МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ІНЖЕНЕРНИЙ НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ
ім. Ю.М. Потебні
ЗАПОРІЗЬКОГО НАЦІОНАЛЬНОГО УНІВЕРСИТЕТУ

КАФЕДРА ЕЛЕКТРОНІКИ, ІНФОРМАЦІЙНИХ СИСТЕМ ТА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Кваліфікаційна робота

перший (бакалаврський)
(рівень вищої освіти)

на тему **Багатофункціональний бот-модератор на платформі Discord**

Виконав: студент 4 курсу, групи 6.1211-пзс
спеціальності 121 Інженерія програмного забезпечення
(код і назва спеціальності)
освітньої програми Програмне забезпечення систем
(код і назва освітньої програми)
О.Я. Жевнерюк
(підпис, ініціали та прізвище)
Керівник доцент, к.ф.-м.н., доцент І.А. Скрипник
(посада, вчене звання, науковий ступінь, підпис, ініціали та прізвище)
Рецензент директор ТОВ «Дискус»
Р.О. Лютий
(посада, вчене звання, науковий ступінь, підпис, ініціали та прізвище)

Запоріжжя
2025

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ІНЖЕНЕРНИЙ НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ
ім. Ю.М. Потебні
ЗАПОРІЗЬКОГО НАЦІОНАЛЬНОГО УНІВЕРСИТЕТУ

Кафедра Електроніки, інформаційних систем та програмного забезпечення
Рівень вищої освіти _перший (бакалаврський)_
Спеціальність 121 Інженерія програмного забезпечення
(код та назва)
Освітня програма Програмне забезпечення систем
(код та назва)

ЗАТВЕРДЖУЮ

Завідувач кафедри_____Тетяна КРИТСЬКА

"_01_" __березня_____2025_року

З А В Д А Н Н Я
НА КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТОВІ

Жевнерюку Олександру Яковлевичу
(прізвище, ім'я, по батькові)

1.Тема роботи Багатофункціональний бот-модератор на платформі Discord
керівник роботи Скрипник Ірина Анатоліївна, к.ф.-м.н., доцент
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)
затверджені наказом ЗНУ від 27.12.2024 р. № 2029-с
2. Строк подання студентом кваліфікаційної роботи 02.06.2025
3. Вихідні дані бакалаврської роботи
- комплект нормативних документів ;
- технічне завдання до роботи.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)
- огляд та збір літератури стосовно теми кваліфікаційної роботи;
- огляд та аналіз існуючих рішень та аналогів;
- дослідження засобів програмної реалізації бота;
- створення програмного продукту та його опис;
- дослідження поставленої проблеми та розробка висновків.

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

6. Консультанти розділів бакалаврської роботи

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

7. Дата видачі завдання 01.03.2025

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів бакалаврської роботи	Строк виконання етапів бакалаврської роботи	Примітка
1	Аналіз предметної області	01.03.2025	виконано
2	Формулювання основної задачі дипломної роботи та узгодження її з науковим керівником	15.03.2025	виконано
3	Аналіз існуючих методів рішення	05.05-07.05.2025	виконано
4	Дослідження засобів реалізації	07.05-08.05.2025	виконано
5	Узгодження подальших дій з науковим керівником	09.05.2025	виконано
6	Розробка основних модулів застосунку	09.05-12.05.2025	виконано
7	Інтеграція БД	12.05-13.05.2025	виконано
8	Програмна реалізація, дописано код застосунку	13.05-24.05.2025	виконано
9	Тестування застосунку	24.05-25.05.2025	виконано
10	Представлення отриманих результатів науковому керівнику та узгодження плану подальшого дослідження	26.05.2025	виконано
11	Перевірка роботоздатності проєкту, виправлення помилок	26.05-27.05.2025	виконано
12	Оформлення звіту	27.05-01.06.2025	виконано
13	Оформлення презентації. Отримання рецензій від опонентів.	01.06.2025	виконано

Студент _____ Жевнерюк О. Я.
(підпис) (прізвище та ініціали)

Керівник роботи _____ Скрипник І.А.
(підпис) (прізвище та ініціали)

Нормоконтроль пройдено

Нормоконтролер _____ Скрипник І.А.
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Сторінок: - 70,

Рисунків: -13,

Джерел: - 13.

Жевнерюк О. Я. Багатофункціональний бот-модератор на платформі Discord: кваліфікаційна робота бакалавра спеціальності 121 «Програмне забезпечення систем» / наук. Керівник І.А. Скрипник. Запоріжжя : ЗНУ, 2025.

У сучасних умовах управління великими онлайн-спільнотами вимагає автоматизації рутинних процесів для забезпечення порядку, ефективної взаємодії з учасниками та підтримки активності. Discord-боти є зручним інструментом для реалізації подібних задач, поєднуючи модераційні функції, інтеграцію з медіа та утилітарні можливості.

Мета проєкту — розробка багатофункціонального Discord-бота для автоматизації модерації, призначення ролей, привітання новачків тощо. Основним завданням є спрощення адміністрування Discord-серверів та підвищення залученості учасників за допомогою інтерактивних інструментів.

Проведено аналіз існуючих ботів-аналогів та вибрано оптимальні технології розробки. Реалізовано модульний бот із використанням мови програмування Python, бібліотеки discord.py та сторонніх сервісів, таких як YouTube, Spotify та API. Система протестована на реальному Discord-сервері, де підтверджено її стабільність, зручність і готовність до розгортання у будь-якій спільноті.

Ключові слова: *Discord, бот, модерація, автоматизація, Python, асинхронність, музика, ролі, спільнота*

SUMMARY

Pages: - 70,

Drawings: - 13,

Sources: - 13.

Zhevneryuk O. Y. Multifunctional Bot-Moderator on Discord platform: qualification thesis of the bachelor of specialty 121 “System software” / Science. Manager I. A. Skrypnyk. Zaporizhzhya : ZNU, 2025.

In modern conditions, managing large online communities requires the automation of routine processes to ensure order, effective interaction with participants, and support activity. Discord bots are a convenient tool for implementing such tasks, combining moderation functions, media integration, and utilitarian capabilities.

The goal of the project is to develop a multifunctional Discord bot to automate moderation, assign roles, welcome newcomers, etc. The main task is to simplify the administration of Discord servers and increase participant engagement using interactive tools.

An analysis of existing similar bots was conducted and optimal development technologies were selected. A modular bot was implemented using the Python programming language, the discord.py library, and third-party services such as YouTube and Spotify API. The system was tested on a real Discord server, where its stability, convenience, and readiness for deployment in any community were confirmed.

Keywords: *Discord, bot, moderation, automation, Python, asynchrony, music, roles, community*

ЗМІСТ

ВСТУП 7
1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ 11
1.1 Визначення Discord-бота.....11
1.2 Історія розвитку 11
1.3 Аналіз програмних продуктів-аналогів.....13
1.4 Постановка задачі 16
2 ДОСЛІДЖЕННЯ ЗАСОБІВ ПРОГРАМНОЇ РЕАЛІЗАЦІЇ DISCORD-БОТА .18
2.1 Аналіз сучасних мов програмування для розробки Discord-бота 18
2.2 Аналіз сучасних систем керування базами даних для розробки Discord-бота 20
2.3 Огляд інших технологій для розробки Discord-бота.....21
3 РОЗРОБКА DISCORD-БОТА МОДЕРАТОРА 25
3.1 Опис предметної області.....25
3.2 Архітектура системи.....26
3.3 Архітектура бази даних.....27
3.4 Функціональні вимоги системи.....29
3.5 Вимоги до програмного та апаратного забезпечення 32
3.6 Модулі та алгоритми 33
3.7 Структури даних 60
3.8 Проєкт інтерфейсу 62
3.9 Реалізація та тестування.....63
ВИСНОВКИ.....68
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ 69

ВСТУП

Актуальність теми

У сучасному цифровому світі, соціальні платформи та комунікаційні сервіси стають невід'ємною частиною повсякденного життя. Однією з таких платформ є Discord, який набуває все більшої популярності з кожним роком. Спочатку спрямований на геймерів Discord стає платформою для всіх, нерідко можна побачити як зараз і навчальні процеси впроваджуються та проводяться в Discord'і. Розвиток популярності можна віднести до розвитку інформаційних технологій в світі та потребі у спілкуванні на відстані.

Актуальність розробки для Discord-бота для модерації в свою чергу обумовлена такими ключовими аспектами як:

1. Зростання популярності. Постійний наплив користувачів впливає на створення все більшої кількості активних серверів на платформі Discord. Це створює нові потреби у адміністраторах чи керівниках для серверів та нові виклики для цих людей, які мають забезпечити належний рівень модерації та порядку у великих спільнотах.
2. Необхідність автоматизації. Ручна модерація великих груп та спільнот стає все більш складною та трудомісткою задачею. Використання бота для таких цілей дозволяє автоматизувати[1] багато процесів, таких як фільтрація небажаного контенту, управління ролями користувачів, попередження та бани. Це значно знижує навантаження на адміністраторів та модераторів серверів.
3. Безпека та комфорт. Забезпечення безпеки та комфортного середовища для користувачів є ключовим фактором успішного функціонування будь-якого сервера. Виходячи з попереднього пункту – бот для модерації допомагає підтримувати високий рівень безпеки, видаляючи небажаний контент та попереджуючи порушення, що сприяє створенню позитивної атмосфери на

сервері.

4. Підвищення ефективності. Автоматизовані інструменти бота можуть працювати цілодобово без перерв, що забезпечить постійний контроль та дотриманням правил на сервері. Також усі логи та система журналу аудита дозволять швидко реагувати на порушення та запобігати недотриманню правил користувачами сервера.

Мета і завдання дослідження

Мета полягає у розробці швидкого та зручного застосунку, який дозволить адміністраторам серверів в Discord ефективно керувати своїми серверами без необхідності постійного втручання. Цей Discord-бот має бути здатний автоматизувати процеси управління серверами, забезпечити ефективний контроль за дотриманням правил, підвищити безпеку та комфорт користувачів на серверах платформи Discord.

Об’єкт дослідження

Об’єктом дослідження є процеси та механізми модерації серверів, які функціонують на платформі Discord. Ці процеси включають управління користувачами, забезпечення безпеки та порядку, а також автоматизацію різних аспектів модерації для підвищення ефективності та надійності управління сервером.

Предмет дослідження

Предметом дослідження є розробка та впровадження Discord-бота для модерації та автоматизації сервера.

Методи дослідження

Теоретичні – методи, ціль існування яких визначається у встановлені прецедентів та визначенні зовнішніх зв’язків поміж ними. За їх допомоги вдається з’ясувати, що саме стало причиною виникнення цих зв’язків та навіщо вони потрібні.

Практичне значення одержаних результатів

Практичне значення одержаних результатів полягає у створенні багатофункціонального Discord-боту, здатного автоматизувати процеси модерації, взаємодії з користувачами та управління сервером без постійного втручання адміністрації. Розроблений програмний засіб може бути впроваджений у різноманітних Discord-спільнотах – від навчальних до геймерських – з метою зниження навантаження на модераторів, покращення комунікації, контролю поведінки учасників та підвищення загального рівня безпеки та взаємодії на сервері.

Отримані результати можуть бути використані як основа для подальшої розробки комерційного продукту для адміністраторів великих онлайн-спільнот.

Глосарій

Discord – Платформа для голосового, відео- та текстового спілкування в реальному часі.

Бот – Програмний агент, який виконує автоматичні дії за заданими командами.

Команда – Вказівка, яку користувач надсилає боту через повідомлення (наприклад, !mute).

Cog – Модуль у архітектурі бота Discord, що групує функціонал за призначенням.

Роль – Певна позначка на серверах Discord, що визначає права користувача на окремому сервері.

Модерація – Процес контролю та управління поведінкою користувачів на сервері.

База даних – Структуроване сховище даних, що використовується для зберігання та організації інформації про користувачів.

Мова програмування – це штучна мова, створена для передачі команд машинам, зокрема комп'ютерам.

API – Набір готових функцій, протоколів та інтерфейсів, який дозволяє програмам взаємодіяти між собою.

Логування – Процес запису дій, подій, помилок або іншої інформації під час виконання програми.

Модуль програми – Самостійна частина програмного забезпечення, яка реалізує певний функціонал і може бути повторно використана.

