МІНІСТЕРСТВО ФІНАНСІВ УКРАЇНИ
ДЕРЖАВНИЙ ПОДАТКОВИЙ УНІВЕРСИТЕТ

ФАКУЛЬТЕТ ФІНАНСІВ ТА ЦИФРОВИХ ТЕХНОЛОГІЙ
КАФЕДРА КОМП'ЮТЕРНИХ ТА ІНФОРМАЦІЙНИХТЕХНОЛОГІЙ І СИСТЕМ

КВАЛІФІКАЦІЙНА РОБОТА

на тему: РОЗРОБКА DISCORD-БОТА

Здобувача вищої освіти (бакалаврського) рівня,
IV курсу, групи КБ-21-1, Спеціальності 122 «Комп'ютерні науки»,
освітня програма «Інформаційні управляючі системи і технології (за галузями)»

(підпис) Лисенка Ярослава Сергійовича
(прізвище, ім'я, по батькові)

Науковий керівник (підпис) Гладченко Оксана Вікторівна, к.пед.н., доцент
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

Керівник освітньо-професійної програми (підпис) Омельчук Антон Анатолійович, к.т.н.
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

Допущено до захисту за рішенням кафедри комп'ютерних та інформаційних технологій і систем протокол від «13» травня 2025 № 18

Завідувач кафедри (підпис) Філоненко Михайло Миколайович, к.ф.-м.н., доцент
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

Ірпінь – 2025

Реєстрація №60 26.05.2025 (дата) (підпис відповідальної особи)

Результати перевірки до захисту (до захисту, на доопрацювання) 26.05.2025 (дата) (підпис)

Результат захисту 5 (відмінно) (оцінка) 05.06.2025 (дата) (підпис)

Голова комісії: (підпис) В.М. Франчук

Члени комісії: (підпис) М.М. Філоненко

(підпис) С.М. Грищенко

(підпис) А.А. Омельчук

БЛАНК ОЦІНЮВАННЯ КВАЛІФІКАЦІЙНОЇ РОБОТИ

Об'єкти контролю	Діапазон оцінки в балах	Фактична кількість балів	Примітки	Дата, П.І.Б.: керівника, голови комісіїчленів комісії, секретаря, (підписи)
ОЦІНЮВАННЯ КВАЛІФІКАЦІЙНОЇ РОБОТИ КОМІСІЄЮ ПО ЗАХИСТУ				
1. Зміст випускної роботи (якість теоретичного матеріалу, його зв'язок з практичним, вміння представляти матеріал в логічній послідовності, наявність висновків та пропозицій)	**від 0 до 30**	29		Гладченко О.В. Франчук В.М. Філоненко М.М. Грищенко С.М. Омельчук А.А. Суховецька Л.П.
2. Дотримання вимог щодо оформлення	**від 0 до 10**	10		
3. Вчасне виконання роботи на всіх етапах у встановлені кафедрою терміни	**від 0 до 5**	5		
4. Рецензент	**від 0 до 5**	4		
РАЗОМ БАЛІВ	**від 0 до 50**	48		

Оцінювання кваліфікаційної роботи науковим керівником – 44 **балів.**

ПЕРЕВЕДЕННЯ ОЦІНКИ ЗА 100-БАЛЬНОЮ ШКАЛОЮ ДО НАЦІОНАЛЬНОЇ ШКАЛИ ТА ШКАЛИ ЄКТС

Оцінка за 100-бальною шкалою	Оцінка за шкалоюЄКТС	Оцінка за національною шкалою	Дата	П.І.Б.: голови, членів комісії, секретаря, (підпис)
92	A	5 (відмінно)	05.06.2025	Франчук В.М. Філоненко М.М. Грищенко С.М. Омельчук А.А. Суховецька Л.П.

ПРАВИЛА ПЕРЕВЕДЕННЯ ОЦІНОК ДО НАЦІОНАЛЬНОЇ ШКАЛИ ТА ШКАЛИ ЄКТС

Оцінка за 100-бальною шкалою	Оцінка за шкалою ЄКТС	Оцінка за національною шкалою
90-100	A	5 (відмінно)
80-89	B	4 (добре)
70-79	C	
60-69	D	3 (задовільно)
50-59	E	
35-49	FX	2 (незадовільно)

Факультет фінансів та цифрових технологій
Кафедра комп'ютерних та інформаційних технологій і систем
Галузь знань: 12 «Інформаційні технології»
Спеціальність: 122 «Комп'ютерні науки»
Освітня програма: «Інформаційні управляючі системи і технології (за галузями)»

Затверджую
Завідувач кафедри _____ Філоненко М.М.
«14» листопада 2024 р.

Завдання на кваліфікаційну роботу
здобувачу вищої освіти
Лисенку Ярославу Сергійовичу
(прізвище, ім'я, по батькові)

1. Тема кваліфікаційної роботи *Розробка Discord-бота.*
Затверджена наказом ректора від «13» листопада 2024 № 17.
2. Строк здачі закінченої роботи «08» травня 2025.
3. Мета роботи *розробка функціонального Discord-бота, здатного автоматизувати ключові процеси, забезпечувати інтерактивну взаємодію та виконувати завдання, що відповідають потребам користувачів.*
Об'єкт дослідження *автоматизація процесів комунікації та взаємодії у середовищі Discord за допомогою інтерактивних ботів.*
Предмет дослідження *розробка алгоритмів та функціональності для Discord-бота.*
Орієнтовна база дослідження *програмна розробка на базі платформи Discord API.*
4. Вихідні дані до роботи *методи та підходи до розробки та створення Discord-бота.*
5. Перелік графічного матеріалу *інтерфейсні елементи, зображення для порівняння, графічне представлення коду.*
6. Зміст кваліфікаційної роботи (перелік питань за кожним розділом)
ВСТУП
РОЗДІЛ 1. ОСОБЛИВОСТІ ЧАТ-БОТІВ В DISCORD
1.1. Аналіз наукової літератури та теоретичне обґрунтування проблеми
1.2. Discord та його можливості
1.3. Discord-боти
1.4. Інструментарій. Мова програмування Python
РОЗДІЛ 2. СИСТЕМНО-АНАЛІТИЧНИЙ АНАЛІЗ ТА ПРОЄКТУВАННЯ
2.1. Обґрунтування вибору технологій для розробки Discord-бота
2.2. Архітектура системи та її компоненти
2.3. Опис алгоритмів та основних модулів
2.4. Моделювання даних
РОЗДІЛ 3. ПРАКТИЧНА РЕАЛІЗАЦІЯ ПРОЄКТУ DISCORD-БОТ
3.1. Вибір інструментів і підготовка середовища
3.2. Реалізація функціональності бота
3.3. Тестування та підсумкова оптимізація
ВИСНОВКИ
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ
ДОДАТКИ

7. Календарний план виконання роботи

№ п/п	Назва етапів кваліфікаційноїроботи	Строк виконання етапів роботи	
		За планом	Фактично
1	2	3	4
1	Вибір теми, підбір літературних джерел, формування структури кваліфікаційної роботи	03.11.024	30.10.2024
2	**І розділ** (розкрити сутність і стан дослідження наукової проблематики роботи, після чого навести її загальну характеристику; огляд літератури за темою і вибір конкретних напрямів досліджень)	27.12.2024	27.12.2024
3	**ІІ розділ** (аналіз експериментальної частини про методики досліджень)	21.02.2025	21.02.2025
4	**ІІІ розділ** (проведені теоретичні та (або) експериментальні дослідження та проєктні рішення)	20.04.2025	20.04.2025
5	**Висновки**	01.05.2025	01.05.2025
6	Доопрацювання роботи з урахуванням зауважень наукового керівника	10.05.2025	10.05.2025
7	Отримання відгуку керівника	27.05.2025	27.05.2025
8	Отримання зовнішньої рецензії	29.05.2025	29.05.2025

8. Дата видачі завдання «14» листопада 2024

9. Керівник кваліфікаційної роботи _____ (підпис) Гладченко О.В (прізвище, ініціали)

10. Керівник освітньо-професійної програми _____ (підпис) Омельчук А.А. (прізвище, ініціали)

11. Завдання до виконання отримав здобувач вищої освіти _____ (підпис) Лисенко Я.С. (прізвище, ініціали)

АНОТАЦІЯ

Розробка Discord-бота

Мета роботи розробити функціонального Discord-бота для оптимізації завдань серверів, забезпечення інтерактивної взаємодії між користувачами та підвищення ефективності адміністрування.

Завданнями роботи є:

- Проведення аналізу платформи Discord та її можливостей.
- Дослідження доступних бібліотек для розробки ботів.
- Розробка та впровадження унікального функціоналу.
- Інтеграція сучасних API для розширення можливостей бота.
- Тестування й оптимізація продуктивності бота.

Предметом дослідження є розробка алгоритмів та функціональності для Discord-бота.

Об'єктом дослідження є автоматизація процесів взаємодії у спільнотах Discord.

Базою дослідження виступає програмна розробка на базі платформи Discord API.

За результатами дослідження сформульовані рекомендації щодо автоматизації адміністрування серверів Discord і покращення користувацького досвіду за допомогою інтерактивних ботів.

Одержані результати можуть бути використані для адміністрування великих спільнот, підвищення активності користувачів та інтеграції зовнішніх сервісів.

Кваліфікаційна робота містить 51 сторінку, 17 рисунків, 4 таблиці, список використаних джерел 40 найменування, 5 додатків.

Ключові слова: Discord, бот, автоматизація, API Discord, інтерактивність, програмування, модерація, управління ролями, спільнота, інтеграція API.

Рік виконання кваліфікаційної роботи 2025.

Рік захисту роботи 2025.

ABSTRACT

Development of a Discord Bot

The objective of this study is to create a functional Discord bot to optimize server tasks, ensure interactive user engagement, and enhance administrative efficiency.

The tasks of this study include:

- Analyzing the Discord platform and its capabilities.
- Researching available libraries for bot development.
- Developing and implementing unique functionalities.
- Integrating modern APIs to expand the bot's capabilities.
- Testing and optimizing the bot's performance.

The subject of the research is the development of algorithms and functionalities for a Discord bot.

The object of the research is the automation of interaction processes within Discord communities.

The foundation of the study is software development based on the Discord API platform.

As a result of this research, recommendations for automating server administration in Discord and improving user experiences through interactive bots were formulated.

The obtained results can be used for managing large communities, increasing user activity, and integrating external services.

The qualification work contains 51 pages, 17 figures, 4 tables, a list of sources comprising 40 entries, 5 applications.

Keywords: Discord, bot, automation, Discord API, interactivity, programming, moderation, role management, community, API integration.

Year of project completion: 2025.

Year of thesis defense: 2025.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ ТА СКОРОЧЕНЬ 3
ВСТУП 4
РОЗДІЛ 1. ОСОБЛИВОСТІ ЧАТ-БОТІВ В DISCORD 7
1.1. Аналіз наукової літератури та теоретичне обґрунтування проблеми 7
1.2. Discord та його можливості 9
1.3. Discord-боти 13
1.4. Інструментарій. Мова програмування Python 16
РОЗДІЛ 2. СИСТЕМНО-АНАЛІТИЧНИЙ АНАЛІЗ ТА ПРОЄКТУВАННЯ 21
2.1. Обґрунтування вибору технологій для розробки Discord-бота 21
2.2. Архітектура системи та її компоненти 24
2.3. Опис алгоритмів та основних модулів 28
2.4. Моделювання даних 30
РОЗДІЛ 3. ПРАКТИЧНА РЕАЛІЗАЦІЯ ПРОЄКТУ DISCORD-БОТ 33
3.1. Вибір інструментів і підготовка середовища 33
3.2. Реалізація функціональності бота 36
3.3. Тестування та підсумкова оптимізація 38
ВИСНОВКИ 47
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ 49
ДОДАТКИ 52

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ ТА СКОРОЧЕНЬ

API – application programming interface (інтерфейс прикладного програмування).

JSON – JavaScript Object Notation (формат структурованого зберігання даних).

SQLite – легка система управління базами даних (СУБД).

ORM – Object-Relational Mapping (об’єктно-реляційне відображення).

HTTP – HyperText Transfer Protocol (протокол передачі гіпертексту).

FFmpeg – мультимедійний фреймворк для обробки аудіо та відео.

AI – Artificial Intelligence (штучний інтелект).

Asyncio – стандартна бібліотека Python для асинхронного програмування.

SQLAlchemy – бібліотека Python для роботи з базами даних через ORM.

ВСТУП

У сучасному світі, де цифрове спілкування стало важливою частиною нашого повсякденного життя, виникає потреба в автоматизації різних процесів. Боти для Discord допомагають автоматизувати модерацію, підтримку серверів, управління ролями, створення опитувань та інші взаємодії з учасниками. Це дозволяє зекономити час і зробити управління спільнотами значно зручнішим.

Актуальність теми. Discord сьогодні є однією з найпопулярніших платформ для спілкування, як голосового, так і текстового. Ще у 2019 році кількість зареєстрованих користувачів перевищила 250 мільйонів, а активних користувачів щомісяця було близько 56 мільйонів. До 2021 року ця цифра зросла до понад 536 мільйонів акаунтів, і місячна активність досягла 196 мільйонів користувачів. Discord активно використовується для створення тематичних спільнот, де люди об’єднуються за інтересами – від геймінгу до навчання і роботи. Тому розробка ботів для Discord є актуальною і корисною темою, яка має практичне значення і перспективи для розвитку в майбутньому.

Стан і ступінь розробки проблеми в спеціальній літературі. Проблематика створення чат-ботів для цифрових платформ, зокрема Discord, активно висвітлюється у вітчизняній та закордонній літературі. Серед українських дослідників варто відзначити Павлова С.М., Коваля О.В., Петренка В.І., які у своїх роботах розглядали застосування ботів для підтримки онлайн-взаємодії. Із закордонних фахівців варто згадати Брайана Кентвелла Сміта, який досліджує інтерактивних агентів у цифрових спільнотах, Джона-Патріка Аллема – з теми соціального впливу ботів, та Алана Джонсона, який аналізує баланс між автоматизацією та живою взаємодією.

Метою роботи є розробити функціонального Discord-бота, здатного автоматизувати ключові процеси, забезпечувати інтерактивну взаємодію та виконувати завдання, що відповідають потребам користувачів.

Завданнями роботи є:

- Проведення аналізу платформи Discord та її можливостей.

- Дослідження доступних бібліотек для розробки ботів.
- Розробка та впровадження унікального функціоналу.
- Інтеграція сучасних API для розширення можливостей бота.
- Тестування й оптимізація продуктивності бота.

Об'єкт дослідження автоматизація процесів комунікації та взаємодії у середовищі Discord за допомогою інтерактивних ботів.

Предметом дослідження є розробка алгоритмів та функціональності для Discord-бота.

Методи дослідження полягають в аналізі наукової та технічної літератури для обґрунтування вибору інструментів, порівняльний аналіз бібліотек і технологій для створення Discord-ботів, моделювання архітектури системи та структури даних, а також методи прототипування і реалізації програмного продукту з подальшим емпіричним тестуванням у середовищі Discord для перевірки функціональності, стабільності та інтерактивності бота.

Базою дослідження виступає програмна розробка на базі платформи Discord API.

Наукова та практична значущість полягає в розробленому Discord-боті, який демонструє можливості автоматизації процесів у межах популярної платформи для комунікації. Бот реалізує функціонал модерації, відтворення музики, взаємодії з користувачами, обробки повідомлень і інтеграції з зовнішніми сервісами, що може бути адаптовано для різних сфер – від освітніх до комерційних.

Наукова новизна здобувача вищої освіти полягає у створенні комплексного Discord-бота з багатофункціональним набором можливостей, що поєднує модерацію, відтворення музики, обробку тексту, взаємодію з користувачами та інтеграцію з зовнішніми API (YouTube, OpenAI, Telegram тощо). На відміну від стандартних реалізацій, запропонований бот використовує комбінацію асинхронного підходу, інтелектуального аналізу повідомлень (через модель Gemini) та збереження історії взаємодії, що забезпечує адаптивність та масштабованість.

Особистий внесок здобувача вищої освіти полягає в здійсненні повного циклу розробки Discord-бота: від аналізу вимог і проєктування архітектури до реалізації, тестування та розгортання програмного продукту. Було створено унікальний програмний код з підтримкою асинхронної обробки подій, реалізовано функції модерації, програвання музики, аналізу тексту за допомогою зовнішніх API, обробки текстових файлів, а також збереження історії взаємодії з користувачами. Особисто виконано інтеграцію з такими сервісами, як YouTube, Gemini API, а також реалізовано систему збереження даних у форматі JSON із забезпеченням структурованого доступу.

РОЗДІЛ 1. ОСОБЛИВОСТІ ЧАТ-БОТІВ В DISCORD

1.1. Аналіз наукової літератури та теоретичне обґрунтування проблеми

Автоматизація комунікації за допомогою ботів стала важливою частиною сучасних цифрових платформ, особливо таких, як Discord. Впровадження ботів дозволяє не тільки модерувати та керувати спільнотами, але й інтегрувати зовнішні сервіси, спрощуючи доступ до інформації. У цьому підрозділі розглядаються основні теоретичні засади функціонування ботів, їхнє місце в сучасній цифровій комунікації та особливості їхньої реалізації на платформі Discord. Згідно зі стандартом ISO/IEC 2382:2015, інформаційні технології охоплюють концепції та терміни, що використовуються в автоматизованій обробці даних [15, 34].

У сучасному цифровому середовищі боти не лише виконують технічні завдання, а й впливають на характер онлайн-взаємодій. Вони здатні обробляти великі обсяги даних, підтримувати користувачів у процесі комунікації та забезпечувати миттєвий доступ до необхідної інформації. Це значно змінює механізми організації спільнот, дозволяючи адміністраторам зосередитися на стратегічному управлінні, а не на рутинних завданнях.

Окрім управління спільнотами, боти також відіграють ключову роль у підтримці безпеки. Завдяки автоматичному аналізу повідомлень та розпізнаванню потенційно шкідливого контенту, вони допомагають запобігати порушенням правил та підтримувати здорову атмосферу на платформах. Такі функції особливо цінні для великих серверів, де модерація вручну стає надзвичайно складною.

