МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

МАТЕМАТИЧНИЙ ФАКУЛЬТЕТ
Кафедра програмної інженерії

КВАЛІФІКАЦІЙНА РОБОТА БАКАЛАВРА

на тему: **«РОЗРОБКА КОМУНІКАЦІЙНОГО БОТА ІЗ ЗАСТОСУВАННЯМ ПЛАТФОРМИ DISCORD»**

Виконав: студент 4 курсу, групи 6.1210-1пі
спеціальності 121 інженерія програмного забезпечення
(шифр і назва спеціальності)
освітньої програми програмна інженерія
(назва освітньої програми)
М.С. Полтавець
(ініціали та прізвище)

Керівник завідувач кафедри фундаментальної та прикладної математики, професор, д.т.н. Гребенюк С.М.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)

Рецензент доцент кафедри комп'ютерних наук, доцент, к.т.н. Решевська К.С.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)

Запоріжжя – 2024

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

Факультет математичний

Кафедра програмної інженерії

Рівень вищої освіти бакалавр

Спеціальність 121 інженерія програмного забезпечення
(шифр і назва)

Освітня програма програмна інженерія

ЗАТВЕРДЖУЮ
Завідувач кафедри програмної інженерії, к.ф.-м.н., доцент

_____ Лісняк А.О.
(підпис)

“ ____ ” _____ 2023 р.

З А В Д А Н Н Я

НА КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТОВІ

Полтавцю Микиті Станіславовичу
(прізвище, ім’я та по-батькові)

1. Тема роботи Розробка комунікаційного бота із застосуванням платформи Discord

керівник роботи Гребенюк Сергій Миколайович, д.т.н., професор
(прізвище, ім’я та по-батькові, науковий ступінь, вчене звання)

затверджені наказом ЗНУ від « 21 » грудня 2023 року № 2180-с

2. Строк подання студентом роботи 03.06.2024 р.

3. Вихідні дані до роботи 1. Постановка задачі.
2. Перелік літератури.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)
1. Постановка задачі.
2. Основні теоретичні відомості.
3. Програмна реалізація комунікаційного бота.

5. Перелік графічного матеріалу (з точним зазначенням обов’язкових креслень)
презентація за темою докладу

6. Консультанти розділів роботи

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

7. Дата видачі завдання 25.12.2023 р.

К А Л Е Н Д А Р Н И Й П Л А Н

№	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1.	Розробка плану роботи.	09.01.2024	
2.	Збір вихідних даних.	23.01.2024	
3.	Обробка методичних та теоретичних	13.02.2024	
	джерел.		
4.	Розробка першого та другого розділу.	29.03.2024	
5.	Розробка третього розділу.	20.05.2024	
6.	Оформлення та нормоконтроль	27.05.2024	
	кваліфікаційної роботи бакалавра.		
7.	Захист кваліфікаційної роботи.	17.06.2024	

Студент _____ М.С. Полтавець
(підпис) (ініціали та прізвище)

Керівник роботи _____ С.М. Гребенюк
(підпис) (ініціали та прізвище)

Нормоконтроль пройдено

Нормоконтролер _____ А.В. Столярова
(підпис) (ініціали та прізвище)

РЕФЕРАТ

Кваліфікаційна робота бакалавра «Розробка комунікаційного бота із застосуванням платформи Discord»: 50 с., 20 рис., 1 табл., 17 джерел, 3 додатка.

БІБЛІОТЕКА, БОТ, ЗАСТОСУНОК, КОМАНДИ, DISCORD, PYTHON, VISUAL STUDIO.

Об’єкт дослідження – середа розробки Visual Studio, бібліотеки для роботи з нею, застосунок Discord.

Предмет дослідження – автоматизований бот для платформи Discord.

Мета роботи: розробка бота для виконання зазначених автоматизованих команд на сервері системи обміну інформацією Discord.

Метод дослідження: методи збору та аналізу вимог до програмного забезпечення, методи проєктування, конструювання та тестування програмного забезпечення.

При розробці додатку був проведений аналіз і актуальність предметної області. Реалізовано серверну частину з використанням API Discord, застосунок для бота було розроблено за допомогою Visual Studio та відповідних бібліотек discord.

В ході виконання роботи отримано застосунок для демонстрування інформації користувачам і виконання певних команд.

SUMMARY

Bachelor’s qualifying paper “Development of a Communication Bot Using the Discord Platform”: 50 pages, 20 figures, 1 table, 17 references, 3 supplements.

APPLICATION, BOT, COMMANDS, DISCORD, LIBRARY, PYTHON, VISUAL STUDIO.

The object of the study are Visual Studio development environment, libraries for working with it, Discord application.

The subject of the study is an automated bot for the Discord platform.

The aim of the study is a development of a bot to execute the specified automated commands on the server of the Discord information exchange system.

Research methods: are methods of collecting and analyzing software requirements provision, methods of software design, construction and testing.

During the development of the application, an analysis and relevance of the subject area was carried out. The server part was implemented using the Discord API, the bot application was developed using Visual Studio and the corresponding discord libraries.

As a result of the work, an application was obtained for displaying information to users and executing certain commands.

ЗМІСТ

ВСТУП

Цей проєкт допоможе зрозуміти всі таємниці світу Discord, відкривши двері в нову еру комунікаційних технологій. Розглядається процес створення та запуску комунікаційного боту для платформи Discord, використовуючи мову програмування Python і інтегровану середу розробки Visual Studio.

З кожним натисканням на клавіши та стрічкою коду, виконується перетворення ідеї в дійсність, змінюючи спосіб, яким користувачі взаємодіють у своїх спільнотах. Цей проєкт не лише здатний автоматизувати рутинні завдання, але й створює можливість для нових форм спілкування чи співпраці в цифровому просторі.

Проєкт базується на сучасних технологіях штучного інтелекту та обробці природної мови, щоб зробити взаємодію з користувачами максимально ефективною та легко зрозумілою. Зроблений бот стане допоміжним кроком у створенні спільнот і створенні нових способів спілкування на платформі Discord.

Такий бот є надзвичайно корисним інструментом для багатьох спільнот. Вони можуть пропонувати широкий спектр функцій та можливостей, які полегшують комунікацію спільноти.

В першу чергу, бот забезпечує автоматизацію рутинних завдань, що звільняє час учасників спільноти для більш важливих справ. Він може виконувати такі завдання, як розклад подій, нагадування про важливі дати або оповіщення про оновлення. Це може допомогти зберегти порядок та організованість в спільноті. Деякі боти навіть можуть створювати та керувати власними каналами для різних видів спілкування чи розваг. Наприклад, він може автоматично створювати окремі канали для обговорення проєктів, запитань або навіть ігор, що сприяє зближенню між учасниками. Бот також може надавати корисну інформацію, відповідати на запитання користувачів та навіть навчати новим навичкам або функціям платформи Discord. Це створює

навколишнє середовище, де користувачі можуть не лише спілкуватися, але і навчатися та розвиватися разом.

Загалом, бот є важливим інструментом для підтримки активної та здорової спільноти на певній платформі, сприяючи зручності та ефективності.

1 ТЕХНІЧНЕ ЗАВДАННЯ ДО ПРОЄКТУ

1.1 Терміни та визначення

1.1.1 Загальні терміни

Discord – сервіс для спілкування онлайн за допомогою відео, голосу або тексту. Кросплатформна пропрієтарна система миттєвого обміну повідомленнями з підтримкою VoIP та відеоконференцій, призначена для використання різними спільнотами за інтересами.

Бот (скорочено від англ. *Robot*) – спеціальна розроблена програма, що автоматично виконує або за заданим розкладом будь-які дії або завдання через інтерфейси, що використовуються і звичайним користувачем [8].

Застосунок – клас програм, призначений для вирішення практичних завдань користувача чи інших таких застосунків і програм для безпосередньої взаємодії з ними [9].

Python – високорівнева мова програмування загального призначення з динамічною строгою типізацією та автоматичним керуванням пам’яттю.

1.1.2 Технічні терміни

Інтегрована середа розробки – програмний застосунок для допомоги у розробці і редагування коду програмістом.

Мова програмування – це штучна розроблена мова, що створена для передачі інформації машинам для подальшої обробки ними. Такі мови програмування використовуються для створення програм і алгоритмів, які контролюють і навіть налаштовують поведінку машини [10].

Бібліотека – зібрання підпрограм або об’єктів, що використовуються для

розробки програмного забезпечення.

Клас – це спеціальна конструкція в об’єктно-орієнтованому програмуванні, що використовується для групування пов’язаних змінних, об’єктів та функцій.

Функція – ділянка програмного коду, яку можливо виконати звернувшись до неї.

Команда – наказ комп’ютерній програмі діяти як деякий інтерпретатор для вирішення задачі.

Сервер (discord) – одна з найважливіших функцій застосунку Discord, що дозволяє користувачам створити структуровану спільноту одразу для багатьох людей, зручно редагувати її та керувати учасниками.

1.2 Інформаційна довідка

1.2.1 Про мову Python

Python – це потужна, високорівнева та ефективна мова програмування, яка поєднує ефективність із простотою використання. Вона була розроблена Гвідо ван Россумом і вперше випущена у 1991 році. Відтоді розробники в усьому світі оцінили її [7].

Гвідо ван Россум у той час працював у CWI (голландському національному дослідницькому інституті математики та інформатики). Метою створення Python було розробити мову програмування, яка була б високоефективною, простою у використанні та здатною виконувати широкий спектр завдань.

Відоме шоу Монті Пайтона «Монті Пайтон і Святий Грааль» надихнув назву «Python». Ван Россум вирішив назвати свою нову мову саме так, щоб підкреслити її унікальний, але простий і зрозумілий синтаксис, який відображено в витончених підзаголовках гумористичних сцен у відомому шоу.

У 1991 році була доступна перша публічна версія Python (версія 0.9.0). З тих пір мова постійно розвивалася і вдосконалювалася, отримуючи нові функції та вдосконалення. Значний внесок у розвиток мови зробила активна спільнота розробників, які постійно вносять свої пропозиції та вдосконалення і до теперішнього часу.

Її простота та читабельність є одними з основних переваг Python. Мова має чистий і простий синтаксис, що робить код більш зрозумілим і простішим у використанні. Це дозволяє розробникам створювати програми та вирішувати проблеми в короткі терміни. Мову можна використовувати для багатьох цілей, включаючи аналіз даних, машинне навчання та веброзробку. Її багатофункціональність навіть дозволяє розробникам легко виконувати кілька проєктів одночасно.