Реакція – це emoji, який прикріплено до певного повідомлення у чаті.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Визначення Discord-бота

Discord-бот – це програмний агент, створений для автоматизації взаємодії з користувачами на платформі Discord. Він працює на основі Discord API та реагує на текстові команди, події, повідомлення або інші дії, що відбуваються на сервері. Discord-боти можуть виконувати широкий спектр завдань: модерацію, управління ролями, автоматичне надсилання повідомлень, відтворення музики, інтеграцію з іншими сервісами, аналітику тощо.

Основна мета створення Discord-ботів полягає в автоматизації рутинних дій, підвищенні ефективності керування спільнотою та наданні учасникам додаткових функцій для зручної взаємодії. Завдяки гнучкості API та широкому обсягу бібліотек (наприклад, discord.py[4] для Python), бота та його функціональність можливо адаптувати під конкретні потреби серверу та легко збільшувати можливості бота.

У межах даної роботи Discord-бот розглядається як багатофункціональна система, що складається з окремих модулів (cogs), кожен з яких відповідає за певну частину логіки: модерація, привітання нових користувачів, музичне відтворення, обробка команд тощо. Такий підхід дозволяє не тільки масштабувати проєкт, а й забезпечити його підтримку та подальший розвиток.

1.2 Історія розвитку

У 2015 році програміст Джейсон Цитрон намагався що є сили пробитися в індустрію відеоігор. На жаль його нова гра, яку він створив разом зі своєю студією розробки Hammer & Chisel, не знайшла відгуку на просторі відео-ігрової

індустрії. Але він стрімко вирішив змінити напрямок діяльності. Він вирішив залишити лише функцію чату та дав назву своєму новому продукту – Discord.

Спочатку Discord був популярним лише в спільнотах геймерів і не дуже швидко набирав користувачів. Але понад шість років, частково завдяки пандемії, він став мейнстрімом. У той час як дорослі, що працюють вдома, збиралися в Zoom, їх діти завантажували Discord, щоб спілкуватися з однолітками за допомогою текстових, аудіо-дзвінків та відео-дзвінків у групах, відомих як сервери.

Зараз на платформі спостерігається понад 200 мільйонів активних користувачів, коли ще у 2019 році їх нараховувалось лише 56мільйонів.

У вересні Discord, що базується в Сан-Франциско, заявила, що залучає фінансування в розмірі $500 млн при оцінці $14,7 млрд за даними PitchBook. У 2021 році кількість співробітників компанії зросла більш ніж удвічі, склавши близько 650 осіб[9].

Перетворення Discord на інструмент масового користування стало несподіваним поворотом у кар'єрі Цитрона. Він розповідає, що ріс на Лонг-Айленді та захоплювався відеоіграми, мало не закінчив Університет Повних Вітрил у Флориді, тому що провів дуже багато часу, граючи у World of Warcraft, і навіть пішов на перше побачення зі своєю майбутньою дружиною до зали грального автоматів.

Він розповідає, що намагається «дозволити людям поводитися безглуздо, щоб вони мали місце, де можна розслабитися», але не терпить розпалювання ненависті чи знущань. За його словами, через ізоляцію, викликану пандемією, ті зв'язки, які формуються в Discord, стали дуже важливими.

Зараз же можна спостерігати постійні офіційні пости на сайті Discord про плани на подальше розвинення продукту та багато інших думок Цитрона, щодо сучасності, розвитку технологій та відео-ігрових новин.

1.3 Аналіз програмних продуктів-аналогів

У межах дослідження було проаналізовано існуючі рішення, які виконують функції автоматизації модерації у Discord-середовищі. Серед них найбільш поширеними є готові боти, які широко використовуються адміністраторами серверів для підтримки порядку, безпеки та інтеграції з іншими сервісами. Нижче розглянуто декілька популярних аналогів:

MEE6 — один із найпопулярніших багатофункціональних ботів. Створений у 2016 році. Профіль на платформі Discord (див. Рис. 1).

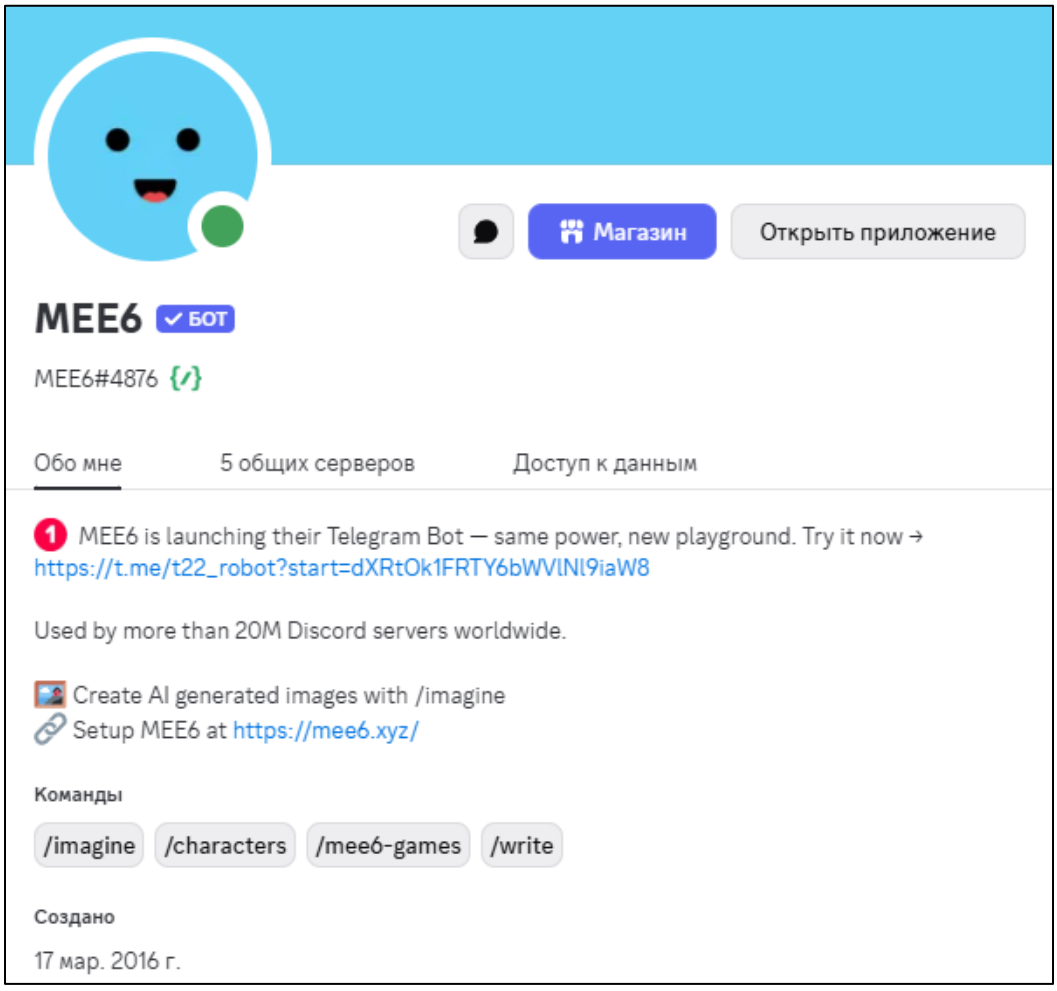


Рисунок *1 — Профіль MEE6 у Discord*

Він надає великий набір можливостей, зокрема:

- автоматична модерація (виявлення спаму, нецензурної лексики, масового тегання тощо);
- вітальні повідомлення для нових учасників;
- система рівнів активності користувачів (із візуальними рангами);
- інтеграція з YouTube та Twitch для автоматичного сповіщення про нові трансляції чи відео;
- планування публікацій;
- автоматичне надання ролей.

MEE6 має зручний веб-інтерфейс, який дозволяє адміністраторам налаштовувати функціонал без необхідності писати код. Однак, низка функцій (зокрема розширена модерація, система рангів, індивідуальні команди тощо) доступні лише у платній версії, що обмежує можливості безкоштовного використання. Код бота є закритим.

JuniperBot — орієнтований на модерацію та безпеку. Створений у 2017 році. Профіль на платформі Discord (див. Рис. 2).

Його головною особливістю є розширені можливості модерації, зокрема:

- фільтрація повідомлень за ключовими словами та регулярними виразами;
- блокування запрошень на інші сервери;
- автоматичне покарання користувачів (мут, кік, бан);
- ведення логів усіх дій;
- зручна веб-панель керування з візуалізацією активності учасників.

Цей бот орієнтований насамперед на великі сервери, де важливо забезпечити швидке виявлення порушень і централізований контроль над діями модераторів. Базовий функціонал є безкоштовним, але частина аналітичних інструментів доступна лише за підпискою.

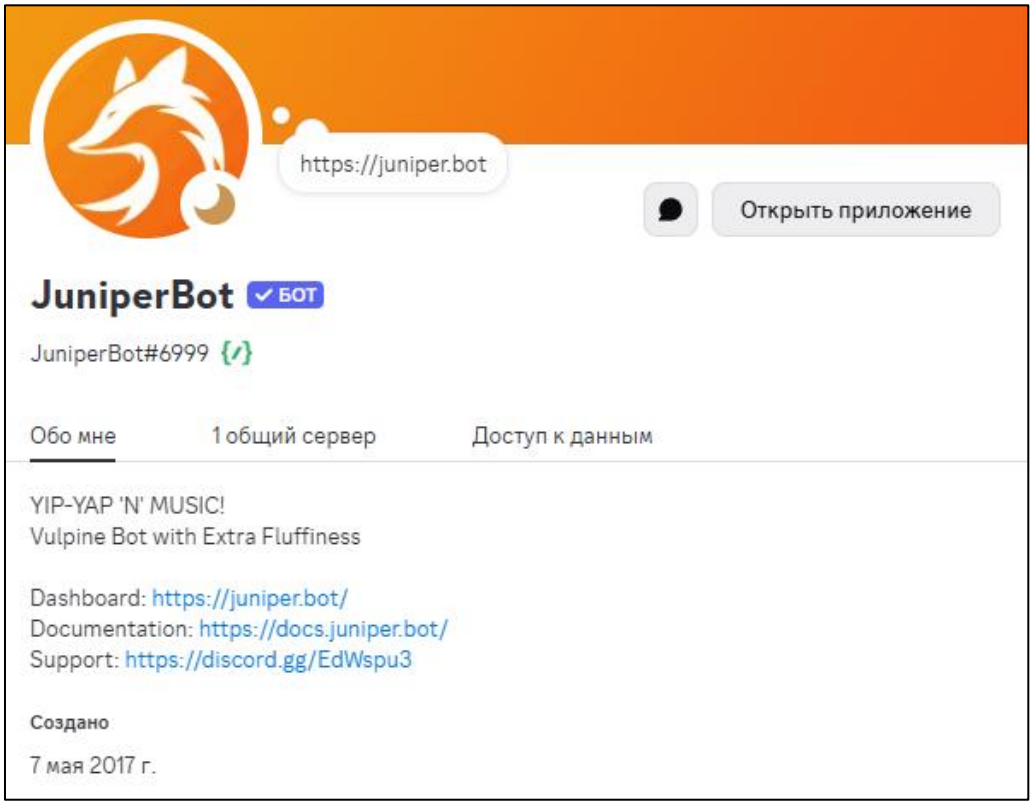


Рисунок *2 — Профіль JuniperBot у Discord*

HydraBot — спеціалізується на музичному функціоналі. Створений у 2019 році. Профіль на платформі Discord (див. Рис. 3).

Його можливості охоплюють:

- відтворення музики з таких джерел, як YouTube, SoundCloud, Deezer, Spotify (використовується пошук через назву або прямі посилання);
- управління чергою треків (додавання, видалення, перегляд черги);
- можливисть отримати роль за реакції;
- привітальні повідомлення для нових користувачів;
- музичні команди, доступні через як текст, так і графічний інтерфейс у вигляді "панелі керування".

Hydra має окрему веб-панель з інтерактивним інтерфейсом, що дозволяє керувати плеєром напряму з браузера. Як і більшість комерційних ботів,

розширений функціонал доступний лише після підписки (наприклад, без реклами, покращена якість звуку, персональні списки відтворення тощо).