З погляду інтеграції, чат-боти в Discord можна запрограмувати на виконання різноманітних функцій: від автоматичної відповіді на запити користувачів до складних алгоритмів модерації та аналізу емоційного забарвлення повідомлень. Це відкриває значні можливості для використання ботів у бізнесі, освіті та інших сферах.

Розвиток ботів для платформ обміну повідомленнями є загальною тенденцією в сучасній IT-індустрії, що не обмежується лише Discord. Подібно до нього, інші популярні сервіси, такі як Slack, що надає власну документацію для розробників API [28], та Telegram з його детальним описом Telegram Bot API [30], також активно підтримують створення автоматизованих помічників. Це свідчить про високий попит на інструменти автоматизації та розширення функціональності комунікаційних платформ.

У світовій науковій літературі значна увага приділяється автоматизованим агентам у соціальних платформах. Дослідження Брайана Кентвелла Сміта показують, що інтерактивні боти можуть значно покращити модерацію серверів, знижуючи навантаження на адміністраторів. Водночас Джонсон А. наголошує на важливості збереження балансу між автоматизацією та живою взаємодією користувачів.

Зі зростанням популярності чат-ботів важливо враховувати не лише їхню технічну ефективність, а й соціальні наслідки їх використання. Автоматизовані системи можуть змінювати характер комунікації, впливаючи на взаємодію користувачів із платформами та один з одним [16]. Саме тому дослідники наголошують на необхідності збереження людського елемента у процесах автоматизації.

Thales, лідер у сфері кібербезпеки, який захищає критично важливі програми, API та дані в будь-якому масштабі, сьогодні оголосив про випуск звіту Imperva Bad Bot Report за 2024 рік, глобального аналізу автоматизованого трафіку ботів в Інтернеті. Майже половина (49,6%) усього інтернет-трафіку надходила від ботів у 2023 році – це на 2% більше, ніж у попередньому році, і це найвищий рівень, про який Imperva повідомила з моменту початку моніторингу автоматизованого трафіку у 2013 році [33].

Водночас варто розрізняти корисні та шкідливі боти. Деякі автоматизовані системи виконують важливі завдання – забезпечують модерування, автоматичне надання інформації або підтримують комунікацію між користувачами. Однак частка ботів використовується для розповсюдження спаму, дезінформації або

кібератак. Зважаючи на це, питання ефективного керування ботами стає ключовим у сучасному цифровому середовищі.

Як зазначив Джон-Патрік Аллем: Соціальні боти мають потенціал впливати на суспільний дискурс, підсилюючи певні наративи та пригнічуючи інші, що робить їх критичним фактором в екосистемах онлайн-інформації [13].

Розмови з чат-ботами на основі штучного інтелекту демонструють чіткий баланс між автоматизацією та взаємодією, де залучення користувачів визначається рівнем автономності, наданим системі [17].

У порівняльному аналізі чат-ботів можна виділити кілька ключових аспектів їхньої еволюції:

1. ранній етап (1960-2000 рр.) – перші текстові боти (ELIZA, Parry), які працювали на основі заданих сценаріїв;
2. перехідний період (2000-2015 рр.) – інтеграція ботів у бізнес-комунікацію (Microsoft Bot Framework, Telegram API);
3. сучасний етап (2015- до тепер) – активне використання AI-ботів у соціальних платформах.

З огляду на швидкий розвиток штучного інтелекту, роль ботів у цифровій комунікації буде лише зростати. Виклики, пов’язані з їхнім регулюванням, безпекою та етичними аспектами, залишаються відкритими для подальших досліджень.

1.2. Discord та його можливості

Discord – це платформа обміну миттєвими повідомленнями та цифрового розповсюдження інформації з функціями VoIP. Користувачі спілкуються за допомогою голосових дзвінків, відеочатів, текстових повідомлень, медіа та файлів у приватних чатах або в межах спільнот, які називаються «серверами». Discord сервер – це набір постійних чатів та каналів голосового чату, доступ до яких можна отримати через запрошення. Discord працює на Windows, macOS, Android, iOS, iPadOS, Linux та у веббраузерах [4]. Станом на 2021 рік у сервісі

зареєстровано близько 563 мільйонів користувачів та понад 196,2 мільйона щомісячних активних користувачів.

З виходом Discord швидко витіснив з конкуренції таку платформу як Skype, яка довго була популярна як серед звичайних користувачів, так і геймерів. Насамперед зміна від Skype до Discord демонструє, як платформи змушені еволюціонувати, щоб залишатися конкурентоспроможними в умовах мінливих потреб користувачів (табл. 1.1). Discord не лише запропонував альтернативу, але й став новим стандартом для сучасної комунікації. Його успіх ґрунтується на здатності адаптуватися до різних груп користувачів, забезпечуючи гнучкість і багатофункціональність.

Discord з самого початку зосередився на потребах користувачів і запропонував не просто засіб для зв'язку, а цілісну платформу для створення спільнот. Месенджер має дуже зручний та інтуїтивно зрозумілий інтерфейс (рис. 1.1).

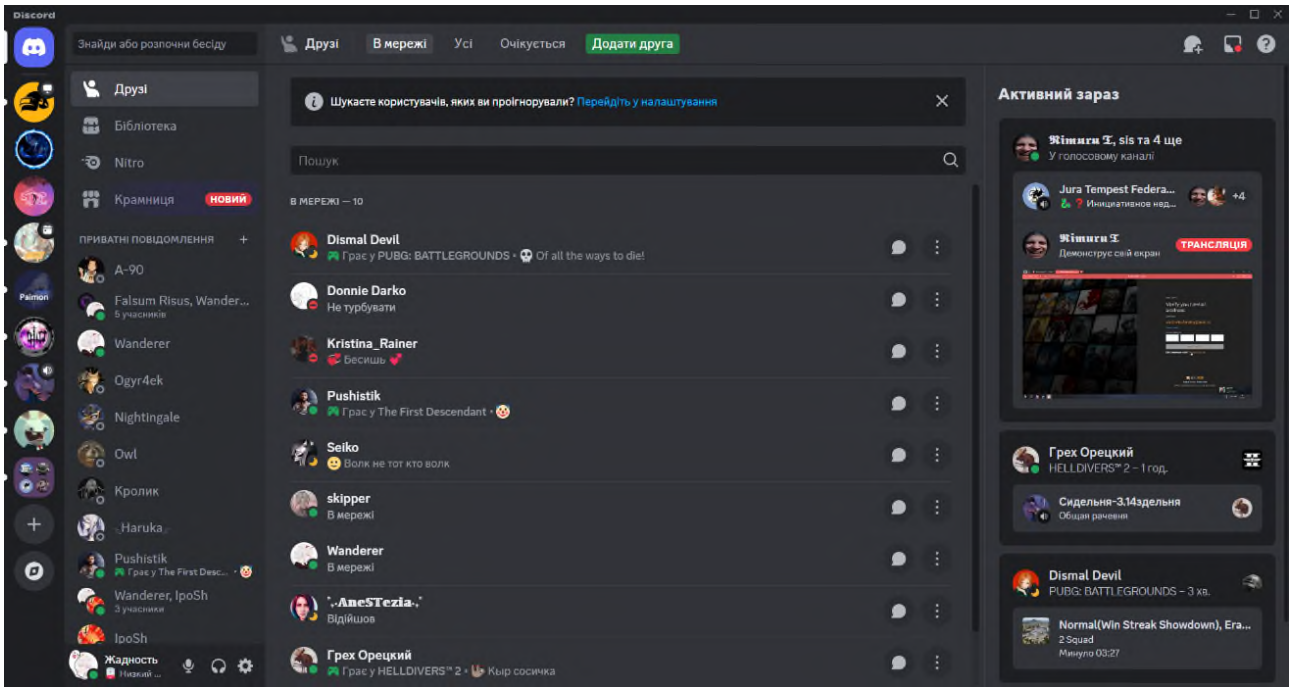


Рис. 1.1 Інтерфейс Discord

В один клік можна перемикатися з одного сервера на інший. Своєю чергою користувачі самі створювали та налаштовували сервери відповідно до їх потреб. Гнучка система голосових та текстових каналів надала можливість фільтрувати інформаційний потік за тематикою, що значно спрощує комунікацію в межах

окремих груп, що дозволяє легко орієнтуватися у великій кількості повідомлень. Це, своєю чергою, сприяє більш ефективній організації дискусій, швидкому обміну інформацією та підтримці порядку в спільнотах. На серверах також є система ролей, що допомагає відокремити звичайних користувачів від адміністративного персоналу, модераторів [4] (рис. 1.2).

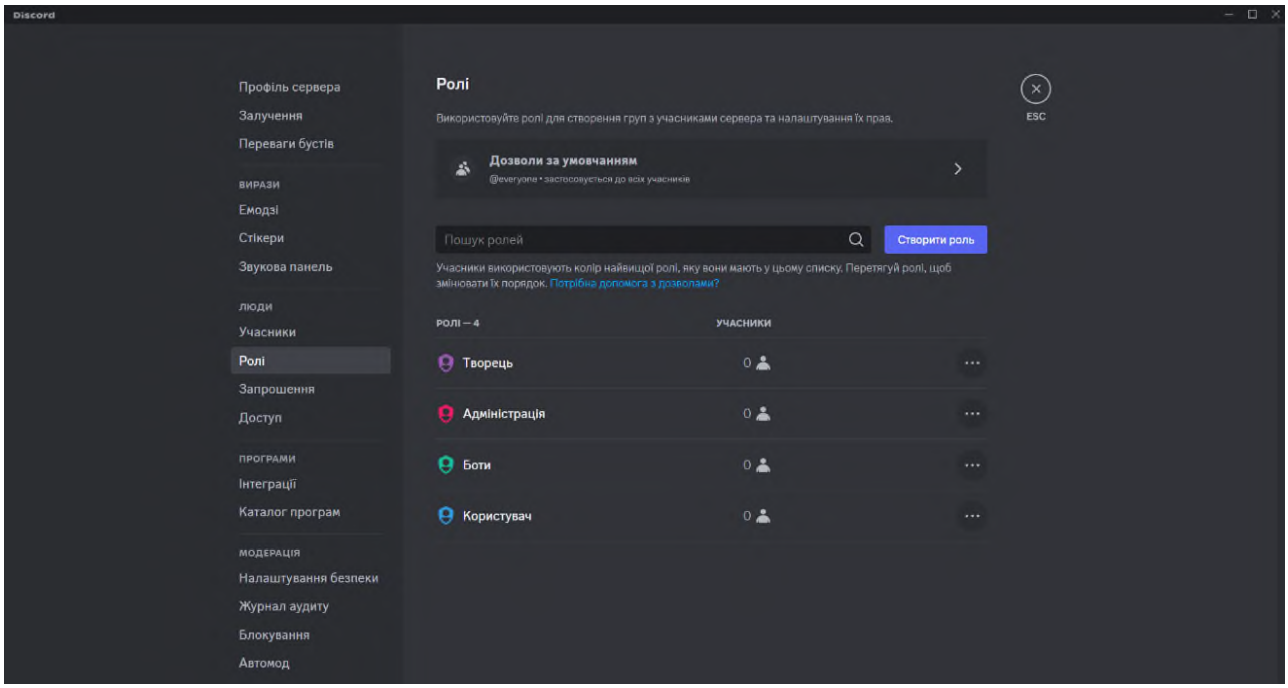


Рис. 1.2 Ролі

Кожну роль можна налаштувати за своїми потребами, та видати потрібні на сервері права та допуски. І на сам перед інтеграція чат-ботів [3].

У контексті сучасної диджиталізації платформи типу Discord стали невіддільною частиною навчання та корпоративної комунікації [19].

Згідно з аналізом European Commission’s Joint Research Centre, цифрові інструменти комунікації підвищують ефективність дистанційного навчання [8].

Серед значущих досліджень варто згадати досвід інтеграції ботів у середовища навчання, що аналізуються в роботах Kim et al. (2021) [17].

Також, згідно з дослідженням IEEE Computer Society, чат-боти здатні забезпечувати до 40% зниження навантаження на викладачів і модераторів [13].

Таблиця 1.1

Порівняння Discord та Skype

Характеристика	**Discord**	**Skype**
Цільова аудиторія	Геймери, онлайн-спільноти за інтересами, творчі групи	Бізнес, особисте спілкування, відеоконференції
Інтерфейс	Сучасний, інтуїтивно зрозумілий, оптимізований для роботи з багатьма каналами та серверами	Традиційний, спрощений, орієнтований переважно на дзвінки та відеозв'язок
Функціональність	Підтримка текстових, голосових та відеоканалів; інтеграція ботів; розширені можливості кастомізації	Основний набір функцій: відео, аудіодзвінки, текстові повідомлення та екранне спільне використання
Структура спільнот	Можливість створення серверів з окремими каналами для різних тематик, налаштування ролей та доступів	Обмежене створення групових чатів без спеціалізованої системи серверів
Система ролей та адміністрування	Гнучка система ролей, що дозволяє розмежовувати права адміністрації, модераторів та звичайних користувачів	Відсутність розвиненої системи ролей – основна увага спрямована на індивідуальне спілкування
Якість зв'язку	Висока якість голосового зв'язку, особливо адаптована для потреб геймерів; стабільність у великій кількості учасників	Стабільна якість відео та аудіозв'язку, оптимізована для бізнес-комунікацій
Моделі монетизації	Безплатний базовий функціонал; додатково – підписка Discord Nitro для розширених можливостей	Безплатний базовий функціонал з можливістю придбання додаткових послуг чи кредитів
Інтеграція ботів та API	Широка підтримка ботів і сторонніх інтеграцій, що дозволяє автоматизувати процеси та додавати нові функції	Обмежені можливості інтеграції ботів та сторонніх додатків

З технічного погляду, існують альтернативи Discord API, зокрема Matrix та Zulip, однак саме Discord підтримує найширший спектр інтеграцій та має високий рівень UX.

1.3. Discord-боти

Одна з найголовніших переваг Discord – це інтеграція ботів. Вони здатні виконувати більшу частину модераційної роботи та спрощувати нагляд за безпекою та порядком сервера. Боти можуть реалізовувати ряд адміністративних задач у режимі реального часу. Наприклад, вони здатні автоматично видаляти небажані повідомлення, реагувати на спам, блокувати порушників та навіть організовувати систему попереджень перед накладенням санкцій. Це значно полегшує роботу модераторів, дозволяючи їм зосередитися на стратегічних питаннях.

Крім модераційних функцій, боти можуть виконувати й додаткові завдання, такі як взаємодія з користувачами та підтримка інформаційного потоку. Вони здатні відповідати на запитання, проводити голосування, надсилати автоматичні нагадування та керувати списками учасників у різних активностях сервера. Це робить їх не просто технічним елементом, а справжнім комунікаційним мостом між користувачами та адміністрацією [3].

Також боти спроможні використовувати зовнішні інтернет-джерела, наприклад, відтворення музики, пошук зображень або інтеграцію штучного інтелекту. Вони можуть підключати API популярних сервісів, таких як YouTube, Wikipedia чи Google Search, що значно розширює можливості взаємодії у межах платформи [14].

Розширення інтерактивних можливостей ботів часто передбачає їхню інтеграцію з популярними зовнішніми сервісами. Наприклад, для реалізації функції відтворення музики з YouTube, пошуку відеокліпів або отримання інформації про канали, розробники можуть використовувати офіційний YouTube Data API v3 [32]. Це дозволяє боту взаємодіяти безпосередньо з величезною базою відеоконтенту та надавати користувачам розширені медіафункції.

Боти в Discord не лише спрощують адміністрування серверів, а й відкривають безліч можливостей для інтеграції зовнішніх сервісів. Вони можуть виконувати складні завдання, такі як автоматичне управління ролями, аналіз активності користувачів, а також створення власних налаштовуваних команд, що робить їх незамінним інструментом для великих спільнот [3].

Зважаючи на їхню багатофункціональність, боти стали важливим елементом у гейміфікації соціальних платформ. Вони можуть організовувати інтерактивні ігри, керувати системами досягнень та рейтингів, а також створювати внутрішню економіку сервера, наприклад, систему балів або віртуальної валюти. Це сприяє підвищенню активності користувачів і створенню більш динамічного середовища [16].

Особливо цікавою є інтеграція штучного інтелекту в ботів. Окрім Midjourney, існують і інші потужні AI-боти, такі як ChatGPT, який дозволяє користувачам вести осмислені бесіди, отримувати консультації та навіть генерувати складні текстові відповіді в межах серверів Discord. Завдяки цьому можна створювати розумні асистенти, які полегшують комунікацію в групах [23].

Midjourney – це популярний інструмент для генерації зображень за допомогою штучного інтелекту. За допомогою команд у чаті та опису побажань можна легко створити потрібне зображення. Бот аналізує ваш текстовий запит і генерує відповідні візуальні образи. Користувачі можуть експериментувати з різними параметрами, такими як стилі, якість зображень, швидкість генерації та інші, що дає змогу отримувати унікальні результати відповідно до запитів [18].

Окрім створення зображень, такі інструменти, як Midjourney, впливають на розвиток цифрового мистецтва. Вони дозволяють художникам та дизайнерам реалізовувати свої ідеї швидше, інтегруючи технології у творчі процеси та вдосконалюючи взаємодію між штучним інтелектом та людиною.

Підписавшись на Midjourney, ви маєте свободу створювати зображення як на їхньому вебсайті, так і на Discord. Більшість функцій і інструментів пропонуються на обох платформах, але іноді вони можуть мати різні назви. Ви

можете вибрати той, який відповідає вашому стилю, або насолоджуватися використанням обох [18].

Редактор на midjourney.com – це ваш вибір для точного налаштування ваших зображень. Усе в одному зручному для користувача місці: можна обрізати, панорамувати та регулювати співвідношення сторін (і більше!) одночасно. Discord також пропонує панорамування та зміну співвідношення сторін, але вам доведеться виконувати кожен крок окремо, що дає вам трохи менше контролю [18].

