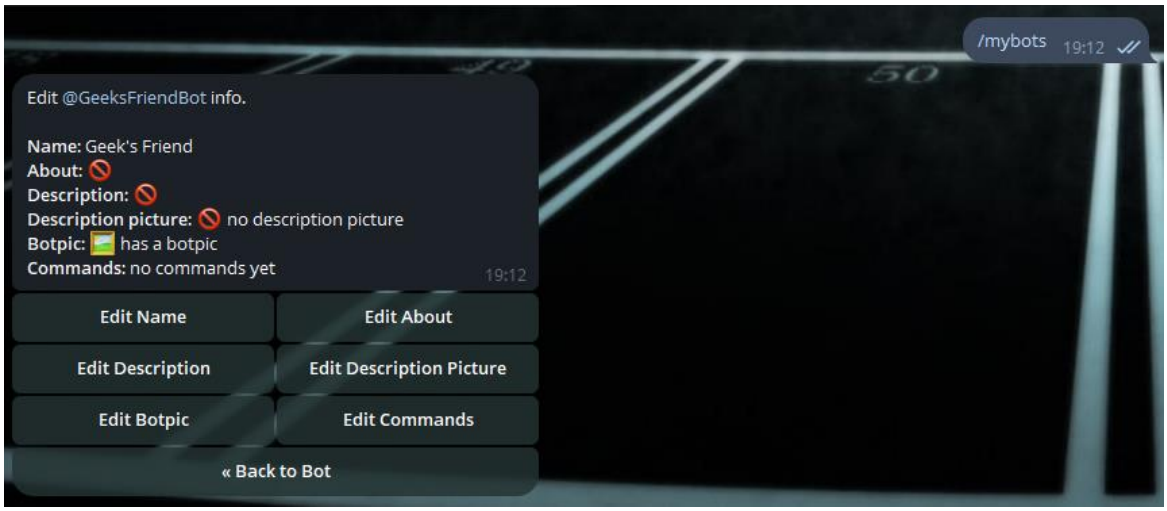


Рис. 3.4 – Інформація про бота

Встановимо фото профіля бота, натиснувши на кнопку «*Edit Botpic*».

Це продемонстровано на рис. 3.5.

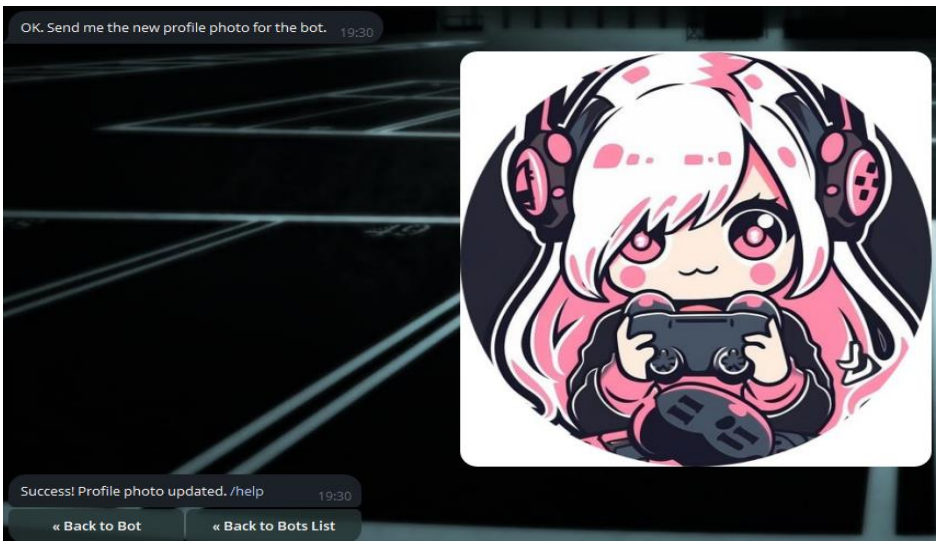


Рис. 3.5 – Встановлення фото профіля бота

Додамо опис бота. Процес додання продемонстровано на рис. 3.6.

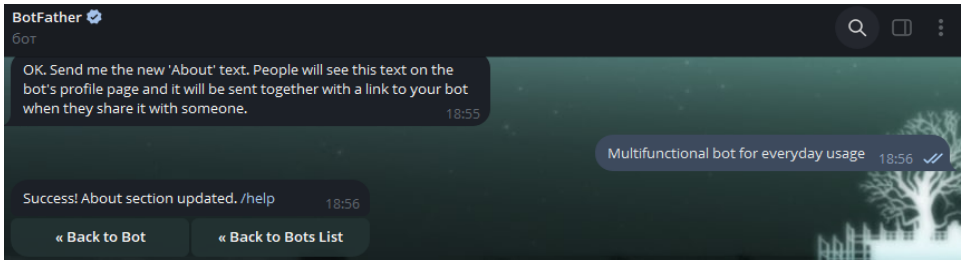


Рис. 3.6 – Додання опису бота

Створимо меню команд, як у *BotFather*, яке розташоване біля поля введення повідомлення боту.

Його можна побачити на рис. 3.7.

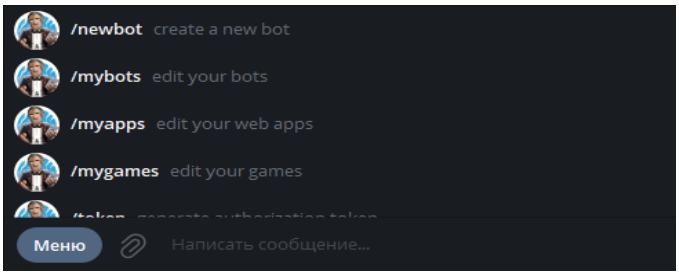


Рис. 3.7 – Меню команд *BotFather*

Щоб створити таке меню для нашого бота, використаємо команду «/*setcommands*». Результат продемонстровано на рис. 3.8.

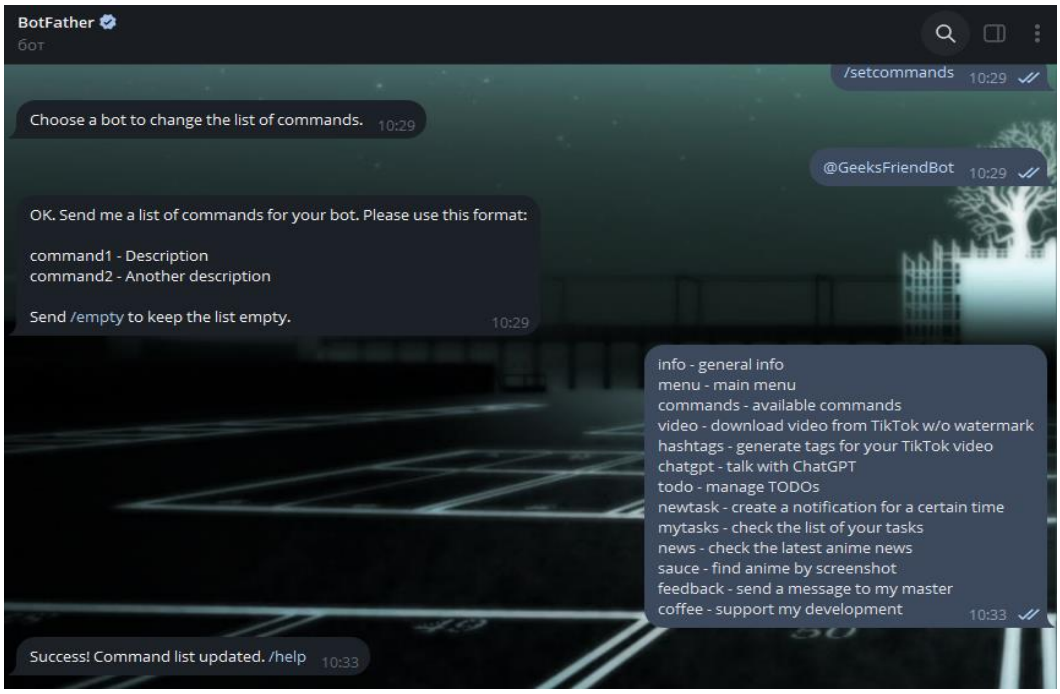


Рис. 3.8 – Створення меню команд

Після всіх налаштувань, ми отримали створеного бота (рис. 3.9) і його меню (рис. 3.10):

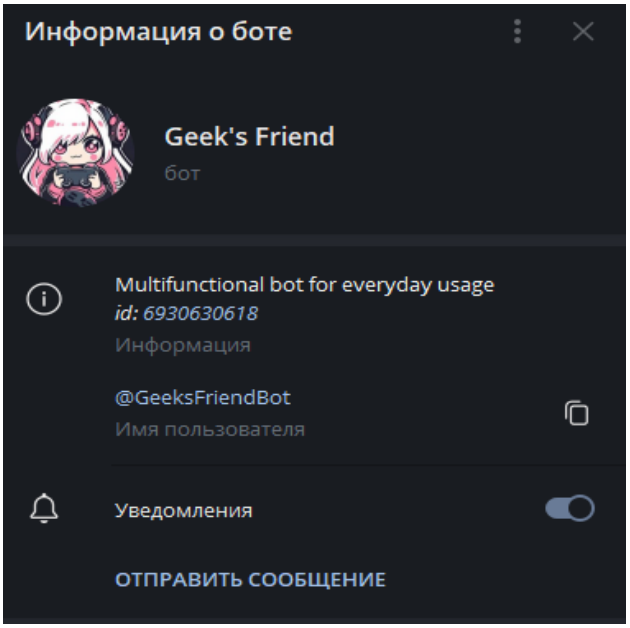


Рис. 3.9 – Бот *Geek's Friend*

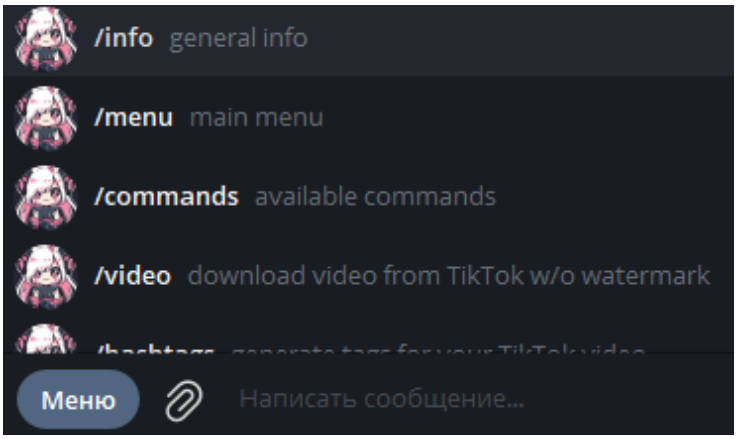


Рис. 3.10 – Меню команд бота *Geek's Friend*

3.3 Розробка файлової структури проекту

Під час розробки багатофункціонального телеграм-бота важливо організувати файлову структуру проекту таким чином, щоб забезпечити зручність розробки, підтримки та масштабування. У цьому підрозділі буде описана структура проекту *Geek's Friend* та призначення основних файлів і каталогів. Структура проекту продемонстрована на рис. 3.11 та 3.12.