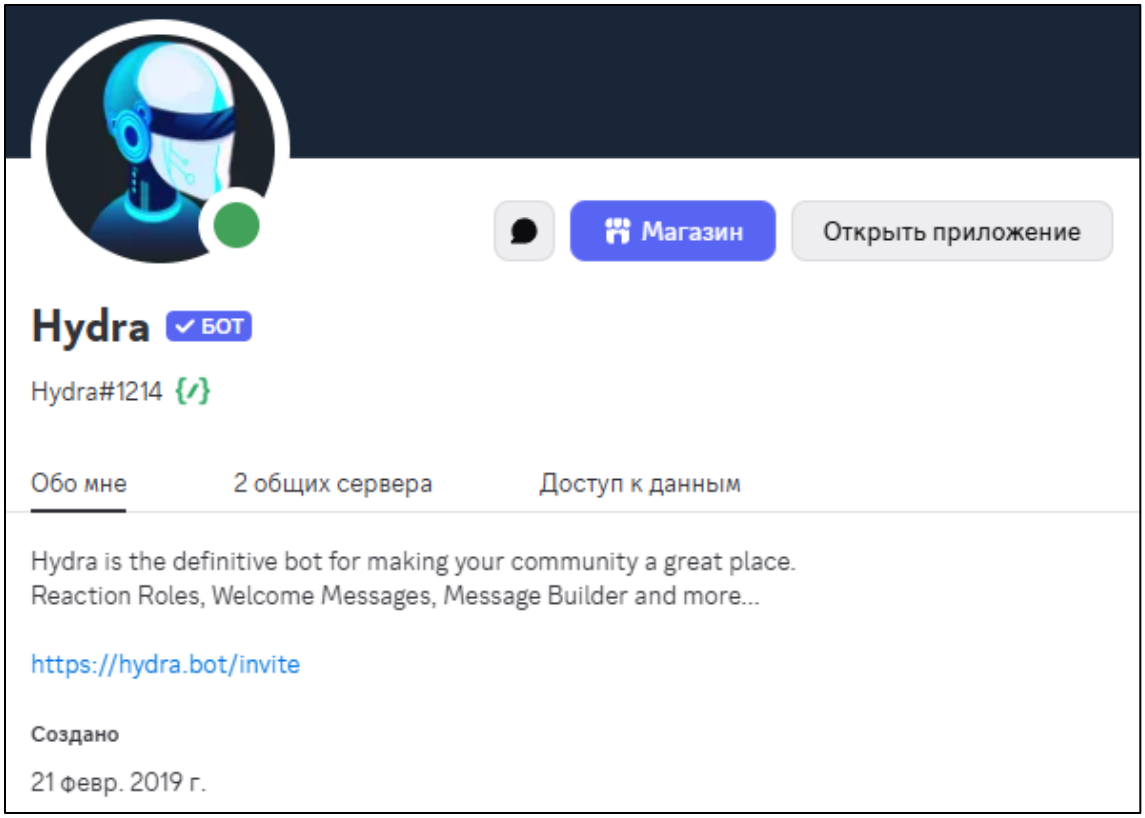


Рисунок *3* — *Профіль HydraBot у Discord*

1.4 Постановка задачі

У сучасному цифровому середовищі спільноти, створені на платформі Discord, активно використовуються як для розваг, так і для професійного спілкування, дистанційного навчання, технічної підтримки тощо. Зі збільшенням кількості учасників зростає і потреба в автоматизованих інструментах керування, модерації та взаємодії з користувачами. Ручна модерація у великих спільнотах є неефективною, потребує значних зусиль з боку адміністраторів та не завжди гарантує об’єктивність і швидкість реагування.

З огляду на вищезазначене, актуальним є створення багатофункціонального Discord-бота, який дозволяє автоматизувати ключові аспекти роботи серверу. Такий бот має забезпечити модерацію учасників, контроль за повідомленнями, видачу попереджень або покарань, підтримку системи реакцій на ролі, а також мати додаткові інструменти для взаємодії.

Мета кваліфікаційної роботи – є розробка багатофункціонального Discord-бота з відкритою архітектурою, що реалізує:

- модераційні функції(бан, кік, мут);
- захист від спаму;
- систему привітання нових користувачів;
- функцію ролей за реакцією;
- можливість програвання музики з YouTube/Spotify;
- логування[8];
- систему рівнів користувачів;
- допоміжні функції(калькулятор);
- розважальні функції(знаходження фото котиків).

Для досягнення цієї мети необхідно:

- проаналізувати доступні технології для розробки Discord-ботів;
- вибрати відповідну мову програмування та бібліотеки;
- спроєктувати архітектуру системи у вигляді окремих незалежних модулів;
- реалізувати кожен функціональний блок у вигляді cog-модуля;
- протестувати роботу бота у реальному Discord-середовищі.

Таким чином, постановка задачі передбачає не лише створення програмного продукту, а й дослідження ефективності використання асинхронного програмування, Discord API та модульного підходу для реалізації гнучкого та масштабованого рішення.

2 ДОСЛІДЖЕННЯ ЗАСОБІВ ПРОГРАМНОЇ РЕАЛІЗАЦІЇ DISCORD-БОТА

2.1 Аналіз сучасних мов програмування для розробки Discord-бота

При виборі мови програмування для створення Discord-бота, важливо враховувати декілька критеріїв: простота написання та читання коду, наявність відповідних бібліотек та фреймворків, продуктивність, можливість масштабування та підтримка серверів. У цьому розділі ми розглянемо декілька популярних мов програмування, які використовуються для розробки Discord-ботів.

Python

Python є однією з найпопулярніших мов програмування для створення Discord-ботів[2]. Основні переваги Python включають:

- Простота синтаксису. Python відомий своєю легкістю вивчення та читання, що робить його ідеальним вибором для початківців.
- Бібліотека Discord.py. Це одна з найбільш використовуваних бібліотек для створення Discord-ботів. Вона забезпечує повний доступ до Discord API та має гарну документацію.
- Підтримка спільноти. Велика спільнота розробників, які активно використовують Python для створення ботів, забезпечує наявність численних ресурсів та прикладів коду.

JavaScript (Node.js)

JavaScript разом з Node.js є ще одним популярним вибором для розробки Discord-ботів. Основні переваги JavaScript включають:

- Асинхронна обробка. Node.js дозволяє легко обробляти асинхронні запити,

що є важливим для реального часу роботи ботів.

- Бібліотека Discord.js. Ця бібліотека забезпечує зручний інтерфейс для роботи з Discord API та має широку функціональність.
- Популярність. JavaScript є однією з найпопулярніших мов програмування у світі, що забезпечує велику кількість ресурсів та підтримку.

C#

C# використовується деякими розробниками для створення Discord-ботів завдяки своїм перевагам:

- Платформа .NET. Використання .NET платформи дозволяє створювати потужні та масштабовані рішення.
- Бібліотека DSharpPlus. Це одна з найпопулярніших бібліотек для роботи з Discord API у світі C#.
- Висока продуктивність. C# забезпечує високу продуктивність, що є важливим для великих серверів з великою кількістю користувачів.

Висновки

Зважаючи на вищезазначені критерії, Python є найбільш оптимальним вибором для створення Discord-бота, особливо для новачків та невеликих проєктів. JavaScript (Node.js) також є чудовим вибором для тих, хто віддає перевагу асинхронній обробці та великій спільноті підтримки. C# може бути використаний для більш складних та масштабованих проєктів, де важлива висока продуктивність.

Для розробки даного застосунку було обрано мову програмування Python через зручність та минулий досвід роботи з цієї мовою програмування. А також через те, що бібліотека Discord.py є найрозвиненішою.

2.2 Аналіз сучасних систем керування базами даних для розробки Discord-бота

При створенні Discord-бота важливо вибрати відповідну систему керування базами даних (СКБД), яка забезпечить ефективне зберігання та обробку даних. Розглянемо декілька популярних СКБД, які можуть бути використані для розробки Discord-бота.

SQLite

SQLite є вбудованою базою даних, яка широко використовується для невеликих проєктів. Основні переваги SQLite включають:

- Простота використання. Не потребує налаштування серверу, оскільки всі дані зберігаються у файлі.
- Швидкість. Забезпечує швидкий доступ до даних для невеликих та середніх проєктів.
- Легка інтеграція. Легко інтегрується з більшістю мов програмування та бібліотек.

PostgreSQL

PostgreSQL є потужною реляційною базою даних, яка підходить для більш масштабних проєктів. Основні переваги PostgreSQL включають:

- Висока продуктивність. Підтримує складні запити та транзакції.
- Масштабованість. Підходить для великих проєктів з великою кількістю даних.
- Надійність. Забезпечує високу надійність та захист даних.

MongoDB

MongoDB є документо-орієнтованою базою даних, яка забезпечує гнучкість у структурі даних. Основні переваги MongoDB включають:

- Гнучкість. Підходить для зберігання нереляційних даних та забезпечує легкість в управлінні.
- Масштабованість. Добре підходить для горизонтального масштабування.
- Швидкість. Забезпечує швидкий доступ до даних, що є важливим для реального часу додатків.

Висновки

Для невеликих проєктів та початківців, SQLite є чудовим вибором завдяки своїй простоті та швидкості. PostgreSQL підходить для масштабніших проєктів, які вимагають високої продуктивності та надійності. MongoDB в свою чергу забезпечує гнучкість для нереляційних даних.

Для проєкту бота модератора було прийнято рішення використати SQLite через його вбудованість, швидкість та сумісність.

2.3 Огляд інших технологій для розробки Discord-бота

Також важливим є огляд та аналіз інших технологій для розробки з ціллю вибору кращої опції задля реалізації застосунку. Окрім мови Python та бібліотеки discord.py, які були обрані для реалізації даного проєкту, існує низка інших популярних технологій і платформ для створення Discord-ботів. Кожна з них має власні переваги, особливості та обмеження, які варто враховувати при виборі інструментів для розробки. Тому у цьому розділі буде розглянуто найпоширеніші альтернативи.

Discord.js (JavaScript/TypeScript)

discord.js — це одна з найпопулярніших бібліотек для створення Discord-ботів з використанням мов JavaScript або TypeScript. Вона має активну спільноту, добре структуровану документацію та широкі можливості інтеграції з іншими веб-технологіями. Перевагами discord.js є:

- повна підтримка всіх функцій Discord API;
- зручна обробка подій і реакцій;
- швидка інтеграція з фронтендом або базами даних через екосистему Node.js;
- хороша масштабованість завдяки TypeScript-типізації.

Недоліком може бути складність для початківців, особливо у випадку використання асинхронного коду з callback-функціями.

Eris (JavaScript)

Eris — легша альтернатива discord.js, орієнтована на продуктивність. Вона менш популярна, але часто використовується у великих ботах, що працюють на великій кількості серверів. Головні переваги якої:

- швидкість роботи;
- знижене споживання пам’яті;
- можливість обробки великої кількості подій.

Проте документація є менш повною, ніж у discord.js, і підтримка спільноти не така активна.

Discord.NET (C#)

Discord.NET — бібліотека для розробки ботів на платформі .NET з використанням мови C#. Вона підходить для розробників, які працюють у середовищі Microsoft і хочуть інтегрувати Discord-функціонал у більші корпоративні рішення. Особливості:

- повноцінна об’єктно-орієнтована модель;
- інтеграція з Windows-сервісами, базами даних MS SQL Server;
- високий рівень безпеки та типізації.

Недоліком є складність у налаштуванні середовища, а також обмеження щодо швидкого прототипування.

JDA (Java Discord API, Java)

JDA — це бібліотека для Java, що надає інтерфейс до Discord API. Вона активно розвивається та підтримує всі ключові можливості платформи:

- подієво-орієнтований підхід;
- інтеграція з Maven або Gradle;
- підтримка голосових функцій.

JDA добре підходить для великих проєктів на Java, але вимагає більше конфігурації в порівнянні з Python або JavaScript, а також має вищий поріг входження.

Порівняння з обраною технологією

Мова Python у поєднанні з бібліотекою discord.py обрана для даного проєкту завдяки:

- простому синтаксису;
- активній спільноті;
- підтримці асинхронного програмування через async/await;
- можливості швидкого створення модульного коду за допомогою системи cogs.