Загалом Python – це потужна та динамічна мова програмування, яка завоювала довіру мільйонів розробників завдяки своїй простоті та широкому спектру застосування.

1.2.2 Про застосунок Discord

Discord – це платформа для відеозв’язку, спілкування голосом і текстових повідомлень, яка швидко набула популярності серед геймерів, різних спільнот і професійних груп. Discord був заснований Джейсоном Цурекі та Стенлі Таном у 2015 році з метою створення платформи для спілкування у онлайн-іграх, але дуже швидко поширився на більшість інших сфер. На популярність Discord сприяють декілька факторів.

Інтерфейс. Інтерфейс простий і легко зрозумілий. На платформі можна використовувати різноманітні функції, такі як голосовий і текстовий чат, створення приватних і публічних серверів, обмін файлами, інтеграцію з ігровими платформами та багато іншого.

Спільнота. Іншим фактором популярності Discord є його величезна

активна спільнота користувачів. Discord надає користувачам можливість легко взаємодіяти та обмінюватися ідеями з людьми з усього світу. Тут мають місце бути гільдії для гравців у комп'ютерні і настільні ігри, спільноти по різним спільним захопленням чи секціям, а останнім часом Discord навіть став основою для комфортної роботи і зв'язку серед працівників різних сфер і галузей.

Постійне оновлення та покращення платформи є останнім але не менш важливим пунктом цього переліку. Розробники Discord постійно покращують функціональність, виправляють помилки та додають нові цікаві можливості, щоб надати користувачам найкращий досвід використання платформи.

Однією з таких функцій стали боти, яких у будь-який момент можна запросити на свій сервер як звичайного користувача для виконання певних завдань. Такі завдання потім іменували «командами» [12].

1.3 Функціональні вимоги

1.3.1 Побудова проблеми

Боти у Discord відіграють важливу роль у покращенні та розширенні функціоналу цієї платформи. Наведемо кілька причин, чому вони є не лише зручними, але й корисними у використанні.

Автоматизація завдань: боти можуть автоматизувати велику кількість рутинних завдань, що звільняє користувачів від необхідності виконувати ці завдання кожного разу вручну. Нагадування, планування подій, модерація чату або навіть оповіщення про нові повідомлення можуть бути частиною їх функціоналу.

Надання інформації: боти можуть надавати корисну інформацію про різні елементи спільноти або реального світу, такі як: правила; вказівки чи розклади подій; новини; корисна інформація; поради і таке інше. Крім того,

вони мають можливість відповідати на запити користувачів і надавати корисні посилання чи ресурси.

Взаємодія з іграми: деякі боти можуть спілкуватися з іграми прямо за допомогою Discord. Вони можуть відстежувати статистику гри, створювати канали для голосування гравців або навіть організовувати турніри.

Спеціалізовані функції: є боти, які спеціалізуються на певних завданнях. Наприклад, вони можуть працювати над музикою, економікою або адмініструванням сервера. Це дозволяє користувачам керувати сервером і використовувати ті функції, які їм потрібні.

Розваги та розвиток спільноти: деякі боти можуть створювати приємну атмосферу для учасників, додаючи розваги на сервер. Сприяючи активнішому спілкуванню та взаємодії між учасниками, вони можуть допомогти розвитку спільноти.

1.3.2 Призначення і цілі створення застосунку

Функціональне призначення застосунку – реалізація виконання запрограмованих завдань програмою-ботом.

Мета створення застосунку – демонстрація елементарних можливостей бота для спрощення взаємодії користувачів одне з одним і з платформою.

Експлуатаційне призначення застосунку: бот використовуватиметься як адміністраторами серверу, так і звичайними користувачами. У них буде спільний функціонал, проте деякі функції зможе використовувати лише адміністратори.

1.3.3 Функціональні можливості застосунку

Функціональні можливості – це те, що буде вміти розроблений бот. Розглянемо основні з них.

Стежити за учасниками серверу: бот матиме можливість «бачити» коли до спільноти доєднується новий користувач. Так само і навпаки, застосунок слідкуватиме за виходом учасника з серверу. В обох випадках, він буде взаємодіяти з користувачами: вітатись і прощатись відповідно.

Аналізувати профіль користувача: застосунок зможе зчитати інформацію аккаунту учасника і надіслати у відповідь повну і актуальну інформацію про профіль того, хто виконав відповідну команду.

Таймер: однією з розроблених команд буде таймер. Бот самостійно сповістить, коли пройде зазначений час.

Демонстрація курсу валют: команда, що виведе актуальну інформацію щодо курсу долара до гривні. Інформація оновлюватиметься самостійно, кожен день.

Фільтр: функціонал бота включатиме подію, що аналізуватиме відправлені на сервері повідомлення і у разі, якщо повідомлення буде містити в собі заборонене слово чи набір літер – видалятиме його.

Однією з важливих функцій серверів Discord є голосовий чат. Він дозволятиме учасникам збиратись у спеціально відведеному місці для спілкування. Функціонал бота зможе взаємодіяти і з цими можливостями, а саме:

- під’єднання: застосунок матиме змогу підключитись до голосового каналу, разом з іншими учасниками серверу, які вже перебувають там;
- транслювання: можливість бота включити аудіофайл у голосовому каналі.

1.3.4 Опис предметної області

Предметною областю є розробка застосунку для взаємодії користувача з платформою і зручним обміном інформації. Ця система повинна виконувати

запрограмовані завдання, що викликаються користувачем. Процес взаємодії з системою проходить наступним чином. Користувач, вводить команду та, за наявності, необхідний для неї текст. Застосунок(бот) встановлює зв’язок із сервером, аналізує введену для нього команду і виконує задачу. Користувач отримує від застосунку відповідь із вирішеною проблемою або отриманим результатом.

У даному завданні, користуватись ботом може кожен користувач, що присутній на сервері. Можливі команди будуть представлені користувачам або у вигляді ознайомчого тексту, або окремою командою бота, що демонструватиме перелік наявних команд.

Виклик команди буде представлено у вигляді написання особливого знаку, щоб бот міг відрізнити звичайне повідомлення від команди і назви команди латинецею. Приклад: !text (текст користувача).

У бота також присутні команди, які зможе виконувати лише адміністратор, або керівник серверу.

Опис сутностей і їх атрибутів наведено у таблиці 1.1.

Таблиця 1.1 – Опис сутностей і атрибутів

Сутність	**Опис**	**Атрибути**
Бот	Програмне забезпечення, що працює на сервері розробника, зв’язується з хмарними серверами Discord, та яке може виконувати різні дії і команди.	– Ім’я; – Опис; – Розробник; – Функціональність.
Користувач	Людина, що є частиною спільноти і присутня на сервері Discord.	– Ім’я; – ID; – Ролі.

Продовження таблиці 1.1.

Сутність	Опис	Атрибути
Адміністратор	Користувач Discord з підвищеними повноваженнями, який може керувати сервером Discord, призначати ролі та видаляти користувачів.	– Ім’я; – ID; – Рівень дозволів.
Хмарні сервера Discord	Інфраструктура, на якій розміщується весь Discord.	– Місцезнаходження; – Потужність; – Дані.

1.3.5 Опис системи

Адміністрування – це мистецтво керування, яке лежить в основі успішної роботи будь-якої організації. Воно подібно до диригента оркестру, який поєднує зусилля окремих інструментів у гармонійну симфонію. Воно полягає в тому, щоб спрямувати діяльність організації до чітко визначених цілей, максимально ефективно використовуючи ресурси. Використання сучасних методів, ресурсів і інструментів адміністрування дозволяє компаніям досягати нових висот і створювати продуктивні результати.

Адміністрування, допомога користувачам і комунікація з ними – головні аспекти у підтримці діяльності і існування серверу. Раніше, все це відбувалось виключно у реальному часі при підтримці адміністрації, тобто реальних людей. Під час розвитку серверу росте і кількість користувачів на ньому, а разом із цим і час, який витрачався на допомогу кожному. На додачу, користувачі можуть бути із різних країн, тому і матимуть різні часові пояси, і

встигати та слідкувати за кожним може бути дуже незручно [15].

Саме тому і було введено функціонал створення спеціальних ботів, що могли би виконувати ці завдання автономно, без присутності адміністраторів Онлайн.

Можливості застосунку:

а) основні:
   - виконання звичайних команд;
   - взаємодія з ботом;

б) адміністративні:
   - виконання адміністративних команд.

1.3.6 Порівняння з подібними застосунками

Існують схожі програми і застосунки на інших відомих платформах спілкування і комунікації. Нижче буде наведено декілька прикладів.

Slack: має власну екосистему ботів, які дозволяють автоматизувати завдання, створювати нагадування, організовувати канали та багато іншого.

Telegram: в ньому існують боти, які можуть виконувати різноманітні завдання, такі як надсилання повідомлень, нагадування, робота з групами та каналами, обробка замовлень тощо.

Microsoft Teams: ця платформа для співпраці також має можливості для розробки ботів, які допомагають у модерації чату, організації робочих процесів та автоматизації рутинних завдань.

WhatsApp Business API: у бізнес-версії WhatsApp існують схожі API для розробників як і у Discord, які дозволяють створювати ботів для спілкування з клієнтами, надсилання повідомлень, відповідей на запитання тощо.

Facebook Messenger: існують боти для Messenger, які можуть автоматизувати клієнтське обслуговування, надсилання повідомлень, обробку замовлень та багато іншого.

Іншими словами, різні месенджери і програми мають свої різні варіації реалізації ботів. Кожен з них програмується по-своєму, згідно мови програмування на якій написано месенджер і середовища розробки. Проте, їх функціонал і швидкість роботи зазвичай дуже схожі, а іноді можуть бути навіть ідентичними через те, що більшість таких ботів-програм створюється саме спільнотою для таких самих користувачів [4].

Потребу, актуальність і необхідність певних завдань ставлять саме користувачі подібних спільнот у месенджерах. А зручне API (опис способів взаємодії однієї програми з іншою, або ж інтерфейс) дозволяє спільноті реалізувати свої потреби уникаючи прямого звернення до розробників конкретної платформи або очікування відповіді та діяльності від них.

2 ПРОЄКТУВАННЯ ЗАСТОСУНКУ

Проєктування є фундаментальною частиною будь-якого успішного проєкту, незалежно від його масштабу чи складності. Проєктування дає змогу чітко визначити цілі та завдання проєкту, що гарантує, стабільний графік для розробки проєкту і зрозуміле завдання. Проєктування – це не просто креслення чи розрахунки, це творчий процес, який допомагає втілити ідеї в життя і реалізувати їх згідно поставленого плану.