З огляду на стрімкий розвиток технологій, роль ботів у цифровому середовищі продовжує зростати. Вони вже сьогодні є незамінними помічниками у модеруванні, комунікації та творчих процесах. Подальше вдосконалення AI-технологій може зробити боти ще більш функціональними, розширюючи межі їх використання у соціальних платформах.

Загалом, розробка та впровадження ботів на таких платформах, як Discord, є яскравим прикладом сучасних підходів до автоматизації процесів у цифрових спільнотах. Ці програмні агенти перебирають на себе рутинні завдання, оптимізують взаємодію та підвищують загальну ефективність функціонування онлайн-платформ, що детально аналізується у дослідженнях з автоматизації цифрових платформ [33].

Очевидно, що роль ботів у цифрових платформах продовжуватиме розвиватись, адаптуючись до потреб користувачів і технічного прогресу. Нові алгоритми машинного навчання та нейромережі дозволять ботам стати ще більш розумними та автономними, покращуючи персоналізацію взаємодії та інтеграцію з іншими сервісами. У майбутньому можна очікувати розширення їхніх можливостей – від аналізу емоційного стану користувачів до проведення складних обговорень у спільнотах. Це відкриває перспективи для глибшого розуміння ролі штучного інтелекту в комунікації та його впливу на цифрові спільноти [13].

1.4. Інструментарій. Мова програмування Python

Python (найчастіше вживане прочитання – «Па́йтон», запозичено назву з британського шоу Монті Пайтон) – інтерпретована об'єктноорієнтована мова програмування високого рівня із суворою динамічною типізацією. Розроблена на початку 1990-х років Гвідо ван Россумом. Структури даних високого рівня разом із динамічною семантикою та динамічним зв'язуванням роблять її привабливою для швидкої розробки програм, а також як засіб поєднування наявних компонентів. Python підтримує модулі та пакети модулів, що сприяє модульності та повторному використанню коду. Інтерпретатор Python та стандартні бібліотеки доступні як у скомпільованій, так і у вихідній формі на всіх основних платформах. В мові програмування Python підтримується кілька парадигм програмування, зокрема: об'єктноорієнтована, процедурна, аспектно-орієнтована та функціональна [27].

Python має довгу і насичену історію, яка починається в 1989 році, коли Гвідо ван Россум розпочав розробку цієї мови. Основною ідеєю було створити простий, зрозумілий і водночас потужний інструмент для програмування. Перша офіційна версія, Python 1.0, вийшла у 1991 році, і з того часу мова пройшла значний шлях розвитку, отримуючи нові можливості, покращення продуктивності та підтримку сучасних технологій [27].

З роками Python отримував важливі оновлення. Python 2.0, випущений у 2000 році, додав можливість збору сміття та спискові вирази, що покращило керування пам'яттю та розширило функціональність мови. Python 3.0, який з'явився у 2008 році, радикально оновив мову, зробивши її ще більш ефективною та сучасною. Сучасні версії, починаючи з Python 3.6, пропонують розширені можливості, включаючи асинхронне програмування, типізацію та оптимізацію швидкодії [27].

У посібнику Walker J. наведено приклади використання сучасних парадигм програмування в Python, включаючи асинхронність і шаблони проєктування [31].

Завдяки своїй простоті та читабельності Python є чудовим вибором для створення Discord-бота. Його синтаксис дозволяє швидко розробляти

функціонал без зайвої складності, що особливо важливо для новачків у програмуванні. Однією з основних переваг мови є потужна бібліотека discord.py, яка значно спрощує роботу з API Discord. Вона дозволяє легко взаємодіяти з сервером, обробляти повідомлення, керувати ролями та виконувати інші функції, необхідні для повноцінного бота [5].

Важливою перевагою Python є підтримка асинхронного програмування через модуль asyncio [27]. Це дозволяє створювати ефективних ботів, здатних обробляти багато запитів одночасно без затримок. Такі можливості є особливо корисними для масштабних серверів, де бот повинен швидко реагувати на команди та події, не уповільнюючи систему.

Гнучкість Python дозволяє легко інтегрувати його з різноманітними базами даних, API, машинним навчанням та іншими технологіями, що відкриває широкий спектр можливостей для створення розумних і функціональних ботів. Велика спільнота розробників, які активно створюють документацію та туторіали, значно спрощує процес розробки, дозволяючи швидко знаходити рішення для складних завдань.

Окрім традиційного використання в програмуванні, Python також підтримує візуальне програмування та середовища розробки, такі як IDLE. Python IDLE є офіційним інтегрованим середовищем розробки для Python, яке пропонує зручний та інтуїтивно зрозумілий інтерфейс для створення та тестування програм (рис. 1.3). Воно забезпечує інтерактивну консоль для виконання команд у реальному часі, редактор коду з підсвіткою синтаксису та базові інструменти для налагодження [27].

У майбутньому розвиток Python і його бібліотек відкриє ще більше можливостей для автоматизації та інтеграції штучного інтелекту в цифрові комунікаційні платформи. Останні тенденції показують зростання інтересу до використання штучного інтелекту для розширення функціональності ботів. Глибока інтеграція нейромереж дозволяє створювати більш адаптивних і розумних ботів, здатних вести природні бесіди, аналізувати контекст повідомлень і персоналізувати взаємодію з користувачами [13].

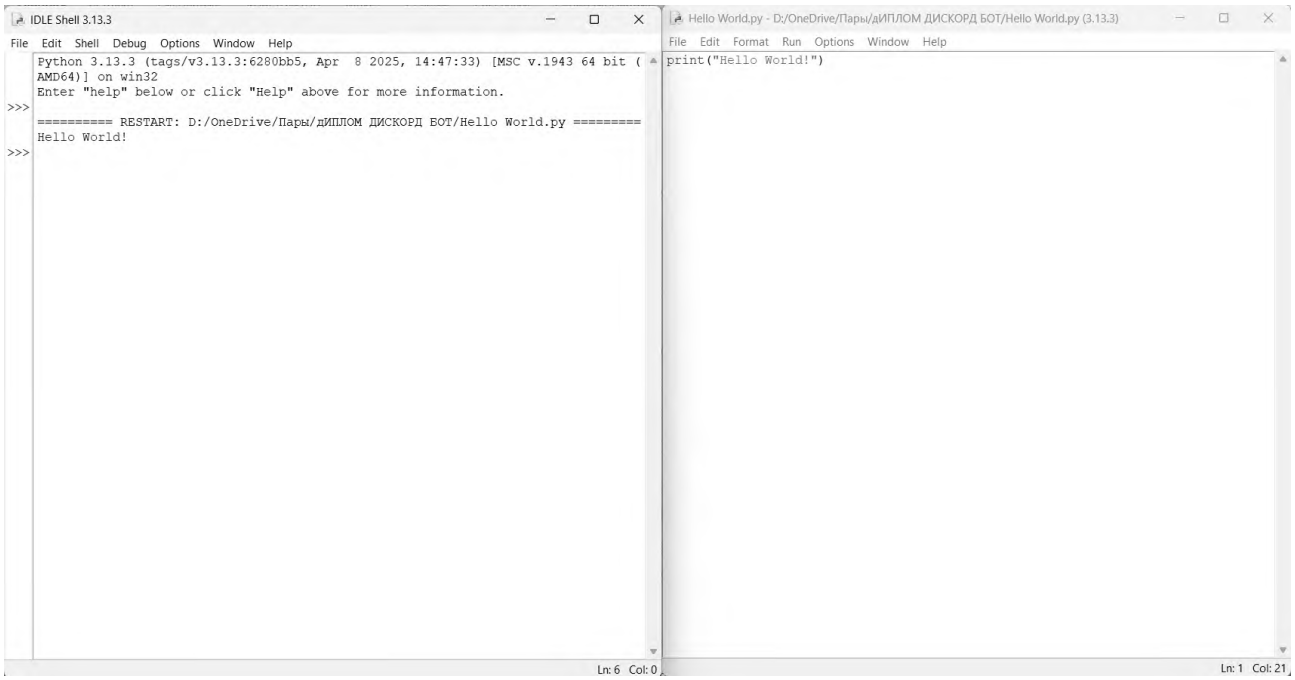


Рис. 1.3 Інтерфейс Python IDLE

Ще один важливий аспект – безпека. Python пропонує безліч інструментів для управління доступом, шифрування даних і захисту від кіберзагроз, що є критичним фактором для великих спільнот [26]. Завдяки цьому розробники можуть створювати надійні системи, які захищають сервери від небажаних атак, спаму та несанкціонованих дій. Безпека є ключовим аспектом при створенні Discord-ботів, особливо для великих спільнот, де ризик зловживань та кіберзагроз значно зростає. Python пропонує широкий спектр інструментів для управління доступом, включаючи бібліотеки aiohttp для безпечного обміну інформацією між серверами (табл. 1.2).

Завдяки відкритості платформи, активному розвитку нових бібліотек і підтримці розробницької спільноти Python залишається одним із найкращих виборів для створення інтелектуальних та ефективних ботів, що робить його ключовим елементом у сфері цифрової автоматизації [37].

Zen of Python (PEP 20) визначає філософію дизайну мови Python, яка відображається в принципах розробки чат-ботів [27].

Таблиця 1.2

Порівняння Python з іншими мовами програмування

Критерій	**Python**	**JavaScript**	**C++**	**Java**
Простота синтаксису	Має зрозумілий і лаконічний код	Складніший через асинхронність	Висока складність	Сувора структура
Робота з Discord API	discord.py – потужна бібліотека	discord.js – ефективний аналог	Вимагає додаткових обгорток	Може використовувати HTTP-запити
Асинхронне програмування	Вбудоване через asyncio	async/await	Не має нативної підтримки	Використовує CompletableFuture
Гнучкість	Легко інтегрується з AI, базами даних	Добре працює в браузерному середовищі	Потребує більше ресурсів	Вимагає додаткової конфігурації
Використання ресурсів	Менш вимогливий	Висока продуктивність	Вимагає більше пам’яті та CPU	Потребує віртуальної машини Java
Спільнота розробників	Дуже велика, активна підтримка	Велика, особливо у веброзробці	Менша, орієнтована на низькорівневі задачі	Велика, популярна в корпоративному секторі
Швидкість розробки	Висока завдяки простому синтаксису	Добра, але потребує додаткової конфігурації	Складніший процес	Вимагає довгого налаштування

На додачу до бібліотеки discord.py, можна розглянути такі інструменти, як interactions.py [14] або nextcord [20], які пропонують нативну підтримку сучасного функціоналу Discord API, включно з ephemeral-повідомленнями та callback-компонентами.

У галузі розподілених обчислень бібліотека aiohttp [1] дозволяє ефективно реалізовувати HTTP-запити у фоновому режимі без втрати продуктивності.

Для зберігання даних, особливо у масштабованих системах, рекомендується використання PostgreSQL [24] з asyncpg [2], яка демонструє кращу продуктивність за sqlite3 [29].

Формат YAML широко використовується для зберігання конфігурацій у багатьох середовищах, включно з Discord-ботами.

З міркувань безпеки, сучасні підходи пропонують інтеграцію механізмів OAuth 2.0 [21, 22] у Discord-ботів [3], що дозволяє реалізовувати авторизований доступ до функцій на основі прав користувача.

Додатково, бібліотека Pydantic [25] значно спрощує валідацію JSON-конфігурацій, а FastAPI [9] може бути використаний для створення backend-служб для інтеграції Discord-ботів із зовнішніми сервісами.

РОЗДІЛ 2. СИСТЕМНО-АНАЛІТИЧНИЙ АНАЛІЗ ТА ПРОЄКТУВАННЯ

2.1. Обґрунтування вибору технологій для розробки Discord-бота

У процесі розробки програмного забезпечення було обґрунтовано доцільність використання мови програмування Python та бібліотеки discord.py. Зазначене рішення є логічним та обґрунтованим з огляду на сучасні тенденції у сфері розробки програм для інтеграції з платформою Discord. Мова Python є однією з найпоширеніших у світі, зокрема завдяки своїй простій, інтуїтивно зрозумілій синтаксичній структурі, високій читабельності коду та широким можливостям для швидкої розробки, тестування й масштабування програмного забезпечення [27]. Вказані характеристики особливо актуальні у контексті створення прикладних систем, таких як Discord-боти, де важливо забезпечити як стабільність, так і гнучкість реалізації [37].

Що стосується самої бібліотеки discord.py, то вона вирізняється дуже зручним і логічно побудованим інтерфейсом. Завдяки використанню принципів асинхронного програмування, бібліотека дозволяє легко працювати з різноманітними подіями та командами, які надсилаються користувачами в реальному часі, при цьому не блокуючи виконання інших процесів [5]. Це стало можливим завдяки інтеграції зі стандартною бібліотекою asyncio, яка є частиною мови Python і дозволяє виконувати кілька задач паралельно, що є надзвичайно зручним під час роботи зі справжнім багатопотоковим середовищем [5].

Крім базового функціоналу, discord.py також підтримує багато сучасних можливостей, які пропонує Discord. Серед них – підтримка Slash-команд, інтерактивних елементів, таких як кнопки та меню, а також розширене управління ролями, каналами та користувачами. Це дає змогу створювати не просто простих ботів з обмеженим функціоналом, а дійсно інтерактивні, динамічні системи, які можуть адаптуватися під потреби спільноти [5].

Вибір бібліотеки discord.py також був обґрунтований порівняльним аналізом з іншими популярними бібліотеками, які використовуються для

створення ботів на Python. У табл. 2.1 наведено ключові характеристики кількох рішень, зокрема nextcord, py-cord та hikari.

Таблиця 2.1

Порівняння discord.py з іншими бібліотеками

Критерій порівняння	**discord.py [9]**	**nextcord [9]**	**py-cord [9]**	**hikari [11]**
Підтримка Discord API	Повна	Повна	Повна	Часткова (фокус на Slash)
Slash-команди та UI-елементи	Так	Так	Так	Так
Активність розробки	Активна, регулярні оновлення	Менш активна	Активна	Активна
Офіційна документація	Докладна, структурована	Базова	Середня	Високоякісна
Сумісність з discord.py кодом	-	Так	Так	Ні
Простота використання	Висока	Висока	Висока	Середня
Асинхронна архітектура (asyncio)	Так	Так	Так	Так (asyncio, trio)
Популярність серед спільноти	Найбільша	Помірна	Зростає	Нижча

Як видно з порівняння, discord.py вирізняється поєднанням високої сумісності, зрозумілого синтаксису, повної підтримки функціоналу Discord та активною спільнотою, що робить її найбільш збалансованим вибором для реалізації проєкту.

Ще одним вагомим плюсом на користь discord.py є її активна спільнота. Проєкт постійно підтримується, оновлюється, і в інтернеті можна знайти безліч прикладів, гайдів і розв’язання типових проблем, що суттєво спрощує роботу з цією бібліотекою як для новачків, так і для досвідчених розробників.

Таким чином, вибір мови програмування став першим кроком до формування технологічного стека майбутнього програмного продукту. Наступним логічним рішенням стало обрання інструменту, що забезпечуватиме прямий зв’язок із платформою Discord. У цьому контексті бібліотека discord.py виявилася оптимальним варіантом, оскільки вона спеціалізована саме на роботі з Discord API та підтримується великою спільнотою розробників.

Бібліотека discord.py, що є відкритим клієнтом для Discord API, надає функціональні засоби для побудови повноцінних ботів із використанням асинхронного підходу до обробки подій. Вона базується на моделі неблокуючого виконання завдань, реалізованої за допомогою стандартної бібліотеки Python asyncio [27], що дозволяє ефективно управляти великою кількістю одночасних запитів без зниження продуктивності системи [7; 9]. Завдяки такій архітектурі реалізація взаємодії з користувачем у режимі реального часу стає не лише можливою, але й продуктивною навіть за значного навантаження [1].

Окрім основної функціональності, яку забезпечує зв’язок з API платформи, бібліотека також підтримує широкий спектр сучасних можливостей, передбачених самим Discord. Суттєвою перевагою discord.py є підтримка таких інструментів, як Slash-команди, інтерактивні елементи (кнопки, спадні меню), а також гнучке управління серверними об’єктами – користувачами, каналами, ролями [5]. Це забезпечує широкі можливості для реалізації сценаріїв взаємодії між користувачами та ботом, включаючи адаптивну реакцію на команди, обробку подій у режимі реального часу та взаємодію з системними об’єктами платформи.

Немаловажним чинником, що впливає на вибір технологій у процесі розробки, є підтримка та активність спільноти. У випадку з discord.py це питання вирішується позитивно: проєкт має розвинену базу користувачів, які активно створюють приклади, документацію, а також беруть участь у вдосконаленні самої бібліотеки. Це значно полегшує процес навчання та розробки, знижує вартість впровадження програмного продукту й мінімізує ризики, пов’язані з помилками при інтеграції.

Крім того, важливою перевагою Python як мови програмування є його багата екосистема [27]. Завдяки цьому можна легко розширити функціональність бота, додавши взаємодію з вебсервісами через бібліотеку aiohttp [1, 40] або організувати збереження даних користувачів за допомогою вбудованої бібліотеки sqlite3 [29] чи складніших ORM-систем, таких як SQLAlchemy [9]. Це дозволяє створювати комплексні, масштабовані рішення, які відповідають вимогам як невеликих, так і великих спільнот у Discord.

Отже, технологічна комбінація Python та discord.py є оптимальним вибором для створення інтелектуальних, масштабованих та зручних у розробці Discord-ботів. Вона поєднує в собі простоту використання, високу продуктивність, підтримку сучасного функціоналу платформи та значну допомогу з боку активної розробницької спільноти [37].