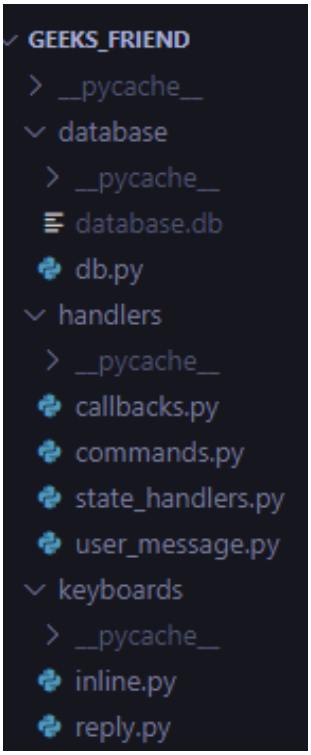


Рис. 3.11 – Файлова структура проекту (частина 1)

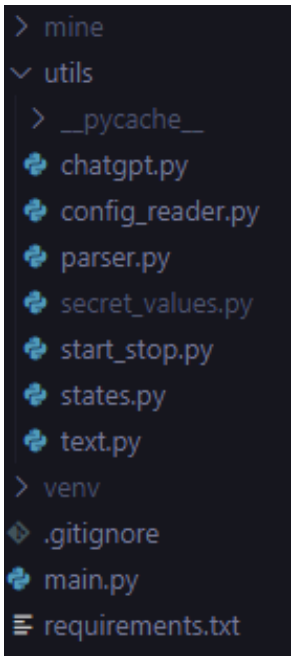


Рис. 3.12 – Файлова структура проекту (частина 2)

Розглянемо файлову структуру проекту:

1. *main.py*: головний файл проекту, в якому відбувається ініціалізація бота, його основні параметри та запуск.

2. *handlers*/: каталог, в якому зберігаються обробники подій (повідомлень, команд, запитів тощо). Кожен файл обробника відповідає за певну функціональність бота.

3. *keyboards*/: каталог, в якому зберігаються файли з визначеннями клавіатур бота (*inline* та *reply*).

4. *states*/: каталог, в якому зберігаються файли, що описують стани машини станів (*FSM*) бота.

5. *utils*/: каталог для допоміжних модулів та функцій.

6. *database/*: каталог файлів, пов’язаних з з базою даних.

7. *mine/*: каталог розробника з додатковими файлами для розробки (*TODO*, картинка профілю бота та ін.).

8. *requirements.txt*: файл зі списком залежностей проекту (бібліотеки *Python*, які необхідно встановити для роботи бота).

9. *.gitignore*: файл, що містить список файлів та каталогів, які *Git* повинен ігнорувати (не відстежувати).

3.4 Розробка бази даних

Використовуючи код, описаний у розділі проектування бази даних, створимо таблиці у програмі *SQLiteStudio*.

Процес створення таблиці «*users*» продемонстровано на рис. 3.13.

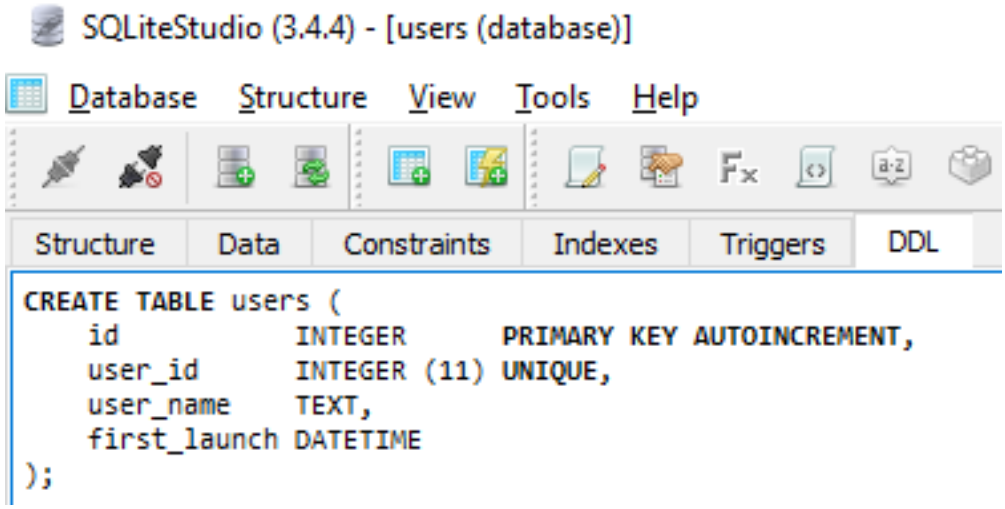


Рис. 3.13 – Створення таблиці *users*

Результат можна побачити на рис. 3.14.

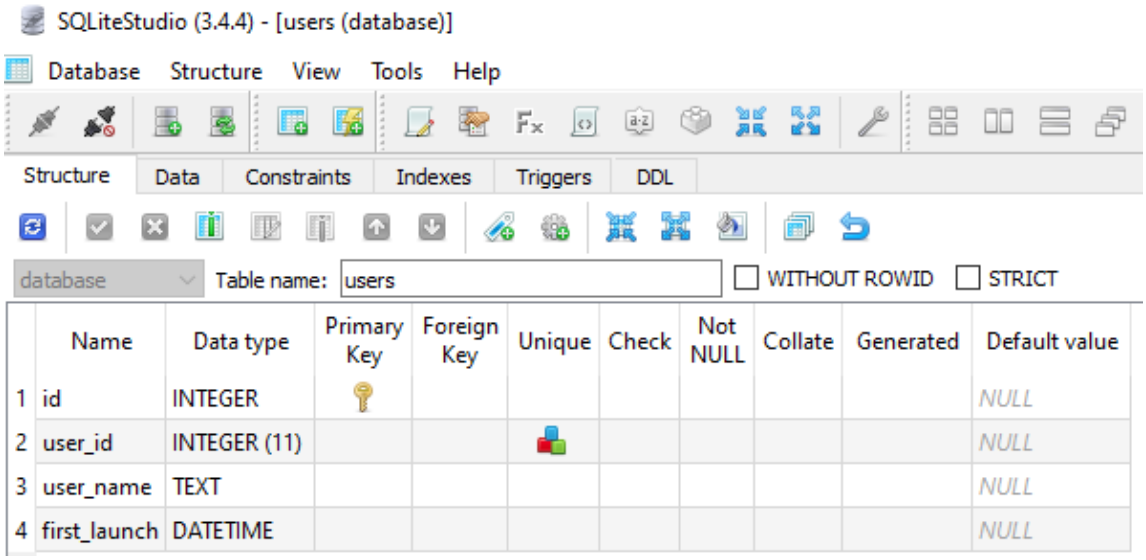


Рис. 3.14 – Таблиця *users*

Тепер створимо таблицю «*tasks*». Процес створення продемонстровано на рис. 3.15.

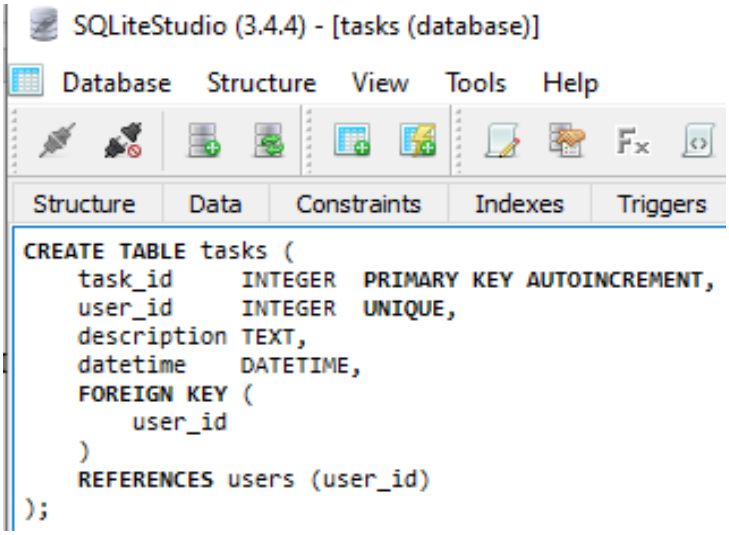


Рис. 3.15 – Створення таблиці *tasks*

Результат можна побачити на рис. 3.16.

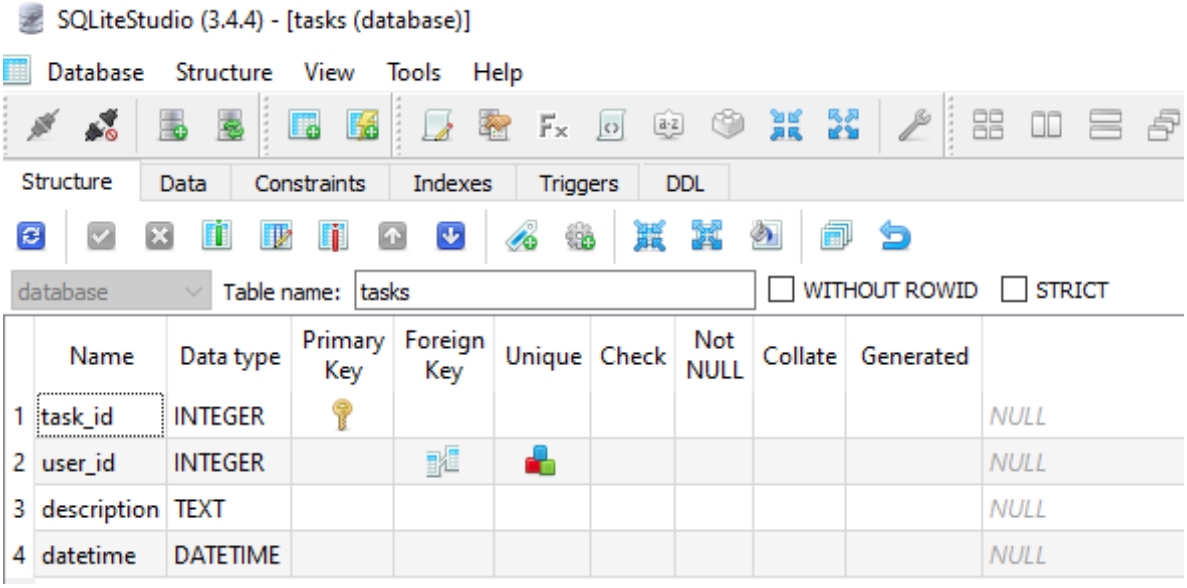


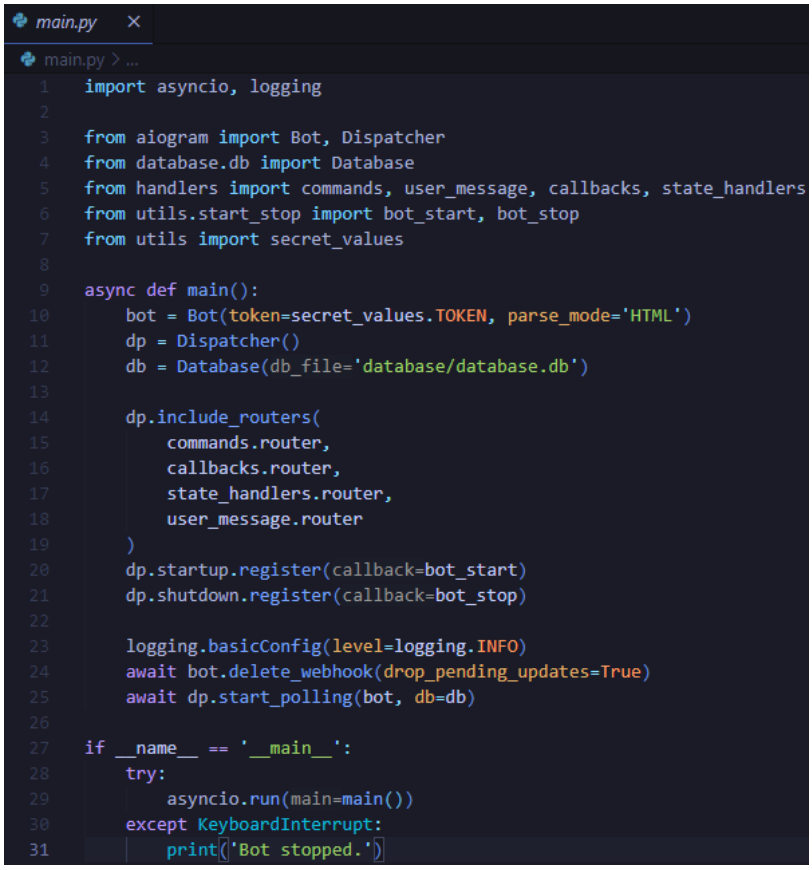
Рис. 3.16 – Таблиця *tasks*

3.5 Розробка бота

3.5.1 Розробка файлу main.py

У цьому підрозділі буде розглянуто процес розробки головного файлу бота *main.py*, який відповідає за ініціалізацію бота, його основні параметри та запуск.