2.1 Розробка і проєктування застосунку

2.1.1 Ідея застосунку

Використовуючи платформу Discord, користувачі часто звертаються до використання ботів чи сторонніх застосунків для цієї мережі. Аналогічним чином довелося зустрітись з ботами і мені. Трохи згодом, під час дискусії з моїми знайомими, мені спала на думку ідея створити власного бота для серверу. Це відбулося напередодні четвертого курсу, саме тоді коли настав час планувати ідею для написання кваліфікаційної роботи. Так і почався мій пошук матеріалу стосовно історії розвитку цієї платформи і появи функціональних ботів.

2.1.2 Використання UML при розробці застосунку

UML (Unified Modeling Language) або уніфікована мова моделювання – це графічна мова, що використовується для опису, візуалізації та проєктування систем програмного забезпечення, а також для моделювання бізнес-процесів,

системного проєктування та відображення організаційних структур. Мова є стандартом, що широко використовується в розробці програмного забезпечення, IT-інфраструктурі та бізнес-системах.

UML створює чіткі та лаконічні діаграми, що описують структуру та поведінку системи, а також допомагає розробникам, аналітикам та користувачам краще зрозуміти систему, її компоненти та їхню взаємодію. Мова моделювання допомагає запланувати систему, визначити її компоненти та встановити зв’язки між ними.

Діаграми UML слугують документацією системи, що полегшує її обслуговування, аналіз та вдосконалення. UML може допомогти скоротити час розробки за рахунок кращого планування та проєктування, а також дає чітке розуміння системи на ранніх стадіях розробки.

2.1.3 Діаграма варіантів використання

Діаграма варіантів використання це інструмент моделювання, який візуально описує функціональні можливості системи та способи їх використання акторами. Це карта системи, яка показує, що вона може робити та хто може з нею взаємодіяти. Вона використовується для розуміння та документування вимог до системи і допомагає у демонструванні, щоб система відповідала потребам користувачів [3].

В рамках діаграми наведеної на рисунку 2.1 присутні чотири актори: хмарний сервіс Discord; користувач; адміністратор; бот.

Актор – це будь-яка сутність, яка взаємодіє з системою і отримує від неї користь. Актор не є частиною самої системи, це зовнішній по відношенню до неї об’єкт або явище і взаємодіє з нею через інтерфейси. На діаграмі варіантів використання актори зазвичай зображуються у вигляді фігурок людей, не залежно від того – людина це чи ні.

На діаграмі важливо чітко визначити акторів системи та описати їхні потреби чи можливості.

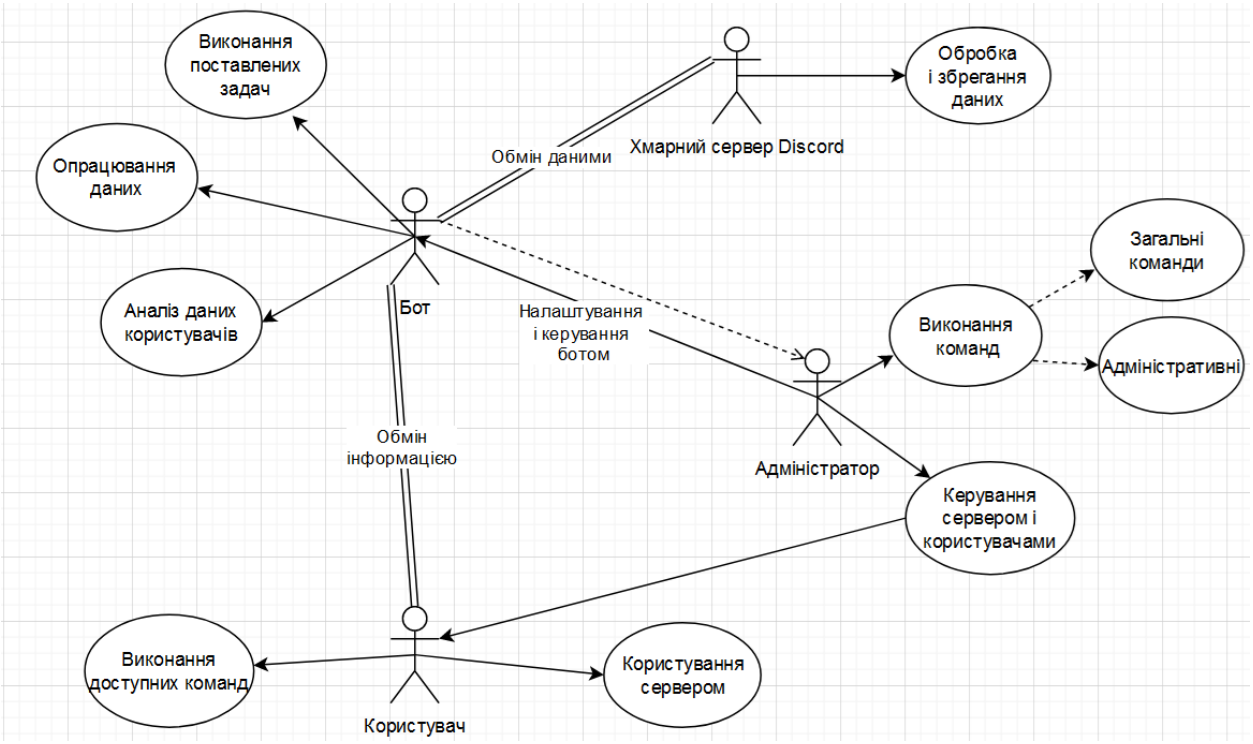


Рисунок 2.1 – Діаграма варіантів використання

На цьому рисунку зображено взаємодію акторів та їх можливості.

Адміністратор. Головна і важлива фігура в управлінні. В його можливості та обов’язки входять керування сервером і налаштування та керування ботом. Адміністратор також слідкує за спільнотою і може керувати учасниками.

Користувач. Не менш важливий учасник спільноти, що перебуває на сервері і користується наданими йому послугами.

Бот. Застосунок, що програмується власником серверу або адміністраторами, призначений на виконання поставлених йому завдань. Він аналізує отриману інформацію, обробляє її і надсилає результат користувачам.

Сервери Discord. Це і є Discord загалом. Сервіси, які підтримують роботу Discord. Зберігають інформацію про активного бота, сервер-спільноту, на якому він знаходиться, а також інформацію щодо користувачів. В тому числі, зберігають основні параметри і дозволи розробки discord-ботів.

Коли користувач авторизується і входить на сервер-спільноту, йому одразу відкривається доступ до можливих команд

2.1.4 Опис варіантів використання

Прецедент (use case) в діаграмі варіантів використання – це опис певної функціональної можливості системи, що розробляється, яка має інформаційну цінність для актора. Тобто, це сценарій, який описує, як актор взаємодіє з системою для досягнення певної мети.

Прецеденти фокусуються на тому, що система робить, а не на тому, як вона це робить. Один прецедент описує один потік подій, який веде до чітко визначеного результату. На діаграмі варіантів використання прецеденти зазвичай зображуються у вигляді еліпсів або кіл.

Прецедент «Виконання команд».

Призначення: варіант використання дає можливість взаємодіяти з ботом шляхом виконання доступних команд.

Основний потік дій: варіант використання починає виконуватись при відправці спеціального префіксу «**!**» і назви команди після нього у доступний чат.

Альтернативний потік дій: деякі команди виконуються тільки якщо користувач знаходиться у голосовому каналі спільноти. У разі невиконання умов, застосунок сповістить користувача необхідними вказівками.

Передумова: перед початком виконання команди краще знаходитись у голосовому каналі.

Прецедент «Керування сервером і користувачами» (прецедент, доступний тільки адміністраторам).

Призначення: варіант використання надає інструмент власнику спільноти і адміністраторам, що дозволяє видаляти користувачів з серверу.

Основний потік дій: варіант використання виконується при відправці спеціального префіксу «**!**», слова «kick» і «@userID», де userID – ID цільового користувача.

Альтернативний потік дій: у виконанні команди буде відмовлено у випадку недостатніх прав на сервері.

Передумова: ви повинні мати певні права, або бути адміністратором на сервері для виконання цієї команди.

Прецедент «Виконання поставлених задач» (прецедент, що виконується самим ботом завдяки користувачу).

Призначення: варіант використання дає можливість отримати результат у відповідь на виконання команд.

Основний потік дій: варіант використання виконуватиметься після отримання ботом команди.

Альтернативний потік дій: команда може бути не виконана при виникненні помилки або перебуванні боту оффлайн.

Передумова: застосунок повинен бути онлайн. Переконатись у цьому можна подивившись на статус бота, що відображатиме поточний стан бота.

3 РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ

Цей розділ кваліфікаційної роботи демонструє процес розробки інтерактивного застосунку, представленого у попередніх двох розділах. Отже, після етапів планування і проєктування, третій розділ розбиратиме сам процес будування проєкту, демонструючи і пояснюючи конкретні ділянки коду.

Розділ детально описує процес розробки програмного забезпечення для Discord бота із наведеним у попередніх розділах функціоналом. Також, конкретний розділ зверне увагу на тестування розробленого застосунку, щоб перевірити функціональність і належну роботу написаної програми. У кінці розділу буде невеликий висновок щодо реалізованого проєкту і його роботі.

3.1 Опис інструментів розробки

Під час реалізації проєкту були використані інтегроване середовище розробки Visual Studio, API Discord і бібліотеку piscord.py.

discord.py – одна з популярних бібліотек Python, що використовується для створення ботів Discord на цій мові. Вона дає можливість взаємодіяти з API Discord, використовуючи мову Python. API Discord також працює з багатьма іншими мовами програмування і має багато різних бібліотек для розробки на них [11].

За допомогою discord.py можна легко створити бота, додати йому токен, налаштувати префікси команд і розробити функціонал. Ця бібліотека дозволяє обробляти різні типи подій: повідомлення, реакції, приєднання/видалення користувачів, зміни каналів тощо. Також, вона має можливість створювати, видаляти, редагувати канали та керувати їхніми налаштуваннями.

Discord.py має простий та зрозумілий інтерфейс API, що робить його доступним для розробників з будь-яким рівнем досвіду, навіть для новачків у

програмуванні таких застосунків. Бібліотека пропонує широкий спектр функцій, що дозволяє створювати складних чи багатофункціональних ботів. Цей пакет має чимале і активне співтовариство, завжди можна знайти людей, готових тобі допомогти. І головне, у discord.py відкритий код, що дає змогу переглянути код та навіть внести свій вклад чи виправлення [13].