Висновки

У процесі аналізу технологій для реалізації Discord-бота було прийнято рішення використовувати мову програмування Python у поєднанні з бібліотекою discord.py з огляду на такі ключові переваги:

- Простота синтаксису. Python є високорівневою мовою, яка дозволяє розробляти логіку швидко та з мінімальним обсягом коду. Це особливо важливо для підтримки, масштабування та читабельності.
- Асинхронне програмування. Discord.py побудована на основі асинхронної моделі asyncio, що дозволяє ефективно обробляти декілька подій одночасно: команди, повідомлення, підключення до голосових каналів тощо, не блокуючи основний потік виконання.
- Широка спільнота та підтримка. Бібліотека має велику кількість користувачів, прикладів, навчальних матеріалів, а також офіційну документацію, що суттєво полегшує процес розробки.
- Модульна архітектура через cogs. Discord.py дозволяє легко організувати код за допомогою розділення на модулі (cogs), що сприяє масштабованості, повторному використанню коду та зручному тестуванню окремих частин системи.
- Інтеграція з іншими бібліотеками Python. Python має потужну екосистему бібліотек, таких як yt_dlp (завантаження відео), aiohttp (HTTP-запити), spotipy (інтеграція з Spotify API[5]), що дозволяє значно розширити функціонал бота.
- Кросплатформеність. Проєкт, написаний на Python, легко розгортається як на локальному сервері, так і на хмарних платформах.

У сукупності ці переваги роблять discord.py найбільш відповідним інструментом для створення багатофункціонального Discord-бота у рамках кваліфікаційної роботи, з акцентом на модерацію, автоматизацію та інтерактивність.

\

3 РОЗРОБКА DISCORD-БОТА МОДЕРАТОРА

3.1 Опис предметної області

Платформа Discord набула значної популярності як серед геймерів, так і серед різних спільнот та організацій у всьому світі. Можна зазначити, що це універсальний інструмент для комунікації, який дозволяє користувачам створювати та долучатись до серверів з текстовими та голосовими каналами, що забезпечує зручний формат для обговорень, співпраці та взаємодії. За минулий рік Discord став доступний на Xbox та PlayStation 5, розвинувши кросплатформність[10]. Також хоча дії розробників спрямовані більше на комфорт для використання геймерами, але це не заважає навіть готувати разом або ж просто розважатись у голосовому чаті чи спілкуватися за рахунок текстових каналів з іншими користувачами.

Для покращення знаходження на платформі та для безпеки і модерації використовують Discord-ботів. Основна роль ботів виступати як автоматизований інструмент, який допомагає адміністраторам та модераторам серверу підтримувати порядок та дотримуватися правил спільноти, визначені цією спільнотою. Але для підтримання конкуренції серед широкого вибору ботів до них додають різноманітний функціонал, наприклад: генерація або пошук фото через API, впровадження плеєра для прослуховування музики прямо в голосовому каналі і багато іншого. Для того щоб досягти популярності та більшої кількості використання своїх ботів, розробники впроваджують нові функції чи роблять існуючі краще та ефективніше.

Наступними перспективами розвитку є інтеграції з штучним інтелектом, машинне навчання та інші передові технології, завдяки чому Discord-боти зможуть стати ще більш ефективними та універсальними, що забезпечить ще

вищий рівень захисту та комфорту для користувачів та адміністраторів серверів, які будуть їх використовувати.

3.2 Архітектура системи

Discord-бот модератор серверу використовує архітектуру клієнт-сервер. Клієнт надає запити використовуючи команди бота, а бот взаємодіє з Discord API для отримання та відправлення даних. Також в боті є БД, до якої клієнт також звертається через команди бота.

Основні складові програми:

- Discord-бот;
- База даних (БД);
- Discord API (Зовнішній сервер).

Взаємодія клієнта продемонстрована на діаграмі розгортання (див. Рис. 4).

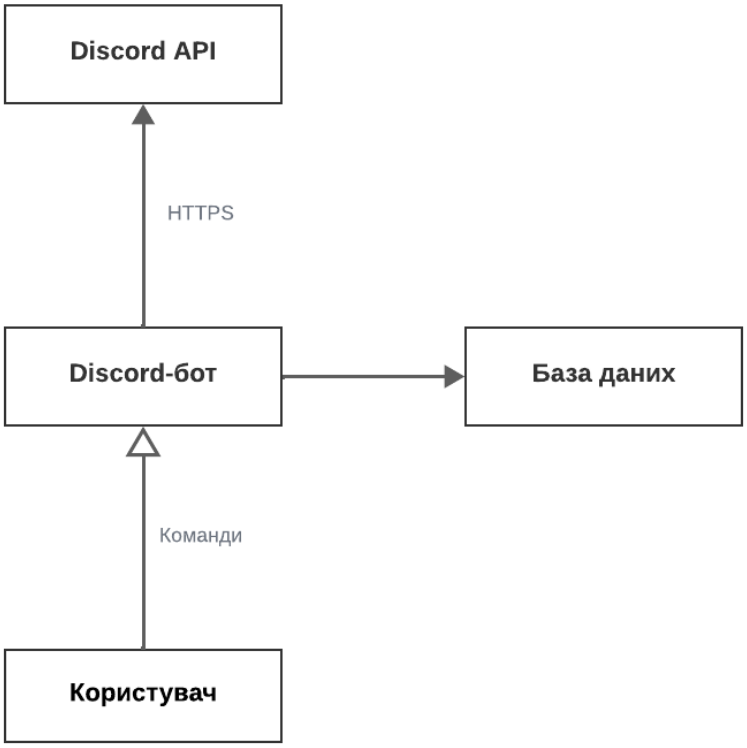


Рисунок *4 — UML діаграма розгортання*

3.3 Архітектура бази даних

У рамках розробки багатофункціонального Discord-бота для модерації, відстеження активності та взаємодії з користувачами було реалізовано просту, але ефективну базу даних для збереження інформації про досвід (XP) та рівні користувачів. Збереження даних реалізовано за допомогою вбудованої реляційної системи управління базами даних SQLite, що не потребує окремого серверного середовища і є ідеальним вибором для локальних проєктів або ботів з помірним навантаженням.

Структура бази даних

База даних зберігається у файлі users.db та містить одну таблицю — users(див. Рис. 5), яка відповідає за збереження статистики активності кожного користувача.

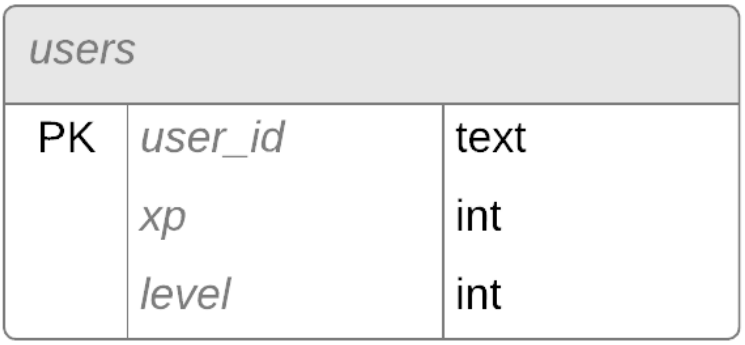


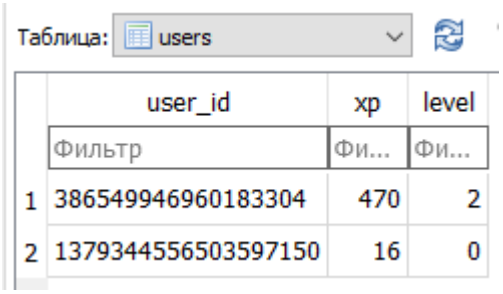
Рисунок *5 — ER діаграма бази даних*

Принцип роботи

- Створення БД. При ініціалізації модуля Leveling бот перевіряє наявність файлу users.db. Якщо база відсутня, вона автоматично створюється, разом із таблицею users.
- Оновлення. Кожного разу, коли користувач надсилає повідомлення в чат,

бот перевіряє його запис у БД. Якщо запис є — оновлює значення XP та рівня. Якщо запису немає — створює новий(див. Рис. 6).

- Обчислення рівня. Алгоритм підрахунку рівня ґрунтується на накопиченому досвіді. Кожен наступний рівень потребує більше XP, що стимулює активність користувачів.
- Отримання даних. Команда !level дозволяє користувачам дізнатися свій поточний рівень і кількість досвіду.



	user_id	xp	level
	Фильтр	Фи...	Фи...
1	386549946960183304	470	2
2	1379344556503597150	16	0

Рисунок *6 — вигляд записів у БД*

Переваги обраної архітектури

- Простота. SQLite не вимагає додаткового встановлення серверу, що спрощує налаштування та підтримку.
- Швидкість. Для невеликих проєктів і локального використання забезпечується висока продуктивність при читанні/записі.
- Гнучкість. За потреби архітектура легко розширюється.

Висновок

Архітектура бази даних є мінімалістичною, але повністю відповідає завданням поточного проєкту. Такий підхід дозволяє ефективно зберігати прогрес користувачів, забезпечує гнучкість у подальшій розробці та підтримує інтеграцію з іншими функціональними модулями Discord-бота.

3.4 Функціональні вимоги системи

Розробка Discord-бота модератора містить чіткі вимоги до функціоналу, які мають бути присутні обов’язково, а також додаткові, які будуть виражені у побічних командах корисних як для адміністраторів, так і користувачів. Також слід відмітити, що весь функціонал доступний лише адміністраторам та модераторам серверу, бо в першу чергу важлива модерація. У цьому розділі буде розглянуто усі функціональні вимоги системи та додаткові вимоги. Додатково ви можете переглянути use-case діаграму адміністратора (див. Рис. 7) та use-case діаграму користувача (див. Рис. 8).

Функціональні вимоги системи краще буде розбити на такі групи:

- Фільтрація повідомлень. Бот повинен постійно перевіряти чат у кожному текстовому каналі. Має бути встановлений ліміт на кількість повідомлень за визначений проміжок часу. У разі недотримання правил користувачеві буде надіслано автоматичне попередження про порушення. Якщо попередження буде проігнороване, то бот самостійно заблокує користувача на визначений проміжок часу. Також кожне попередження та блокування буде записано до логу самого бота та сервера для відстеження і швидкого реагування адміністратора у разі потреби.
- Модерація. Має бути реалізовано набір команд для адміністраторів, які дозволять швидко виконувати модераційні дії(тимчасове блокування, видалення користувача з серверу, блокування назавжди тощо).
- Управління ролями. Призначення ролей користувачам на основі їхньої активності, належності до визначених груп чи інтересів або інших визначених критеріїв. Бот повинен мати команди для адміністраторів щоб призначати або знімати ролі, а також автоматичне призначення та зняття ролі користувачів у певних випадках.
- Логування. Бот повинен мати можливість збереження логів повідомлень та

подій для подальшого аналізу адміністраторами серверу та відстеження активності і подій.

- !help. Інструмент для швидкого перегляду усіх доступних команд на сервері. Є автоматично згенерованим за допомогою API при правильному призначенні груп команд та самих команд. Показує доступні команди виключно для користувача який її викликав.