Код файлу *main.py* продемонстровано на рис. 3.17.



```python
import asyncio, logging

from aiogram import Bot, Dispatcher
from database.db import Database
from handlers import commands, user_message, callbacks, state_handlers
from utils.start_stop import bot_start, bot_stop
from utils import secret_values

async def main():
    bot = Bot(token=secret_values.TOKEN, parse_mode='HTML')
    dp = Dispatcher()
    db = Database(db_file='database/database.db')

    dp.include_routers(
        commands.router,
        callbacks.router,
        state_handlers.router,
        user_message.router
    )
    dp.startup.register(callback=bot_start)
    dp.shutdown.register(callback=bot_stop)

    logging.basicConfig(level=logging.INFO)
    await bot.delete_webhook(drop_pending_updates=True)
    await dp.start_polling(bot, db=db)

if __name__ == '__main__':
    try:
        asyncio.run(main=main())
    except KeyboardInterrupt:
        print('Bot stopped.')
```

Рис. 3.17 – Файл *main.py*

Асинхронна функція *main()*:

1. Створюється об'єкт *Bot* з використанням токена *secret_values.TOKEN* і встановлюється режим парсингу *HTML*.

2. Створюється об'єкт *Dispatcher*, який керує обробниками.

3. Створюється об'єкт *Database* для роботи з базою даних.

4. Включаються роутери (обробники маршрутів) із різних модулів: *commands*, *callbacks*, *state_handlers*, *user_message*.

5. Реєстрація функцій для надсилання повідомлень адміністратуру, коли бот вмикається та вимикається (*bot_start* та *bot_stop*).

6. Встановлюється базовий рівень логування. Це дозволяє отримувати у консоль повідомлення, про всі дії бота. Це дуже зручно і знадобиться при розробці і тестуванні. Приклад логування наведено на рис. 3.18.

```
INFO:aiogram.dispatcher:Start polling
INFO:aiogram.dispatcher:Run polling for bot @GeeksFriendBot id=6930630618 - "Geek's Friend"
INFO:aiogram.event:Update id=746608690 is handled. Duration 297 ms by bot id=6930630618
INFO:aiogram.event:Update id=746608691 is handled. Duration 452 ms by bot id=6930630618
INFO:aiogram.event:Update id=746608692 is handled. Duration 312 ms by bot id=6930630618
INFO:aiogram.event:Update id=746608693 is handled. Duration 203 ms by bot id=6930630618
```

Рис. 3.18 – Приклад логування

7. Видаляється вебхук, якщо його було встановлено раніше, і скидаються очікувані оновлення.

8. Запускається процес опитування (*polling*), який дозволяє боту отримувати оновлення від сервера телеграм.

Запуск фукції *main* (рис. 3.19).

```
27  if __name__ == '__main__':
28      try:
29          asyncio.run(main=main())
30      except KeyboardInterrupt:
31          print('Bot stopped.')
```

Рис. 3.19 – Запуск функції *main*

1. Перевіряється, що скрипт запускається безпосередньо, а не імпортується як модуль.

2. У блоці *try* виконується асинхронна функція *main*().

3. У разі переривання виконання скрипта (наприклад, при натисканні *Ctrl+C*), виводиться повідомлення "*Bot stopped*."

3.5.2 Розробка інтерфейсу бота

У цьому підрозділі буде розглянуто реалізацію інтерфейсу бота, створення меню, кнопок та обробку команд користувача.

Основним засобом взаємодії я обрав *inline* клавіатури, прікріплені до повідомлень. Їх буде чотири і знаходитися вони будуть у директорії *keyboards*, у файлі *inline.py*.

Основна клавіатура *main_kb*. Ця клавіатура буде використовуватися як основне меню бота. На ній мають бути розміщені кнопки основного

функціоналу бота. Код клавіатури продемонстровано на рис. 3.20.

```python
from aiogram.types import InlineKeyboardButton, InlineKeyboardMarkup

main_kb = InlineKeyboardMarkup(
    inline_keyboard=[
        [
            InlineKeyboardButton(text='🚀 TikTok Assistant', callback_data='tiktok'),
            InlineKeyboardButton(text='💬 ChatGPT', callback_data='chatgpt')
        ],
        [
            InlineKeyboardButton(text='📝 To-Do List', callback_data='todo'),
            InlineKeyboardButton(text='⛩ Anime Assistant', callback_data='anime')
        ],
        [
            InlineKeyboardButton(text='📪 Send Feedback', callback_data='feedback'),
        ],
        [
            InlineKeyboardButton(text='☕ Support the Bot', callback_data='buy_coffee')
        ]
    ]
)
```

Рис. 3.20 – Код клавіатури *main_kb*

У результаті отримали клавіатуру, яку можна додавати у кінці повідомлень бота (рис. 3.21):

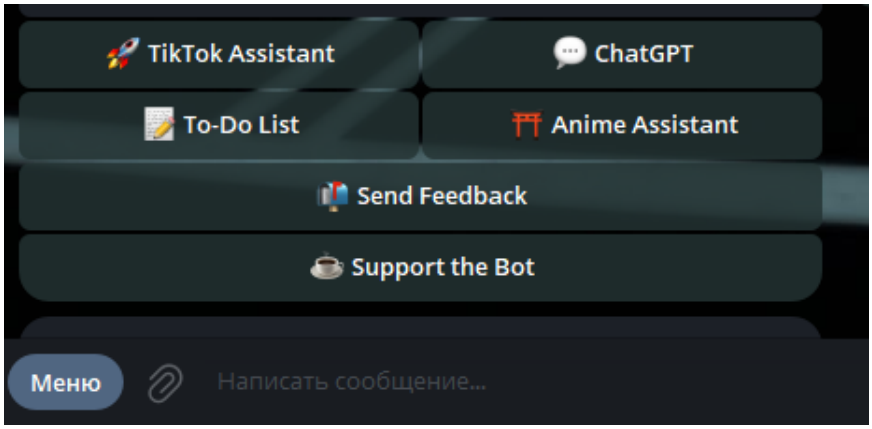


Рис. 3.21 – Клавіатура *main_kb*

Клавіатура *tiktok_assistant_kb* – ця клавіатура буде використовуватися як меню функціоналу *TikTok Assistant*.

Код для її створення продемонстровано на рис. 3.22.

```python
tiktok_assistant_kb = InlineKeyboardMarkup(
    inline_keyboard=[
        [
            InlineKeyboardButton(text='Video Downloader 📹', callback_data='download_video'),
            InlineKeyboardButton(text='Tags Generator 🏷', callback_data='generate_tags')
        ]
    ]
)
```

Рис. 3.22 – Код клавіатури *tiktok_assistant_kb*

Результат можна побачити на рис. 3.23:

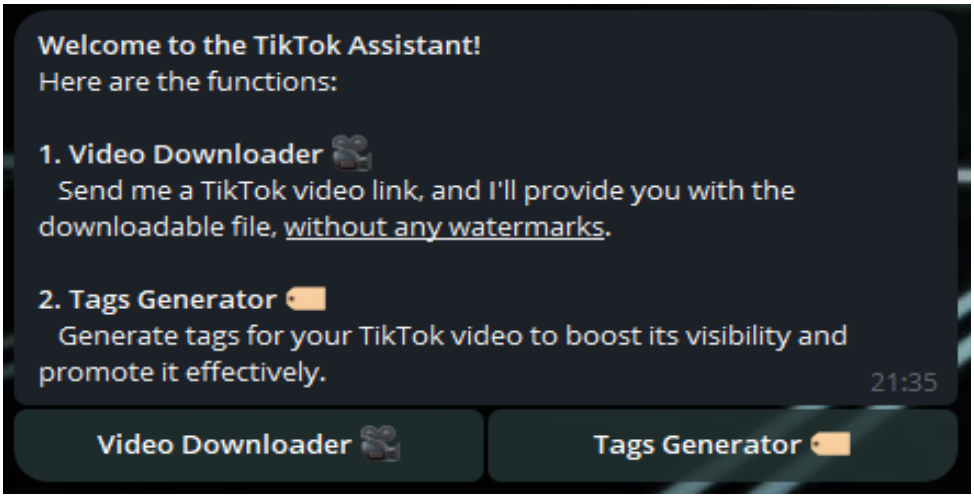


Рис. 3.23 – Клавіатура *tiktok_assistant_kb*

Клавіатура *todo_kb* – ця клавіатура буде використовуватися як меню функціоналу *To-Do list.*

Код для її створення продемонстровано на рис. 3.24.

```python
todo_kb = InlineKeyboardMarkup(
    inline_keyboard=[
        [
            InlineKeyboardButton(text='Add task ➕', callback_data='add_task'),
            InlineKeyboardButton(text='My tasks 📋', callback_data='my_tasks')
        ]
    ]
)
```

Рис. 3.24 – Код клавіатури *todo_kb*

Результат можна побачити на рис. 3.25.