3.2 Розробка системи

3.2.1 Підготовка до написання програми

Першим і головним кроком у розробці системи буде встановлення на комп’ютер пакетів python. Для цього з офіційного сайту завантажується і виконується застосунок**.** Результат встановлення наведено нижче на рисунку 3.1.

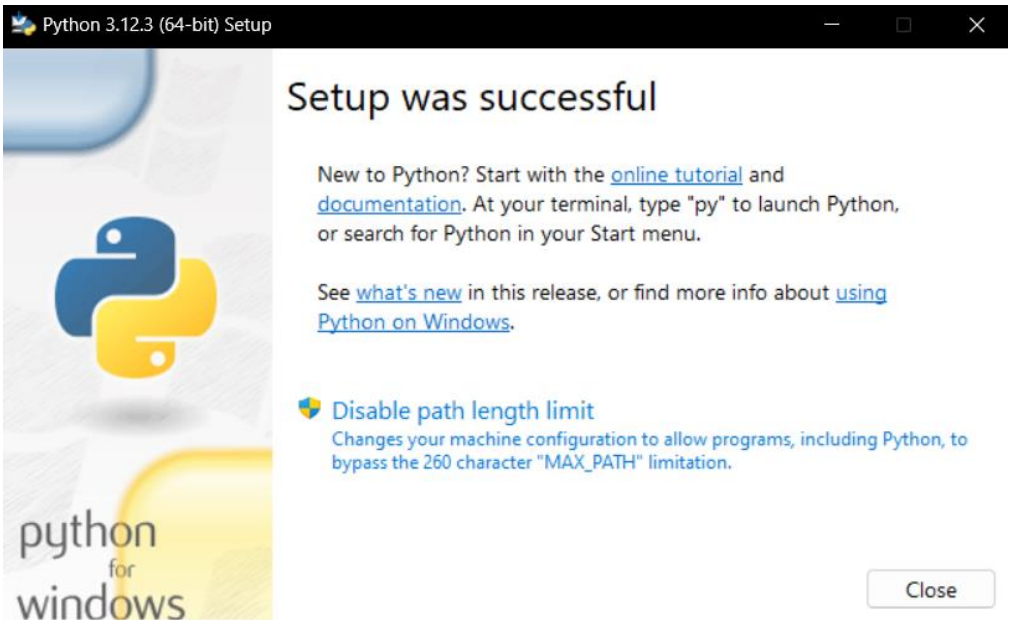


Рисунок 3.1 – Встановлення Python

Далі, в терміналі комп’ютеру пишемо наступну команду: pip install discord.py [1]. Ця команда встановить необхідну бібліотеку для розробки

застосунку. Як тільки процес завершено, можна переходити до наступного кроку.

Після цього, нам потрібно перейти на сайт discord для розробників під назвою “discord developer portal”. На цьому сайті необхідно створити самого бота, щоб він з’явився у полі зору Discord [14]. Даємо йому права адміністратора на сервері, створюємо спеціальний ключ bot-token, за яким він буде зв’язуватись з глобальними серверами і генеруємо посилання, яке вставляємо у нову вкладку браузера. Через це посилання, нашого бота можна додати до власного серверу-спільноти. Для цієї роботи було створено окремий сервер під назвою DisBot і запрошено нашого бота туди. Весь цей процес продемонстровано на рисунках 3.2 та 3.3.

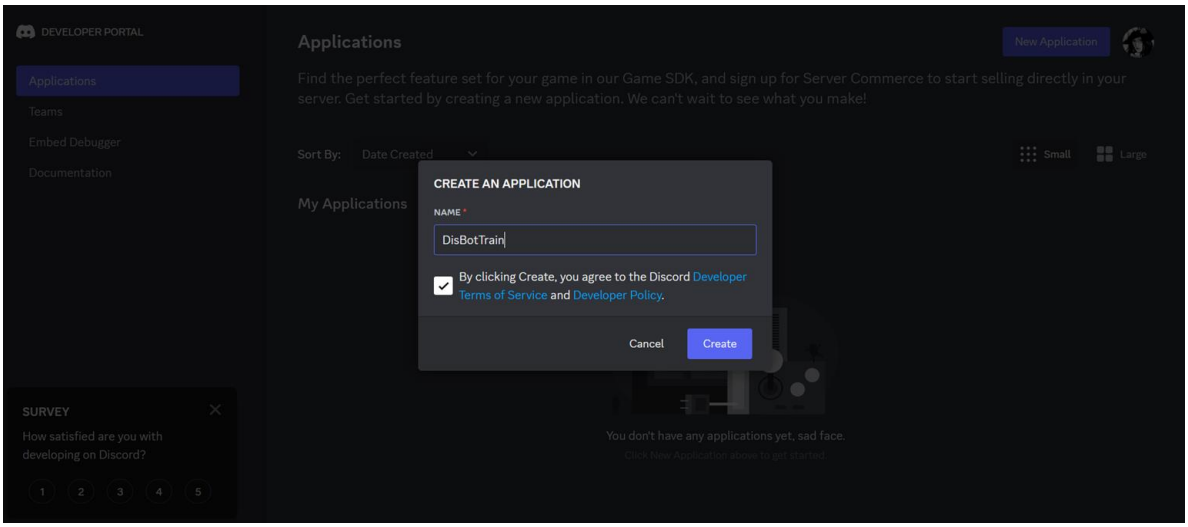


Рисунок 3.2 – Створення застосунку

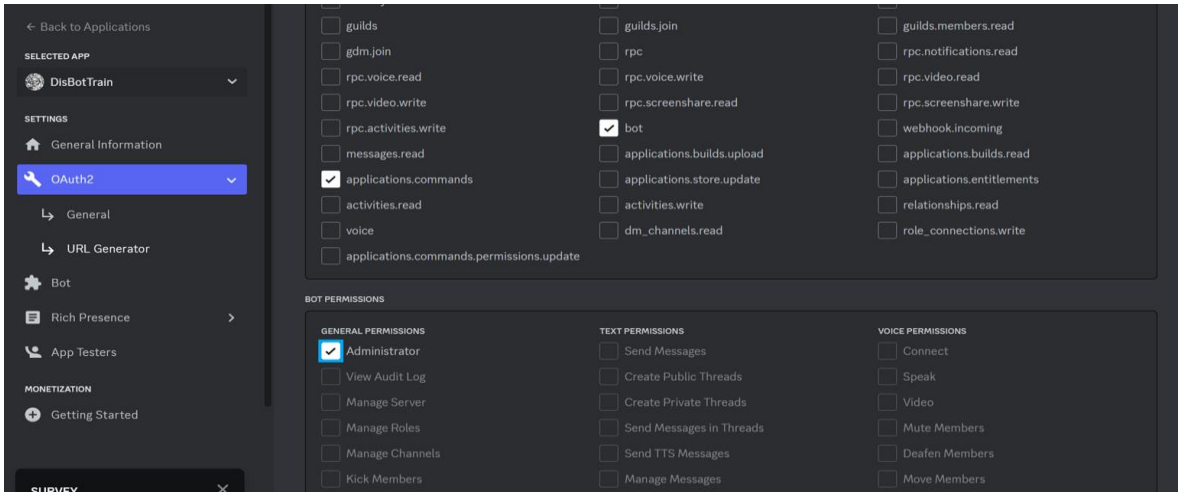


Рисунок 3.3 – Встановлення дозволів для бота

Як тільки ми налаштували бота і додали його на сервер, наш застосунок з’явився серед учасників нашої спільноти. До цього на сервері був присутній лише я. Наразі, наш бот зовсім порожній і немає ніякого функціоналу. Крім того, зараз він перебуває не в мережі, на рисунку 3.4 це проілюстровано.

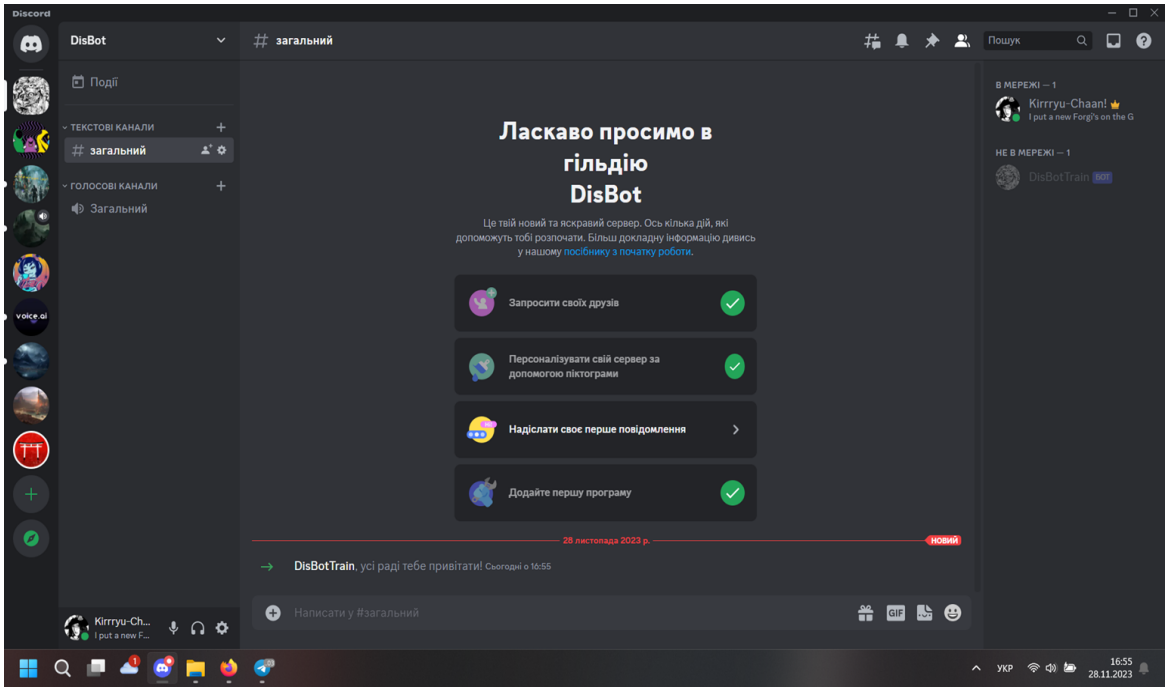


Рисунок 3.4 – Демонстрація спільноти з ботом

3.2.2 Розробка подій застосунку

На початку, необхідно виконати налаштування і імпорт необхідних бібліотек, а також розробити початкові події.

Для початку імпортуємо всі бібліотеки (див. рис. 3.5).

```
import discord
import time
import asyncio
import requests
from discord import FFmpegPCMAudio
from requests import get
from discord import Member
from discord.ext.commands import has_permissions, MissingPermissions
from discord.ext import commands
```

Рисунок 3.5 – Імпорт бібліотек

Імпортування “discord” виконується для отримання функціоналу з бібліотеки discord.py, яку ми встановили раніше.