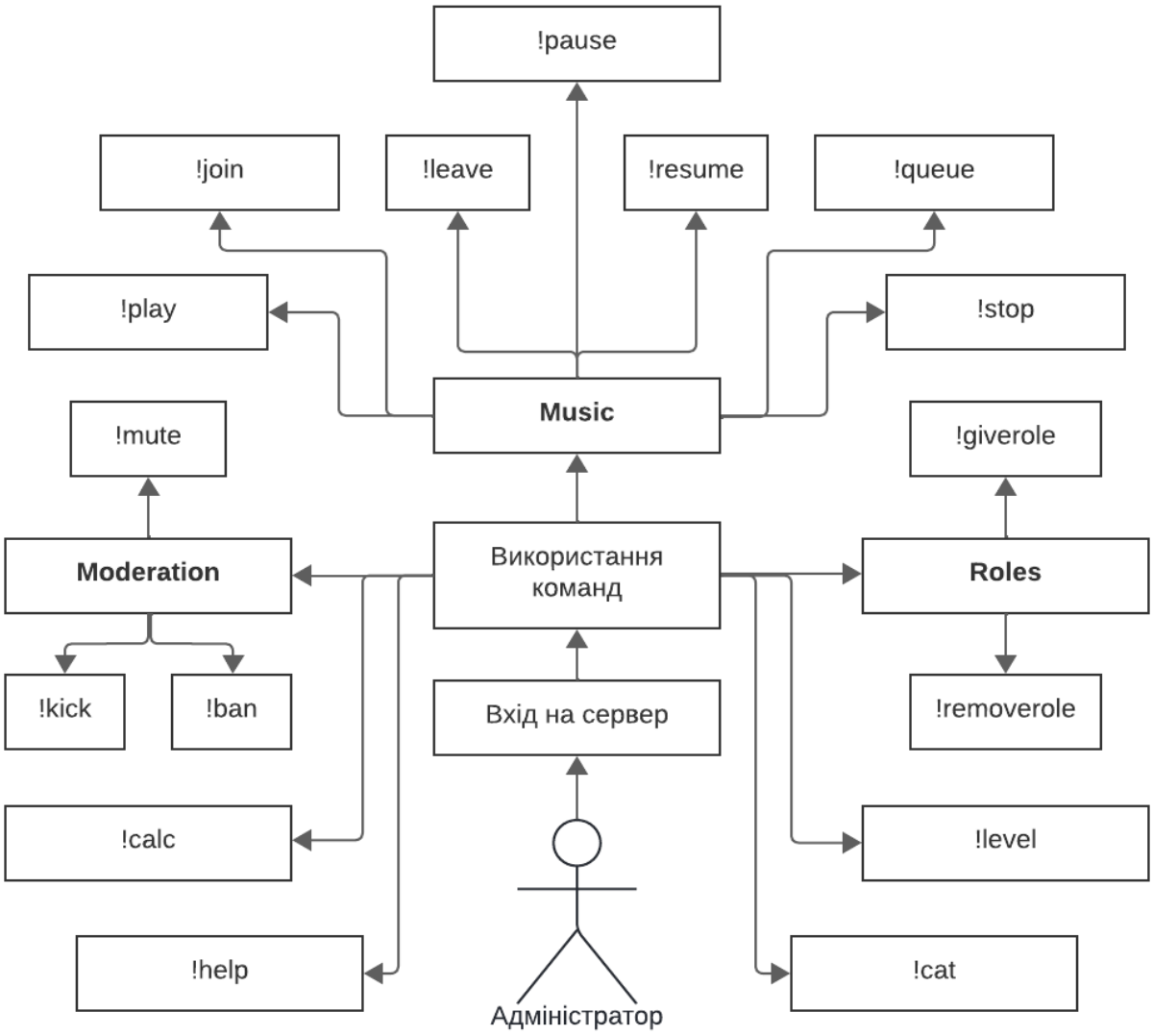


Рисунок *7 — Use-case діаграма адміністратора*

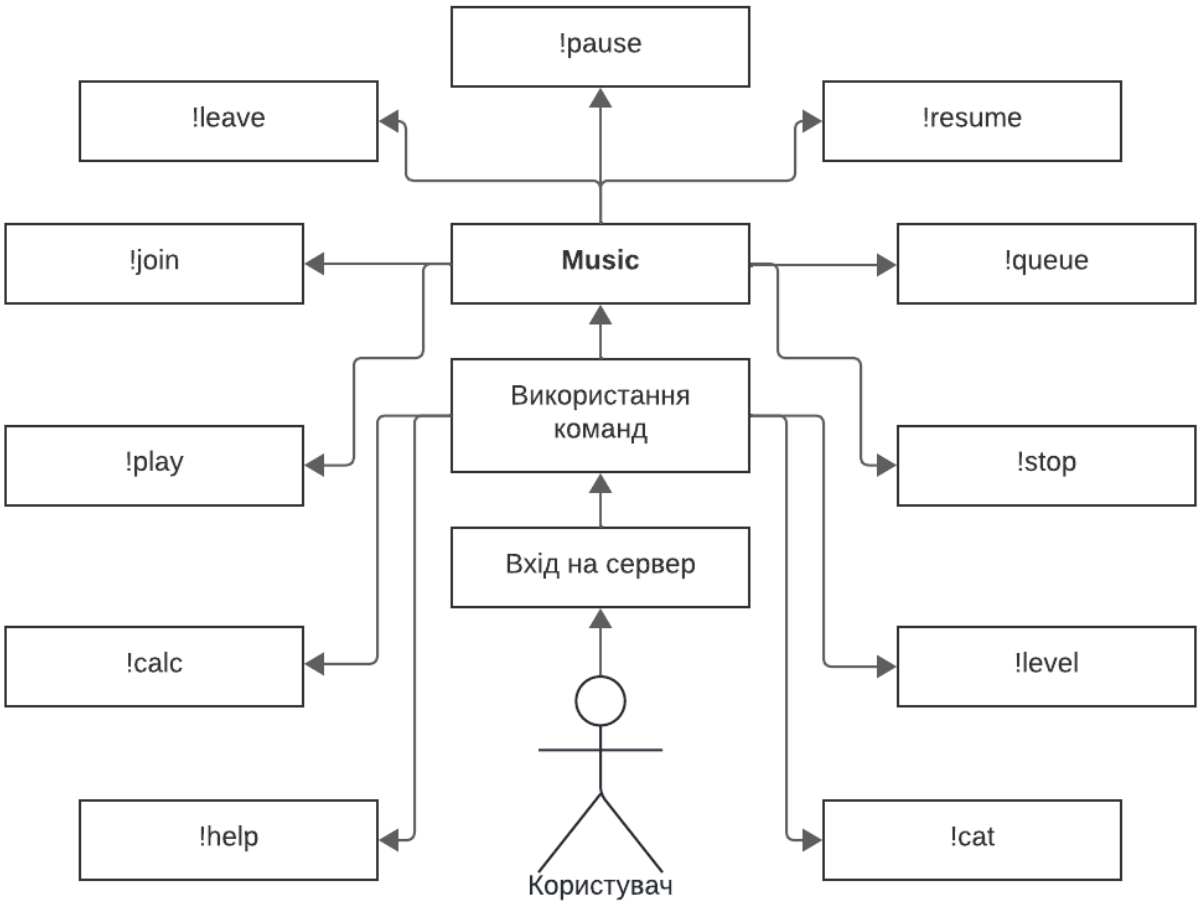


Рисунок *8* — *Use-case діаграма користувача*

Додаткові функціональні вимоги(тільки ті що реалізовані в програмі):

- Інтеграція з базами даних або іншими сервісами. Реалізована база даних для розважальної системи рівнів на сервері.
- Система рівнів. Кожен користувач на сервері має свій рівень(level) та кількість досвіду(xp). За кожне повідомлення в будь-якому текстовому каналі користувачеві надається певна кількість досвіду. Кожен користувач може подивитися свій рівень та кількість досвіду. При переході на наступний рівень, бот повідомляє про це користувачу.
- Калькулятор. Звичайний калькулятор який підтримує такі оператори як +, –, /, *, %, ().
- Запит !cat(Отримання випадкового фото кота за допомогою API).

- Музика. За допомогою бібліотеки ffmpeg[6] реалізована функція програвання музики у голосовому каналі. Бот підтримує посилання на треки з Spotify та YouTube. Перелік функцій – приєднання та від'єднання з голосового каналу, пауза та відновлення програвання треку, зупинка треку, додавання треку у чергу, черга. Також при тривалому простої у голосовому каналі бот автоматично покине його.

Забезпечення відповідності функціональних вимог дозволить створити ефективний та надійний інструмент для модерації серверів на платформі Discord, що підвищить якість спілкування та забезпечить комфортне середовище для всіх користувачів.

3.5 Вимоги до програмного та апаратного забезпечення

Для досягнення успішної реалізації проєкту Discord-бота модератора необхідно визначити вимоги до програмного та апаратного забезпечення, які забезпечать стабільну та ефективну роботу бота. Розглянемо необхідні інструменти, бібліотеки, мови та інші компоненти для вимог програмного забезпечення, а також все що стосується вимог апаратного забезпечення для даного проєкту. Описані будуть застосовані інструменти та технології безпосередньо для розробленого та описуваного в цій роботі застосунку.

Вимоги до програмного забезпечення:

1. Операційна система – 64-бітна версія Windows 10 або вище.
2. Мова програмування – Python 3.7.1 або вище.
3. База даних – SQLite для зберігання даних користувачів.
4. Інструменти розробки та управління проєктом – Git для контролю версій.
5. Інтеграційні сервіси та API – Discord API для взаємодії з Discord.

6. Бібліотеки – в основі discord.py, усі інші представлені у файлі requirements.txt, який знаходиться в папці проєкту.
7. Інструмент ffmpeg[7] – набір бібліотек та інструментів для керування аудіо та відео файлами, потрібний для впровадження музичного плеєра.
8. .env-файл, розташований в кореневій папці проєкту, в який потрібно надати токен бота, отриманий з Discord Developer Portal[3], та Spotify ID і secret, отриманий з Spotify for Developers.

Вимоги до апартаного забезпечення:

1. Серверне обладнання. Задля виконання цієї потреби можна використати хмарні сервіси, віртуальні машини або локальні чи окремі сервери.
2. Мережне обладнання. Широкосмугове підключення до Інтернету з низькою затримкою й високою пропускною здатністю.

Для реалізації проєкту було використано персональний комп’ютер, на якому велась розробка як локальний сервер.

Ці вимоги до програмного та апаратного забезпечення забезпечать стабільну роботу Discord-бота модератора, дозволяючи йому ефективно виконувати всі необхідні функції для підтримання порядку та безпеки на Discord-сервері.

3.6 Модулі та алгоритми

Програма містить 9 файлів розширення .py, один з них — це головний файл, а інші являють собою модулі. Модулі реалізовані через Cogs в бібліотеці discord.py – це спосіб організації та управління кодом бота. Вони використовуються для створення відокремлених модулів, де кожен з них містить набір команд та/або подій під різні категорії. У такий спосіб ми маємо більш

зручний спосіб розбиття програми на різні компоненти, які структуровані логічно, а також дозволяють простіше підтримувати та розширяти функціонал програми.

У цьому розділі буде розглянуто усі 8 модулів програми, а саме:

- cogs.calculator;
- cogs.cat_command;
- cogs.leveling;
- cogs.moderation;
- cogs.music;
- cogs.reaction_role;
- cogs.spam_filter;
- cogs.welcome.

1. Модуль cogs.calculator

Являє собою звичайний калькулятор. Він дозволяє користувачам обчислювати математичні вирази безпосередньо у будь-якому текстовому каналі Discord, використовуючи команду “!calc”. Містить лише одну команду та її реалізацію(лістинг 1).

Лістинг *1 — Код команди !calc*

```
@commands.command(name='calc')
    async def calculate(self, ctx, *, expression: str):
        """Evaluates a mathematical expression."""
        try:
            # Safely evaluate the expression
            result = eval(expression, {"__builtins__": None},
{})
            await ctx.send(f'The result of `{expression}` is
`{result}`.')
```

```
            logging.info(f'Calculated      {expression}      =
{result}')
        except Exception as e:
            await   ctx.send(f'Error   evaluating   expression:
{e}')
            logging.error(f'Error    evaluating    expression:
{expression} - {e}')
```

Висновок

Модуль “cogs.calculator” є простим, але ефективним прикладом функціональності, яку можна додати до Discord-бота. Він демонструє основні принципи створення команд та обробки користувацьких запитів, а також включає методи ведення журналу для відстеження подій та помилок, які будуть присутні і в усіх інших модулях. Цей модуль може бути легко інтегрований до будь-якого бота для розширення його можливостей.

2. Модуль cogs.cat_command

Реалізує команду !cat, яка надсилає випадкове зображення кота з відкритого API (https://thecatapi.com). Для надсилання запитів використовується асинхронна бібліотека aiohttp. У випадку невдалого запиту користувачеві надсилається повідомлення про помилку.

Лістинг *2 — Код команди !cat*

```
@commands.command(name='cat')
    async def fetch_cat_picture(self, ctx):
        """Fetches a random cat picture from an open API."""
        async with aiohttp.ClientSession() as session:
            async                                        with
session.get('https://api.thecatapi.com/v1/images/search'
```

```
                                    ) as response:
                if response.status != 200:
                    await ctx.send('Failed to fetch a cat
picture.')
                    logging.error(f'Failed to fetch a cat '
                                  f'picture:
{response.status}')
                    return

                data = await response.json()
                cat_image_url = data[0]['url']
                await ctx.send(cat_image_url)
                logging.info('Sent a cat picture.')
```

Висновок

Модуль cogs.cat_command додає інтерактивну та розважальну функцію до бота. Він є прикладом використання зовнішніх REST API в асинхронному середовищі Discord і демонструє обробку HTTP-запитів у режимі реального часу.

3. Модуль cogs.leveling

Цей модуль реалізує систему рівнів активності користувачів. Він відслідковує повідомлення, начисляє кількість очок досвіду(xp) користувачеві, обчислює кількість очок досвіду(xp) та надає відповідний рівень користувачеві. Дані зберігаються у локальній базі даних.