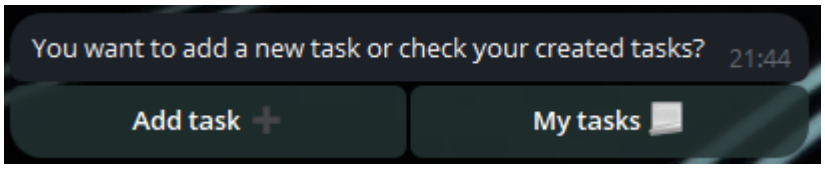


Рис. 3.25 – Код клавіатури *todo_kb*

Клавіатура *anime_assistant_kb* – ця клавіатура буде використовуватися як меню функціоналу *Anime Assistant*. Код для її створення продемонстровано на рис. 3.26.

```
anime_assistant_kb = InlineKeyboardMarkup(
    inline_keyboard=[
        [
            InlineKeyboardButton(text='News ⛩', callback_data='anime_news'),
            InlineKeyboardButton(text='Name finder 🔎', callback_data='sauce')
        ]
    ]
)
```

Рис. 3.26 – Код клавіатури *anime_assistant_kb*

Результат можна побачити на рис. 3.27:

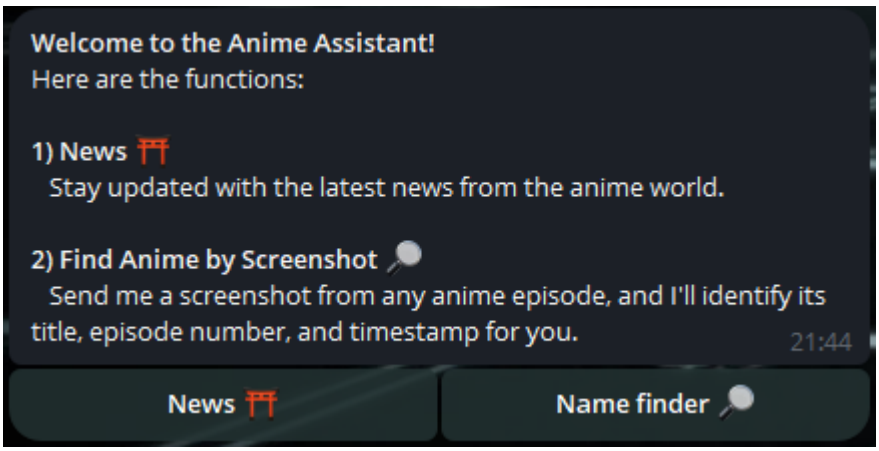


Рис. 3.27 – Клавіатура *anime_assistant_kb*

Щоб ці клавіатури працювали, потрібно реалізувати логіку колбеків (*callbacks*). Саме для цього при створенні клавіатур ми задавали значення параметрам «*callback_data*» кожної кнопки. Напишемо відповідний код у файлі *callback.py*, в директорії *handlers*. Він продемонстрований на рис. 3.28.

Повний код наведено у Додатку А.

```python
from keyboards import inline
from aiogram import Router, F, Bot, Dispatcher
from aiogram.types import CallbackQuery
from aiogram.fsm.context import FSMContext
from utils import text, secret_values, parser
from utils.states import PickState
from database.db import Database as db

router = Router()
dp = Dispatcher()
bot = Bot(token=secret_values.TOKEN, parse_mode='HTML')

@router.callback_query(F.data=='tiktok')
async def cb_tiktok(cb: CallbackQuery, state: FSMContext):
    await cb.answer()
    await state.set_state(state=PickState.tt_assistant)
    await cb.message.answer(text=text.tt_assistant_menu, reply_markup=inline.tiktok_assistant_kb)

@router.callback_query(F.data=='download_video')
async def cb_download_video(cb: CallbackQuery, state: FSMContext):...

@router.callback_query(F.data=='generate_tags')
async def cb_generate_tags(cb: CallbackQuery, state: FSMContext):...
```

Рис. 3.28 – Файл *callbacks.py*

Розглянемо код на рисунку 3.28:

1. Декоратори (*@router.callback_query()*) реєструють функції як обробники подій *callback*-запитів. Це означає, що коли користувач натискає кнопку, яка відправляє *callback*-запит з даними, наприклад, «*tiktok*», відповідна функція буде викликана для обробки запиту.

2. У кожній функції відбуваються наступні події:

   a) *cb.answer()* – відповідь на коллбек. Це необхідно робити, бо якщо накопиться велика кількість коллбеків зареєстрованих без відповідей, *BotFather* повідомить про це і примусово відключить бота. Так як немає необхідності відповідати якимось повідомленням, параметр можна не передавати;

   b) *state.set_state()* – функціонал машини станів. Змінює стан бота на відповідний функції. Це робиться для коректнох взаємодії бота з користувачем;

   c) *cb.message.answer(text.tt_assistant_menu,*

reply_markup=inline.tiktok_assistant_kb) – бот надсилає повідомлення із заготовленим текстом та *inline* клавіатурою.

Тепер реалізуємо обробку команд, доступних користувачу, у файлі *commands.py*, в директорії *handlers*. Код продемонстровано на рис. 3.29. Повний код наведено у Додатку А.

commands.py

handlers > commands.py > cmd_animenews

```python
import asyncio

from aiogram import Router
from aiogram.filters.command import Command, CommandStart
from aiogram.types import Message
from aiogram.fsm.context import FSMContext
from datetime import datetime
from keyboards import inline
from utils import text, secret_values, parser
from utils.states import PickState
from database.db import Database as db

router = Router()

@router.message(CommandStart())
async def cmd_start(msg: Message, state: FSMContext, db: db):
    if not db.user_exists(user_id=msg.from_user.id):
        formated_datetime = datetime.now().strftime(format/'%Y-%m-%d %H:%M')
        db.add_user(user_id= msg.from_user.id, user_name= msg.from_user.username, first_launch= formated_datetime)
    await state.set_state(state=PickState.info_viewing)
    await msg.answer(text=text.greeting.format(name=msg.from_user.full_name.title()), reply_markup=inline.main_kb)

@router.message(Command('info'))
async def cmd_info(msg: Message, state: FSMContext):
    await state.set_state(state=PickState.info_viewing)
    await msg.answer(text=text.info)

@router.message(Command('menu'))
async def cmd_menu(msg: Message, state: FSMContext):···

@router.message(Command('commands'))
async def cmd_commands(msg: Message, state: FSMContext):···
```

Рис. 3.29 – Файл *commands.py*

Розглянемо код на рисунку 3.29, а саме реакцію бота на команду «/*start*»:

1. У даному випадку декоратори виконують таку ж саму роль, що і під час реалізації коллбеків.

2. При отриманні команди «/*start*», бот перевіряє, чи є користувач у базі даних. Якщо ні – вносить його до бази даних у таблицю *users*.

3. Стан бота змінюється. Це робиться для коректної взаємодії бота з користувачем.

4. Бот надсилає користувачу привітальне повідомлення із заготовленим текстом та *inline* клавіатурою.

3.5.3 Розробка основного функціоналу бота

У цьому підрозділі буде розглянуто процес розробки основного функціоналу бота та як він взаємодіє з API.

Завантаження відео з соц. мережі *TikTok* без водяного знака. Цей функціонал буде реалізований у файлі *state_handlers.py*, в директорії *handlers*. Код продемонстровано на рис. 3.30.

```
@router.message(PickState.tt_downloading)
async def download_video(msg: Message):
    if ('http' not in msg.text
        or 'tiktok' not in msg.text
        or 'video' not in msg.text):

        await msg.answer(text=text.tt_wrong_format)
    else:
        link = msg.text
        proc = await msg.answer(text='⏳ Processing...')
        url = "https://tiktok-video-no-watermark2.p.rapidapi.com/"
        querystring = {"url":link, "hd":"1"}
        headers = {
            "X-RapidAPI-Key": secret_values.RAPIDAPI_KEY,
            "X-RapidAPI-Host": "tiktok-video-no-watermark2.p.rapidapi.com"
        }
        response = requests.get(url=url, headers=headers, params=querystring)
        video_link = response.json()['data']['play']

        await proc.edit_text(text=text.tt_sending_video)
        await msg.answer_video(video=URLInputFile(url=video_link))
        await proc.delete()
```

Рис. 3.30 – Код функціоналу *Video Downloader*

Розглянемо код на рисунку 3.30:

1. Декоратор «*@router.message(PickState.tt_downloading)*» реєструє функцію «*download_video*» як обробник повідомлень, коли бот перебуває у стані «*PickState.tt_downloading*». Це означає, що ця функція буде викликатись при отриманні повідомлення від користувача, якщо бот знаходиться в цьому стані.

2. Перевірка коректності повідомлення: бот перевіряє, чи містить повідомлення текст, який передбачає наявність посилання на відео з *TikTok*. Якщо в тексті повідомлення немає слів «*http*», «*tiktok*» або «*video*», то бот відправляє повідомлення у відповідь з текстом «*text.tt_wrong_format*» (повідомлення, заготовлене на випадок отримання неправильного формату

даних від користувача), вказуючи, що формат повідомлення неправильний.

3. Обробка коректного повідомлення: якщо повідомлення містить коректне посилання, текст повідомлення зберігається в змінну «*link*», і бот відправляє повідомлення у відповідь, яке вказує користувачеві, що запит обробляється. Повідомлення у відповідь зберігається в змінну proc для подальшого редагування.

4. Надсилання запиту до *API* – цей блок коду формує запит до *API* сервісу «*tiktok-video-no-watermark2.p.rapidapi.com*» для завантаження відео з *TikTok* без водяних знаків. Запит надсилається методом «*requests.get*». Відповідь *API* розбирається для отримання посилання відео. На запит передаються:

a) *url* – адреса *API*;

b) *querystring* — параметри запиту, що включають посилання на відео (*link*) та параметр якості відео (*hd*);

c) *headers* — заголовки запиту, що включають ключ *API* (*RAPIDAPI_KEY*) і хост.

5. Надсилання відео користувачу

a) редагується текст повідомлення «*proc*» на «*text.tt_sending_video*», щоб повідомити користувача, що відео надсилається;

b) бот відправляє відео користувачу, використовуючи посилання на «*video_link*»;

c) видаляється повідомлення «*proc*», щоб очистити чат.

6. Спілкування з *ChatGPT*

Цей функціонал буде реалізований у файлі *chatgpt.py*, в директорії *utils*. Код продемонстровано на рис. 3.31.

Розглянемо код на рисунку 3.31. Функція *generate_answer()* – ця асинхронна функція приймає повідомлення користувача (*user_message)* і надсилає його до *API OpenAI* для створення відповіді. Запит надсилається методом *client.chat.completions.create()*, який створює завершення чату.

```python
from openai import AsyncOpenAI
from utils import secret_values
from utils import text

client = AsyncOpenAI(api_key=secret_values.OPENAI_API_KEY)

async def generate_answer(user_message):
    chat_completion = await client.chat.completions.create(
        messages=[
            {
                "role": "user",
                "content": f"{user_message}",
            }
        ],
        model="gpt-3.5-turbo",
    )
    return chat_completion.choices[0].message.content

async def generate_tags(vid_theme):
    chat_completion = await client.chat.completions.create(
        messages=[
            {
                "role": "user",
                "content": f"{text.tt_generate_tags_prompt.format(vid_theme=vid_theme)}",
            }
        ],
        model="gpt-3.5-turbo",
    )
    return chat_completion.choices[0].message.content
```

Рис. 3.31 – Код функціоналу *ChatGPT*

У параметр «*messages*» передається список повідомлень, де кожне повідомлення є словником з роллю (тут «*user*») і вмістом (саме повідомлення користувача).

Вказується модель «*gpt-3.5-turbo*» для створення відповіді. Функція повертає вміст першого повідомлення у відповідь моделі (*chat_completion.choices[0].message.content*).

Функція *generate_tags()* – ця асинхронна функція приймає тему відео «*vid_theme*» і надсилає запит до *API OpenAI* для створення тегів. У параметрі messages передається повідомлення з роллю «*user*» та вмістом, сформованим з використанням шаблону «*text.tt_generate_tags_prompt*». Шаблон форматується за допомогою змінної «*vid_theme*». Також використовується модель «*gpt-3.5-turbo*» для генерації тегів. Функція повертає вміст першого повідомлення у відповідь моделі (*chat_completion.choices[0].message.content*).

Anime Assistant – пошук аніме-серіалу за скриншотом. Код продемонстровано на рис. 3.32.