Рядок “import random” в Python використовується для імпорту модуля random, який надає функції для генерування випадкових чисел та виконання інших команд, що використовують випадкові значення.

Бібліотека asyncio – це бібліотека вбудованих модулів Python, яка використовується для написання асинхронного коду. Асинхронний код – це тип коду, який може виконувати декілька завдань одночасно, не блокуючи основний потік [2].

Бібліотека requests в Python це цікавий і сильний інструмент серед розробників на цій мові, що полегшує роботу з HTTP-запитами. Вона дозволяє надсилати запити до вебсервісів і отримувати відповідь від них і опрацьовувати отримані дані. В цьому проєкті вона використовується для зв’язку з сайтом “ExchangeRate”, який видає спеціальний API ключ для актуалізації ботом даних курсу валют [6].

На рисунку 3.6 проілюстровано початок програмування застосунку. Цей код налаштовує бота на роботу.

Рядок “client = commands.Bot” визначає, який префікс буде використовуватись ботом, щоб розпізнавати команди. У нашому випадку це символ оклику – “!”.

Далі, використовується клієнтський івент, тобто подія, де «клієнт» і є нашим ботом. Подія on_ready виконується коли наш застосунок запускається і бот переходить в стан «в мережі». Ця подія виводить у консоль повідомлення, щоб розробнику було простіше зрозуміти, що бот зараз працює. Також, ця подія налаштовує іконку поточного стану активності бота і його активність в жартівливій формі: «Грає в Minecraft».

Активність на сервері-спільноті, або ж як їх ще називають «гільдії», означає зайнятість користувача конкретною активністю. Тобто Discord при дозволі користувача може аналізувати, чим зараз зайнята система, на якій встановлено Discord. Таким чином, якщо користувач займається якоюсь

відомою для дискорду справою, наприклад грає у популярну гру, або прослуховує музику на платформі Spotify, то дискорд може транслювати цю поточну активність у профіль користувача. Саме цей опис-активність і можна вручну встановити нашому боту.

```
intents = discord.Intents.all()

client = commands.Bot(command_prefix = '!',intents=intents)
@client.event
async def on_ready():
    await client.change_presence(status=discord.Status.idle,
activity=discord.Game('Minecraft'))
    print("Hello, i'm ready!")
    print("--------------------------")

***

client.run('Bot Token')
```

Рисунок 3.6 – Початкові налаштування

Рядок “client.run(‘Bot Token’)” отримує згенерований нами код під час створення бота. У цілях безпеки і захищеності застосунку і проєкту загалом неможливо продемонструвати цей токен, тому замість нього у цьому прикладі і подальшій демонстрації буде вказано “Bot Token”.

Наразі, можна запускати нашого бота. Коли програма скомпілюється і запуститься, наш застосунок з’явиться онлайн на нашому сервері і відправить повідомлення у консоль про готовність. Проте, поки що він нічого не вміє, тому зупиняємо застосунок і переходимо далі.

Наступним кроком є написання інших івентів, що використовуватимуться на сервері. Їх продемонстровано на рисунку 3.7

Розроблено наступні події:

а) подія on_message аналізує відправлені користувачами повідомлення у чат (у разі, якщо будь-яке з них буде містити в собі вказане слово “swearword” – таке повідомлення буде негайно видалене ботом, а сам він відправить у чат відповідь «Ви не можете таке казати»);

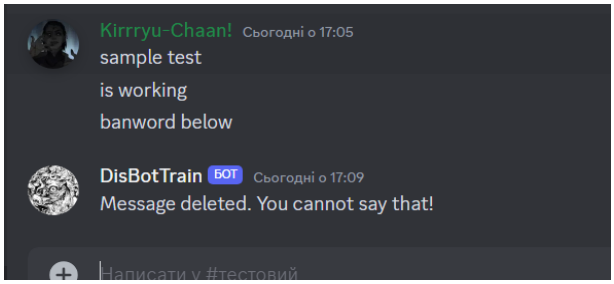
б) подія “on_member_join” спостерігає, коли новий користувач

доєднується до спільноти (коли це відбувається, бот буде вітати цього користувача повідомленням у спеціально відведений чат для цього; також, застосунок відмітить цього користувача через “@”, щоб той точно побачив це повідомлення);

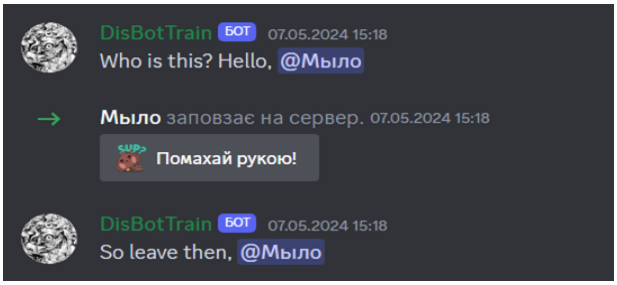
в) подія “on_member_remove” працює аналогічним чином, як і попередня, але її функціонал прощається з користувачем, який покинув спільноту, вийшовши з серверу (результат роботи цих подій наведено на рисунку 3.8).

```
@client.event
async def on_message(message):
   print(message)
   if message.content == "swearword":
      await message.delete()
       await message.channel.send("Message deleted. You cannot say that!")

@client.event
async def on_member_join(member):
    print("someone joined")
    channel = client.get_channel(1179069994559094806)
    await channel.send(f'Who is this? Hello, <@{member.id}>')

@client.event
async def on_member_remove(member):
    print("someone leaved")
    channel = client.get_channel(1179069994559094806)
    await channel.send(f'So leave then, <@{member.id}>')
```

Рисунок 3.7 – Розроблені події



а) видалення повідомлення



б) привітання користувачів

Рисунок 3.8 – Подія “on_member_remove”

3.2.3 Розробка простих команд застосунку

Далі розробимо декілька простих команд (див. рис. 3.9). Їх структура написання нагадує створення подій, але процес роботи таких функцій дещо інший і виконується безпосередньо викликом.

Команди під назвами “hello” та “bye” виконують прості функції: вітання і представлення ботом та прощання.

Команда “kick” дозволяє вигнати користувача з серверу. Цю команду можна виконати лише маючи необхідні права. Також створено функцію, що ловитиме виключення, якщо у користувача не буде прав на виконання цієї команди.

```
@client.command()
async def hello(ctx):
    await ctx.send("Hi there! I'm Maxwell!")

@client.command()
async def bye(ctx):
    await ctx.send("See ya!")

@client.command()
@has_permissions(kick_members=True)
async def kick(ctx, member: discord.Member, *, reason=None):
    await member.kick(reason=reason)
    await ctx.send(f'User {member} has been kicked')

@kick.error
async def kick_error(ctx, error):
    if isinstance(error, commands.MissingPermissions):
        await ctx.send("You don't have permission to do that")
```

Рисунок 3.9 – Прості команди

Ці команди є початковими функціями для тестування і аналізу працездатності бота. Далі вони не будуть нам необхідні, тому не обов’язково їх реалізовувати у кінцевому проєкті. Вони слугували початковими опорами для перевірки роботи застосунку.

3.2.4 Розробка розширених команд застосунку

Для подальшої розробки проєкту, потрібно розширити функціонал і розробити інші команди.

Для розробки команди, що виводить користувачу його профіль, ми використовуємо “embed” повідомлення [5], тобто вбудовані повідомлення.

Embed повідомлення в Discord – це тип повідомлення, який використовується для представлення інформації більш структуровано та візуально привабливіше, порівняно зі звичайними текстовими повідомленнями.

Embed повідомлення включають в себе:

а) заголовок: заголовок embed повідомлення відображається зверху і використовується для опису теми повідомлення;

б) опис: опис embed повідомлення використовується для надання більш детальної інформації про тему повідомлення;

в) поле: поля embed повідомлень використовуються для представлення додаткової інформації, такої як автор, URL-адреса, зображення або файли.

Це основні властивості таких повідомлень, проте вони також можуть містити файли, зображення, відео, посилання, тощо.

Щоб реалізувати демонстрацію профілю, використаємо таке вбудоване повідомлення, його розробку наведено на рисунку 3.10.

Рядок embed.add_field додає поле, в яке передається отримана інформація. Рядок await ctx.send() відправляє збудоване повідомлення у вигляді невеликої інформаційної рамки (див. рис. В.1).

Команда вміє звертатись до даних користувача Discord і виводити отриману інформацію. Ця команда збирає наступну інформацію: Ім’я користувача; аватар користувача; ID користувача; поточний статус; дату приєднання до спільноти а також наявні ролі, які йому були видані.

```
@client.command()
async def profileCOOL(ctx):
    member = ctx.message.author
    roles = [role.name for role in member.roles if role.name != "@everyone"]  #
отримуєм список ролей учасника

    # створення embed повідомлення
    embed = discord.Embed(
        title="Профіль учасника",
        description=f"Профіль {member.mention}",
        color=discord.Color.blue()
    )
    embed.set_author(name=str(member), icon_url=member.avatar_url)
    embed.add_field(name="🆔 ID", value=member.id, inline=True)
    embed.add_field(name="🟢 Статус", value=str(member.status).title(), inline=True)
    embed.add_field(name="📅 Доєднався до серверу",
value=member.joined_at.strftime("%d/%m/%Y, %H:%M:%S"), inline=True)
    embed.add_field(name="🎖️ Ролі", value=", ".join(roles), inline=True)
    embed.set_thumbnail(url=member.avatar_url)

    await ctx.send(embed=embed)  # відправка embed повідомлення
```

Рисунок 3.10 – Embed повідомлення

Як вже зазначалось у попередніх розділах, однією з важливих функцій користування Discord-спільнотою – це здатність приєднуватись до голосових каналів для колективного зібрання. В такому разі, було б доцільно «навчити» бота приєднуватись до саме такого голосового каналу. На рисунку 3.11 зображено перелік команд для взаємодії застосунку з голосовими чатами.

Тепер нам знову знадобиться встановити деякі пакети, а саме: пакет під назвою “FFmpeg”.

FFmpeg – безкоштовний, багатофункціональний програмний пакет з відкритим кодом, який використовується для обробки мультимедійних даних. FFmpeg може бути корисним для широкого кола завдань, пов’язаних з обробкою мультимедійних даних.