2.2. Архітектура системи та її компоненти

Архітектура Discord-бота, реалізованого за допомогою бібліотеки discord.py, побудована на принципах модульності та асинхронного програмування. Такий підхід забезпечує високу продуктивність системи, зручність у підтримці та можливість масштабування проєкту. Основу архітектури становить подієво-орієнтована модель, що дозволяє обробляти численні одночасні запити без блокування основного потоку виконання. У цьому контексті використання асинхронної бібліотеки asyncio [27], яка входить до стандартної бібліотеки Python, є ключовим елементом реалізації неблокуючого оброблення подій. Завдяки такій архітектурі можна легко інтегрувати нові функціональні модулі, адаптуючи бота до змінних вимог і сценаріїв використання (табл. 2.2). Крім того, ефективне управління асинхронними задачами дозволяє мінімізувати затримки у відповідях, покращуючи взаємодію користувачів із системою.

Таблиця 2.2

Основні компоненти архітектури Discord-бота:

Компонент	**Опис**
Модульність та асинхронність	Архітектура побудована на принципах модульності та асинхронного програмування, що забезпечує високу продуктивність і масштабованість.
Подієво-орієнтована модель	Модель, яка дозволяє обробляти численні одночасні запити без блокування основного потоку виконання, використовуючи бібліотеку asyncio.
Головний модуль	Відповідає за ініціалізацію з'єднання з Discord API, автентифікацію та запуск основного циклу обробки подій.
Клас commands.Bot	Основний клас для реалізації командної логіки бота, включаючи перевірки, групування команд і обробку помилок.
Поділ на модулі (cogs)	Модульна структура, де кожен модуль (cog) відповідає за окремі функції, такі як обробка команд, робота з базами даних, API-запити тощо.
Комунікація між модулями	Використання асинхронних викликів для забезпечення ефективної взаємодії між модулями, що дозволяє працювати з високим навантаженням.
Модульна архітектура	Можливість незалежного масштабування та зміни компонентів, що дозволяє легко додавати нові функції без втручання в основний код.
Система збереження даних	Використання бібліотеки sqlite3 для зберігання інформації, з можливістю інтеграції зі складнішими СУБД через SQLAlchemy.
Інтерактивність з користувачем	Реалізація Slash-команд, кнопок та спадних меню для інтерактивної взаємодії з користувачем, яка підтримується через Discord API.
Конфігурація через JSON	Використання зовнішніх JSON-файлів для зручного налаштування параметрів бота без потреби перекомпіляції коду.
Загальна архітектура	Система базується на розподіленій логіці та асинхронному виконанні, що забезпечує стабільність, ефективність і масштабованість бота.

Цей підхід дозволяє ефективно справлятися з великою кількістю запитів, що можуть надходити одночасно, що особливо важливо при розробці програм, орієнтованих на роботу в реальному часі. Без блокувань і затримок, система

здатна оперативно обробляти кожну задачу, що дозволяє зберігати високу ефективність і стабільність роботи.

У структурі бота важливе місце посідає головний модуль, який відповідає за ініціалізацію з'єднання з Discord API, автентифікацію та запуск основного циклу обробки подій. Сам бот створюється як екземпляр класу commands.Bot, що надається бібліотекою discord.ext.commands і дає змогу реалізувати командну логіку з розширеним функціоналом, включаючи перевірки, групування команд і обробку помилок [5].

Ключова перевага такої організації полягає в тому, що кожен елемент системи має свою чітко визначену роль. Це дозволяє без проблем оновлювати чи змінювати конкретні компоненти без порушення роботи всього бота. Крім того, бібліотека discord.py дозволяє легко додавати нові функції чи змінювати старі без значних змін у структурі основного коду.

Система також передбачає поділ на окремі модулі (cogs), кожен з яких відповідає за певний функціональний блок: обробку команд, реакцію на події, роботу з базою даних, API-запити тощо. Такий підхід дозволяє ізолювати логіку, полегшити налагодження й реалізувати незалежне масштабування компонентів [5]. Комунікація між модулями здійснюється за допомогою асинхронних викликів, що забезпечує високу ефективність навіть при значному навантаженні на систему.

Важливо зазначити, що така архітектура дозволяє значно зменшити кількість помилок і проблем, що можуть виникати через неправильну взаємодію між компонентами. Завдяки чітко визначеним інтерфейсам і залежностям між модулями, система стає значно більш надійною [39].

У процесі розробки програмного забезпечення на базі платформи Discord було реалізовано модульну архітектуру, що дозволяє ефективно розподіляти функціональні компоненти системи. Такий підхід забезпечує гнучкість і масштабованість, оскільки кожен компонент відповідає за окрему функціональність і може бути незалежно змінений або вдосконалений без суттєвого впливу на інші частини проєкту.

Також важливим аспектом є можливість додавати нові можливості або модулі без необхідності переписувати наявний код. Кожен новий функціональний блок може бути підключений окремо, що дозволяє зберігати гнучкість і динамічність в роботі з ботом [5].

Для реалізації голосових функцій, таких як підключення до голосових каналів і передача аудіо, бібліотека discord.py використовує модуль PyNaCl, який забезпечує доступ до низькорівневої криптографічної обробки звуку. Завдяки PyNaCl [26] забезпечується шифрування голосових даних відповідно до вимог Discord API [3].

Окрему роль в архітектурі відіграє система збереження даних. У поточній реалізації для цього використовується вбудована бібліотека sqlite3 [29], яка дозволяє зберігати інформацію про користувачів, їхню активність, конфігурацію бота тощо. У випадку ускладнення логіки або збільшення обсягу даних можлива інтеграція з повноцінними СУБД через ORM-систему SQLAlchemy, яка підтримує асинхронну взаємодію з базами даних.

Збереження даних є важливим аспектом для забезпечення стабільної роботи бота, особливо коли необхідно відстежувати історію взаємодії з користувачами чи зберігати налаштування для подальшого використання. Така система дозволяє зберігати та обробляти великі обсяги інформації без значного впливу на швидкість роботи бота.

Інтерактивність у взаємодії з користувачем реалізована через Slash-команди, кнопки та спадні меню, які активно підтримуються бібліотекою discord.py. Завдяки реалізації цієї функціональності в межах Discord API [3], бот може оперативно відповідати на дії користувача, надсилаючи адаптовані повідомлення та оновлюючи елементи інтерфейсу в реальному часі.

Ця інтерактивність є основною перевагою бота для користувачів, оскільки вона дозволяє швидко реагувати на запити та змінювати інтерфейс відповідно до потреб користувачів. Завдяки зручним та доступним елементам взаємодії, користувачі можуть ефективно використовувати всі функції бота без зайвих труднощів.

Додатково, система включає механізм конфігурації на основі зовнішніх JSON-файлів. Такий підхід спрощує налаштування логіки бота, дозволяє швидко змінювати параметри без перекомпіляції або модифікації коду [27].

Цей механізм забезпечує значну гнучкість при налаштуванні параметрів бота. Користувачі або адміністратори можуть змінювати конфігурацію без необхідності втручатися у код програми, що дозволяє уникнути помилок та значно спрощує процес адміністрування.

Таким чином, архітектура системи базується на розподіленій логіці, асинхронному виконанні та гнучкій модульній побудові. Такий підхід забезпечує стабільність, розширюваність та ефективність роботи бота, що є необхідними критеріями для сучасних програмних продуктів, які функціонують у середовищі з великою кількістю одночасних користувачів [5].

2.3. Опис алгоритмів та основних модулів

У цьому підрозділі розглядається опис алгоритмів та основних модулів, які забезпечують функціональність Discord-бота. Столяров А. В. у своїй праці акцентує на важливості алгоритмізації при створенні ефективних програмних рішень. Архітектура побудована так, щоб забезпечити ефективну взаємодію між компонентами системи, зберігаючи гнучкість та масштабованість.

При розробці подібних асинхронних систем особливу роль відіграє використання бібліотеки asyncio, яка забезпечує неблокуюче виконання коду та дозволяє масштабувати обробку подій [27]. Ця бібліотека підтримує асинхронну обробку запитів, що дозволяє виконувати численні операції одночасно без блокування основного потоку. Це особливо важливо для систем, які працюють у реальному часі та мають високе навантаження.

Одним із ключових алгоритмів є ініціалізація бота. На цьому етапі створюється екземпляр класу commands.Bot, який відповідає за взаємодію з Discord API. Бот проходить процес автентифікації через отриманий токен, після чого починається робота з подієво-орієнтованою моделлю. Завдяки цьому бот

може реагувати на різноманітні події, такі як надсилання повідомлень, приєднання нових користувачів або зміна ролей [5].

Процес автентифікації виконується через функцію bot.run(TOKEN), де TOKEN – це унікальний ключ, згенерований на Discord Developer Portal [4]. З погляду безпеки, використання змінних середовища або .env файлів є найкращою практикою для захисту токену від випадкового витоку, особливо при роботі з публічними репозиторіями [4, 38].

Алгоритм обробки команд побудований таким чином, щоб реагувати на текстові запити користувачів. Насамперед перевіряється синтаксис команди. Якщо команда відповідає заданим критеріям, викликається прив'язана функція, яка обробляє запит. У разі некоректного введення команди бот автоматично інформує користувача про помилку у зрозумілому форматі.

Перевірка прав доступу до команд забезпечується за допомогою декораторів, наприклад, @commands.has_permissions(). Вони дозволяють обмежити виконання певних функцій тільки для адміністраторів або користувачів із конкретними ролями [5]. Це допомагає уникнути непередбачуваних дій з боку неавторизованих користувачів.

Обробка подій є важливою частиною роботи бота. Discord-бот реагує на різноманітні події за допомогою асинхронних функцій. Наприклад, при приєднанні нового користувача до сервера бот може автоматично надсилати персоналізоване вітальне повідомлення або додавати йому специфічну роль залежно від налаштувань сервера.

Для моніторингу та налагодження роботи бота використовується модуль logging. Цей модуль надає зручний спосіб вести журнали логів, у яких можна відстежувати системні виклики, помилки та успішно виконані операції. Регулярний аналіз логів допомагає розробникам оптимізувати функціональність системи [35].

Структура бота організована за принципом модульності. Головний модуль відповідає за автентифікацію та основну взаємодію з Discord API, тоді як модулі команд (cogs) реалізують специфічний функціонал, наприклад, управління

ролями, взаємодію з базами даних або інтеграцію з зовнішніми API. Модульний підхід дозволяє з легкістю розширювати функціональність бота, тестувати окремі модулі незалежно один від одного та підтримувати масштабованість проєкту [5].

Система збереження даних базується на бібліотеці sqlite3 [29], яка використовується для зберігання конфігурацій та інформації про користувачів. У випадках, коли обсяг даних стає занадто великим, можна інтегрувати ORM-систему SQLAlchemy, яка забезпечує гнучкість і масштабованість для роботи зі складнішими базами даних.

Інтерактивність у роботі бота забезпечується через використання знаку “!”. Це дозволяє створити інтуїтивно зрозумілий та зручний інтерфейс для користувачів. Завдяки цьому бот може не тільки оперативно обробляти запити, але й адаптувати свою логіку до потреб сервера у реальному часі [5].

Налаштування конфігурацій бота здійснюється за допомогою JSON-файлів. Це зручний формат для зберігання структурованих даних, який дозволяє легко змінювати налаштування бота без необхідності модифікації основного коду [27]. JSON також використовується для тестування та розгортання, що спрощує інтеграцію бота у нові середовища.

Використання асинхронних функцій, модульної архітектури та інтерактивних елементів забезпечує ефективність і зручність, необхідні для сучасних потреб користувачів.

2.4. Моделювання даних

У процесі реалізації проєкту було використано низку інструментів, серед яких ключовими є: середовище розробки, бібліотеки Python, онлайн-сервіси та ресурси для тестування.

При створенні технічної документації проєкту, а також при обговоренні архітектурних рішень, важливо забезпечити однозначне розуміння використовуваних термінів. Для уніфікації термінології в галузі інформаційних

технологій існують міжнародні стандарти, такі як ISO/IEC 2382:2015 [15], що містить узгоджений словник термінів.

Розробка бота здійснювалася в середовищі Visual Studio Code, що забезпечує зручний інтерфейс, підтримку розширень, автоформатування коду та інтеграцію з Git (рис. 2.1). Цей редактор став стандартом де-факто серед Python-розробників завдяки своїй легкості та функціональності.

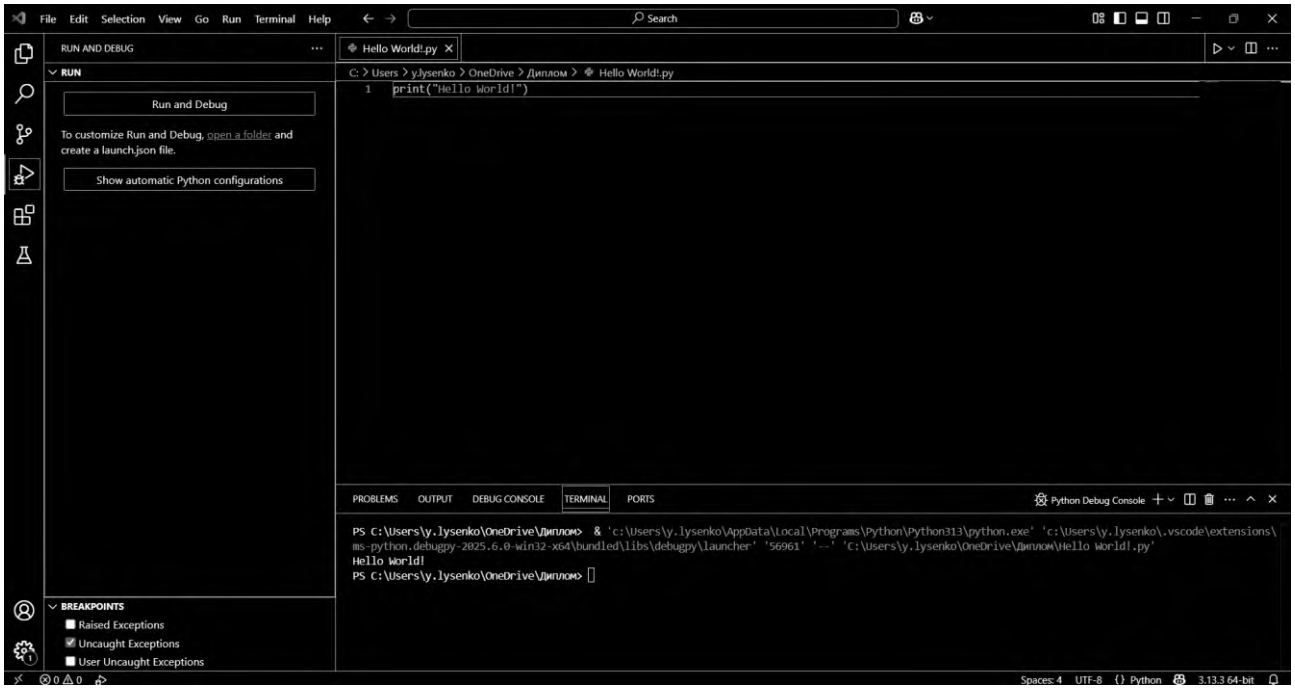


Рис. 2.1 Інтерфейс Visual Studio Code

Для забезпечення стабільного з’єднання з Discord API використовувався токен, згенерований на Discord Developer Portal [4]. Крім основної документації API [3], під час розробки активно застосовувалися приклади коду з офіційного GitHub-репозиторію бібліотеки discord.py [6, 10], що значно пришвидшило створення та налагодження функціоналу.

Для зберігання локальних даних застосовувався sqlite3, а в перспективі передбачено міграцію до PostgreSQL [24] або іншої більш масштабованої СУБД. Розширення функціоналу баз даних здійснювалося з використанням ORM-бібліотеки SQLAlchemy в асинхронному режимі.

Крім того, в роботі використовувалися сервіси на кшталт replit.com для швидкого тестування коду онлайн, а також ngrok для тунелювання локального сервера, коли виникала потреба в перевірці взаємодії бота з Discord у реальному часі [19].

Окрім вибору середовища розробки та бібліотек, важливим етапом життєвого циклу програмного продукту є його розгортання та підтримка. Для забезпечення портативності, масштабованості та ізоляції середовища виконання Discord-бота часто розглядається використання технологій контейнеризації. Docker, згідно з його офіційною документацією [7], дозволяє "запакувати" застосунок з усіма необхідними залежностями в легко переносний контейнер, що значно спрощує процес деплою на різноманітні серверні інфраструктури.

Альтернативою локальному хостингу або використанню віртуальних серверів для розгортання Discord-ботів є платформи як сервіс (PaaS). Платформа Heroku, наприклад, надає інструменти для деплою Python-застосунків, автоматизуючи багато аспектів управління інфраструктурою, що описано в її документації [12]. Такий підхід може бути зручним для швидкого запуску та тестування бота в реальному середовищі.

Хоч сучасні бібліотеки для розробки ботів, такі як discord.py, значною мірою абстрагують розробника від деталей мережевої взаємодії, глибоке розуміння принципів прикладного програмування в мережі, як описано у відповідній літературі [36], залишається корисним. Це допомагає при діагностиці проблем зі з'єднанням, оптимізації трафіку та забезпеченні безпеки передачі даних між ботом та серверами Discord.

РОЗДІЛ 3. ПРАКТИЧНА РЕАЛІЗАЦІЯ ПРОЄКТУ DISCORD-БОТ

3.1. Вибір інструментів і підготовка середовища

У процесі розробки Discord-бота важливим першим кроком є вибір інструментів та налаштування середовища розробки. Цей етап забезпечує основу для реалізації програмного продукту, а також створює умови для ефективної розробки й тестування. Враховуючи результати аналізу в розділі 2, було прийнято рішення використовувати мову програмування Python разом із бібліотекою discord.py, а також інтегроване середовище розробки Visual Studio Code.

Python обрано через його простоту, зрозумілий синтаксис та підтримку модульного програмування. Це дозволяє швидко реалізувати складні функціональні рішення навіть розробникам із базовим рівнем знань. Додатковою перевагою є підтримка асинхронного програмування через стандартну бібліотеку asyncio, що критично важливо для роботи Discord-бота, який має реагувати на події в реальному часі без затримок.