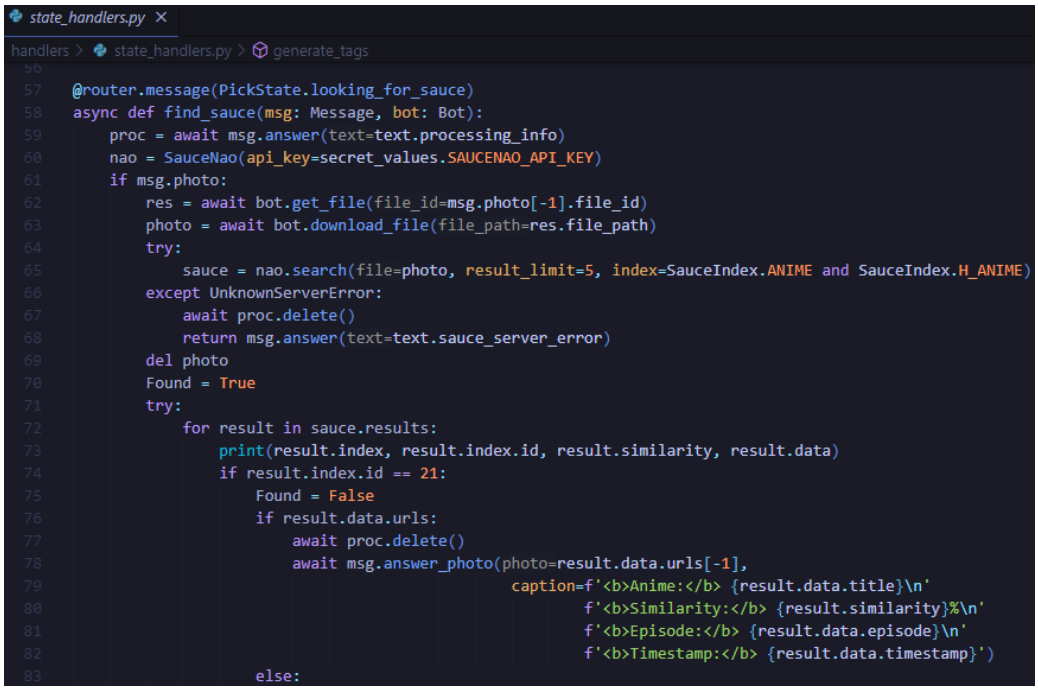


Рис. 3.32 – Фрагмент коду функціоналу *Name finder*

Розглянемо код на рисунку 3.32:

1. Обробка повідомлення – у рядку 59, бот надсилає повідомлення користувачу, що бот почав обробку запиту. У рядку 60, бот створює об'єкт SauceNao за допомогою API-ключа.

2. Перевірка наявності фотографії – якщо повідомлення містить фотографію, бот отримує інформацію про файл фотографії та завантажує її.

3. Пошук джерела зображення – у рядку 65, бот шукає джерело зображення за допомогою *SauceNao API*, обмежуючи кількість результатів до п’яти. Якщо відбувається помилка *UnknownServerError*, бот видаляє повідомлення про процес і надсилає повідомлення про помилку користувачеві.

4. Обробка результатів пошуку – бот проходить через результати пошуку та аналізує кожен результат. Якщо ідентифікатор результату (*result.index.id*) дорівнює 21 або 22, це означає, що знайдено збіг. Якщо є *URL*, надсилає фотографію та опис. Інакше надсилає текстове повідомлення з інформацією про збіг. Якщо збіг не знайдено, надсилає повідомлення, що джерело не знайдено.

5. Обробка випадків без фотографії – якщо повідомлення не містить фотографії, бот надсилає повідомлення про неправильний формат і видаляє повідомлення про процес.

6. Отримання останніх аніме-новин – для реалізації цього функціоналу, використовуємо бібліотеку «*asyncpraw*» для асинхронної взаємодії з *Reddit API*. Вона призначена для отримання останніх новин із субредиту «*animenews*». Код продемонстровано на рис. 3.33.

parser.py

utils > parser.py > ...

```python
import asyncpraw

from utils import text, secret_values

async def get_news():
    reddit = asyncpraw.Reddit(
        client_id=secret_values.CLIENT_ID,
        client_secret=secret_values.CLIENT_SECRET,
        user_agent=secret_values.USER_AGENT
    )
    subreddit = await reddit.subreddit('animenews')
    result = text.anime_news
    counter = 1

    async for submission in subreddit.new(limit=10):
        result += f'\n<b>{counter}) {submission.title}</b>'
        result += f'\n<a href="{submission.url}">Read more</a>\n'
        counter += 1

    return result
```

Рис. 3.33 – Код функціоналу *Anime News*

Розглянемо код на рисунку 3.33:

1. Асинхронна функція get_news – ця функція створює асинхронний об'єкт «*reddit*» для взаємодії з *Reddit API*, використовуючи ідентифікатор клієнта, секретний ключ і власний агент з файлу «*secret_values*».

2. Отримання субредиту animenews:

   a) *subreddit = await reddit.subreddit('animenews')* – отримує об'єкт субредиту animenews;

   b) *result = text.anime_news* – ініціалізує змінну result з базовим текстом новин, взятим із файлу «*text.anime_news*»;

   c) *counter=1* – ініціалізує лічильник для нумерації новин.

3. Цикл для отримання останніх 10 новин:

a) *async for submission in subreddit.new(limit=10).* Асинхронний цикл для отримання останніх 10 публікацій із субредиту animenews.

b) *result += f'\n{counter}) {submission.title}'.* Додає заголовок кожної публікації до результату з номером.

c) *result += f'\nRead more\n'.* Додає посилання на повну публікацію.

d) *counter+=1.* Збільшує лічильник для наступної публікації.

4. Повернення результату.

To-Do list – цей функціонал дозволяє користувачу створювати та керувати своїми завданнями. На рисунку 3.34 продемонстровано реалізацію функціонала перегляду існуючих завдань.

```
87  @router.message(Command('mytasks'))
88  async def cmd_mytasks(msg: Message, state: FSMContext, db: db):
89      await state.set_state(state=PickState.browsing_tasks)
90      db.delete_old_tasks()
91      await msg.answer(text=db.get_formated_tasks(user_id=msg.from_user.id))
```

Рис. 3.34 – Функціонал перегляду створених завдань

Перш за все, стан користувача змінюється і командою *db.delete_old_tasks()* видаляються прострочені завдання. Після цього бот отримає форматовані дані із бази даних та надсилає їх користувачу у вигляді списку.

На рисунку 3.35 продемонстровано реалізацію функціонала створення нового завдання.

Обробка повідомлення та поділ даних задачі:

task_info = msg.text.replace(«/newtask», "").strip()

Видаляє команду /newtask з тексту повідомлення та обрізає зайві прогалини.

task_data = task_info.split('#')

Розділяє текст, що залишився, на частини, використовуючи символ # як

роздільник. Очікується, що перша частина буде описом завдання, а друга – часом виконання.

```python
@router.message(Command('newtask'))
async def cmd_newtask(msg: Message, state: FSMContext, db: db):
    await state.set_state(state=PickState.adding_new_task)
    db.delete_old_tasks()
    try:
        task_info = msg.text.replace(old/"/newtask", new/"").strip()
        task_data = task_info.split(sep='#')

        if len(obj/task_data) >= 2:
            task_description = task_data[0].strip()
            task_datetime_str = task_data[1].strip()
            task_datetime = datetime.strptime(date_string/task_datetime_str, format/"%Y-%m-%d %H:%M")
            user_id = msg.from_user.id

            db.add_task(user_id=user_id, task_description=task_description, task_datetime=task_datetime)

            await msg.reply(text=text.adding_task_success.format(task=task_description, time= task_datetime))
            time_difference = task_datetime - datetime.now()

            await asyncio.sleep(delay=time_difference.total_seconds())
            await msg.answer(text=text.task_remind.format(task= task_description, time= task_datetime))

        else:
            await msg.reply(text=text.adding_task_error)

    except Exception:
        await msg.reply(text=text.adding_task_error)
```

Рис. 3.35 – Функціонал створення нового завдання

Перевірка та обробка даних завдання:

db.add_task(user_id, task_description, task_datetime)

Додає нове завдання до бази даних.

time_difference = task_datetime – datetime.now()

Обчислює різницю між поточним часом та часом виконання завдання.

await asyncio.sleep(time_difference.total_seconds())

Очікує до моменту виконання завдання.

Обробка помилок та неправильного формату даних – якщо даних у *task_data* недостатньо, надсилається повідомлення про помилку. Якщо відбувається будь-яка помилка в процесі обробки, також надсилається повідомлення про помилку.

Висновок до третього розділу

Було описано процес практичної реалізації телеграм-бота *Geek's Friend*. Було розглянуто вибір та обґрунтування використання мови програмування *Python*, СУБД *SQLite*, системи контролю версій *Git* та середовища розробки

Visual Studio Code. Описано процес створення бота за допомогою *BotFather*, а також розроблено структуру проекту, що включає основні файли та каталоги.

Описано процес розробки бази даних, включаючи створення таблиць для зберігання інформації про користувачів та їх завдання. Було розглянуто основний функціонал бота, такий як завантаження відео з соц. мережі *TikTok,* генерація хештегів, пошук аніме за скриншотами, керування списками завдань, отримання новин з *Reddit* та взаємодія з *ChatGPT*. У результаті було створено багатофункціонального телеграм-бота, який відповідає сучасним вимогам та запитам користувачів.

РОЗДІЛ 4
ТЕХНІКО-ЕКОНОМІЧНА ЧАСТИНА

4.1. Організаційно-економічне обґрунтування проекту

У даній роботі проведено аналіз технологій розробки багатофункціонального телеграм-бота та розроблено програмне забезпечення, яке дозволяє виконувати різні задачі та отримувати необхідну інформацію без необхідності перемикатися між різними програмами чи ботами. Головною метою проекту було створення зручного інструменту для користувачів, що надає такі можливості:

1. Завантаження відео з *TikTok* без водяного знака.
2. Генерація хештегів для публікацій *TikTok.*
3. Пошук аніме-серіалу за скриншотом.
4. Керування списками завдань (*ToDo list).*
5. Надсилання останніх аніме-новин.
6. Спілкування із *ChatGPT.*
7. Надсилання фідбеку розробнику бота.
8. Можливість фінансової підтримки.

Для реалізації проекту було використано сучасні технології та інструменти. Зокрема, для написання програмного коду було обрано мову програмування *Python*, яка забезпечує високу продуктивність та зручність у розробці. Бібліотека *Aiogram* була використана для взаємодії з телеграм *API,* що дозволяє ефективно обробляти запити користувачів. Для асинхронного програмування була використана бібліотека *Asyncio,* яка дозволяє обробляти численні запити одночасно без затримок. База даних *SQLite* забезпечує надійне зберігання даних про користувачів та їх завдання.

Перед початком розробки було проведено аналіз існуючих аналогів, визначено основні проблеми та завдання, які потрібно вирішити. Було досліджено моделі та методи проектування програмного забезпечення для

телеграм-ботів.

У результаті було створено багатофункціонального телеграм-бота, який може взаємодіяти з різними *API*, такими як *OpenAI* для генерації текстів, *SauceNAO* для пошуку аніме за зображеннями, та *Reddit* для отримання останніх новин.

Цей проект має потенціал для подальшого розвитку та вдосконалення, оскільки забезпечує користувачів широким спектром можливостей, які відповідають сучасним вимогам та запитам. Він забезпечує зручний та ефективний спосіб взаємодії з різними сервісами, що робить його корисним інструментом для широкого кола користувачів.