Серед його можливостей є:

а) перетворення форматів аудіо та відео: FFmpeg може конвертувати

аудіо та відео з одного формату в інший, наприклад, з MP3 в WAV або з AVI в MP4;

б) обрізка та об’єднання аудіо та відео: FFmpeg може використовуватись для обрізки частин аудіо або відеоматеріалів, а також для об’єднання декількох файлів в один;

в) зміна розміру та роздільної здатності аудіо та відео: FFmpeg може використовуватися для зміни розміру та роздільної здатності аудіо- та відеофайлів;

г) додавання субтитрів та водяних знаків: FFmpeg може використовуватися для додавання субтитрів та водяних знаків до відеофайлів;

д) створення потокового медіа: FFmpeg може використовуватися для створення потокового медіа з аудіо- та відеофайлів;

е) запис екрану: FFmpeg може використовуватися для запису екрану вашого комп’ютера.

Нам потрібна функція д) – «Створення потокового медіа». Таким чином бот отримає «голос» і зможе відтворити аудіофайл. Слово «голос» використано не випадково, саме так називається властивість бота відтворювати звук у середі розробці Discord.

Перед розробкою цих можливостей спочатку треба виконати декілька простих кроків.

Крок 1. Переходимо на сайт ffmpeg за посиланням, наведеним у додатку А. Завантажуємо архів з трьома необхідними нам сторонніми застосунками, щоб змусити бота «говорити».

Крок 2. Розпаковуємо архів з застосунками у зручну тецю.

Крок 3. Відкриваємо меню виконання через “Win+R” і вписуємо “sysdm.cpl”. Ця команда відкриває Властивості системи. Переходимо у «Додатково» і натискаємо «Змінні оточення».

Крок 4. Двічі натискаємо на “Path”. Це відкриває нам можливі шляхи, запам’ятовані системою.

Крок 5. Додаємо новий шлях і вказуємо тецю, куди ми зберегли завантажені ffmpeg застосунки.

```
@client.command(pass_context = True)
async def join(ctx):
    if(ctx.author.voice):
        channel = ctx.message.author.voice.channel
        voice = await channel.connect()
        source = FFmpegPCMAudio('house.mp3')
        player = voice.play(source)
    else:
        await ctx.send("Please join the channel first and repeat.")

@client.command(pass_context = True)
async def pause(ctx):
    voice = discord.utils.get(client.voice_clients, guild=ctx.guild)
    if voice.is_playing():
        voice.pause()
    else:
        await ctx.send("I'm not singing right now")

@client.command(pass_context = True)
async def resume(ctx):
    voice = discord.utils.get(client.voice_clients, guild=ctx.guild)
    if voice.is_paused():
        voice.resume()
    else:
        await ctx.send("I'm not paused right now")

@client.command(pass_context = True)
async def stop(ctx):
    voice = discord.utils.get(client.voice_clients, guild=ctx.guild)
    voice.stop()


@client.command(pass_context = True)
async def leave(ctx):
    if(ctx.voice_client):
        await ctx.voice_client.disconnect()
        await ctx.send("I left the voice channel")
    else:
        await ctx.send("I'm chilling already!")
```

Рисунок 3.11 – Реалізація голосових команд

Головне натиснути «ок» і зберегти наші зміни, бо звичайне закриття цих вікон не зберігає новий шлях. Також, необхідно перезавантажити систему, на якій розроблюється проєкт. Було витрачено чимало часу на з’ясування проблеми непрацюючого пакету ffmpeg. Не повторюйте моїх помилок.

Перевірити чи система дійсно «бачить» цей пакет і може вільно звертатись до нього нам допоможе введення команди “ffmpeg --version” до командного рядку нашої системи.

Тепер, можемо підготувати аудіофайл будь-якого відомого аудіоформату. В моєму випадку, я завантажив файл з мелодією, переніс його до теці де лежить мій Visual Studio проєкт і підключив цей файл до самого проєкту у середовищі розробки.

Відтепер ми можемо використовувати пакет FFmpeg і можемо перейти до продовження розробки.

Спочатку команда join лише приєднувала бота до голосового каналу. Але згодом, її функціонал було трохи перероблено. Тепер ця команда приєднує бота до каналу і програє завантажений аудіофайл.

Хочу звернути увагу, що планування цього боту починалось ще у 2022 – 2023 роках. Під час планування, у мене була ідея розробити музикальний застосунок. Бота, який міг би відтворювати аудіо за посиланням на відомі аудіосервіси, наприклад: YouTube, Spotify, тощо. Але, політика YouTube і схожих сервісів дуже стрімко змінюється і посилюється, і відтепер вони забороняють подібні функції і незаконні дії з контентом своїх медіапросторів. Discord був вимушений піти назустріч, підкоритись політиці цих сервісів і заборонити подібні функції взаємодії з такими онлайн ресурсами. А вже існуючі застосунки-боти піддаються частим блокуванням, як самого боту, так і аккаунту розробника. Через це, щоб уникнути незаконних дій і запобігти бану застосунку і своїх сторінок, я був вимушений відмовитись від подібної ідеї і переробити функціонал під місцевий аудіопрогравач. Тобто, який працюватиме стаціонарно, із попередньо встановленими аудіофайлами.

Отож, розробка команди join.

Ця команда містить в собі умову if, яка визначає чи є користувач зараз у голосовому каналі. Якщо ні – бот сповістить про те, що користувачу необхідно зайти до такого каналу і повторити команду (див. рис. В.2).

У разі, якщо умови відповідають необхідності, бот приєднається до

голосового чату з користувачем, який виконав необхідну команду і відтворить аудіофайл під назвою “house.mp3”.

Наш бот працює і відтворює мелодію у голосовому каналі. Але нам зараз не вистачає функціональності. Можна зробити так, щоб була можливість ставити аудіо на паузу та навпаки, продовжити відтворення.

Для цього було розроблео функції “pause” та “resume” відповідно.

Команда паузи звертається до спеціальних утиліт і також має умовний блок if. Якщо бот відтворює файл – тоді він ставиться на паузу. Якщо бот вже на паузі, він про це сповістить користувача.

Команда повторного відтворення працює схожим чином. Тільки перевіряє чи бот стоїть зараз на паузі. Відтворює звук, якщо відповідь позитивна.

Також присутня команда “stop”. Вона теж звертається до стандартних Discord утиліт, але нічого не перевіряє. Команда зупиняє програвач від відтворення, щоб була можливість програти наступну мелодію.

Наприкінці йде команда “leave”, яка використовує умову. Якщо застосунок не знаходиться в каналі – бот про це сповістить. У разі, якщо він до сих пір перебуває в голосовому чаті, бот буде вимушений полишити канал користувача.

З командами взаємодії застосунку з голосовими каналами ми закінчили, далі будуть продемонстровані інші корисні функції, які бот зможе представити користувачам.

Одна з таких команд це “timer”. Команда повністю відповідає своїй назві і запускає таймер на вказану кількість секунд (див. рис. 3.12).

```
@client.command()
async def timer(ctx, seconds: int):
    await ctx.send(f'{seconds} seconds timer started ⏲')
    await asyncio.sleep(seconds)
    await ctx.send(f'Timer stoped! {seconds} seconds passsed ⏲')
```

Рисунок 3.12 – Команда таймер

Ця команда відправляє у чат повідомлення про початок таймеру. За допомогою бібліотеки asyncio, у нас є можливість почати відлік паралельно з виконанням команди. Коли таймер скінчиться, тобто пройде вказана кількість секунд, застосунок сповістить, що таймер зупинено і час вийшов.

Наступною йде дуже цікава команда, яка може бути актуальною для розробників, підприємців чи інших працівників, що мають справу з валютами і їх курсом (див. рис. 3.13). Ця команда використовує встановлену бібліотеку “requests”, яка дозволяє взаємодіяти з онлайн ресурсами. Для її належної роботи, нам також необхідно виконати декілька простих кроків підготовки.

Крок 1. Встановити бібліотеку requests. Це ми вже зробили аналогічно як встановлювали бібліотеки python і asyncio, через командний рядок на комп’ютері.

Крок 2. Перейти на сайт під назвою “ExchangeRate-API”. Це безкоштовний ресурс, який дозволяє отримати актуальний курс валют через такі бібліотеки, як requests за допомогою спеціального ключа.

Крок 3. Реєструємось на сайті, приймаємо політику сайту і переходимо до персонального кабінету.

Крок 4. Отримуємо індивідуальний API-ключ і зберігаємо його собі.

На цьому нам сайт більше не потрібен, тому можемо його вільно закривати, хоча і рекомендую перед цим зберегти його до своїх закладок.

Можна перейти до розробки команди. Особистий API ключ “api_key” для цієї функції я також не можу показати в цілях безпеки.

Отриманий нами трохи раніше API ключ ми можемо передати у змінну перед самою командою.

Команда складається з самої команди і винесеною за її межі функцією. Команда працює таким чином, що потребує введення валюти. Команда розпізнає три валюти: USD – долар, EUR – євро, GBP(Great Britain Pound) – фунт. Таким чином, щоб вказати валюту яка цікавить, після введення команди !kurs треба вказати і назву валюти з переліку. Якщо просто виконати команду – буде обрана перша валюта за замовчуванням, тобто долар.

```
api_key = "My API key"

async def get_currency_rate(currency):
  # Перевірка правильної назви валюти
  if currency.upper() not in ["USD", "EUR", "GBP"]:
    return f"Невідома валюта: {currency}"

  response = get(f"https://api.exchangerate-
api.com/v4/latest/{currency}?apiKey={api_key}")
  data = response.json()
  # Отримання курсу валюти до UAH
  uah_rate = data["rates"]["UAH"]
  # Заокруглення курсу до 2 десяткових знаків
  uah_rate = round(uah_rate, 2)

  # Повернення тексту з актуальним курсом
  return f"1 {currency} = {uah_rate} UAH"

# Команда для отримання курсу валют
@client.command()
async def kurs(ctx, currency="USD"):
  # Отримання курсу валюти
  currency_rate = await get_currency_rate(currency.upper())

  # Надсилання повідомлення з курсом (якщо валюта вірна)
  if currency_rate != "Unknown currency:":
    await ctx.send(currency_rate)
  else:
    await ctx.send(f"Unknown currency: {currency}")
```

Рисунок 3.13 – Команда актуалізації курсу валют

Як результат цієї команди, бот відправляє у чат відповідь, в якій продемонстровано відношення гривні і обраної валюти.

Приклад результату: 1 USD = 39.39 UAH (див. рис. В.3).

Якщо в команді вказана валюта, яку застосунку не вдалося розпізнати, він поверне повідомлення, що валюта невірна. Це перевіряє функція за межами самої команди.