Бібліотека discord.py є найпоширенішим рішенням для розробки ботів на платформі Discord. Вона забезпечує повну інтеграцію з Discord API, надаючи розширені інструменти для роботи з командами, подіями, користувачами та іншими елементами платформи. Перевагами бібліотеки є її активна підтримка спільнотою, регулярні оновлення та наявність вичерпної документації, що значно спрощує процес розробки.

Для налаштування середовища розробки Discord-бота було обрано мову програмування Python у зв’язці з бібліотекою discord.py, що забезпечує ефективну розробку асинхронних додатків із повною інтеграцією до платформи Discord. Під час підготовки середовища було виконано низку важливих кроків, серед яких: інсталяція Python, налаштування бібліотеки discord.py, встановлення FFmpeg, підготовка зручного середовища розробки у Visual

Studio Code, реєстрація застосунку на Discord Developer Portal, а також створення структури проєкту.

Розпочато з інсталяції останньої стабільної версії Python, яку було завантажено з офіційного сайту python.org. Під час встановлення було обрано опцію Add Python to PATH, що забезпечило можливість доступу до Python з командного рядка. Для перевірки коректності встановлення було використано команду python --version, яка при успішній установці виводить номер версії Python (рис. 3.1).

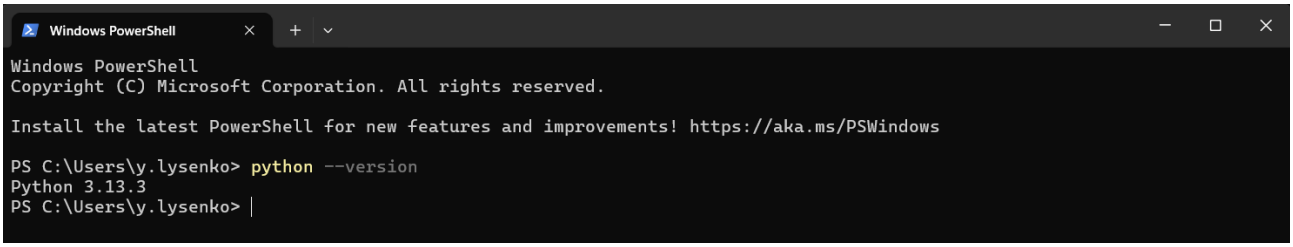


Рис 3.1 Перевірка установки Python

Після цього було встановлено бібліотеку discord.py, яка є ключовою для взаємодії з Discord API. Установку було здійснено за допомогою команди python -m pip install discord.py (рис. 3.2), а для перевірки її наявності використовувалася команда python -m pip show discord.py (рис. 3.3), що підтвердила успішне підключення бібліотеки до проєкту.

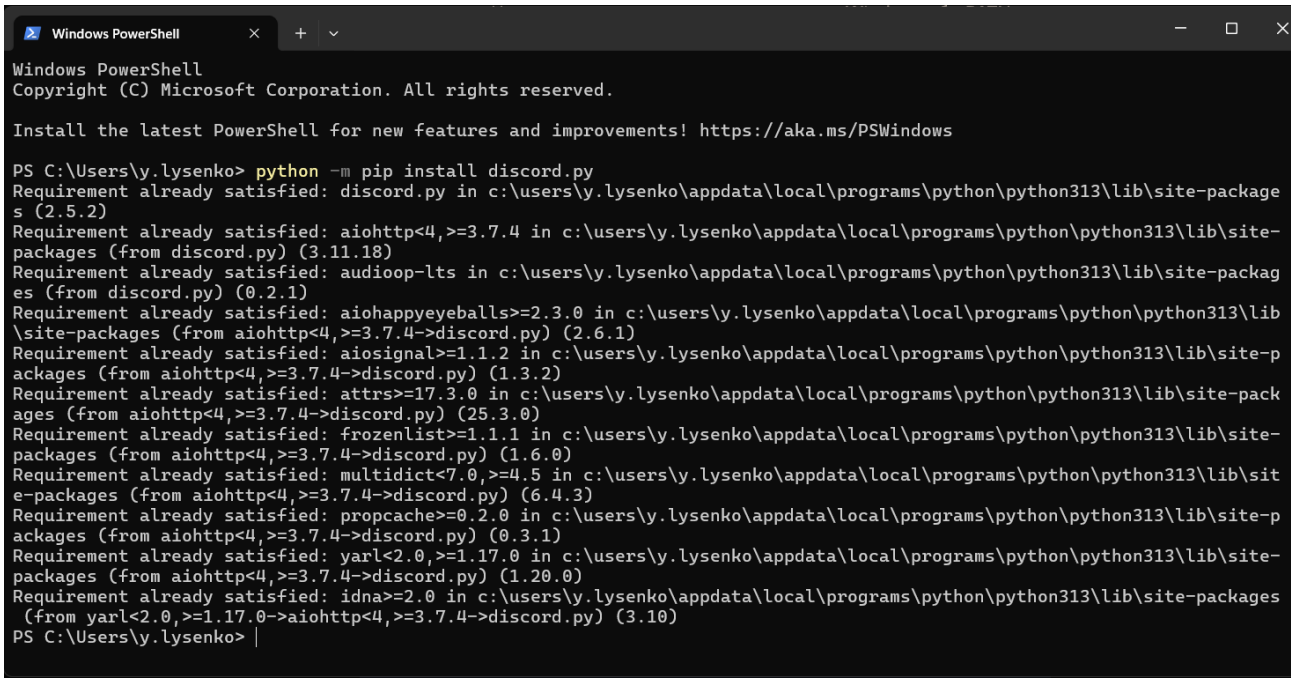


Рис 3.2 Встановлення discord.py

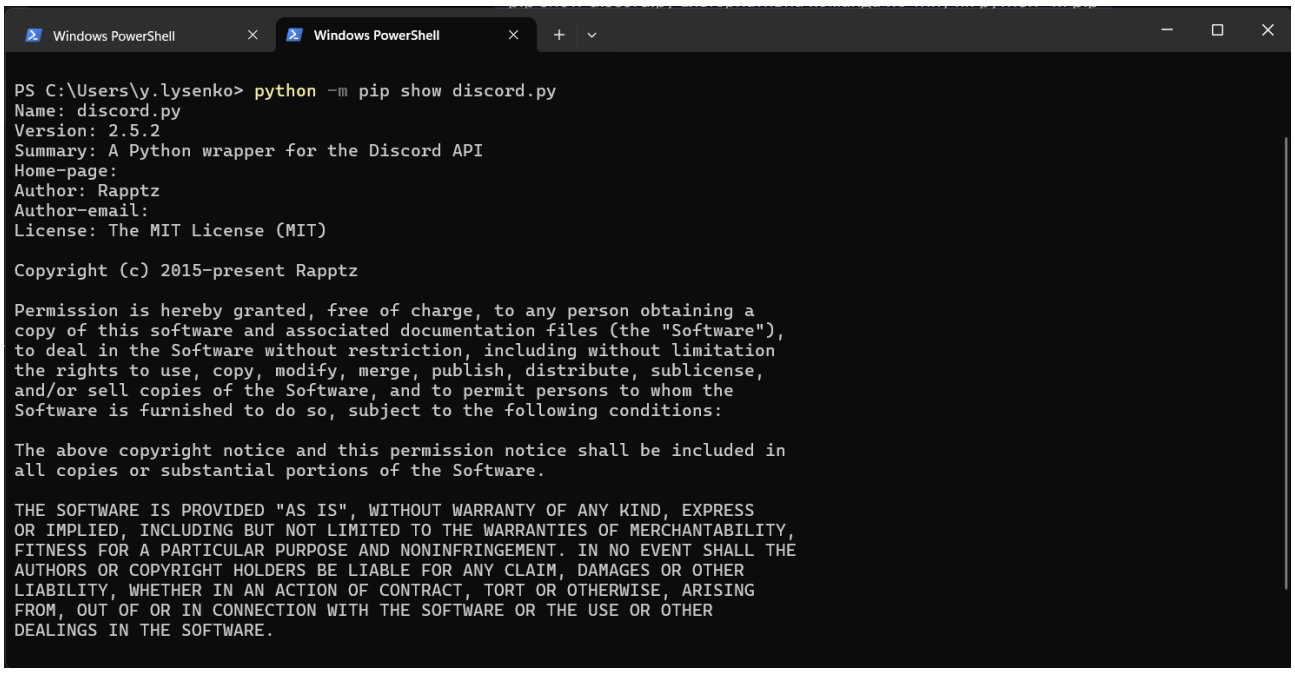


Рис. 3.3 Перевірка встановлення

Оскільки бот має підтримувати функціональність відтворення аудіо з мережі, було також встановлено FFmpeg. Це досягається через пакет ffmpeg-python, який інсталюється разом із додатковими інструментами за допомогою команди pip install discord.py yt-dlp ffmpeg-python.

Середовищем розробки було обрано Visual Studio Code через його гнучкість, інтуїтивно зрозумілий інтерфейс та підтримку Python. Після завантаження редактора з офіційного сайту code.visualstudio.com було встановлено розширення для роботи з Python, що забезпечує зручне автозаповнення, автоматичне форматування коду та налагодження.

Далі відбулася реєстрація нового застосунку на Discord Developer Portal, де було створено бота, згенеровано унікальний токен авторизації, який з міркувань безпеки було збережено у файлі .txt у форматі token.txt. Це дозволяє ізолювати конфіденційну інформацію від основного коду.

Завершальним етапом підготовки стало створення самого проєкту. У локальній папці було ініціалізовано структуру з основним файлом bot.py, який містить стартову логіку запуску бота. Було закладено основу модульної структури, що дозволяє в майбутньому розширювати функціонал бота шляхом додавання окремих компонентів у вигляді модулів.

3.2. Реалізація функціональності бота

Базовий код для перевірки роботи бота. Перший скрипт виконує авторизацію бота та підключення до Discord API:

```
import discord
from discord.ext import commands
intents = discord.Intents.default()
bot = commands.Bot(command_prefix="!", intents=intents)
@bot.event
async def on_ready():
   print(f'Logged in as {bot.user}')
@bot.command()
async def ping(ctx):
   await ctx.send("Pong!")
bot.run("Токен")
```

Цей початковий код, що успішно перевіряє з'єднання з Discord та підтверджує автентифікацію бота (рис. 3.4), є лише фундаментом, на якому можна звести значно складнішу та функціональнішу систему. Усі подальші дії та можливості бота можуть бути експоненційно розширені на основі цієї базової інтеграції. Наприклад, функціонал обробки повідомлень може еволюціонувати від простих команд до складних систем розпізнавання природної мови, автоматичних відповідей на часті запитання, модерації чату з використанням фільтрів та алгоритмів виявлення небажаного контенту. Управління ролями може бути автоматизоване для надання або скасування дозволів на основі активності користувачів, їхніх досягнень чи реакцій на певні повідомлення, створюючи динамічні спільноти. Використання асинхронного програмування, є ключовим аспектом для забезпечення високої продуктивності та чутливості бота, особливо на активних серверах з великою кількістю користувачів та одночасних подій.

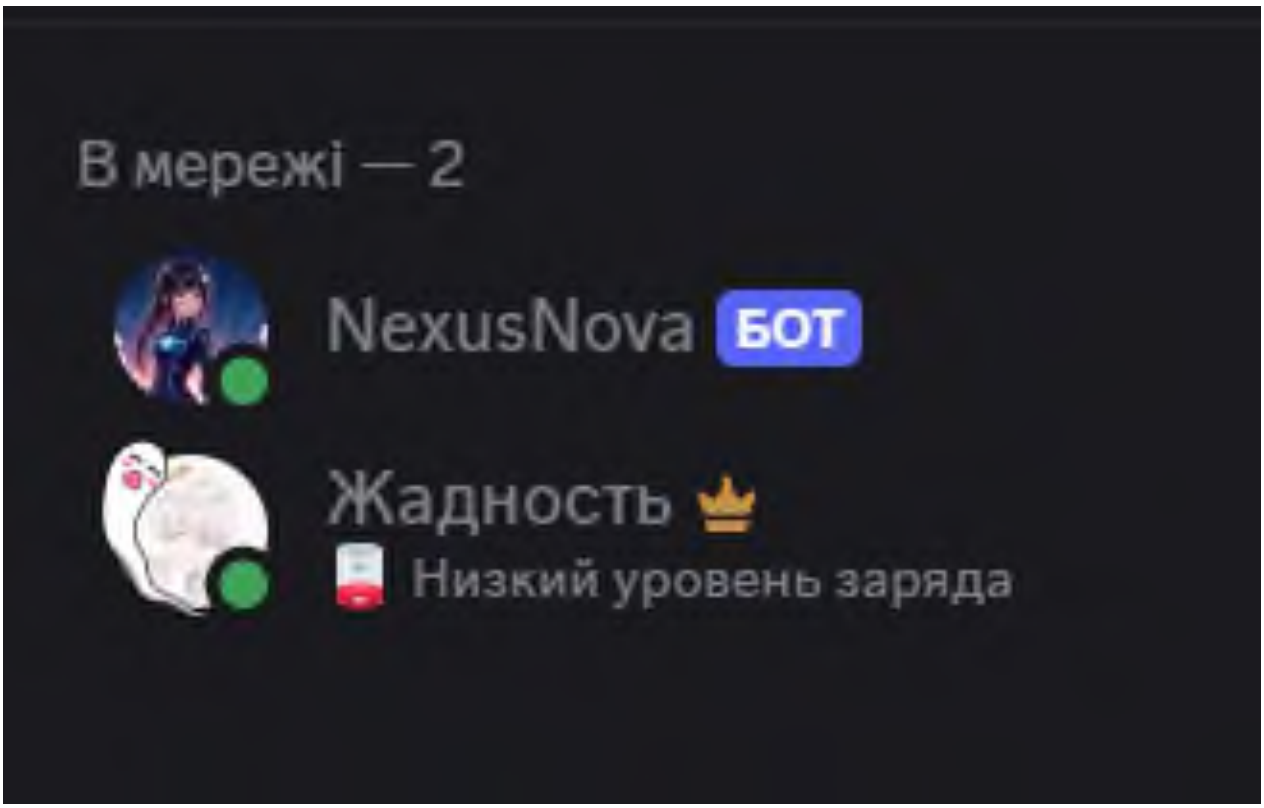


Рис. 3.4 Авторизований бот

Після успішного запуску бота він почав реагувати на тестову команду, таку як !ping, на яку він відповідає Pong! (рис. 3.5). Окрім базової відповіді, бот також надає інформацію про затримку API, що дозволяє користувачам оцінити продуктивність системи у реальному часі. Така функція є важливим елементом моніторингу мережевого з'єднання та може бути доповнена іншими діагностичними командами, наприклад, перевіркою стану сервера або статистики використання. Це підтверджує правильність роботи бота та його готовність до розширення функціоналу відповідно до потреб користувачів.

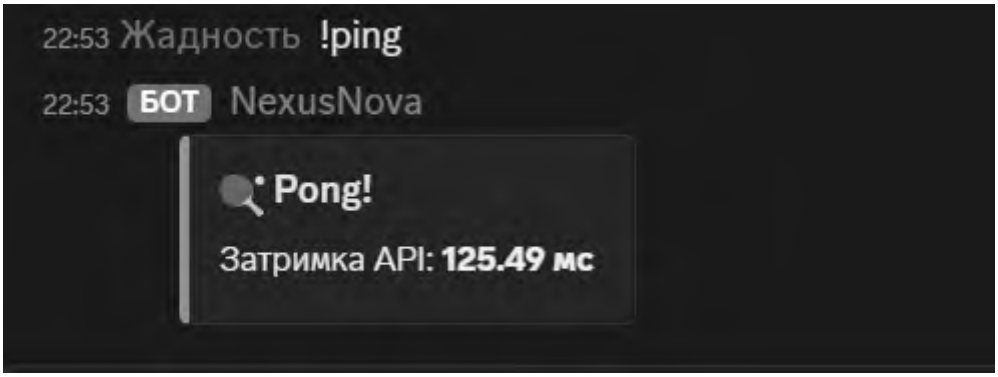


Рис 3.5 Реакція бота на команду

3.3. Тестування та підсумкова оптимізація

Для більш зручного орієнтування по командах бота була створена кастомна команда help, яка виводить зручний та зрозумілий список всіх доступних команд з коротким описом їх функціонала та розбиттям по блоках функціоналу (рис. 3.6). Це значно покращує взаємодію користувача з ботом, особливо коли функціональність охоплює велику кількість команд. Завдяки використанню форматованих вбудованих повідомлень (embedded messages), інформація подається у структурованому вигляді з заголовками, розділами та підказками щодо спеціальних фраз, які підтримуються ботом. Такий підхід дозволяє швидко зорієнтуватися в можливостях бота, не звертаючись до сторонньої документації або коду, що робить роботу з ботом інтуїтивно зрозумілою навіть для нових користувачів.