Наступні етапи розвитку проекту включають:

1. Розширення функціональності бота.
2. Оптимізація та покращення користувацького інтерфейсу.
3. Впровадження нових технологій та інтеграцій з іншими сервісами.
4. Маркетингова стратегія для залучення нових користувачів та підвищення впізнаваності продукту.

Підсумовуючи, розроблений телеграм-бот є важливим кроком у напрямку створення багатофункціонального інструменту, який може значно полегшити користувачам доступ до різних сервісів та інформації.

У таблиці 4.1 представлена порівняльна характеристика існуючих аналогів із розробленим під час виконання цієї роботи багатофункціональним ботом (*Geek's Friend*):

Таблиця 4.1

Порівняння функціоналу та характеристик існуючих аналогів

Функціонал чи характеристика	***TikTok Download Bot***	***Taskobot***	***WAIT: What Anime Is This***	***ChatGPT on Telegram-Bot***	***Geek's Friend***
Завантаження відео з соц. мережі *TikTok*	+	-	-	-	+

Продовження таблиці 4.1

Функціонал чи характеристика	***TikTok Download Bot***	***Taskobot***	***WAIT: What Anime Is This***	***ChatGPT on Telegram-Bot***	***Geek's Friend***
Генерація хештегів для просунення публікації у соц. мережі *TikTok*	-	-	-	+	+
Керування списками справ	-	+	-	-	+
Пошук назви аніме за скриншотом	-	-	+	-	+
Надання останніх аніме-новин	-	-	-	-	+
Спілкування с моделлю штучного інтелекту *ChatGPT*	-	-	-	+	+
Відкритий вихідний код	-	-	+	+	+
Стабільна робота бота	+	+	+	+	+
Можливість зворотного зв'язку	+	-	-	+	+
Наявність опису можливостей бота з інструкціями щодо використання	+	+	+	+	+

4.1.1 Організаційне обґрунтування

Проект з розробки багатофункціонального телеграм-бота вимагає ретельного планування та організації всіх етапів робіт для досягнення поставлених цілей. Організаційне обґрунтування включає в себе оцінку та класифікацію проекту, що дозволяє забезпечити ефективне управління та реалізацію.

Класифікаційна оцінка проекту:

1. Клас проекту – монопроект, це проект з розробки багатофункціонального телеграм-бота є монопроектом, оскільки він орієнтований на створення одного конкретного продукту.

2. Тип проекту – змішаний, він включає технічні, організаційні та комерційні аспекти. Технічний аспект полягає у розробці програмного забезпечення, організаційний – у плануванні та управлінні проектом, комерційний – у потенційній монетизації та просуванні продукту.

3. Вид проекту – інноваційний, проект впроваджує нові рішення та технології, такі як інтеграція з різними *API*, асинхронне програмування та автоматизація завдань.

4. Тривалість проекту – короткостроковий.

5. Складність – середня. Проект має середній рівень складності, оскільки вимагає інтеграції з кількома *API*, розробки асинхронної архітектури та забезпечення безпеки даних.

6. Розмір – невеликий, оскільки він орієнтований на конкретну нішу користувачів телеграм.

7. Рівень проекту – корпоративний, проект спрямований на створення продукту, який може бути використаний широким колом користувачів.

Класифікаційна оцінка проекту подана в таблиці 4.2.

Таблиця 4.2

Класифікаційна оцінка проекту

Критерій	**Оцінка**
Клас проекту	Монопроект
Тип проекту	Змішаний
Вид проекту	Інноваційний
Тривалість проекту	Короткостроковий
Складність і розміри	Середня складність, невеликий розмір
Розмір	Невеликий
Рівень проекту	Корпоративний

Етапи виконання розділів кваліфікаційної роботи. Розробка багатофункціонального телеграм-бота включає наступні етапи виконання, кожен з яких має визначені терміни:

1. Підготовчий етап: визначення цілей і завдань проекту, аналіз існуючих рішень і визначення основних вимог до системи. Тривалість: 1 тиждень.

2. Проектування: розробка технічного завдання, визначення архітектури системи та вибір технологічного стека. Тривалість: 1 тиждень.

3. Розробка: написання коду, інтеграція з *API*, тестування та відладка. Тривалість: 1 місяць.

4. Впровадження: розгортання бота на сервері, налаштування та забезпечення безперебійної роботи. Тривалість: 3 дні.

5. Підтримка та розвиток: регулярні оновлення та вдосконалення функціональності, робота з фідбеком користувачів та реалізація нових можливостей. Тривалість: постійно.

4.1.2 Склад робіт по життєвому циклу проекту

Для успішної реалізації проекту з розробки багатофункціонального телеграм-бота необхідно визначити основні етапи його життєвого циклу та розподілити роботи за часом. Це дозволить забезпечити ефективне управління проектом та своєчасне виконання всіх завдань. Життєвий цикл проекту представлено в таблиці 4.3.

Таблиця 4.3

Життєвий цикл проекту

№	Код роботи	Назва роботи	Тривалість (дні)
1	1-2	Аналіз предметної області, дослідження існуючих рішень та визначення основних вимог до системи	7
2	2-3	Проектування архітектури бота та вибір технологічного стеку	5
3	2-4	Розробка функціоналу бота, інтеграція з API	30
4	3-5	Тестування та налагодження роботи бота	14
5	4-5	Розгортання бота на сервері та налаштування	3
6	5-6	Створення документації та інструкцій для користувачів	7

Загальна тривалість проекту: 66 днів

На основі визначених етапів та їх тривалості побудовано мережевий графік (рис. 4.1), який відображає взаємозв'язки між роботами та критичний шлях проекту. Критичний шлях визначає мінімальний час, необхідний для завершення проекту, і включає роботи, затримка яких призведе до затримки всього проекту.

Рисунок 4.1 – Мережевий графік проекту

4.2 Маркетингове обґрунтування проекту

Маркетингове обґрунтування проекту з розробки багатофункціонального телеграм-бота має на меті дослідити ринкові можливості, цільову аудиторію та конкурентне середовище, а також розробити стратегії просування продукту для досягнення комерційного успіху.

Дослідження ринку – телеграм, як одна з найпопулярніших платформ для обміну повідомленнями, пропонує величезний потенціал для розвитку ботів. Постійне зростання кількості користувачів телеграм створює сприятливі умови для просування та монетизації ботів.

Цільовою аудиторією багатофункціонального телеграм-бота є широке коло користувачів, які шукають зручні та ефективні інструменти для вирішення різноманітних завдань. Це можуть бути як індивідуальні користувачі, так і представники бізнесу, освітніх установ та інших організацій.

Конкурентне середовище – на ринку телеграм-ботів існує безліч рішень, проте більшість з них є вузькоспеціалізованими і не пропонують такого широкого спектру функцій, як розроблюваний бот. Це створює конкурентну перевагу для *Geek's Friend*, який може задовольнити потреби користувачів у різних сферах.

Стратегії просування – для успішного просування бота *Geek's Friend* необхідно використовувати комплексний підхід, який включає в себе:

1. Контент-маркетинг: створення та публікація корисного контенту (статті, відео, інструкції) на тему використання бота та його функцій; розміщення контенту на тематичних ресурсах, форумах та соціальних мережах.

2. Партнерські програми: співпраця з іншими телеграм-каналами та групами для взаємного просування; розробка партнерської програми для залучення нових користувачів.

3. Реклама: таргетована реклама в телеграм та інших соціальних мережах; контекстна реклама в пошукових системах.

4. *PR* та заходи: участь у тематичних конференціях та виставках; організація конкурсів та акцій для залучення уваги до бота.

5. Очікувані результати – завдяки комплексному підходу до маркетингу та просування очікується: збільшення кількості користувачів бота; підвищення впізнаваності бренду *Geek's Friend*; розширення ринкової частки; збільшення прибутку від монетизації бота.

Аналізуючи ринок в Україні, Європейському Союзі (ЄС) та Сполучених Штатах Америки (США), можна затвердити, що вони мають суттєві відмінності, зокрема й у ціноутворенні.

Ці відмінності зумовлені низкою факторів, серед яких:

1. Економічний розвиток та рівень доходів. Середній рівень доходів населення в США та країнах ЄС вищий, ніж в Україні. Це дозволяє розробникам встановлювати вищі ціни в цих регіонах, розраховуючи на більшу платоспроможність замовників.

2. Податкове законодавство в Україні, ЄС та США має свої особливості. Різні ставки податків та зборів впливають на кінцеву вартість розробки проекту.

3. Конкурентне середовище – рівень конкуренції на ринку може відрізнятися в Україні, ЄС та США. Більша конкуренція може спонукати розробників знижувати ціни, щоб отримати більше замовників.

4. Регуляторні норми – в ЄС діють специфічні правила щодо захисту прав споживачів та цифрового контенту, що також може впливати на ціноутворення розробки.

Враховуючи ці фактори, ціни на розробку бота можуть суттєво відрізнятися в Україні, ЄС та США. Розуміння цих відмінностей є важливим як для споживачів, так і для учасників ринку.

Отже, маркетингове обґрунтування є невід'ємною частиною проекту з розробки багатофункціонального телеграм-бота *Geek's Friend*. Успішна реалізація запропонованих стратегій просування дозволить досягти поставлених цілей та забезпечити комерційний успіх продукту.

4.3 Економічні розрахунки проекту

4.3.1 Визначення трудомісткості розробки програмного продукту (ПП)

Тривалість створення програмного продукту залежить від його обсягу, складності розробки, кваліфікації команди розробників та термінів, встановлених ринковими умовами. Для визначення трудомісткості розробки програмного продукту використовувати дані про типові етапи та укрупнені норми часу, представлені в методичних вказівках. За структурною аналогією та інформацією з каталогу аналогів ПЗ (табл. 4.4) визначається обсяг програмного забезпечення в тисячах умовних машинних команд.

Таблиця 4.4

Каталог аналогів ПЗ

Найменування типу ПЗ	**Обсяг функції ПЗ-Vo, ум. машинних команд**
ПЗ СУБД	2500-9800
ПЗ система ведення лінійних файлів	860-6600
Комплексні системи ведення БД	950-7430
ПЗ уведення інформації	1060-5750
ПЗ автоматизації засобів по каталозі	680-7000
ПЗ автоматизованих розрахунків	1300-8600
ПЗ загальної математики й ПЗ імітаційного моделювання	7800-8800
ПЗ організації обчислювального процесу	1300-10200
ПЗ оптимизаційних розрахунків	1300-4200

Виходячи з функціоналу та складності розробленого телеграм-бота, можна провести аналогію з ПЗ системи ведення лінійних файлів (позиція 2 у таблиці 4.4). Припустимо, що обсяг такого ПЗ становить 1000 умовних машинних команд.

Для визначення трудомісткості розробки скористаємося даними з таблиці 4.5 (також узята з методичних вказівок), де наведено укрупнені норми часу на розробку ПЗ залежно від його обсягу.

Таблиця 4.5

Каталог аналогів ПЗ

Обсяг ПЗ, тис. умовних машинних команд	**Норма часу, чол/год**
1.00	229
2.00	244
3.00	262
4.00	283
5.00	306

Згідно з таблицею 4.5, трудомісткість розробки 1000 умовних машинних команд становить 229 люд/год. Враховуючи, що розробка телеграм-бота є відносно нескладним проектом, застосуємо поправочний коефіцієнт Кн = 0,8, який враховує умови розробки (рівень кваліфікації розробників, доступність інструментів тощо).