Потім команда встановлює зв’язок з сайтом за посиланням і нашому API ключу і записує отримані дані. Далі відбувається округлення значення до двох знаків після крапки для скорочення отриманого результату. Це візуально покращить і спростить сприйняття користувачем отриманої інформації.

Далі командою повертається результат і відправляється у чат як відповідь.

ВИСНОВКИ

У кваліфікаційній роботі представлено розробку функціонального бота для Discord з багатьма функціями, що робить його важливим інструментом для модерації, інформаційної підтримки та автоматизації завдань на Discord-серверах. Розробка відбувалась за допомогою загальнодоступних в інтернеті бібліотек на середовищі Visual Studio. Мова програмування Python значно пришвидшила та спростила реалізацію цього проєкту. Реалізація проєкту на мові Python представляє широкий функціонал можливостей, що і дозволило розробити застосунок у повному обсязі.

Розроблений бот отримав функціонал, який йому планувалось встановити. Через зміну політики користування YouTube і схожих сервісів, змін зазнала лише одна функція з програвання аудіо за посиланням. Проте, не зважаючи на цей момент, бот не втратив своїх можливостей і впевнено продемонстрував працездатність цієї і інших представлених функцій. Розширені команди управління ботом, такі як відтворення музики, модерація користувачів та інтеграція з зовнішніми API, були успішно реалізовані. Це дало можливість створити бота, функціонал якого дозволяє виконувати кілька завдань на Discord-сервері.

На основі зворотного зв’язку від користувачів, які тестували бота, було виявлено задовільний рівень користування його функціоналом та можливостями. Це підтверджує ефективність проведених робіт та правильність обраних технічних рішень.

Також, під час реалізації проєкту було створено детальний і зрозумілий опис підготовки до розробки застосунку. Сама реалізація аналогічним чином описана у доступній і зрозумілій формі. Описано і помилки, з якими можуть стикнутися розробники, що спростить процес реалізації у майбутньому і запобігатиме повторним виникненням подібних помилок і складнощів. Детальна документація дозволить легко налаштувати та використовувати

бота. Вона забезпечить можливу підтримку проєкту та сприятиме його подальшому розвитку.

Загалом, проєкт з розробки функціонального бота для мережі Discord виявився успішним. Результати підтверджують можливість створення складних інтеграційних рішень, які використовують сучасні інструменти та технології. Досвід може бути використаний для нових проєктів автоматизації та інтеграції в Discord та інших платформах, а також для подальших розробок і покращень.

ПЕРЕЛІК ПОСИЛАНЬ

1. Chollet F. Deep Learning with Python : first ed. Manning, 2017. 384 p. URL: https://tanthiamhuat.wordpress.com/wp-content/uploads/2018/03/deeplearningwithpython.pdf (дата звернення: 08.04.2024).
2. Jones B., Beazley D. Python Cookbook: Recipes for Mastering Python 3 : 3rd ed. O'Reilly Media, 2013. 708 p. URL: https://paper.bobylive.com/Programming/python-cookbook-third-edition.pdf (дата звернення: 06.04.2024).
3. Ramalho L. Fluent Python. 2015. 979 p. URL: https://elmoukrie.com/wp-content/uploads/2022/05/luciano-ramalho-fluent-python_-clear-concise-and-effective-programming-oreilly-media-2022.pdf (дата звернення: 10.04.2024).
4. Slatkin B. Effective Python: 59 Specific Ways to Write Better Python. United States, RR Donnelley, Crawfordsville, Indiana, 2015. 680 p.
5. Summerfield M. Programming in Python 3: A Complete Introduction to the Python language. 2008. 636 p.
6. Sweigart A. Automate the Boring Stuff with Python: Practical Programming for Total Beginners. 2015. 505 p.
7. Python. URL: https://www.python.org/ (дата звернення: 07.04.2024).
8. Що таке бот. URL: http://surl.li/sqiun (дата звернення: 08.04.2024).
9. Що таке застосунок. URL: http://surl.li/afoaz (дата звернення: 08.04.2024).
10. Що таке мова програмування. URL: http://surl.li/tuaz (дата звернення: 09.04.2024).
11. Gautam H., Agnihotri A. Discord bot introduction. URL: http://www.ir.juit.ac.in:8080/jspui/bitstream/123456789/3628/1/Discord%20Bot.pdf (дата звернення: 03.04.2024).
12. Cahyani N. D. W., Pratama D. S., Rahman H. N. A. Proactive Acquisition using Bot on Discord. *International Journal of Advanced Computer Science and*

Applications. 2023. Vol. 14, No. 5. P. 315–325. DOI: 10.14569/IJACSA.2023.0140533.

13. Verma A., Tyagi S., Mathur G. A Comprehensive Review on Bot – Discord Bot. *International Journal of Scientific Research in Computer Science, Engineering and Information Technology*. 2021. Vol. 7, Iss. 2. P. 532‑536. URL: https://doi.org/10.32628/CSEIT2172100 (дата звернення: 03.04.2024).
14. How to make a Discord bot. URL: https://www.ionos.com/digitalguide/server/know-how/creating-discord-bot/ (дата звернення: 02.04.2024).
15. Ronquillo A. How to Make a Discord Bot in Python. URL: https://realpython.com/how-to-make-a-discord-bot-python/ (дата звернення: 02.04.2024).
16. I’ve Read Over 100 Books on Python. Here are the Top 3. YT Channel: Python Programmer. URL: https://www.youtube.com/watch?v=MqywbqLmjp4 (дата звернення: 01.04.2024).
17. 3 BEST Python Books for Learning Python 2024 (from Beginner to Pro). YT Channel: Daniel Dan. URL: https://www.youtube.com/watch?v=hD6UVZ3nm-U (дата звернення: 01.04.2024).

ДОДАТОК А

Корисні сервіси та посилання

Сайт з генерування API ключа для актуалізації курсу валют. URL: https://www.exchangerate-api.com/ (дата звернення: 08.05.2024).

Сайт, де можна завантажити пакет застосунків FFmpeg. URL: https://ffmpeg.org/download.html (дата звернення: 09.05.2024).

Сайт для розробників в середовищі Discord. URL: https://discord.com/developers/docs/intro (дата звернення: 02.04.2024).

Сайт гуглу на основі ШІ як інструмент пошуку документації і матеріалу: Google Gemini. URL: https://gemini.google.com/app?hl=uk (дата звернення: 02.05.2024).

Сайт для скорочення посилань. URL: https://hyperhost.ua/tools/ru/surli (дата звернення: 18.05.2024).

Сайт офіційної підтримки дискорд. URL: https://support.discord.com/hc/en-us (дата звернення: 08.05.2024).

ДОДАТОК Б

Код розробленого застосунку

```
import discord
import random
import time
import asyncio
import requests
import schedule
from discord import FFmpegPCMAudio
from requests import get
from discord import Member
from discord.ext.commands import has_permissions, MissingPermissions
from discord.ext import commands

intents = discord.Intents.all()

client = commands.Bot(command_prefix = '!',intents=intents)
@client.event
async def on_ready():
    await client.change_presence(status=discord.Status.idle,
activity=discord.Game('Minecraft'))
    print("Hello, i'm ready!")
    print("--------------------------")

#@client.event
#async def on_message(message):
#   print(message)
#   if message.content == "swearword":
#      await message.delete()
#       await message.channel.send("Message deleted. You cannot say that!")

@client.event
async def on_member_join(member):
    print("someone joined")
    channel = client.get_channel(1179069994559094806)
    await channel.send(f'Who is this? Hello, <@{member.id}>')

@client.event
async def on_member_remove(member):
    print("someone leaved")
    channel = client.get_channel(1179069994559094806)
    await channel.send(f'So leave then, <@{member.id}>')


#-------------------------BOT COMMANDS BELOW-----------------------------

@client.command()
async def hello(ctx):
    await ctx.send("Hi there! I'm Maxwell!")

@client.command()
async def bye(ctx):
    await ctx.send("See ya!")

@client.command()
@has_permissions(kick_members=True)
async def kick(ctx, member: discord.Member, *, reason=None):
    await member.kick(reason=reason)
```

```
    await ctx.send(f'User {member} has been kicked')

@kick.error
async def kick_error(ctx, error):
    if isinstance(error, commands.MissingPermissions):
        await ctx.send("You don't have permission to do that")

@client.command()
async def menu(ctx):
    embed = discord.Embed(title="Commands list", description="Here are some commands for
you:", color=0x4dff4d)
    embed.add_field(name="!hello", value="Says hello", inline=True)
    embed.add_field(name="!bye", value="Says Goodbye", inline=True)
    embed.add_field(name="!profileCOOL", value="Shows your profile info", inline=True)
    embed.add_field(name="!join", value="Joins the bot to channel and plays music",
inline=True)
    embed.add_field(name="!timer", value="Sets the timer for a entered seconds. Exmp:
!timer 4", inline=True)
    embed.add_field(name="!kurs", value="Shows the current exchange rate for USD, EUR,
GBP", inline=True)
    embed.set_footer(text="Try yourself")

    await ctx.send(embed=embed)

@client.command()
async def profileCOOL(ctx):
    member = ctx.message.author
    roles = [role.name for role in member.roles if role.name != "@everyone"]

    embed = discord.Embed(
        title="Профіль учасника",
        description=f"Профіль {member.mention}",
        color=discord.Color.blue()
    )
    embed.set_author(name=str(member), icon_url=member.avatar_url)
    embed.add_field(name="🆔 ID", value=member.id, inline=True)
    embed.add_field(name="🟢 Статус", value=str(member.status).title(), inline=True)
    embed.add_field(name="📅 Доєднався до серверу",
value=member.joined_at.strftime("%d/%m/%Y, %H:%M:%S"), inline=True)
    embed.add_field(name="🎖️ Ролі", value=", ".join(roles), inline=True)
    embed.set_thumbnail(url=member.avatar_url)

    await ctx.send(embed=embed)

@client.command(pass_context = True)
async def join(ctx):
    if(ctx.author.voice):
        channel = ctx.message.author.voice.channel
        voice = await channel.connect()
        source = FFmpegPCMAudio('house.mp3')
        player = voice.play(source)
    else:
        await ctx.send("Please join the channel first and repeat.")

@client.command(pass_context = True)
async def pause(ctx):
    voice = discord.utils.get(client.voice_clients, guild=ctx.guild)
    if voice.is_playing():
        voice.pause()
    else:
        await ctx.send("I'm not singing right now")

@client.command(pass_context = True)
async def resume(ctx):
```

```
    voice = discord.utils.get(client.voice_clients, guild=ctx.guild)
    if voice.is_paused():
        voice.resume()
    else:
        await ctx.send("I'm not paused right now")

@client.command(pass_context = True)
async def stop(ctx):
    voice = discord.utils.get(client.voice_clients, guild=ctx.guild)
    voice.stop()


@client.command(pass_context = True)
async def leave(ctx):
    if(ctx.voice_client):
        await ctx.voice_client.disconnect()
        await ctx.send("I left the voice channel")
    else:
        await ctx.send("I'm chilling already!")

@client.command()
async def timer(ctx, seconds: int):
    await ctx.send(f'{seconds} seconds timer started ⏲')
    await asyncio.sleep(seconds)
    await ctx.send(f'Timer stoped! {seconds} seconds passsed ⏲')


api_key = "My API Key"

#client = discord.Client()
async def get_currency_rate(currency):
  if currency.upper() not in ["USD", "EUR", "GBP"]:
    return f"Невідома валюта: {currency}"

  response = get(f"https://api.exchangerate-
api.com/v4/latest/{currency}?apiKey={api_key}")
  data = response.json()

  uah_rate = data["rates"]["UAH"]
  uah_rate = round(uah_rate, 2)
  return f"1 {currency} = {uah_rate} UAH"

@client.command()
async def kurs(ctx, currency="USD"):
  currency_rate = await get_currency_rate(currency.upper())

  if currency_rate != "Unknown currency:":
    await ctx.send(currency_rate)
  else:
    await ctx.send(f"Unknown currency: {currency}")



#-------------------------BOT TOKEN BELOW-----------------------------

client.run('My Bot Token')
```