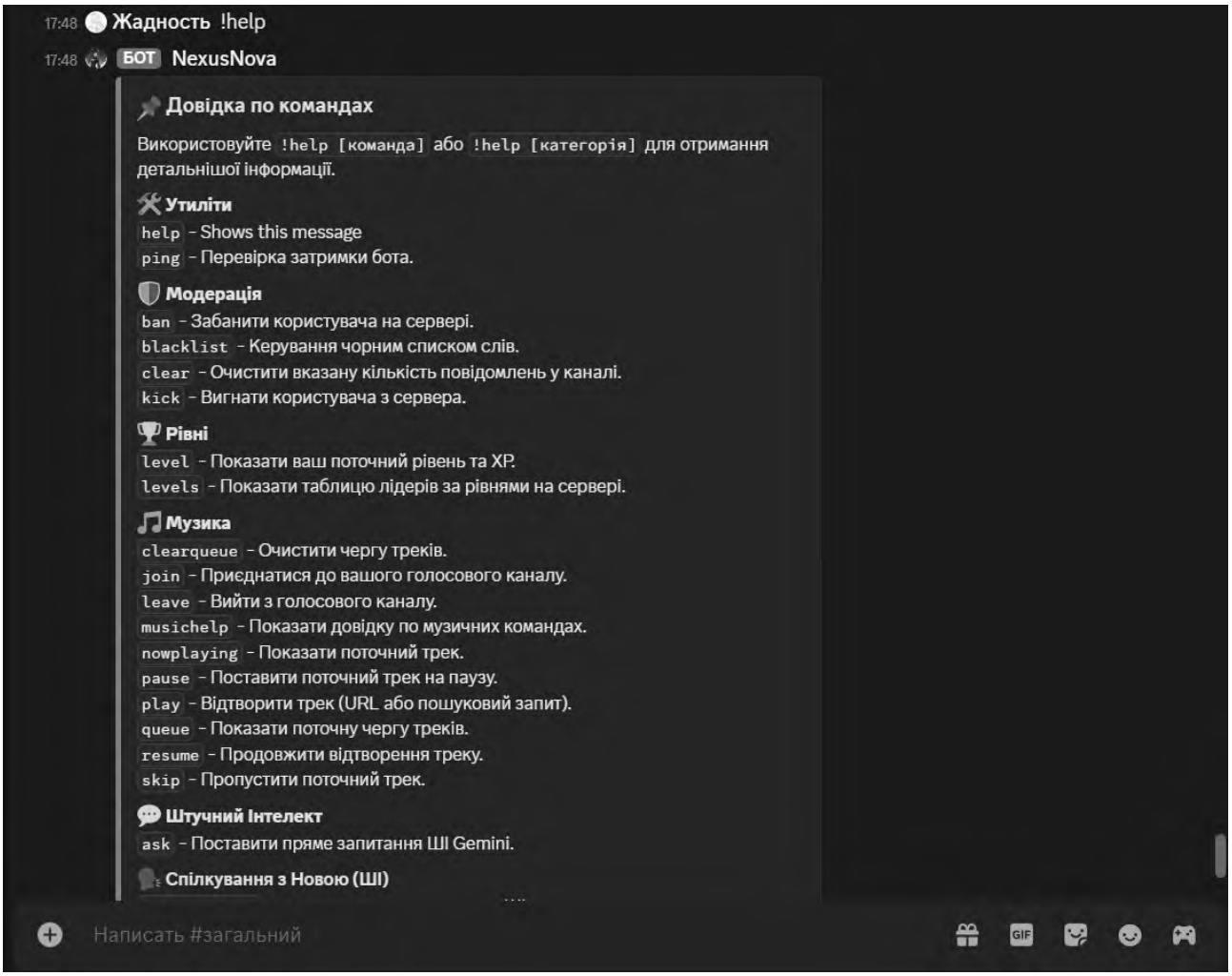


Рис. 3.6 Доступні команди

Однією з ключових функцій створеного Discord-бота є фільтрація небажаного контенту на основі списку заборонених слів. Ця функція реалізована шляхом попереднього завантаження списку слів із текстового файлу blacklist.txt. Під час обробки кожного вхідного повідомлення бот виконує перевірку на наявність в ньому будь-якого слова зі списку. У разі виявлення забороненого слова повідомлення автоматично видаляється, а користувачу виводиться попередження (рис. 3.7). Такий підхід дозволяє ефективно підтримувати порядок у чаті та забезпечити дотримання правил спільноти. Реалізація є гнучкою, оскільки список заборонених слів можна редагувати без необхідності змінювати сам код бота.

```
blacklist_file = "blacklist.json"
# Завантаження чорного списку
if os.path.exists(blacklist_file):
    with open(blacklist_file, "r", encoding="utf-8") as f:
        blacklist = json.load(f)
else:
    blacklist = []
@bot.command(help="Додати слово до чорного списку")
@commands.has_permissions(administrator=True)
async def add_blacklist(ctx, word: str):
    if word.lower() not in blacklist:
        blacklist.append(word.lower())
        with open(blacklist_file, "w", encoding="utf-8") as f:
            json.dump(blacklist, f, indent=4)
        await ctx.send(f"Слово додано до чорного списку.")
    else:
        await ctx.send(f" Слово вже є в чорному списку.")
```

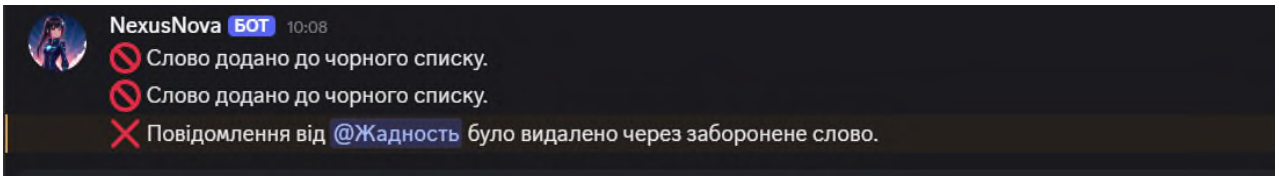


Рис. 3.7 Видалення небажаних повідомлень

Також була додана можливість підключатися до голосового каналу та відтворювати аудіо з YouTube за посиланням (рис. 3.8). Крім того, бот підтримує чергу відтворення, дозволяючи користувачам додавати кілька треків у список очікування. Це забезпечує безперервне звучання музики без необхідності вручну запускати кожну композицію. Також реалізована команда !music_list, яка дає змогу переглянути поточний список треків, а !cmlist—очистити його за потреби. Завдяки команді !skip користувачі можуть перемикатися між треками, а !resume і !stop забезпечують зручне управління паузами та зупинкою відтворення. Така гнучкість функціоналу робить бота ефективним мультимедійним інструментом для інтерактивних спільнот, надаючи їм можливість налаштовувати музичний супровід відповідно до власних уподобань. (додаток А).

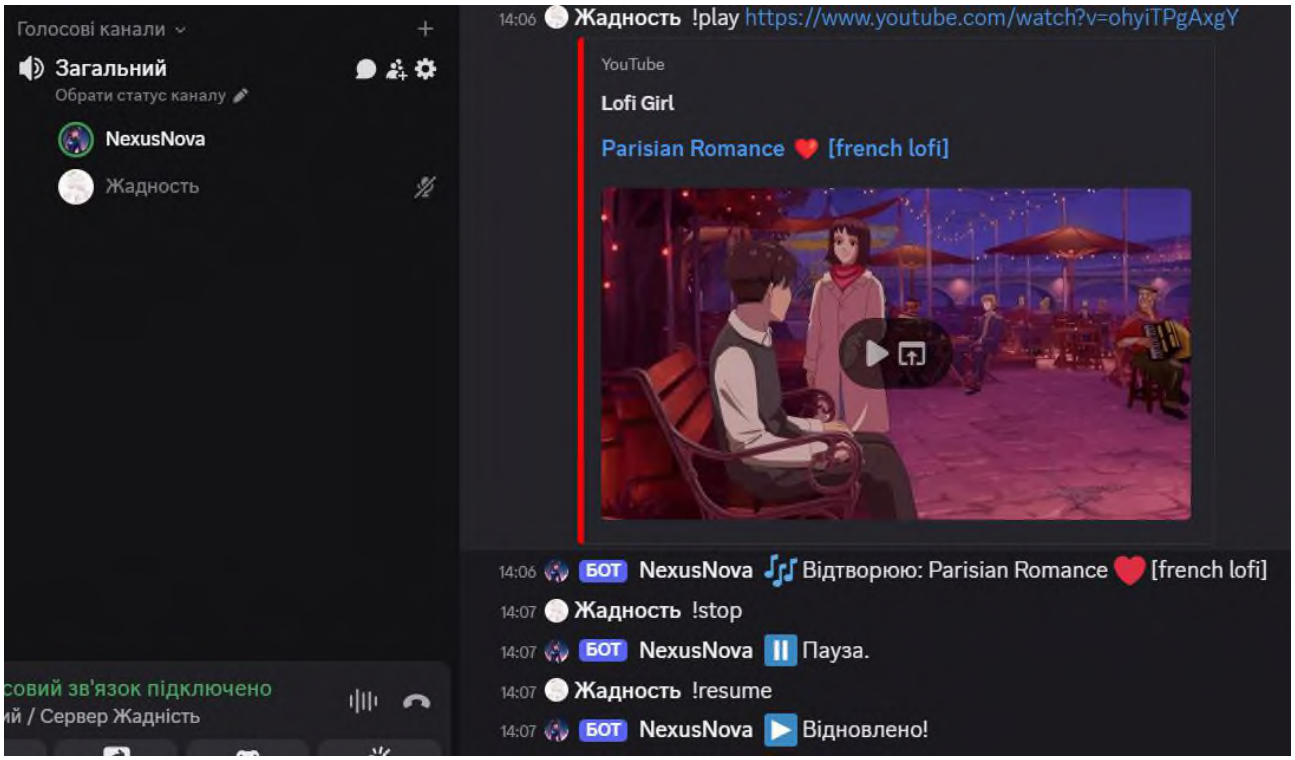


Рис 3.8 Функціональність музичних команд

Зі стрімким розвитком технологій, за останні роки штучний інтелект поступово почав ставати невіддільною частиною нашого життя. На цей момент ще не так багато Discord-ботів використовують ШІ, тож було вирішено додати його до функціональності нашого бота (рис. 3.9). Впровадження ШІ дозволить значно розширити можливості бота, зокрема в таких аспектах, як відповіді на запити користувачів. Зокрема, інтеграція з Gemini API дозволяє реалізувати передові можливості для обробки природної мови та створення більш розумних і динамічних взаємодій у Discord-середовищі.

```
# API-ключ Gemini
with open("aitoken.txt", "r", encoding="utf-8") as f:
    gemini_api_key = f.read().strip()
genai.configure(api_key=gemini_api_key)
model = genai.GenerativeModel("gemini-2.0-flash")
 @bot.command(help="Постав запитання ШІ Gemini")
async def ask(ctx, *, prompt: str):
    try:
        async with ctx.typing():
            loop = asyncio.get_event_loop()
            response = await loop.run_in_executor(None, model.generate_content, prompt)
            reply = response.text
            if len(reply) > 2000:
                chunks = [reply[i:i+2000] for i in range(0, len(reply), 2000)]
                for chunk in chunks:
                    await ctx.send(chunk)
            else:
                await ctx.send(reply)
    except Exception as e:
        await ctx.send("⚠ Сталася помилка при зверненні до Gemini API.")
        print(f"Gemini error: {e}")
```

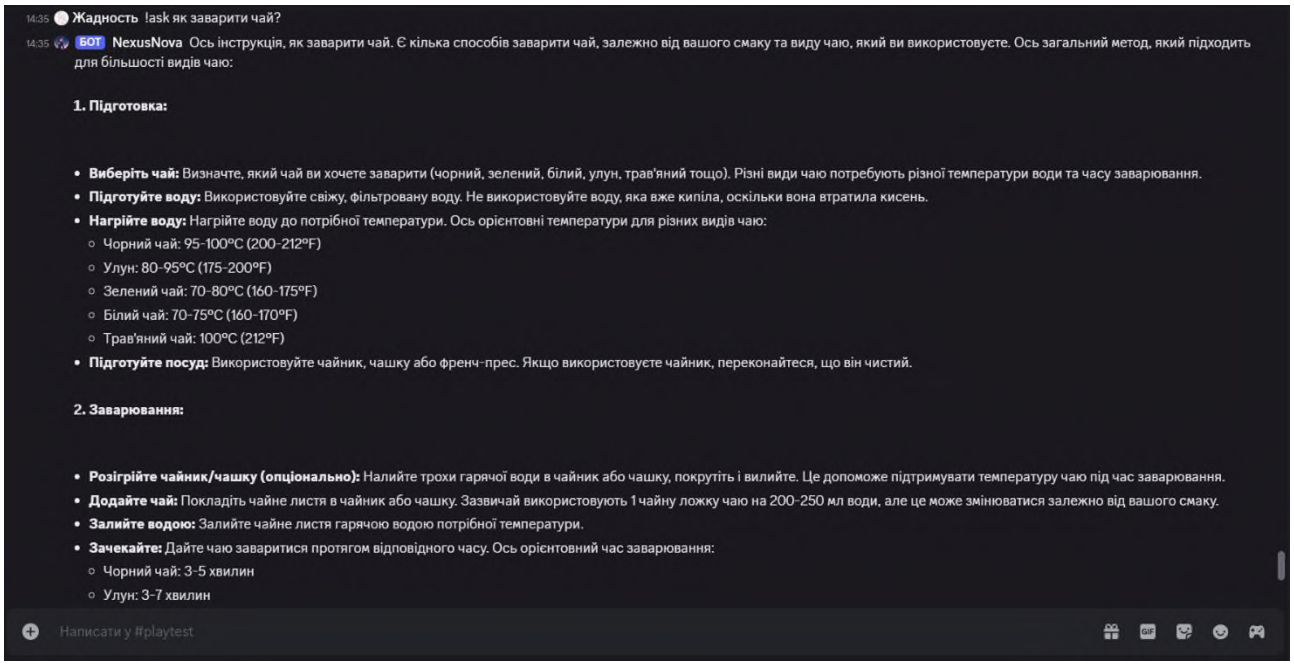


Рис. 3.9 Інтеграція з Gemini API

Для більшої зручності для користувачів бот починає відповідати просто, коли хтось пише його ім’я “Нова”. Також була введена система історії діалогу для кожного користувача, що дозволяє боту зберігати контекст попередніх повідомлень і давати більш осмислені та релевантні відповіді на подальші запити (рис. 3.10). Це значно покращує взаємодію з ботом, оскільки він може підтримувати логічний зв’язок між повідомленнями, створюючи відчуття реального діалогу, замість того, щоб кожен запит сприймався як нове питання без контексту. В результаті, користувачі отримують більш персоналізовані та точні відповіді, що покращує досвід взаємодії з ботом. Крім цього, механізм збереження контексту дає змогу боту адаптувати відповіді відповідно до стилю спілкування конкретного користувача, покращуючи природність взаємодії. Це особливо корисно у довготривалих діалогах, де користувач може повернутися до попередніх тем без необхідності повторно вводити всю інформацію (додаток Б).

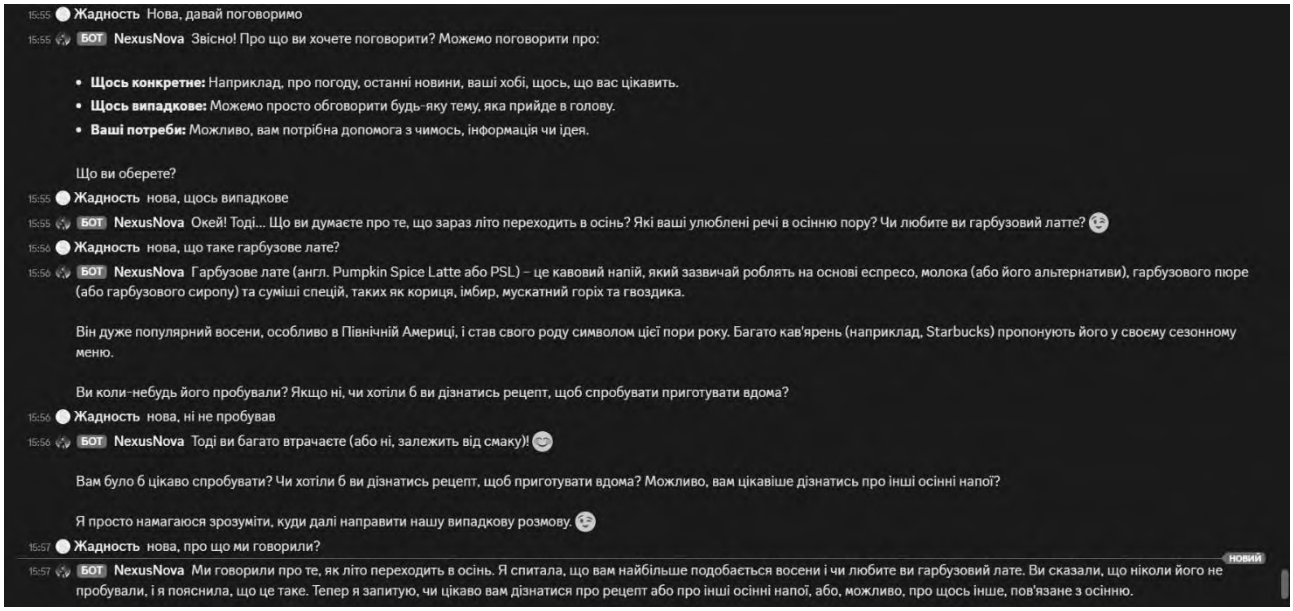


Рис. 3.10 Діалог з ботом

Також було вирішено додати функцію аналізу чату. З історії спілкування береться 100 останніх повідомлень і відправляється Gemini API з запитом на аналіз їхніх даних і формування короткої теми спілкування. Для зручнішого використання і більш натурального вигляду, функція аналізу запускається фразою: “Нова, проаналізуй чат”. Цей функціонал дозволяє отримати швидке резюме останніх подій або обговорень у каналі, що особливо корисно для

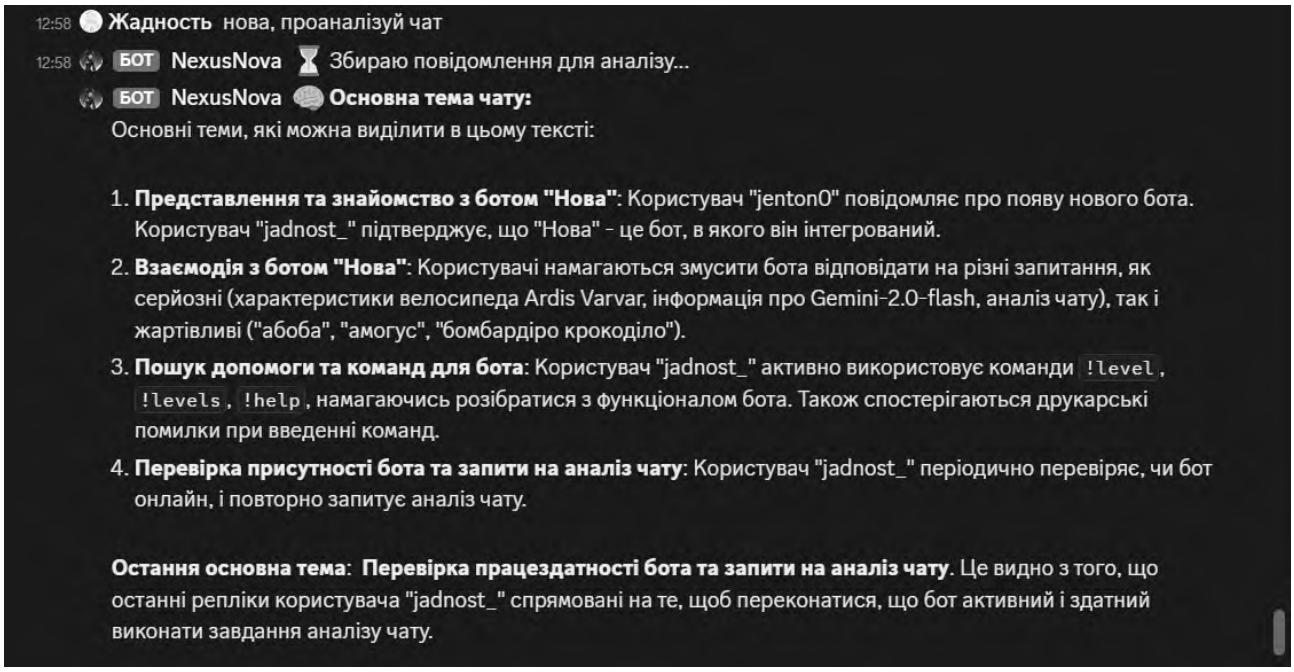


Рис. 3.11 Аналіз чату

нових учасників, які щойно приєдналися до дискусії, або для користувачів, які тривалий час були неактивними (рис. 3.11, додаток В).