Таким чином, трудомісткість розробки телеграм-бота ($Т_р$) розраховується за формулою:

$$Т_р = К_н * Т_р \quad (4.1)$$

$Т_р = 0,8 * 229 = 183,2$ люд/год

Трудомісткість розробки ПП повинна включати розробку наступних етапів:

1. технічного завдання (ТЗ);
2. технічного проекту (ТП);
3. робочого проекту (РП);
4. впровадження (ВН).

Трудомісткість розроблювального ПП визначається для кожного етапу окремо на підставі трудомісткості аналога з урахуванням складності розробки, ступеня новизни й ступеня використання в розробці стандартних модулів на підставі формул:

$$T_{тз} = T_р * L_1 * К_н \quad (4.2)$$

$$T_{тп} = T_р * L_2 * К_н \quad (4.3)$$

$$T_{рп} = T_р * L_3 * К_н * К_т \quad (4.4)$$

$$T_{вн} = T_р * L_4 * К_н \quad (4.5)$$

де:

$T_р$ – укрупнена норма часу на розробку аналога ПЗ, люд/год (183,2), що коректується поправочним коефіцієнтом $К_н = 0,7$ (враховує умови розробки ПЗ на комп'ютері);

L_i – питома вага і-го етапу розробки;

$К_н$ – поправочний коефіцієнт, що враховує ступінь новизни;

$К_т$ – поправочний коефіцієнт, що враховує ступінь використання типових програм.

Значення питомих коефіцієнтів трудомісткості стадії в загальній трудомісткості розробки ПЗ представлено в таблиці 4.6.

Таблиця 4.6

Значення питомих коефіцієнтів трудомісткості стадії в загальній трудомісткості розробки ПЗ

Код стадії	**Ступінь новизни**		
	А	**Б**	**В**
ТЗ	0.15	0.12	0.12
ТП	0.16	0.15	0.11
РП	0.55	0.58	0.61
ВП	0.14	0.15	0.16

Ступінь проекта можна визначити як Б. Тому що його можна вважати розвитком існуючих рішень. Тобто $L_1 = 0.12$, $L_2 = 0.15$, $L_3 = 0.58$, $L_4 = 0.15$.

Розрахуємо трудомісткість розробки ПП для кожного етапу окремо згідно з формулами (4.2) – (4.5):

$T_{тз} = 183,2 * 0,7 * 0,12 * 0.5 = 7,69$ люд/год

$T_{тп}$ = 183,2 * 0,7 * 0,15 * 0.5 = 9,62 люд/год

$T_{рп}$ = 183,2 * 0,7 * 0,58 * 0.5 * 0,8 = 36,95 люд/год

$T_{вн}$ = 183,2 * 0,7 * 0,15 * 0.5 = 9,62 люд/год

Загальна трудомісткість розробки ПП складає:

$\sum T_i$ = 7,69 + 9,62 + 36,95 + 9,62 = 63,88 люд/год

Тривалість розробки ПП у роках визначається за формулою 4.6:

$$T_{пп} = \frac{\sum T_{ij}}{8*0.73} \quad (4.6)$$

$$T_{пп} = \frac{63.88}{8*0.73} = 11 \text{ (дн.)} = 0{,}0301\text{(р.)}$$

4.3.2 Визначення ціни ПП

У цьому підрозділі буде визначено ціну розробки бота. Для цього необхідно розрахувати основну заробітну плату розробників, матеріальні витрати, вартість машинного часу та інші витрати, пов'язані зі створенням продукту. Оскільки телеграм-бот є програмним продуктом, що підлягає багаторазовому тиражуванню та відчуженню від розробників, для розрахунку ціни використовується наступна формула:

$$Ц = К * С + П_р \quad (4.7)$$

де:

Ц – ціна розробки програмного продукту;

К – коефіцієнт обліку витрат на виготовлення дослідного зразка ПП (К = 1,1);

С – витрати на розробку програмної продукції (кошторисна собівартість);

$П_р$ – нормативний прибуток.

Нормативний прибуток розраховується за формулою:

$$П_р = (С – С_м) * Р_н / 100 \quad (4.8)$$

де:

$Р_н$ – норматив рентабельності (приймаємо 25%);

$С_м$ – матеріальні витрати.

Матеріальні витрати (См)

Для розробки телеграм-бота матеріальні витрати будуть мінімальними, оскільки основні ресурси – це час розробників та обчислювальні потужності. Враховуючи це, приймемо $С_м$ = 500 грн.

Спеціальне обладнання (Се)

Витрати, пов'язані з використанням обчислювальної техніки, визначаються за формулою:

$$С_{еом} = t^{еом} * К_и^{еом} * Ц^{еом} * К_{бд}^{еом} * К_е^{еом} \quad (4.9)$$

де:

$t^{еом}$ – час використання ЕОМ для розробки телеграм-бота (приймаємо 80 годин);

$К_и^{еом}$ – поправочний коефіцієнт обліку часу використання ЕОМ (1,08);

$Ц^{еом}$ – ціна однієї години роботи ЕОМ (приймаємо 50 грн/год);

$К_{бд}^{еом}$ – коефіцієнт обліку ступеня використання СУБД (1,1, оскільки використовується SQLite);

$К_е^{еом}$ – коефіцієнт обліку швидкодії ЕОМ (1,0, оскільки швидкодія ЕОМ більше 20*10^30 оп/с).

Підставивши значення у формулу, отримаємо:

$С_{еом}$ = 80 * 1,08 * 50 * 1,1 * 1,0 = 4752 грн.

Основна заробітна плата ($С_{зо}$)

Враховуючи, що над проектом працює один розробник із середньомісячним окладом 25 000 грн, а трудомісткість розробки становить 63,88 людино-годин (згідно з розрахунками у підрозділі 4.3.1), основна заробітна плата розраховується за наступною формулою:

$$С_{зо} = (З_і * К_о * \tau_і) / Д_р \quad (4.10)$$

де:

$З_і$ – середньомісячний оклад і-того виконавця, грн;

$К_о$ – коефіцієнт обліку окладу керівників і консультантів проекту (рекомендується 0,1);

$τ_і$ – трудомісткість робіт

$Д_р$ – середня кількість робочих днів у місяці (рекомендовано 21-22).

Підставивши значення у формулу, отримаємо:

$С_{зо} = (25000 * 0,1 * (63,88 / 8)) / 21 = 948,21$ грн.

Додаткова заробітна плата ($С_{зд}$)

Додаткова заробітна плата розраховується як 10% від основної заробітної плати:

$С_{зд} = С_{зо} * 0,1 = 948,21 * 0,1 = 94,82$ грн.

Відрахування на соціальне страхування ($С_{сс}$)

Відрахування на соціальне страхування складають 22% від суми основної та додаткової заробітної плати:

$С_{сс} = 0,22 * (С_{зо} + С_{зд}) = 0,22 * (948,21 + 94,82) = 231,65$ грн.

Накладні витрати ($С_н$)

Накладні витрати приймаємо у розмірі 50% від основної заробітної плати:

$С_н = 0,5 * С_{зо} = 0,5 * 948,21 = 474,11$ грн.

Кошторисна собівартість (С):

Загальна кошторисна собівартість розробки телеграм-бота розраховується як сума всіх витрат:

$$С = С_м + С_е + С_{зо} + С_{зд} + С_{сс} + С_н \quad (4.11)$$

$С = 500 + 4752 + 948,21 + 94,82 + 231,65 + 474,11 = 7000,79$ грн.

Нормативний прибуток ($П_р$):

$$П_р = (С – С_м) * Р_н / 100 \quad (4.12)$$

$П_р = (7000,79 – 500) * 25 / 100 = 1625,20$ грн.

Ціна розробки (Ц):

$$Ц = К * С + П_р \quad (4.13)$$

Ц = 1,1 * 7000,79 + 1625,20 = 9325,07 грн.

Отже, розрахункова ціна розробки багатофункціонального телеграм-бота *Geek's Friend* становить 9325,07 грн.

Висновки до четвертого розділу

Економічна частина дипломної роботи присвячена аналізу та обґрунтуванню економічної доцільності розробки багатофункціонального телеграм-бота *Geek's Friend*. Було проведено дослідження ринку, визначено цільову аудиторію та конкурентне середовище, а також розроблено маркетингову стратегію просування продукту.

Розрахунок трудомісткості розробки показав, що для створення бота потрібно 63,88 людино-годин, що еквівалентно 10,98 робочим дням або 0,044 років. Ціна розробки складає 9325,07 грн, враховуючи витрати на оплату праці та інші витрати. Таким чином, економічний аналіз підтверджує доцільність розробки та впровадження багатофункціонального телеграм-бота *Geek's Friend,* який має потенціал для успішної монетизації та задоволення потреб широкого кола користувачів.

РОЗДІЛ 5
ОХОРОНА ПРАЦІ

5.1 Основні положення охорони праці

Охорона праці – це система правових, соціально-економічних, організаційно-технічних, санітарно-гігієнічних і лікувально-профілактичних заходів та засобів, спрямованих на збереження життя, здоров’я і працездатності людини у процесі трудової діяльності (Закон України «Про охорону праці»). Вона є невід'ємною частиною будь-якої трудової діяльності, включаючи розробку програмного забезпечення. Дотримання вимог законодавства та правил безпечної роботи дозволить уникнути нещасних випадків та професійних захворювань, забезпечить збереження здоров'я та працездатності працівників.

Відповідно до статті 13 Закону України «Про охорону праці»[30], роботодавець зобов'язаний створити на робочому місці умови праці відповідно до нормативно-правових актів, а також забезпечити дотримання вимог законодавства щодо прав працівників у галузі охорони праці.

Працівники, зі свого боку, зобов'язані знати і виконувати вимоги нормативно-правових актів з охорони праці, проходити у встановленому порядку попередні та періодичні медичні огляди, а також дотримуватися правил безпечної поведінки на робочому місці.

Держава здійснює державне управління охороною праці, встановлює єдині вимоги щодо охорони праці, здійснює контроль за дотриманням законодавства про охорону праці та проводить інші заходи, спрямовані на забезпечення безпечних і здорових умов праці.

Важливим аспектом охорони праці є навчання та інструктаж працівників з питань безпеки праці. Роботодавець зобов'язаний забезпечити проведення інструктажів, навчання та перевірку знань працівників з питань охорони праці.

5.2 Недоліки та умови роботи за комп’ютером

Робота за комп’ютером має свої особливості та потенційні ризики для здоров’я працівників. Серед основних недоліків можна виділити:

1. Навантаження на зір – тривала робота за монітором може призвести до втоми очей, погіршення зору, сухості та подразнення очей.

2. Навантаження на опорно-руховий апарат – неправильна постава, тривале сидіння в одній позі та повторювані рухи можуть спричинити біль у спині, шиї, руках та кистях.

3. Психологічне навантаження – інформаційне перевантаження, стресові ситуації та недостатність спілкування можуть негативно впливати на психічне здоров’я працівників.

Для мінімізації цих ризиків необхідно дотримуватися певних умов праці:

1. Ергономіка робочого місця – правильно підібрані стіл, стілець, монітор та клавіатура допоможуть забезпечити комфортну та безпечну роботу.

2. Освітлення – рівномірне та достатнє освітлення робочого місця допоможе зменшити навантаження на зір.

3. Режим праці та відпочинку – регулярні перерви, фізичні вправи та розминка допоможуть уникнути втоми та перенапруження.

4. Психологічний комфорт – створення сприятливої атмосфери на робочому місці, підтримка колег та можливість спілкування допоможуть зберегти психічне здоров’я працівників.