ДОДАТОК В

Скриншоти роботи розробленого застосунку

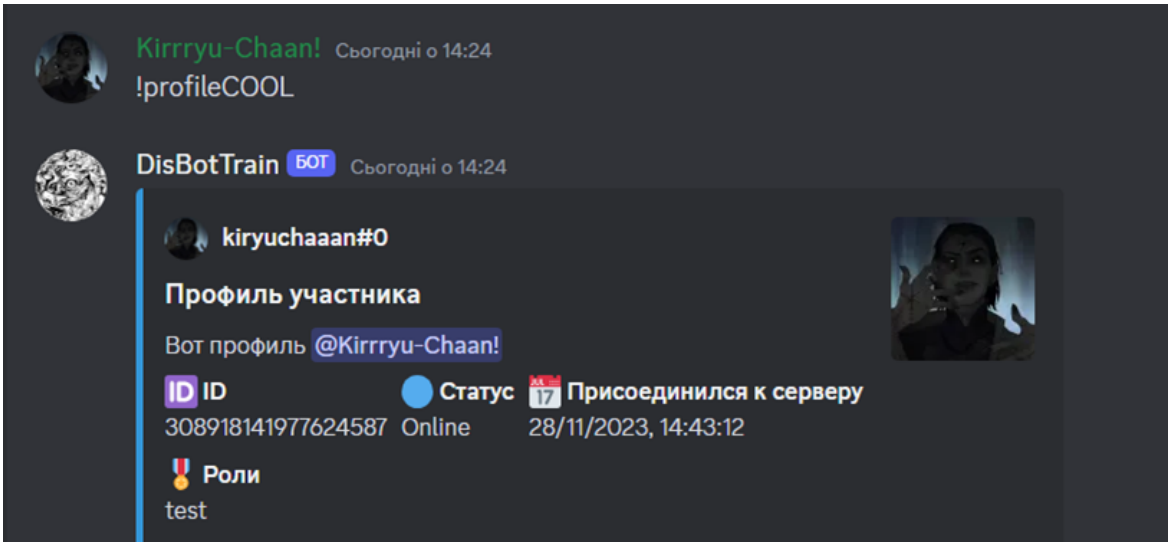


Рисунок В.1 – Embed повідомлення з профілем користувача

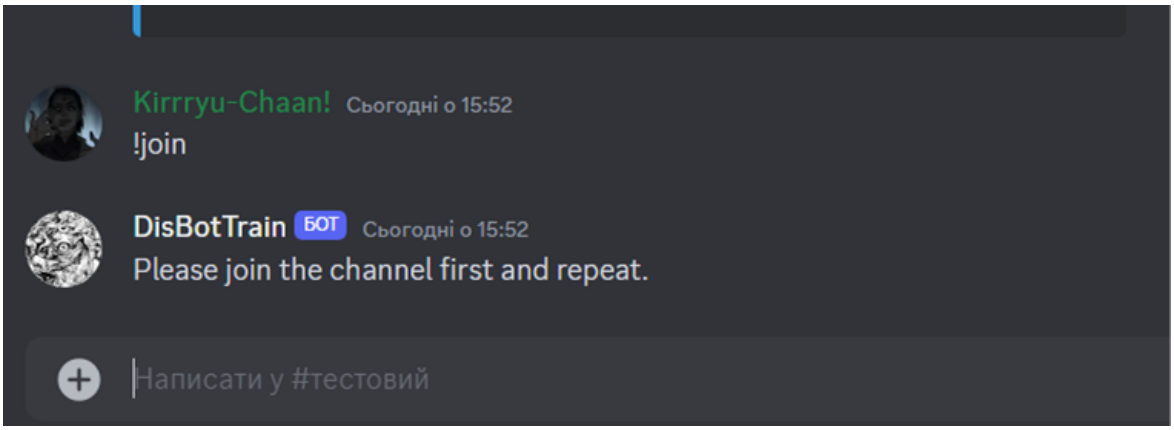


Рисунок В.2 – Попередження про невиконання умов

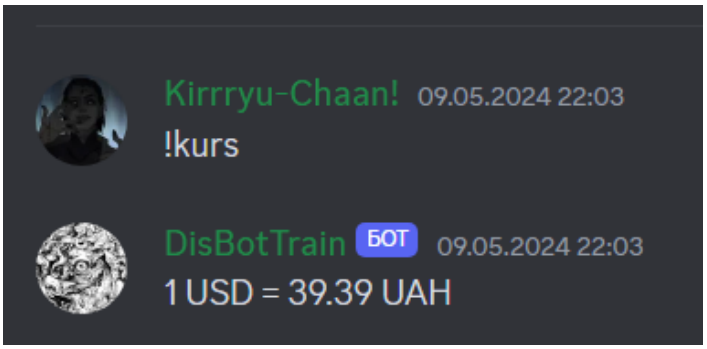


Рисунок В.3 – Демонстрація команди з курсом валют

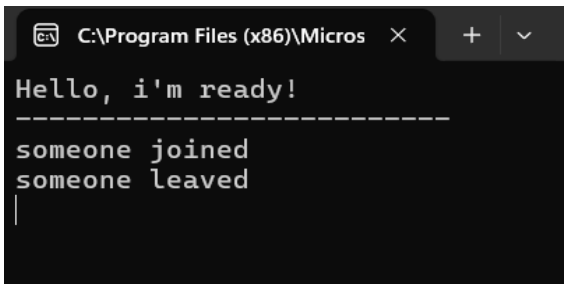


Рисунок В.4 – Звіт нових користувачів у терміналі розробника

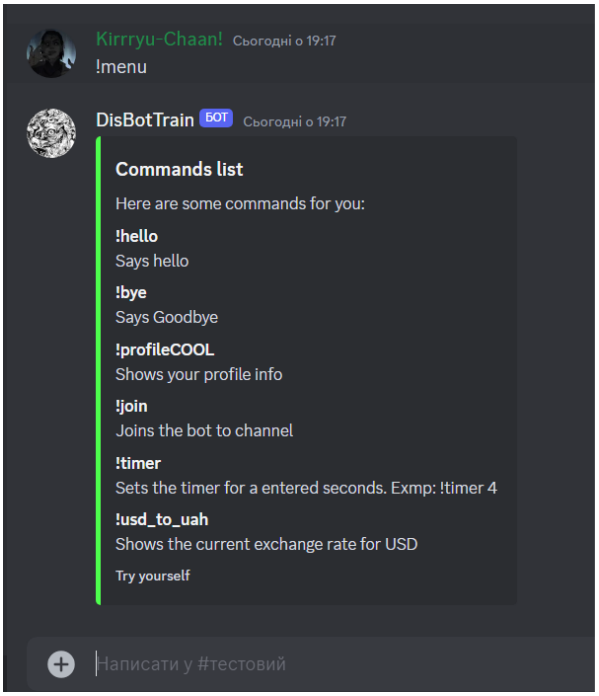


Рисунок В.5 – Команда-перелік можливих функцій

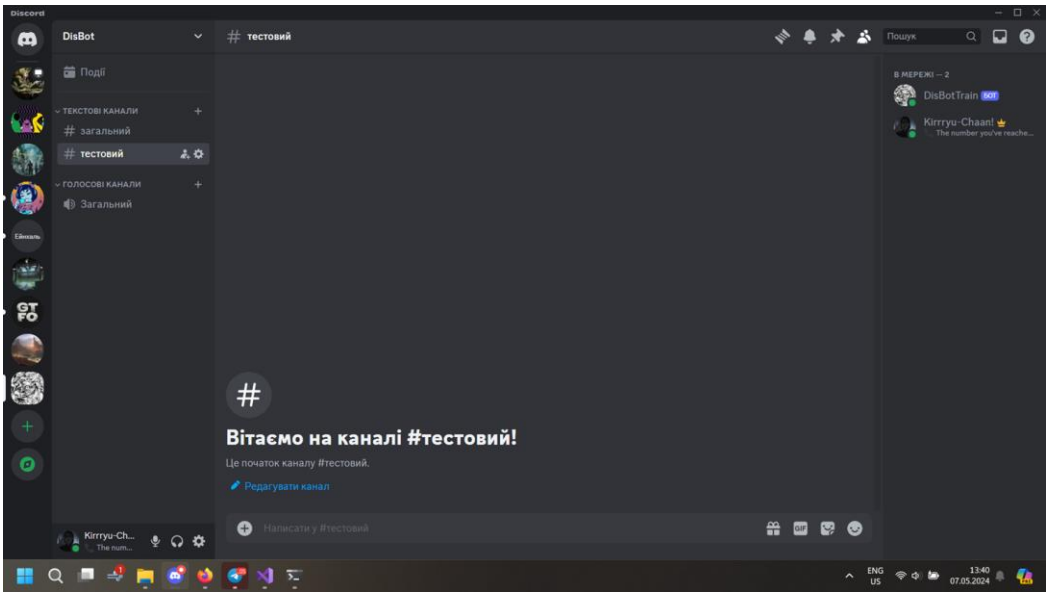


Рисунок В.6 – Готова спільнота з ботом в стані «онлайн»