Лістинг *3 — Функція створення бази даних*

```
def create_database(self):
    try:
        conn = sqlite3.connect(self.database_path)
        c = conn.cursor()
```

```
        # Create users table
        c.execute('''
            CREATE TABLE IF NOT EXISTS users (
                user_id TEXT PRIMARY KEY,
                xp INTEGER NOT NULL,
                level INTEGER NOT NULL
            )
        ''')

        conn.commit()
        conn.close()
        print("Database created successfully or already exists.")
    except sqlite3.Error as e:
        print(f"Error creating database: {e}")
```

Лістинг *4 — Код функцій зарахування досвіду та поведінка реагування на повідомлення*

```
@commands.Cog.listener()
    async def on_message(self, message):
        if message.author == self.bot.user:
            return

        user_id = str(message.author.id)
        print(f"Adding XP for user_id: {user_id}")
        await self.add_xp(message, user_id, 2)  # Add 2 XP for each message

    async def add_xp(self, message, user_id, xp_to_add):
```

```
        try:
            conn = sqlite3.connect(self.database_path)
            c = conn.cursor()
            c.execute('SELECT xp, level FROM users WHERE
user_id = ?',
                      (user_id,))
            result = c.fetchone()
            if result:
                xp, level = result
                xp += xp_to_add
                new_level = self.calculate_level(xp)
                if new_level > level:
                    level = new_level
                    await
message.channel.send(f"Congratulations "

f"{message.author.mention}! "
                                                f"You got to
level {level}")
                c.execute('UPDATE users SET xp = ?, '
                          'level = ? WHERE user_id = ?',
                          (xp, level, user_id))
            else:
                xp = xp_to_add
                level = self.calculate_level(xp)
                c.execute('INSERT INTO users '
                          '(user_id, xp, level) VALUES
(?, ?, ?)',
                          (user_id, xp, level))
                if level > 0:
```

```
                    await
message.channel.send(f"Congratulations "

f"{message.author.mention}! "
                                                f"You  got  to
level {level}")
            conn.commit()
            conn.close()
        except sqlite3.Error as e:
            print(f"Error updating/inserting XP: {e}")
```

Лістинг *5 — Функції розрахунку рівнів та досвіду*

```
def calculate_level(self, xp):
    level = 0
    xp_required = 100
    while xp >= xp_required:
        level += 1
        xp -= xp_required
        xp_required += 150
    return level

def xp_for_next_level(self, current_xp):
    xp_required = 100
    while current_xp >= xp_required:
        current_xp -= xp_required
        xp_required += 150
    return xp_required - current_xp

def get_user_level(self, user_id):
    try:
        conn = sqlite3.connect(self.database_path)
```

```
        c = conn.cursor()
        c.execute('SELECT xp, level FROM users WHERE user_id
= ?',
                  (user_id,))
        result = c.fetchone()
        conn.close()
        if result:
            return result
        else:
            return 0, 0
    except sqlite3.Error as e:
        print(f"Error fetching user level: {e}")
        return 0, 0
```

Лістинг *6 — Команда !level для відображення поточного рівня користувача*

```
@commands.command(name='level', help='Check your level')
    async def level(self, ctx):
        user_id = str(ctx.author.id)
        xp, level = self.get_user_level(user_id)
        xp_needed = self.xp_for_next_level(xp)
        await ctx.send(
            f"{ctx.author.mention}, you are at level {level}
with {xp} XP."
            f" You need {xp_needed} more XP for the next
level.")
```

Висновок

Модуль cogs.leveling являє собою застосування БД у застосунку. Він є прикладом роботи з постійним зберіганням даних, обробки подій та

математичного розріахунку прогресу. Це дозволяє створити змагальний елемент між користувачами, підвищити їхню залученість та стимулювати участь у житті спільноти.

4. Модуль cogs.moderation

Відповідає за базові модераційні функції: !ban, !kick, !mute. Команди перевіряють права доступу користувача, запобігають спробам самозабанення або взаємодії з ботом. Мут реалізовано із вказаним часом, після чого користувача автоматично розм'ючується. Всі дії додаються до логу подій.

Лістинг *7 — Код команди !ban*

```
@commands.command(name='ban')
    @commands.has_permissions(ban_members=True)
    async def ban(self, ctx, member: discord.Member, *, reason=None):
        """Ban a user from the server."""
        if member == ctx.author:
            await ctx.send("You cannot ban yourself!")
            logging.warning(f'{ctx.author} tried to ban themselves.')
            return

        if member == self.bot.user:
            await ctx.send("I cannot ban myself!")
            logging.warning(f'{ctx.author} tried to ban the bot.')
            return

        await member.ban(reason=reason)
```

```
        await ctx.send(f'{member.mention} has been banned. Reason: {reason}')
        logging.info(f'{member} was banned by {ctx.author}. Reason: {reason}')
```

Лістинг *8 — Код команди !kick*

```
@commands.command(name='kick')
    @commands.has_permissions(kick_members=True)
    async def kick(self, ctx, member: discord.Member, *, reason=None):
        """Kick a user from the server."""
        if member == ctx.author:
            await ctx.send("You cannot kick yourself!")
            logging.warning(f'{ctx.author} tried to kick themselves.')
            return

        if member == self.bot.user:
            await ctx.send("I cannot kick myself!")
            logging.warning(f'{ctx.author} tried to kick the bot.')
            return

        await member.kick(reason=reason)
        await ctx.send(f'{member.mention} has been kicked. Reason: {reason}')
        logging.info(f'{member} was kicked by {ctx.author}. Reason: {reason}')
```

Лістинг *9 — Код команди !mute та функція автоматичного створення відповідної ролі*

```
@commands.command(name='mute')
    @commands.has_permissions(manage_roles=True)
    async def mute(self, ctx, member: discord.Member, time: int,
*,
                   reason=None):
        """Mute a user for a specified number of minutes."""
        if member == ctx.author:
            await ctx.send("You cannot mute yourself!")
            logging.warning(f'{ctx.author}   tried   to   mute
themselves.')
            return

        if member == self.bot.user:
            await ctx.send("I cannot mute myself!")
            logging.warning(f'{ctx.author} tried to mute the
bot.')
            return

        mute_role     =     discord.utils.get(ctx.guild.roles,
name="Muted")
        if mute_role is None:
            mute_role = await self.create_mute_role(ctx.guild)

        await member.add_roles(mute_role)
        await ctx.send(f'{member.mention} has been muted'
                       f'   for   {time}   minutes.   Reason:
{reason}')
        logging.info(f'{member} was muted by {ctx.author} '
                     f'for {time} minutes. Reason: {reason}')
```

```
        await asyncio.sleep(time * 60)
        await member.remove_roles(mute_role)
        await ctx.send(f'{member.mention} has been unmuted.')
        logging.info(f'{member} was unmuted after {time}
minutes.')

    async def create_mute_role(self, guild):
        """Create a Muted role if it doesn't exist."""
        mute_role = await guild.create_role(
            name="Muted",

permissions=discord.Permissions(send_messages=False,
speak=False))
        for channel in guild.channels:
            await channel.set_permissions(mute_role,
                                          send_messages=False,
speak=False)
        return mute_role
```

Висновок

cogs.moderation є критичним модулем для полегшеного управління користувачами на сервері. Завдяки простим, але важливим командам, модуль значно полегшує роботу адміністраторів та підвищує рівень безпеки спільноти. Передбачені можливості невірного застосування команди та додано захищений допуск для їх виконання.

5. Модуль cogs.music

Один із найскладніших модулів, відповідає за програвання музики. Реалізовано підтримку

команд: !play, !join, !leave, !pause, !resume, !stop, !skip, !queue. Для завантаження та обробки аудіо використовується yt_dlp, ffmpeg, а також інтеграція зі Spotify через spotipy. Модуль шукає трек за посиланням та програє його у голосовому каналі до котрого підключений. Підтримує функції черги, призупинення треку, відновлення треку, відтворення наступного треку, повна зупинка програвання.

Лістинг *10 — Код команди !join*

```
@commands.command(name='join',
                  help='Tells the bot to join the voice
channel')
    async def join(self, ctx):
        if ctx.author.voice:
            channel = ctx.author.voice.channel
            await channel.connect()
        else:
            await ctx.send("You are not in a voice channel!")
```

Лістинг *11 — Код команди !leave*

```
@commands.command(name='leave',
                  help='Tells the bot to leave the voice
channel')
    async def leave(self, ctx):
        if ctx.voice_client:
            await ctx.guild.voice_client.disconnect()
        else:
            await ctx.send("I am not in a voice channel!")

    async def extract_audio_info(self, url, retries=3):
        loop = asyncio.get_event_loop()
        attempt = 0
```

```
        while attempt < retries:
            try:
                with youtube_dl.YoutubeDL(self.YDL_OPTIONS) as
ydl:
                    info = await loop.run_in_executor(
                        None, ydl.extract_info, url, False)
                    if 'url' in info:
                        return info['url'], info.get('title',
'Unknown Title')
                    else:
                        return None, None
            except Exception as e:
                attempt += 1
                print(f"Error extracting audio info: {e}. "
                      f"Attempt {attempt}/{retries}")
                await asyncio.sleep(2)
        return None, None
```

Лістинг *12 — Код команди !play*

```
@commands.command(name='play',
                      help='Plays a song from a YouTube or
Spotify URL')
    async def play(self, ctx, url):
        if not ctx.voice_client:
            if ctx.author.voice:
                await ctx.author.voice.channel.connect()
            else:
                await ctx.send("You are not in a voice
channel!")
                return
```

```
        if 'open.spotify.com' in url:
            track_id = url.split("/")[-1].split("?")[0]
            track = self.sp.track(track_id)
            track_name = track['name']
            artist_name = track['artists'][0]['name']
            query = f"{track_name} {artist_name}"

            with youtube_dl.YoutubeDL(self.YDL_OPTIONS) as
ydl:
                try:
                    results = ydl.extract_info(
                        f"ytsearch:{query}",
download=False)['entries'][0]
                    youtube_url = results['webpage_url']
                    audio_url, title = await
self.extract_audio_info(
                        youtube_url)
                    if audio_url:
                        self.queue.append({'url': audio_url,
'title': title})
                        await ctx.send(f"Added {track_name} "
                                       f"by {artist_name} to
the queue.")
                    else:
                        await ctx.send("Could not extract
audio URL.")
                except Exception as e:
                    await ctx.send("Could not find the track on
YouTube.")
                    print(f"YouTube search failed: {e}")
                    return
```

```
        else:
            audio_url, title = await
self.extract_audio_info(url)
            if audio_url:
                self.queue.append({'url': audio_url, 'title':
title})
                await ctx.send(f"Added {title} to the queue.")
            else:
                await ctx.send("Could not extract audio URL.")

        if (not ctx.voice_client.is_playing() and
                not ctx.voice_client.is_paused()):
            await self.play_next(ctx)

    async def play_next(self, ctx):
        if self.queue:
            song = self.queue.pop(0)
            self.currently_playing = song
            audio_url = song['url']
            title = song['title']
            try:
                source = await
discord.FFmpegOpusAudio.from_probe(
                    audio_url,
                    executable=self.FFMPEG_PATH,
**self.FFMPEG_OPTIONS)
                ctx.voice_client.play(
                    source, after=lambda e:
self.bot.loop.create_task(
                        self.play_next(ctx)))
                await ctx.send(f"Now playing: {title}")
```

```
            except Exception as e:
                await ctx.send(f"An error occurred while trying
"
                               f"to play the track: {e}")
                print(f"Error playing track: {e}")
                await self.play_next(ctx)
        else:
            self.currently_playing = None
```

Лістинг *13 — Код команди !pause*

```
@commands.command(name='pause', help='Pauses the song')
    async def pause(self, ctx):
        if ctx.voice_client and ctx.voice_client.is_playing():
            ctx.voice_client.pause()
        else:
            await ctx.send("No audio is playing.")
```

Лістинг *14 — Код команди !resume*

```
@commands.command(name='resume', help='Resumes the song')
    async def resume(self, ctx):
        if ctx.voice_client and ctx.voice_client.is_paused():
            ctx.voice_client.resume()
        else:
            await ctx.send("The audio is not paused.")
```

Лістинг *15 — Код команди !skip*

```
@commands.command(name='skip', help='Skips the current song'
                                    ' and plays the next in
the queue')
    async def skip(self, ctx):
```

```
        if ctx.voice_client and ctx.voice_client.is_playing():
            if self.queue:
                await ctx.send("Skipped to the next track.")
            else:
                await ctx.send("Skipped. Queue is empty now.")
            ctx.voice_client.stop()
        else:
            await ctx.send("Cannot skip. No audio is playing.")

    @commands.command(name='pause', help='Pauses the song')
    async def pause(self, ctx):
        if ctx.voice_client and ctx.voice_client.is_playing():
            ctx.voice_client.pause()
        else:
            await ctx.send("No audio is playing.")
```

Лістинг *16 — Код команди !stop*

```
@commands.command(name='stop', help='Stops all songs')
    async def stop(self, ctx):
        self.queue.clear()
        self.currently_playing = None
        if ctx.voice_client and ctx.voice_client.is_playing():
            ctx.voice_client.stop()
        await ctx.send("Playback stopped.")
```

Лістинг *17 — Код команди !queue*

```
@commands.command(name='queue',  help='Displays  the  current
song queue')
    async def show_queue(self, ctx):
        queue_message = ""
```

```
        if self.currently_playing:
            queue_message += (f"Playing "
                              f"now:
{self.currently_playing['title']}\n")
        if self.queue:
            for idx, song in enumerate(self.queue):
                entry = f"{idx + 1}. {song['title']}\n"
                if len(queue_message) + len(entry) > 2000:
                    await ctx.send(queue_message)
                    queue_message = ""
                queue_message += entry
        else:
            queue_message += "The queue is empty."

        await ctx.send(queue_message)
```

Висновок

Модуль cogs.music демонструє комплексну інтеграцію декількох бібліотек, роботу з аудіо та потоками. Його реалізація вимагає асинхронної обробки, обробки помилок, динамічного керування станом черги та голосовим підключенням.