5.3 Електробезпека

При роботі з комп'ютерною технікою важливо пам'ятати про потенційну небезпеку ураження електричним струмом. Це може статися через несправне обладнання, пошкоджені дроти або неправильне поводження з електроприладами.

Для забезпечення електробезпеки на робочому місці слід дотримуватися наступних правил:

1. Заземлення – всі електроприлади повинні бути заземлені, щоб

уникнути накопичення статичної електрики та забезпечити безпечний шлях для струму у разі короткого замикання.

2. Використання захисних пристроїв – рекомендується використовувати джерела безперебійного живлення (ДБЖ) та мережеві фільтри для захисту обладнання від перепадів напруги та імпульсних перешкод.

3. Відключення від мережі – після завершення роботи або у разі тривалої перерви комп'ютер та інше електрообладнання слід відключати від мережі.

4. Заборона на самостійний ремонт – ремонт електрообладнання повинен проводитися лише кваліфікованими фахівцями.

5. Інструктаж з електробезпеки – всі працівники, які працюють з комп'ютерною технікою, повинні пройти інструктаж з електробезпеки та знати правила безпечного поводження з електроприладами.

Дотримання цих правил допоможе забезпечити безпеку працівників та уникнути нещасних випадків, пов'язаних з ураженням електричним струмом.

5.4 Пожежна безпека при роботі з комп’ютером

Комп'ютерна техніка та супутні електронні пристрої є джерелом підвищеної пожежної небезпеки. Несправності в електропроводці, перевантаження мережі, неправильна експлуатація обладнання або його несправність можуть призвести до займання.

Для запобігання пожежі на робочому місці, де використовується комп'ютерна техніка, необхідно дотримуватися правил пожежної безпеки в Україні, затверджених наказом Міністерства внутрішніх справ України від 30 грудня 2014 року № 1417 [31]. Зокрема, слід звернути увагу на наступні пункти:

1. Розміщення техніки – комп'ютери та інше електронне обладнання повинні бути розміщені на відстані не менше 1 метра від легкозаймистих матеріалів.

2. Електропроводка – електропроводка має бути справною, без пошкоджень ізоляції. Необхідно використовувати розетки та подовжувачі із заземленням.

3. Охолодження – вентиляційні отвори комп'ютера та інших пристроїв повинні бути вільними, щоб забезпечити належне охолодження.

4. Захист від перегріву – слід уникати перегріву обладнання, не накривати його тканинами або іншими матеріалами, що можуть перешкоджати вентиляції.

5. Регулярна перевірка та обслуговування – періодично слід проводити перевірку та обслуговування електрообладнання кваліфікованими спеціалістами.

6. Наявність засобів пожежогасіння – на робочому місці повинні бути первинні засоби пожежогасіння (вогнегасники, покривала з негорючих матеріалів), а працівники повинні знати, як ними користуватися.

7. Дії у разі пожежі – у разі виникнення пожежі необхідно негайно відключити електрообладнання від мережі, повідомити пожежну службу та вжити заходів для гасіння пожежі.

Дотримання цих правил допоможе забезпечити пожежну безпеку на робочому місці та запобігти виникненню небезпечних ситуацій.

5.5 Вентиляція

Відповідно до державних санітарних норм та правил роботи з візуальними дисплейними терміналами електронно-обчислювальних машин [32], приміщення, де працюють з комп'ютерами, повинні бути обладнані системами вентиляції, що забезпечують оптимальні параметри мікроклімату.

Недостатня вентиляція може призвести до:

1. Підвищення температури та вологості повітря – це може призвести до перегріву обладнання, зниження його продуктивності та збільшення ризику виникнення пожежі. Крім того, висока температура та вологість негативно впливають на самопочуття працівників, знижуючи їх працездатність та концентрацію уваги.

2. Накопичення шкідливих речовин – комп'ютерна техніка виділяє невелику кількість шкідливих речовин, таких як озон та формальдегід. Недостатня вентиляція може призвести до накопичення цих речовин у повітрі,

що може негативно вплинути на здоров'я працівників.

3. Зниження рівня кисню – у приміщеннях з недостатньою вентиляцією може знижуватися рівень кисню в повітрі, що призводить до сонливості, головного болю та зниження працездатності.

Для забезпечення належної вентиляції приміщення необхідно:

1. Регулярне провітрювання – рекомендується провітрювати приміщення кожні 1-2 години, відкриваючи вікна або кватирки.

2. Використання систем вентиляції – у приміщеннях з великою кількістю комп'ютерної техніки або відсутністю можливості регулярного провітрювання необхідно встановити системи припливно-витяжної вентиляції.

3. Контроль параметрів мікроклімату – необхідно регулярно контролювати температуру, вологість та рівень кисню в повітрі, щоб забезпечити відповідність вимогам ДСН 3.3.2.007-98 [32].

4. Очищення повітря – для видалення пилу та інших забруднень з повітря можна використовувати повітряні фільтри.

Забезпечення належної вентиляції є важливим заходом для створення безпечних та комфортних умов праці при роботі з комп'ютерною технікою. Це сприятиме збереженню здоров'я працівників та підвищенню їхньої продуктивності.

Висновок до п’ятого розділу

У цьому розділі було розглянуто питання охорони праці при роботі з комп’ютерною технікою, включаючи ергономіку робочоко місця, електробезпеку, пожежну безпеку та вентиляцію. Було проаналізовано основні вимоги та правила, встановлені законодавством України, щодо забезпечення безпечних та комфортних умов праці для працівників, які взаємодіють з комп’ютерною технікою.

ЗАГАЛЬНІ ВИСНОВКИ

У результаті аналізу предметної області було визначено основні принципи створення телеграм-ботів, їх архітектуру та функціональні можливості. Було досліджено ринок існуючих телеграм-ботів та виявлено відсутність багатофункціональних рішень, що поєднують у собі широкий спектр можливостей.

Спроектовано архітектуру телеграм-бота, включаючи вибір технологічного стеку (Python, Aiogram), проектування бази даних (*SQLite*) та визначення основних функціональних можливостей бота.

Розроблено багатофункціональний телеграм-бот *Geek's Friend*, який надає користувачам широкий спектр можливостей, таких як:

1. Завантаження відео з *TikTok* без водяного знака.
2. Генерація хештегів для публікацій *TikTok.*
3. Пошук аніме-серіалу за скриншотами.
4. Керування списками завдань (*To-Do list*).
5. Отримання останніх аніме-новин.
6. Спілкування зі штучним інтелектом *ChatGPT.*
7. Надсилання фідбеку розробнику бота.

Проведено тестування розробленого бота, яке підтвердило його працездатність та відповідність поставленим вимогам.

Проведено економічне обґрунтування проекту, включаючи розрахунок трудомісткості та вартості розробки, що підтверджує його економічну доцільність та потенціал для подальшого розвитку та монетизації.

Розроблено рекомендації щодо подальшого розвитку та вдосконалення бота, включаючи розширення функціоналу, оптимізацію роботи та впровадження нових технологій.

Розглянуто питання охорони праці при роботі з комп'ютерною технікою, включаючи ергономіку робочого місця, електробезпеку, пожежну безпеку та

вентиляцію.

Таким чином, у результаті виконання дипломної роботи було створено багатофункціонального телеграм-бота *Geek's Friend*, який відповідає сучасним вимогам та потребам користувачів. Розроблений бот має потенціал для подальшого розвитку та вдосконалення, що відкриває перспективи для його використання у різних сферах діяльності.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Async PRAW Documentation. URL: https://asyncpraw.readthedocs.io/en/stable/getting_started/quick_start.html (дата звернення: 30.11.2023).

2. Aiogram 3.7.0 Documentation. URL: https://docs.aiogram.dev/en/dev-3.x/index.html (дата звернення: 04.12.2023).

3. Telegram API Documentation. URL: https://core.telegram.org/bots/api#formatting-options (дата звернення: 04.12.2023).

4. SauceNAO API Documentation. URL: https://pypi.org/project/saucenao-api/ (дата звернення: 11.12.2023).

5. Beautiful Soup Documentation. URL: https://www.crummy.com/software/BeautifulSoup/bs4/doc/#installing-a-parser (дата звернення: 08.12.2023).

6. Кодов А., Як працювати з модулем asyncio в Python. 2023. URL: https://sky.pro/media/kak-rabotat-s-modulem-asyncio-v-python/ (дата звернення: 04.01.2024).

7. Python Language Documentation. URL: https://docs.python.org/3/ (дата звернення: 04.01.2024).

8. Кодов А., Як використовувати Python для роботи з API Reddit. 2023. URL: https://sky.pro/media/kak-ispolzovat-python-dlya-raboty-s-api-reddit/ (дата звернення: 09.01.2024).

9. Карманов И., За границей Hello World: полный гайд по разработке Telegram ботов с помощью Python и Aiogram 3, Часть 1. 2023. URL: https://habr.com/ru/articles/732136/ (дата звернення: 15.02.2024).

10. Groosha, Пишемо Telegram-ботів з aiogram 3.x. URL: https://mastergroosha.github.io/aiogram-3-guide/ (дата звернення: 18.02.2024).

11. GeekForGeeks, Python Tutorial.URL: https://www.geeksforgeeks.org/python-programming-language-tutorial/?ref=outind (дата звернення: 18.02.2024).

12. Tiktok video no watermark2 API. URL: https://rapidapi.com/yi005/api/tiktok-video-no-watermark2 (дата звернення: 21.02.2024).

13. Fsoky YouTube channel. URL: https://www.youtube.com/@fsoky (дата звернення: 21.02.2024).

14. Sudo teach IT YouTube channel. URL: https://www.youtube.com/@sudoteach (дата звернення: 11.02.2024).

15. Shcoder YouTube channel. URL: https://www.youtube.com/@shcoder001

16. OpenAI API Documentation. URL: https://platform.openai.com/docs/api-reference/introduction (дата звернення: 22.03.2024)

17. W3schools, Python Tutorial. URL: https://www.w3schools.com /python/default.asp (дата звернення: 15.03.2024)

18. Reddit, r/learnpython. URL: https://www.reddit.com/r/learnpython/ (дата звернення: 15.03.2024)

19. Stack overflow. URL: https://stackoverflow.com (дата звернення: 17.03.2024).

20. Axel Baudot, Project based learning. URL: https://github.com/practical-tutorials/project-based-learning?tab=readme-ov-file#additional-resources (дата звернення: 09.04.2024).

21. Nishant Sahoo, Scraping Top 50 Movies on IMDb using BeautifulSoup, Python. 2018. URL: https://medium.com/@nishantsahoo/which-movie-should-i-watch-5c83a3c0f5b1 (дата звернення: 09.04.2024).

22. Наказ Про затвердження Правил пожежної безпеки в Україні: Наказ М-ва внутр. справ України від 30.12.2014 р. № 1417: станом на 7 квіт. 2023 р. URL: https://zakon.rada.gov.ua/laws/show/z0252-15#Text (дата звернення: 21.04.2024).

23. Закон Про охорону праці в Україні: Відомості Верховної Ради України (ВВР), 1992, № 49. URL: https://zakon.rada.gov.ua/laws/show/2694-12#Text (дата звернення: 17.04.2024).

24. ДСанПІН 3.3.2.007-98. Державні санітарні правила і норми роботи з візуальними дисплейними терміналами електронно-обчислювальних машин. URL: https://zakon.rada.gov.ua/rada/show/v0007282-98#Text (дата звернення: 17.04.2024).