Бот має можливість аналізувати відправлені .txt файли для подальшої роботи з ними. Достатньо просто відправити його в чат і вказати що потрібно з ним зробити. Це може значно полегшити роботу з файлами на сервері (рис. 3.12). Бот також може обробляти текстові файли для складнішого аналізу, наприклад, витягати певні дані, конвертувати їх у таблиці або здійснювати автоматичний переклад. Крім цього, він може перевіряти текст на орфографічні та граматичні помилки, що робить його корисним інструментом для роботи з документами (додаток Г).

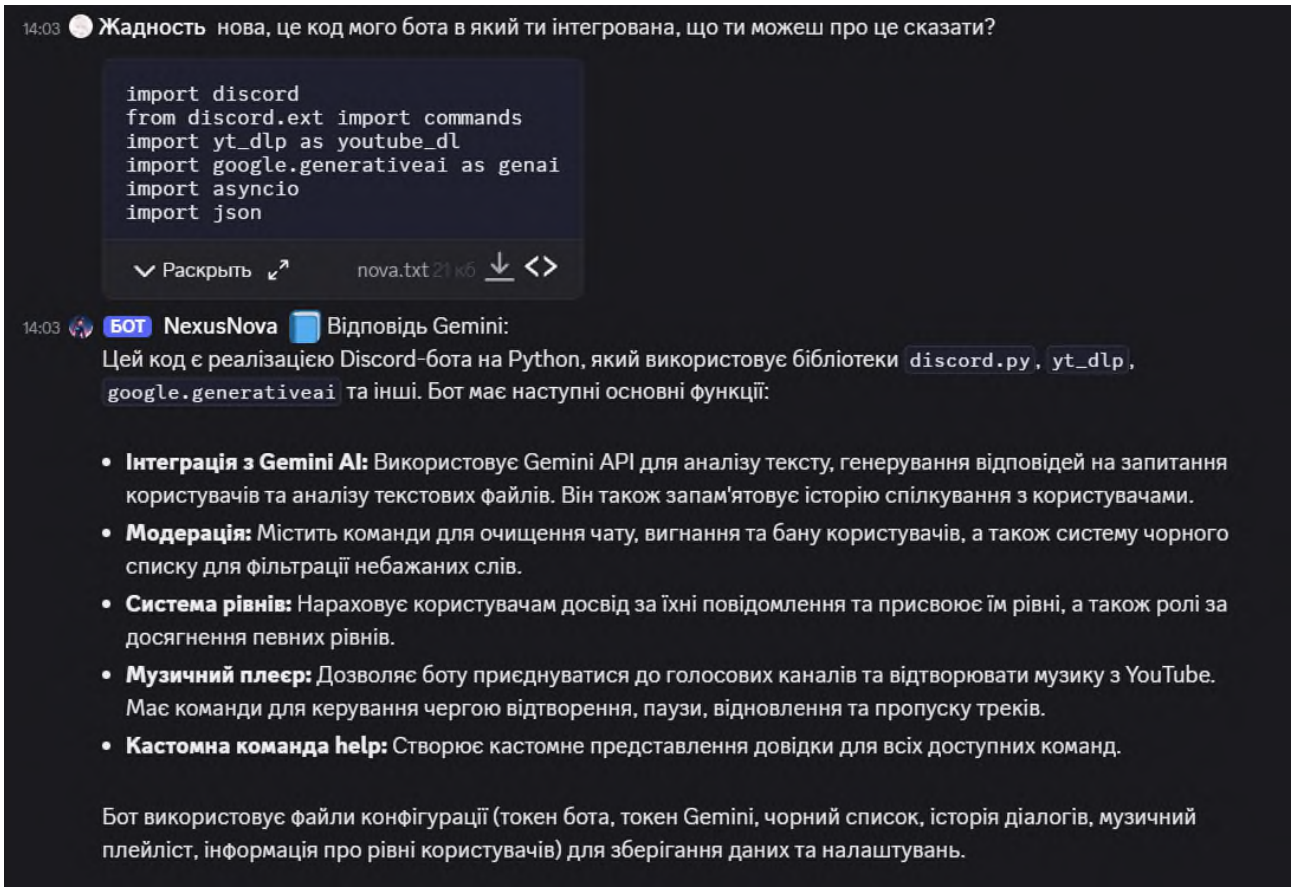


Рис. 3.12 Аналіз файлу

Для залучення користувачів активно спілкуватися в чаті бот має систему накопичення рівня. Рівень залежить від кількості відправлених повідомлень. Команда level покаже поточний рівень користувача, а команда levels виведе список рейтингу користувачів на сервері (рис. 3.13). Також бот кожні 10 рівнів

видає роль цього рівня. Якщо ролі немає, то він її автоматично створює. Така система рівнів стимулює активність у спільноті, роблячи спілкування більш інтерактивним та мотиваційним. Окрім звичайного підвищення рівня, можна додати додаткові бонуси, такі як доступ до ексклюзивних каналів, унікальні кольори для ніків або навіть спеціальні команди для високорівневих учасників. Автоматичне створення ролей кожні 10 рівнів забезпечує зручне керування привілеями без необхідності ручного адміністрування (додаток Д).

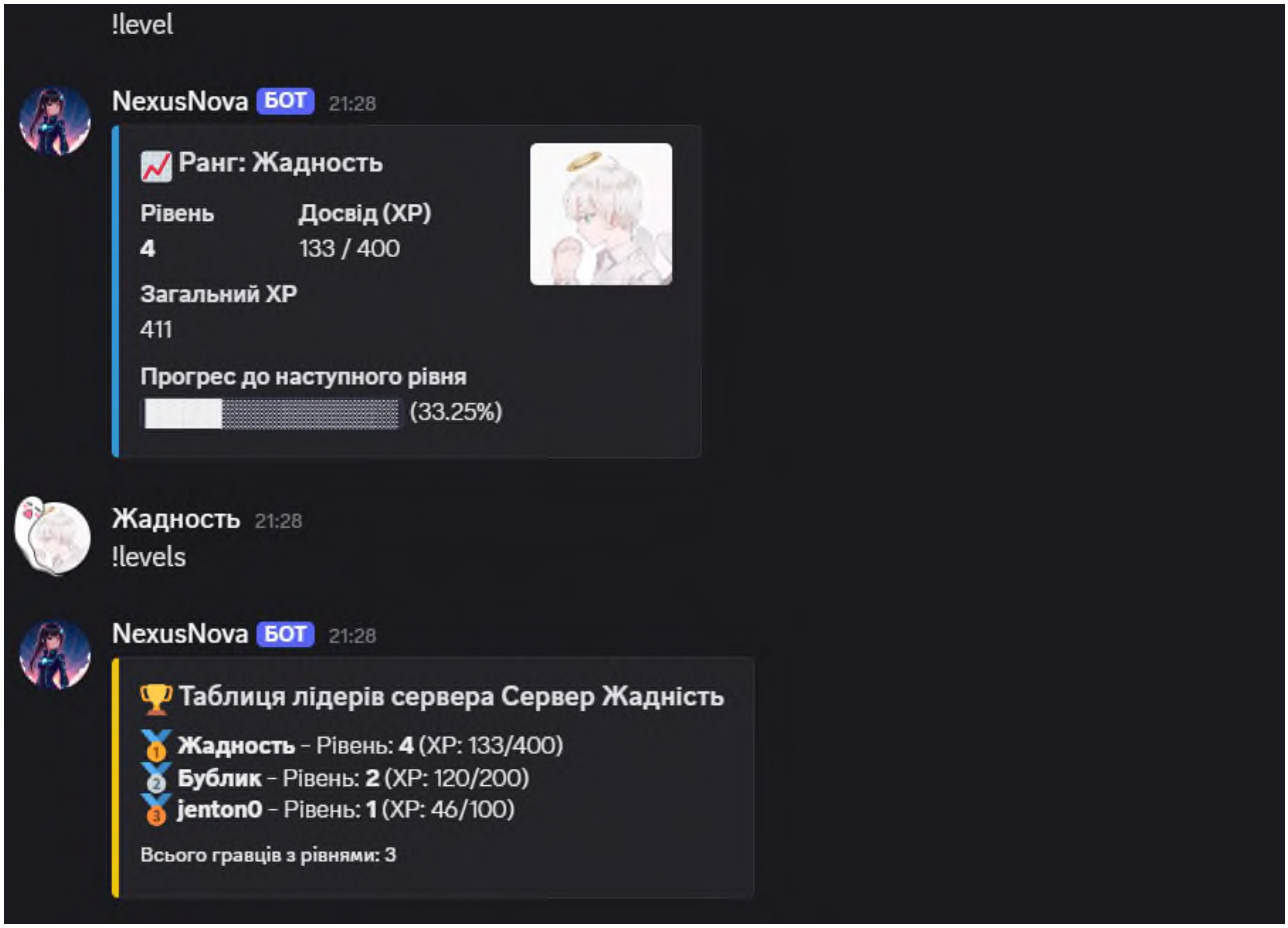


Рис. 3.13 Система рівнів і рейтингу

Для ще більшого заохочення користувачів до взаємодії з ботом було реалізовано спеціальний ког, присвячений немаловідомій настільній рольовій грі Dungeons and Dragons (D&D). У цьому модулі Discord-бот виступає в ролі Майстра Підземелля, використовуючи можливості Gemini API для створення динамічного, інтерактивного ігрового світу. Гравці отримують змогу повністю зануритися у фентезійну атмосферу: брати участь у захопливих пригодах, розгадувати таємниці древніх руїн, вести переговори з вигаданими персонажами,

вступати в битви з монстрами, знаходити артефакти й скарби, а також відчути себе справжніми героями, що формують долю вигаданого світу.

Однією з особливостей реалізації є те, що бот не обмежується заздалегідь визначеними сценаріями. Завдяки штучному інтелекту, кожна сесія гри є унікальною: Gemini API аналізує дії гравців і відповідно до них генерує нові сюжетні повороти, непередбачувані події, перешкоди та виклики. Це робить гру дійсно інтерактивною і захопливою, оскільки учасники не просто слідують написаній історії, а створюють її у реальному часі. Таким чином, бот фактично виконує функцію живого наратора, з яким можна взаємодіяти як із досвідченим Майстром Підземелля.

Інтеграція такого функціоналу в Discord-бота створює інноваційний підхід до цифрової рольової гри, поєднуючи потужність сучасного ШІ з класичною механікою D&D. У результаті користувачі отримують захопливий досвід, який розгортається просто в чаті Discord і не потребує сторонніх інструментів чи майданчиків (рис. 3.14). Це відкриває безмежний простір для творчості, імпровізації та занурення у світ епічних саг, які учасники створюють разом, крок за кроком.

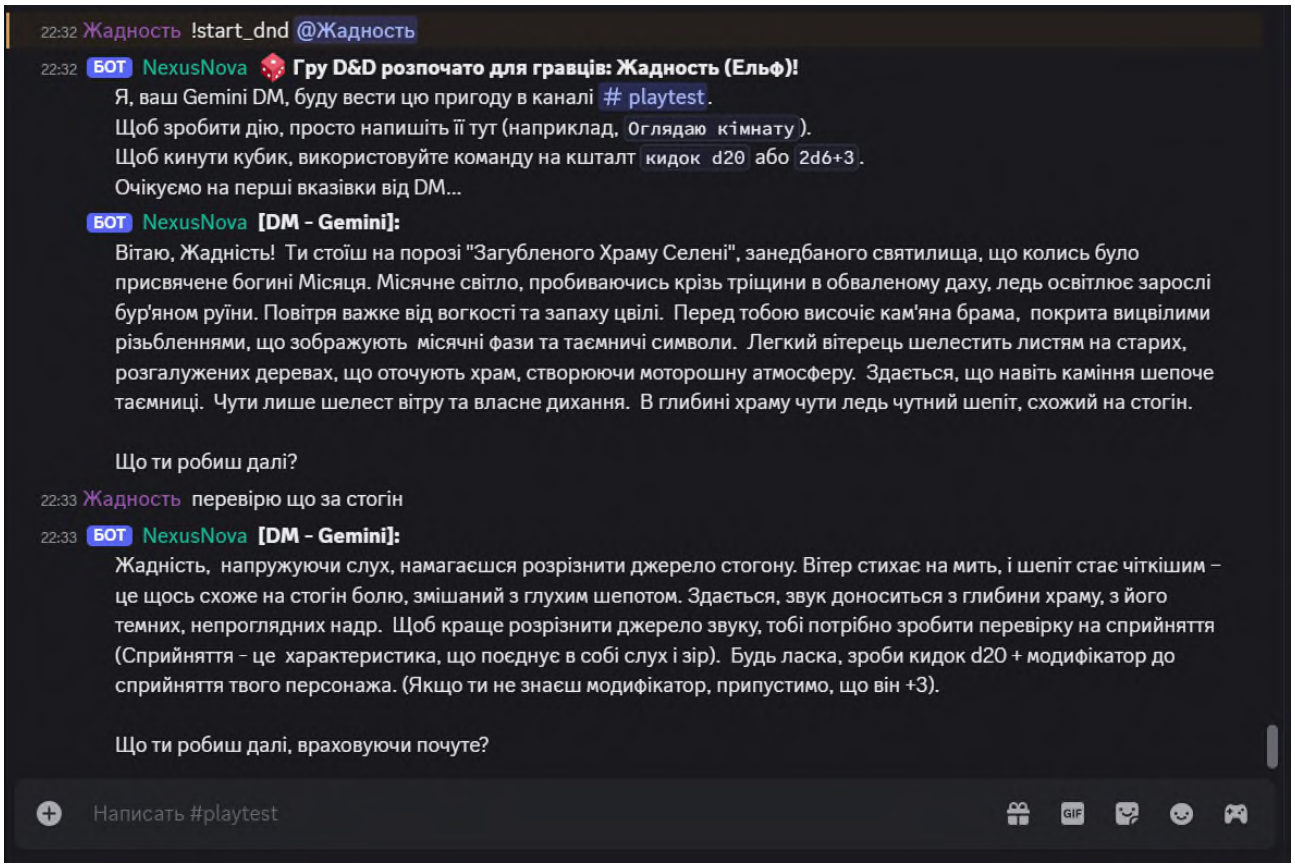


Рис. 3.14 Гра D&D

ВИСНОВКИ

У процесі виконання роботи на тему «Розробка Discord-бота» було досліджено, спроєктовано та реалізовано програмний засіб, що забезпечує інтерактивну взаємодію з користувачами Discord-серверів у режимі реального часу. Головна ідея була у створенні функціонального бота, здатного обробляти як текстові, так і голосові команди. Зокрема відтворювати музику та керувати чергою композицій.

Одним з основних завдань стало вивчення можливостей бібліотеки discord.py, яка виступає основним інтерфейсом для зв'язку з API Discord. Ця бібліотека дозволила легко та ефективно реалізовувати всі необхідні функції, такі як: обробкою подій, реакцією на повідомлення та взаємодією з голосовими каналами. Асинхронний характер роботи бібліотеки дозволив глибше зануритись у концепцію async/await у Python, що стало важливою частиною практичного досвіду.

Під час розробки було реалізовано підтримку відтворення музики з YouTube, використовуючи інструменти yt-dlp та FFmpeg. Бот отримав підтримку основних музичних команд: відтворення, пауза, відновлення, пропуск, зупинка. А також автоматичне покидання голосового каналу після завершення черги. Для шифрування та передачі голосових даних було використано бібліотеку PyNaCl, яка відповідає вимогам Discord до обміну голосовими даними.

Окрему увагу приділено структурі проєкту та зберіганню даних. Для зручності було впроваджено механізм логування подій та помилок, а також системи контролю доступу до окремих функцій бота. Так деякі команди доступні лише адміністраторам або користувачам з певними правами. Такий підхід дозволяє уникнути зловживання ботом і забезпечити належний рівень безпеки при його використанні.

Також були розглянуті ймовірні нестабільності в роботі голосових функцій, пов'язані з особливостями мережевої інфраструктури Discord або проблемами зі сторонніми сервісами. Для цього реалізовано систему обробки

винятків і повідомлень користувачу, що дозволяє уникнути некоректної поведінки та дає зворотний зв’язок.

Результати створення бота підтверджують, що у межах навчального проєкту можна створити функціональний і зручний у користуванні інструмент. При цьому він здатен не лише задовольнити основні потреби користувачів Discord-серверів, а й бути основою для майбутнього вдосконалення.