6. Модуль cogs.reaction_role

Надає функціонал надавання та прибирання ролі у користувача через відповідні команди !qiverole, !removerole. Також реалізує механізм автоматичного надання ролей на основі реакцій під повідомленням. Після додавання або видалення реакції бот відповідно надає або знімає роль. Реакції прив’язані до конкретних емодзі та повідомлень.

Лістинг *18 — Код команди !qiverole*

```
@commands.command()
    @commands.has_permissions(manage_roles=True)
    async def giverole(self, ctx, member: discord.Member, role:
discord.Role):
        try:
            await member.add_roles(role)
            await ctx.send(f'Successfully assigned the role'
                           f'          {role.name}          to
{member.mention}.')
            logging.info(f'Assigned role {role.name} '
                         f'to {member.name} via command.')
        except discord.Forbidden:
            await   ctx.send(f'Failed   to   assign   the   role
{role.name} '
                             f'to  {member.mention}  -  Missing
Permissions.')
            logging.error(f'Failed to assign role: {role.name}
'
                          f'to   {member.name}   -   Missing
Permissions.')
        except Exception as e:
            await ctx.send(f'An error occurred: {e}')
            logging.error(f'Failed to assign role via command:
{e}')
```

Лістинг *19 — Код команди !removerole*

```
@commands.command()
    @commands.has_permissions(manage_roles=True)
    async def removerole(self, ctx, member: discord.Member,
```

```
                        role: discord.Role):
        try:
            await member.remove_roles(role)
            await ctx.send(f'Successfully removed the role'
                           f'         {role.name}         from
{member.mention}.')
            logging.info(f'Removed role {role.name} '
                         f'from {member.name} via command.')
        except discord.Forbidden:
            await  ctx.send(f'Failed  to  remove  the  role
{role.name} '
                            f'from  {member.mention}  -  Missing
Permissions.')
            logging.error(f'Failed to remove role: {role.name}
'
                          f'from   {member.name}   -   Missing
Permissions.')
        except Exception as e:
            await ctx.send(f'An error occurred: {e}')
            logging.error(f'Failed to remove role via command:
{e}')
```

Лістинг *20 — Код функції додавання та прибирання ролі за реакцію*

```
def __init__(self, bot):
    self.bot = bot
    self.message_id = 1379343596406444122  # Replace with your
msg ID
    self.role_id = 1249701926468325407  # Replace with the role
ID
    self.emoji = "😁"  # The emoji to react with
```

```
@commands.Cog.listener()
async def on_raw_reaction_add(self, payload):
    logging.info(f'Reaction added: {payload.emoji} by
{payload.user_id}')
    if payload.message_id == self.message_id and str(
            payload.emoji) == self.emoji:
        guild = self.bot.get_guild(payload.guild_id)
        if guild is None:
            logging.warning(f'Guild not found:
{payload.guild_id}')
            return

        role = guild.get_role(self.role_id)
        if role is None:
            logging.warning(f'Role not found: {self.role_id}')
            return

        try:
            member = await guild.fetch_member(payload.user_id)
            if member is None:
                logging.warning(f'Member not found:
{payload.user_id}')
                return

            await member.add_roles(role)
            logging.info(f'Assigned role {role.name} to
{member.name}')
        except discord.Forbidden:
            logging.error(
                f'Failed to assign role:'
```

```
                f' {role.name} to {member.name} - Missing
Permissions')
        except Exception as e:
            logging.error(f'Failed to assign role: {e}')

@commands.Cog.listener()
async def on_raw_reaction_remove(self, payload):
    logging.info(f'Reaction removed: {payload.emoji} by
{payload.user_id}')
    if payload.message_id == self.message_id and str(
            payload.emoji) == self.emoji:
        guild = self.bot.get_guild(payload.guild_id)
        if guild is None:
            logging.warning(f'Guild not found:
{payload.guild_id}')
            return

        role = guild.get_role(self.role_id)
        if role is None:
            logging.warning(f'Role not found: {self.role_id}')
            return

        try:
            member = await guild.fetch_member(payload.user_id)
            if member is None:
                logging.warning(f'Member not found:
{payload.user_id}')
                return

            await member.remove_roles(role)
```

```
            logging.info(f'Removed role {role.name} from
{member.name}')
        except discord.Forbidden:
            logging.error(
                f'Failed to remove role: '
                f'{role.name} from {member.name} - Missing
Permissions')
        except Exception as e:
            logging.error(f'Failed to remove role: {e}')
```

Висновок

Модуль дозволяє адміністраторам знімати чи надавати певну роль користувачеві, а також автоматичне зняття чи додавання за реакцію. Значно спрощує роботу адміністраторів із розподілом ролей та створює інтерактивну взаємодію з користувачами. Він є прикладом реактивного програмування на Discord-події.

7. Модуль cogs.spam_filter

Являє собою продовження функцій для модерації серверу. Фільтрує спам-повідомлення, які надсилаються занадто часто одним користувачем. При виявленні підозрілої активності — надсилається попередження, а надмірну активність обмежують за допомогою автоматичного муту.

Лістинг *21 — Код для відстеження повідомлень та визначення параметрів*

```
def __init__(self, bot):
    self.bot = bot
    self.message_counts = defaultdict(list)    # Track user
messages
    self.warnings = defaultdict(int)  # Track user warnings
```

```
@commands.Cog.listener()
async def on_message(self, message):
    if message.author.bot:
        return

    # Parameters for spam detection
    time_limit = 10  # Time frame in seconds
    message_limit = 5  # Max messages allowed in the time frame
    warning_limit = 3  # Warnings before mute
    mute_duration = 5  # Mute duration in minutes

    # Track user messages
    current_time = datetime.utcnow()

self.message_counts[message.author.id].append(current_time)

    # Remove messages outside the time frame
    self.message_counts[message.author.id] = [
        msg_time           for           msg_time           in
self.message_counts[message.author.id]
        if       msg_time       >       current_time       -
timedelta(seconds=time_limit)
    ]

    # Check for spam
    if      len(self.message_counts[message.author.id])      >
message_limit:
        self.warnings[message.author.id] += 1
        await message.channel.send(
            f'{message.author.mention}, '
```

```
            f'you are sending messages too quickly.'
            f'            This            is            warning
#{self.warnings[message.author.id]}.')

        # Mute the user if they exceed the warning limit
        if self.warnings[message.author.id] >= warning_limit:
            await self.mute_user(message.author, mute_duration,
                                 message.channel)
```

Лістинг *22 — Функції для муту/анмуту та створення відповідної ролі*

```
async def mute_user(self, member, duration, channel):
    mute_role     =     discord.utils.get(member.guild.roles,
name="Muted")
    if mute_role is None:
        mute_role = await self.create_mute_role(member.guild)

    await member.add_roles(mute_role)
    await channel.send(f'{member.mention} has '
                       f'been muted for {duration} minutes.')
    logging.info(f'{member.name} has been muted for {duration}
minutes.')

    # Unmute after the duration
    await     self.unmute_after_duration(member,     duration,
mute_role)

async def create_mute_role(self, guild):
    mute_role = await guild.create_role(
        name="Muted",
        permissions=discord.Permissions(send_messages=False,
speak=False))
```

```
    for channel in guild.channels:
        await channel.set_permissions(mute_role,
                                      send_messages=False,
speak=False)
    return mute_role

async def unmute_after_duration(self, member, duration,
mute_role):
    await asyncio.sleep(duration * 60)
    await member.remove_roles(mute_role)
    logging.info(f'{member.name} has been unmuted.')
```

Висновок

Модуль cogs.spam_filter забезпечує захист від зловживань чат-функціоналом. Він демонструє використання лічильників, асинхронних таймерів та внутрішнього кешу для відстеження активності.

8. Модуль cogs.spam_filter

Відповідає за автоматичне надсилання привітальних повідомлень при приєднанні нового користувача до серверу. Реалізовано обробку події on_member_join. Повідомлення налаштовується та відправляється у визначений канал.

Лістинг *23 — Код функції привітання нового користувача*

```
def __init__(self, bot):
    self.bot = bot
    script_dir = os.path.dirname(os.path.abspath(__file__))
    self.greetings_path   =   os.path.join(script_dir,   '..',
'greetings.json')

@commands.Cog.listener()
```

```
async def on_member_join(self, member):
    try:
        with open(self.greetings_path, 'r') as f:
            greetings = json.load(f)
        greeting = random.choice(greetings)
        channel = discord.utils.get(member.guild.text_channels,
                                    name='welcome')

        if channel:
            await
channel.send(greeting.format(user=member.mention))
        else:
            logging.warning('Welcome channel not found.')
    except Exception as e:
        logging.error(f'Error sending greeting: {e}')
```

Висновок

Модуль cogs.welcome підвищує залученість нових учасників, дозволяє створити перше позитивне враження від сервера, а також сигналізує адміністрації про нові входи.

3.7 Структури даних

Для забезпечення ефективної роботи Discord-бота у процесі розробки були використані різні типи структур даних, які відповідають вимогам окремих модулів. Вибір структур ґрунтувався на необхідності зберігання інформації про черги, користувачів, ролі, повідомлення та інші об’єкти взаємодії у Discord-середовищі.

Нижче наведено основні структури даних, що використовуються в проєкті:

1. Списки (list)

У Python списки є базовою структурою даних, яка активно використовувалася для реалізації черги треків у модулі music. Зокрема, для зберігання порядку треків (self.queue) використовується список, що дозволяє додавати нові елементи в кінець черги та вилучати з початку за допомогою методу pop(0).

2. Словники (dict)

Словники використовуються у модулях leveling, spam_filter, reaction_role для зберігання зв’язків між унікальними ключами (наприклад, ID користувачів або повідомлень) та відповідними значеннями (досвід, час останнього повідомлення тощо).

3. Множини (set)

Використовуються для обробки унікальних об’єктів, наприклад, для відстеження вже оброблених реакцій у reaction_role або запобігання дублюванню певних подій.

4. Файли формату JSON

Для збереження даних між перезапусками бота (наприклад, у модулі leveling) використовується серіалізація у формат JSON. Це дозволяє зберігати стан прогресу користувачів (рівні, досвід) у текстовому вигляді та завантажувати його при наступному запуску.

5. Черги (queue)

У модулі music реалізована логіка черги пісень на базі списків, що дозволяє впорядковано додавати та відтворювати треки.

Використані структури даних відповідають вимогам продуктивності та логіки кожного окремого модуля. Списки забезпечують впорядкованість дій (черги), словники дозволяють організовувати дані у форматі ключ-значення (наприклад, рівень користувача або реакції), а JSON-файли — збереження стану.

Завдяки правильному підбору структур даних забезпечено як ефективну роботу окремих функцій, так і цілісність проєкту в цілому.

3.8 Проєкт інтерфейсу

Інтерфейс Discord-бота базується на взаємодії користувача з ботом через текстові повідомлення у середовищі Discord. Оскільки Discord сам по собі надає зручну платформу для спілкування, весь функціонал реалізований через командну строку та реакції, без необхідності створення окремого графічного інтерфейсу.