Використана система баз даних для збереження інформації про користувачів. Зокрема для збереження рівнів для системи рейтингу, список треків для відтворення в черзі, історії спілкування з ШІ. А також такі дані як токени для запуску бота та інтеграції з Gemini були збережені в файлах для їх конфіденційності під час використання в коді.

Під час виконання роботи вдалося поглибити розуміння принципів клієнт-серверної взаємодії, роботи з API, обробки подій, синхронізації потоків та керування ресурсами. Ці знання є надзвичайно важливими для майбутньої професійної діяльності у сфері ІТ.

Варто зазначити, що робота над проєктом сприяла розвитку навичок самостійного навчання, а також роботи з технічною документацією. У ході реалізації неодноразово виникали нетипові ситуації, вирішення яких вимагало звернення до офіційної документації, пошуку рішень у спільнотах, експериментів з кодом і налагодження.

Поставлену мету кваліфікаційної роботи досягнуто повністю. Було створено програмний продукт. Це демонструє ефективне використання сучасних інструментів, знання мов програмування, бібліотек і загальних принципів створення асинхронних мережевих застосунків. Отриманий результат має практичне і навчальне значення, а робота може слугувати основою для майбутніх проєктів у сфері автоматизації, розробки чат-ботів або розширення функціональності соціальних платформ.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Asynchronous HTTP Client/Server for asyncio and Python. URL: https://docs.aiohttp.org/ (Дата звернення: 20.04.2025)

2. asyncpg. URL: https://magicstack.github.io/asyncpg (Дата звернення: 21.04.2025)

3. Discord API Documentation. URL: https://discord.com/developers/docs (Дата звернення: 21.04.2025)

4. Discord Developer Portal. Getting Started. URL: https://discord.com/developers/docs/intro (Дата звернення: 20.04.2025)

5. discord.py Documentation. URL: https://discordpy.readthedocs.io/ (Дата звернення: 21.04.2025)

6. Discord.py Examples Repository. URL: https://github.com/Rapptz/discord.py (Дата звернення: 25.04.2025)

7. Docker Documentation. URL: https://docs.docker.com/ (Дата звернення: 02.05.2025)

8. European Commission Joint Research Centre. Digital Education Policies in Europe. URL: https://joint-research-centre.ec.europa.eu (Дата звернення: 20.04.2025)

9. FastAPI – Modern web framework for Python. URL: https://fastapi.tiangolo.com/ (Дата звернення: 21.04.2025)

10. GitHub. Discord.py Examples Repository. URL: https://github.com/Rapptz/discord.py (Дата звернення: 20.04.2025)

11. Glynska T. B. Інформаційні системи в економіці. Київ: КНЕУ, 2020. 240 с.

12. Heroku Documentation. URL: https://devcenter.heroku.com/ (Дата звернення: 02.05.2025)

13. IEEE Computer Society. Trends in Chatbot Technology and Use. URL: https://ieeexplore.ieee.org (Дата звернення: 20.04.2025)

14. interactions.py – Modern Discord API wrapper. URL: https://github.com/interactions-py/library (Дата звернення: 21.04.2025)

15. ISO/IEC 2382:2015. Information technology – Vocabulary. International Organization for Standardization.

16. Jenkins T. Designing Bots for Interaction. New York: Apress, 2020. 198 p.

17. Kim H., Park Y. Adaptive Chatbots for E-Learning Environments. *Journal of Educational Technology*. 2021. Vol. 38(4). P. 233-245.

18. Learn all about Midjourney's Discord features and start chatting! URL: https://docs.midjourney.com/hc/en-us/articles/33330535666445-Discord-Overview (Дата звернення: 20.04.2025)

19. nextcord – An enhanced fork of discord.py. URL: https://nextcord.dev (Дата звернення: 21.04.2025)

20. Nextcord Documentation. URL: https://docs.nextcord.dev/ (Дата звернення: 20.04.2025)

21. OAuth 2.0 Overview. URL: https://oauth.net/2/ (Дата звернення: 21.04.2025)

22. OAuth 2.0 Authorization Framework. URL: https://datatracker.ietf.org/doc/html/rfc6749 (Дата звернення: 25.04.2025)

23. OpenAI API Reference. URL: https://platform.openai.com/docs/ (Дата звернення: 25.04.2025)

24. PostgreSQL Documentation. URL: https://www.postgresql.org/docs/ (Дата звернення: 25.04.2025)

25. Pydantic Documentation. URL: https://docs.pydantic.dev/ (Дата звернення: 20.04.2025)

26. PyNaCl – Python binding to the Networking and Cryptography library. URL: https://pynacl.readthedocs.io (Дата звернення: 05.05.2025)

27. Python Software Foundation. Python 3.11 Documentation. URL: https://docs.python.org/3.11/

28. Slack API Documentation. URL: https://api.slack.com/ (Дата звернення: 21.04.2025)

29. SQLite. Lightweight Database Engine. URL: https://www.sqlite.org/ (Дата звернення: 25.04.2025)

30. Telegram Bot API. URL: https://core.telegram.org/bots/api (Дата звернення: 02.05.2025)

31. Walker J**.** Modern Programming Paradigms with Python. Boston: O’Reilly Media, 2021. 312 p.

32. YouTube Data API v3. URL: https://developers.google.com/youtube/v3 (Дата звернення: 21.04.2025)

33. Зінченко О.П. Автоматизація цифрових платформ. Дніпро: ДНУ, 2022. 189 с.

34. Зозуля П.М. Технології клієнт-серверної архітектури. Київ: КПІ, 2023. 195 с.

35. Литвин В.В., Яковенко В.М. Програмування мовою Python. Львів: ЛьвДУ, 2021. 342 с.

36. Літвінов І.Б. Прикладне програмування в мережі. Харків: ХНУРЕ, 2023. 301 с.

37. Маклаков С.В. Створення чат-ботів за допомогою Python. Харків: Фоліо, 2020. 188 с.

38. Матвієнко А.С. Безпека веб-ресурсів. Київ: КНУ, 2021. 216 с.

39. Плотніков Ю.С. Основи програмної інженерії. Львів: ЛПІ, 2020. 280 с.

40. Явдоха С. Ю. Веб-програмування. Одеса: ОНУ, 2021. 220 с.

ДОДАТКИ

Додаток А

```
# Музичні команди
@bot.command(name="music", help="Список музичних команд")
async def help_music(ctx):
    embed = discord.Embed(title="Музичні команди", color=discord.Color.blue())
    music_commands = """
play <URL> – Відтворити трек
resume – Продовжити після паузи
stop – Поставити на паузу
skip – Пропустити трек
cmlist – Очистити чергу
music_list – Показати чергу
"""
    embed.add_field(name="Команди", value=music_commands, inline=False)
    await ctx.send(embed=embed)
@bot.command(help="Приєднатися до голосового каналу")
async def join(ctx):
    if ctx.author.voice:
        await ctx.author.voice.channel.connect()
    else:
        await ctx.send("Будь ласка, приєднайтесь до голосового каналу!")
@bot.command(help="Вийти з голосового каналу")
async def leave(ctx):
    if ctx.voice_client:
        await ctx.voice_client.disconnect()
    else:
        await ctx.send("Я не в голосовому каналі!")
@bot.command(help="Відтворити трек")
async def play(ctx, url: str = None):
```

```
    if url is None:
        await ctx.send(" Не вказаний URL для відтворення.")
        return
    if not ctx.voice_client:
        if ctx.author.voice:
            await ctx.author.voice.channel.connect()
        else:
            await ctx.send("Потрібно бути у голосовому каналі.")
            return
    queue.append(url)
    save_music_list()
    if not ctx.voice_client.is_playing():
        await play_next(ctx)
@bot.command(help="Продовжити відтворення після паузи")
async def resume(ctx):
    if ctx.voice_client and ctx.voice_client.is_paused():
        ctx.voice_client.resume()
        await ctx.send("Відновлено!")
@bot.command(help="Поставити музику на паузу")
async def stop(ctx):
    if ctx.voice_client and ctx.voice_client.is_playing():
        ctx.voice_client.pause()
        await ctx.send("⏸ Пауза.")
@bot.command(help="Пропустити поточний трек")
async def skip(ctx):
    if ctx.voice_client and ctx.voice_client.is_playing():
        ctx.voice_client.stop()
        await ctx.send("⏭ Пропущено!")
@bot.command(help="Очистити чергу треків")
```

```
async def cmlist(ctx):
    queue.clear()
    save_music_list()
    await ctx.send("🗑 Черга очищена!")
@bot.command(help="Показати список треків у черзі")
async def music_list(ctx):
    if not queue:
        await ctx.send("Черга порожня!")
        return
    embed = discord.Embed(title="🎶 Черга треків", color=discord.Color.purple())
    for i, url in enumerate(queue, 1):
        embed.add_field(name=f"Трек {i}", value=url, inline=False)
```

```
def load_json_data(filepath):
    try:
        with open(filepath, ‘r’, encoding=‘utf-8’) as file:
            data = json.load(file)
            return data if isinstance(data, dict) else {}  # Ensure it’s a dict
    except FileNotFoundError:
        return {}  # Return empty dict if file is not found
    except json.JSONDecodeError:
        return {}  # Return empty dict if file is corrupted
    except Exception as e:
        print(f"Error loading {filepath}: {e}")
        return {}  # Return empty dict on other errors
def save_json_data(filepath, data):
    with open(filepath, ‘w’, encoding=‘utf-8’) as file:
        json.dump(data, file, ensure_ascii=False, indent=4)
queue = []
user_conversations = load_json_data(history_file)
important_memory = load_json_data(important_memory_file)


if re.search(r"\bнова\b", message.content.lower()):
        await message.channel.typing()
        try:
            # Об’єднуємо історії: важлива пам’ять та тимчасову історію спілкування
            combined_history = []
            if user_id in important_memory:
                combined_history.extend(important_memory[user_id])
            if user_id in user_conversations:
                combined_history.extend(user_conversations[user_id])
            # Можна додатково відсортувати історію за часовими мітками,
            # якщо зберігати їх при записі, але наразі використовується просто
об’єднання.
            # Стартуємо чат із заданою історією
            chat = model.start_chat(history=combined_history)
            response = chat.send_message(message.content)
```

```
        reply = response.text.strip()

        # Зберігаємо саме нове звернення у user_conversations (як тимчасова історія)
        if user_id not in user_conversations:
            user_conversations[user_id] = []
        user_conversations[user_id].append({
            "role": "user",
            "parts": [message.content]
        })
        user_conversations[user_id].append({
            "role": "model",
            "parts": [reply]
        })
        save_json_data(history_file, user_conversations)
        # Вивід відповіді у Discord (якщо текст занадто довгий, розбиваємо його на частини)
        if len(reply) > 2000:
            chunks = [reply[i:i + 2000] for i in range(0, len(reply), 2000)]
            for chunk in chunks:
                await message.channel.send(chunk)
        else:
            await message.channel.send(reply)

    except Exception as e:
        await message.channel.send("⚠ Сталася помилка при зверненні до Gemini API.")
        print(f"Gemini error: {e}")
```

```
# Перевірка на команду аналізу чату
    if re.search(r"\bнова[,] проаналізуй чат\b", message.content.lower()):
        await message.channel.typing()  # Відправляємо типування, щоб повідомити, що йде обробка
        await analyze(message.channel, limit=100)
        return
    await bot.process_commands(message)
#блок аналізу чату
async def analyze(ctx, limit: int = 20):
    await ctx.typing()
    await ctx.send("⏳ Збираю повідомлення для аналізу...")
    messages = []
    async for msg in ctx.history(limit=limit):
        if not msg.author.bot:
            messages.append(f"{msg.author.name}: {msg.content}")
    messages.reverse()
    combined_text = '\n'.join(messages)
    # Prompt з конкретним запитом на тему
    prompt = (
        "Проаналізуй цей текст та визнач, про що йшла мова. "
        "Сформулюй останню основну тему серед цього тексту:\n\n"
        f"{combined_text}"
    )
    try:
        async with ctx.typing():
            loop = asyncio.get_event_loop()
            response = await loop.run_in_executor(None, model.generate_content, prompt)
            reply = response.text.strip()
```

```
        if not reply:
            await ctx.send("⚠ Не вдалося визначити тему обговорення.")
            return
        # Надсилання відповіді
        if len(reply) > 2000:
            chunks = [reply[i:i + 2000] for i in range(0, len(reply), 2000)]
            for chunk in chunks:
                await ctx.send(chunk)
        else:
            await ctx.send(f"🧠 **Основна тема чату:**\n{reply}")

except Exception as e:
    await ctx.send("⚠ Сталася помилка при зверненні до Gemini API.")
    print(f"Gemini analyze error: {e}")
```

```
# Аналіз .txt файлу
    if message.attachments:
        for attachment in message.attachments:
            if attachment.filename.endswith(".txt"):
                async with message.channel.typing():
                    try:
                        file_bytes = await attachment.read()
                        file_text = file_bytes.decode("utf-8")
                        prompt = (
                            "Проаналізуй вміст цього текстового файлу та дай короткий
зміст:\n\n"
                            f"{file_text}"
                        )
                        loop = asyncio.get_event_loop()
                        response = await loop.run_in_executor(None, model.generate_content,
prompt)
                        reply = response.text.strip()
                        if not reply:
                            await message.channel.send("⚠ Gemini не повернув зміст.")
                            return
                        if len(reply) > 2000:
                            chunks = [reply[i:i + 2000] for i in range(0, len(reply), 2000)]
                            for chunk in chunks:
                                await message.channel.send(chunk)
                        else:
                            await message.channel.send(f"📄 Відповідь Gemini:\n{reply}")
                    except Exception as e:
                        await message.channel.send("⚠ Не вдалося обробити файл.")
```

```
        print(f"File/Gemini error: {e}")
    return
```

```
    @commands.command(name="level", aliases=["lvl", "rank"], help="Показати ваш поточний рівень та XP.")
    @commands.cooldown(1, 5, commands.BucketType.user) # Затримка 5 секунд на користувача
    async def display_level(self, ctx: commands.Context, member: discord.Member = None):
        """Відображає рівень та XP вказаного користувача або автора повідомлення."""
        await ctx.typing()
        target_member = member or ctx.author
        if target_member.bot:
            await ctx.send("🤖 Боти не мають рівнів.")
            return
        guild_id = str(ctx.guild.id)
        user_id = str(target_member.id)
        levels_data = await db.load_levels_data()
        if guild_id in levels_data and user_id in levels_data[guild_id]:
            user_data = levels_data[guild_id][user_id]
            current_level = user_data["level"]
            current_xp = user_data["xp"]
            xp_for_next = required_xp_for_level(current_level)
            total_xp = user_data.get("total_xp", current_xp) # Для зворотної сумісності
            embed = discord.Embed(
                title=f"📈 Ранг: {target_member.display_name}",
                color=target_member.color if target_member.color != discord.Color.default() else discord.Color.blue()
            )
            embed.set_thumbnail(url=target_member.display_avatar.url)
            embed.add_field(name="Рівень", value=f"**{current_level}**", inline=True)
            embed.add_field(name="Досвід (XP)", value=f"{current_xp} / {xp_for_next}", inline=True)
            embed.add_field(name="Загальний XP", value=f"{total_xp}", inline=False)
            # Прогрес-бар (простий текстовий)
            progress_percentage = (current_xp / xp_for_next) * 100
            bar_length = 20 # Довжина прогрес-бару
            filled_length = int(bar_length * current_xp // xp_for_next)
            bar = '█' * filled_length + '░' * (bar_length - filled_length)
            embed.add_field(name="Прогрес до наступного рівня", value=f"`{bar}` ({progress_percentage:.2f}%)", inline=False)
            await ctx.send(embed=embed)
        else:
```

```
        await ctx.send(f"{target_member.mention}, ви ще не маєте рівня на цьому сервері. Напишіть щось, щоб почати!")

    @commands.command(name="levels", aliases=["leaderboard", "top"], help="Показати таблицю лідерів за рівнями на сервері.")
    @commands.cooldown(1, 10, commands.BucketType.guild) # Затримка 10 секунд на сервер
    async def display_leaderboard(self, ctx: commands.Context):
        """Відображає топ-10 користувачів за рівнями на сервері."""
        await ctx.typing()
        guild_id = str(ctx.guild.id)
        levels_data = await db.load_levels_data()

        if guild_id not in levels_data or not levels_data[guild_id]:
            await ctx.send("На цьому сервері ще ніхто не має рівня.")
            return
        # Сортуємо користувачів за рівнем, потім за XP, потім за загальним XP
        # (x[1].get('total_xp', 0) для зворотної сумісності)
        sorted_users = sorted(
            levels_data[guild_id].items(),
            key=lambda item: (item[1]['level'], item[1]['xp'], item[1].get('total_xp', 0)),
            reverse=True
        )
        embed = discord.Embed(
            title=f"🏆 Таблиця лідерів сервера {ctx.guild.name}",
            color=discord.Color.gold()
        )
        embed.set_thumbnail(url=ctx.guild.icon.url if ctx.guild.icon else None)
        description_lines = []
        rank_emojis = ["🥇", "🥈", "🥉"] # Емодзі для перших трьох місць

        for i, (user_id_str, data) in enumerate(sorted_users[:10]): # Топ-10
            user = ctx.guild.get_member(int(user_id_str)) # Отримуємо об'єкт Member
            if user:
                name = user.display_name
            else: # Якщо користувач покинув сервер
                name = f"Користувач {user_id_str} (покинув сервер)"
            rank_display = rank_emojis[i] if i < len(rank_emojis) else f"{i + 1}."
            description_lines.append(
                f"{rank_display} **{name}** - Рівень: **{data['level']}** (XP: {data['xp']}/{required_xp_for_level(data['level'])})"
            )
        if not description_lines:
```

```
        embed.description = "Немає даних для відображення."
    else:
        embed.description = "\n".join(description_lines)
          embed.set_footer(text=f"Всього гравців з рівнями:
{len(levels_data[guild_id])}")
    await ctx.send(embed=embed)
```