Командний інтерфейс

Кожна функція бота активується за допомогою певної команди, яка починається з символу-префікса (в даному застосунку – !). Наприклад:

- !play — команда для відтворення музики;
- !mute @user 1 — накласти тимчасовий мут користувачу на хвилину;
- !calc 5*7+2 — обчислення виразу(див. Рис. 9);
- !level — перегляд рівня користувача;
- !queue — перегляд черги треків.

Користувачі взаємодіють із ботом у звичному для себе текстовому форматі. Результати виконання команд бот надсилає у вигляді відповідей на повідомлення або у вигляді embed’ів (вбудованих кольорових блоків з інформацією).

Виведення інформації

Вивід результатів реалізовано через:

- текстові повідомлення (звичайні відповіді бота) (див. Рис. 9);
- ембеди (структуровані блоки з кольором, іконками, назвами);
- мультимедіа (зображення, аудіо).

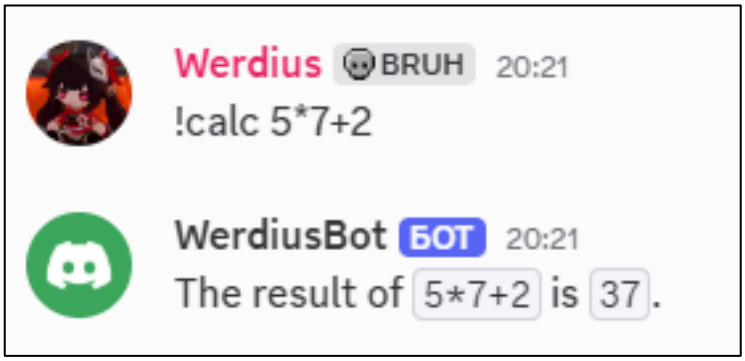


Рисунок *9 — Приклад виконання команди та її виведення до чату*

Взаємодія з голосовими каналами

Інтерфейс модуля music передбачає голосову взаємодію — бот підключається до каналу та відтворює музику. Користувач керує відтворенням через команди (!play, !stop, !skip), а бот повідомляє статус: "Now playing", "Skipped", "Queue is empty" тощо.

Висновок

Хоча Discord-бот не має класичного графічного інтерфейсу, він повністю забезпечує зручну, зрозумілу та інтуїтивну взаємодію з користувачами через текстові команди, повідомлення, реакції та голосовий супровід. Такий підхід дозволяє інтегрувати бота в будь-яку спільноту без додаткових інструментів, зберігаючи простоту використання та широку доступність функцій.

3.9 Реалізація та тестування

У межах дипломної роботи було реалізовано повноцінний багатофункціональний Discord-бот із модульною архітектурою, кожен компонент якого відповідає за окрему функціональність. Для реалізації проєкту використовувалися такі ключові технології:

- Мова програмування. Python.
- Основна бібліотека. Discord.py — для взаємодії з Discord API.
- Сторонні бібліотеки. Yt_dlp, ffmpeg, aiohttp, spotipy, python-dotenv, asyncio та інші.

Код бота структуровано за допомогою механізму cogs, де кожен модуль є окремим Python-файлом зі своїми командами, подіями та логікою(див. Рис. 10). Основна логіка завантаження модулів реалізована у головному файлі запуску (main.py), який під час старту ініціалізує об'єкт бота, підключає всі cogs та запускає клієнт Discord.

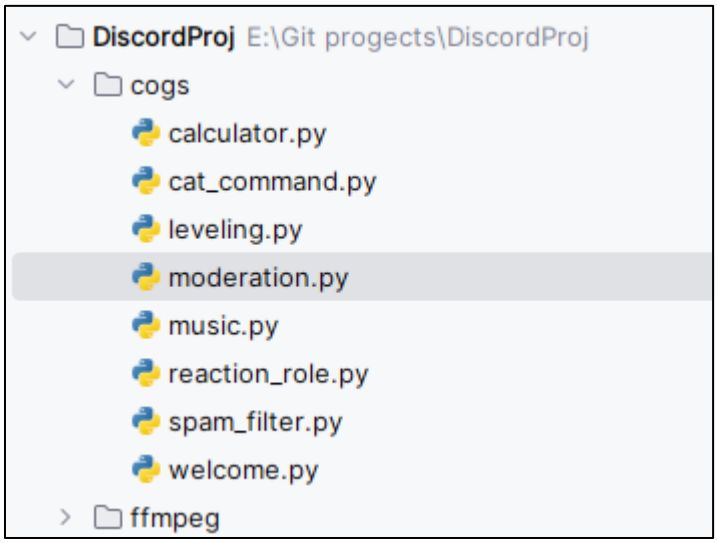


Рисунок *10 — Структура модулів(cogs) проєкту*

Реалізація функціоналу

Під час розробки були реалізовані такі функціональні модулі:

- moderation — бан(див. Рис. 11), кік, мут користувачів(захищений доступ);
- spam_filter — виявлення спаму, надання попередження та автоматичний мут;
- welcome — надсилання привітального повідомлення новим користувачам;
- reaction_role — команди додавання/прибирання ролей та

надання/видалення ролей за реакціями під повідомленням;

- music — відтворення музики з YouTube/Spotify;
- calculator — обчислення математичних виразів;
- cat_command — надсилання зображенньь через API;
- leveling — система рівнів активності користувачів.

Усі модулі реалізовано асинхронно з обробкою винятків, веденням логів та повідомленнями про помилки (див. Рис. 12).

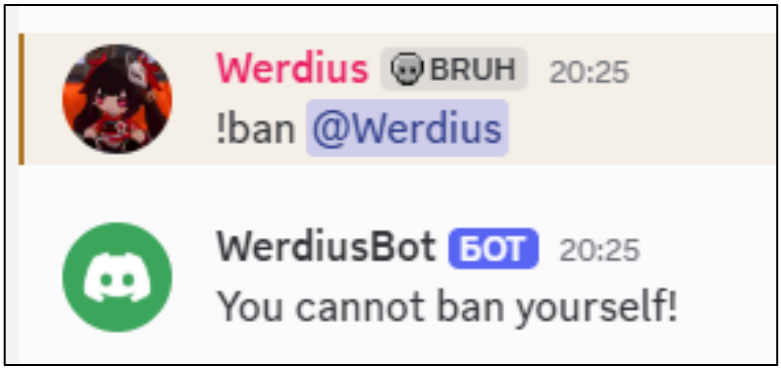


Рисунок *11 — Приклад виконання некоректної команди*

```
WARNING:root:werdius tried to ban themselves.
```

Рисунок *12 — Приклад виводу попередження у лозі терміналу про некоректну команду*

Тестування

Проєкт пройшов поетапне тестування в умовах тестового Discord-серверу. Було виконано:

1. Ручне функціональне тестування:

- перевірка виконання команд у чаті (!play, !kick, !level тощо);
- тестування поведінки реакцій, привітань, роботи з голосовими каналами;

- перевірка коректного реагування на помилкові дії або неправильний синтаксис.

2. Перевірка на відповідність стилю коду:

- за допомогою flake8[11] і pylint[12] було проаналізовано код на відповідність PEP8;

3. Логування:

- усі події та помилки записуються до лог-файлів;
- це дозволяє аналізувати роботу бота після тестування.

Виявлені помилки та усунення

Під час тестування були виявлені й усунені наступні помилки:

- відсутність повідомлення при команді !skip без активного треку у черзі >> виправлено логіку відображення (див. Рис. 13).



Рисунок *13 — Приклад виконання команди !skip після виправлення помилки*

- неправильна обробка реакцій під повідомленнями >> додано перевірку ID повідомлення;
- некоректне очищення черги після команди !stop >> додано обнулення черги та прапору currently_playing;
- блокування доступу до YouTube через CAPTCHA >> реалізовано резервний пошук;

Тестування показало, що бот стабільно виконує всі передбачені функції в реальному середовищі Discord. Завдяки модульному підходу, обробці помилок та логуванню система легко розширюється, а окремі компоненти можуть бути повторно використані або оновлені без втручання в інші частини проєкту.

ВИСНОВКИ

1. У процесі виконання кваліфікаційної роботи бакалавра була проведена робота з літературними джерелами, які корелюються з темою автоматизації модерації, Discord API, чат-ботів та архітектури програмних систем.
2. Проведено аналіз існуючих Discord-ботів (MEE6, JuniperBot, HydraBot), визначено їх переваги та недоліки, а також обґрунтовано доцільність створення власного рішення з відкритим кодом та адаптованим функціоналом.
3. Здійснено вибір мови програмування Python і бібліотеки discord.py, які забезпечили гнучкість, простоту реалізації та підтримку асинхронної обробки подій.
4. Спроєктовано та реалізовано модульну архітектуру Discord-бота з використанням системи cogs, що дозволяє розділити функціональність на незалежні блоки для зручного тестування й масштабування.
5. Реалізовано ключові модулі: moderation, spam_filter, welcome, reaction_role, music, calc, а також система рівнів активності та розважальні функції, що дозволяє використовувати бота як повноцінний інструмент керування Discord-сервером.
6. Проведено ручне тестування і аналіз стилю та коду всіх функціональних компонентів.
7. Розроблений програмний засіб має практичну цінність для адміністраторів Discord-серверів, сприяє зменшенню ручної модерації, покращенню організації спільноти та підвищенню рівня залученості користувачів.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Automate the Boring Stuff with Python : Al Sweigart, 2019. 592 p. (дата звернення: 05.02.2025).
2. Python Crash Course : Eric Matthes, 2019. 544 p. (дата звернення: 05.02.2025).
3. Документація до Discord Developer Portal : веб-сайт. URL: https://discord.com/developers/docs/intro (дата звернення: 01.03.2025).
4. Документація до discord.py : веб-сайт. URL: https://discordpy.readthedocs.io/en/stable/ext/tasks/index.html (дата звернення: 01.03.2025).
5. Документація до Spotify for Developers : веб-сайт. URL: https://developer.spotify.com/documentation/web-api (дата звернення: 15.03.2025).
6. Головна сторінка ffmpeg : веб-сайт. URL: https://github.com/FFmpeg/FFmpeg (дата звернення: 15.03.2025).
7. Документація до ffmpeg: веб-сайт. URL: https://ffmpeg.org/ffmpeg.html (дата звернення: 15.03.2025).
8. Документація до Discord Logging : веб-сайт. URL: https://discordpy.readthedocs.io/en/stable/logging.html (дата звернення: 15.03.2025).
9. Стаття щодо розвитку Discord у New York Times: веб-сайт. URL: https://www.nytimes.com/2021/12/29/business/discord-server-social-media.html (дата звернення: 20.03.2025).
10. Сторінка офіційного блогу Discord : веб-сайт. URL: https://discord.com/blog/discords-next-chapter (дата звернення: 21.03.2025).
11. Головна сторінка Flake8(Tool For Style Guide): веб-сайт. URL: https://flake8.pycqa.org/en/latest/ (дата звернення: 24.05.2025).
12. Документація до PyPI(pylint, static code analyser): веб-сайт. URL: https://docs.pypi.org/ (дата звернення: 24.05.2025).

13. Жевнерюк О., бакалавр, Скрипник І.А. к. ф.-м. н., доцент — науковий керівник. Багатофункціональний бот-модератор на платформі discord. // ЗБІРНИК наукових праць студентів, здобувачів PhD, докторантів і молодих вчених «Молода наука-2025». Запоріжжя : Запорізький національний університет, 2025. Т. 5. С. 390–391.

Декларація
академічної доброчесності
здобувача ступеня вищої освіти ЗНУ

Я, Жевнерюк Олександр Яковлевич, студент 4 курсу, форми навчання денної, Інженерного навчально-наукового інституту ім. Ю.М. Потебні ЗНУ, спеціальність 121 Інженерія програмного забезпечення освітньої програми "Програмне забезпечення систем", адреса електронної пошти ipz20bd-205@stu.zsea.edu.ua, — підтверджую, що написана мною кваліфікаційна робота на тему «**Багатофункціональний бот-модератор на платформі Discord**» відповідає вимогам академічної доброчесності та не містить порушень, що визначені у ст.42 Закону України «Про освіту», зі змістом яких ознайомлений.

- заявляю, що надана мною для перевірки електронна версія роботи є ідентичною її друкованій версії;

- згоден на перевірку моєї роботи на відповідність критеріям академічної доброчесності у будь-який спосіб, у тому числі за допомогою інтернет-системи, а також на архівування моєї роботи в базі даних цієї системи.

Дата 02.06.2025 _____ Жевнерюк Олександр Яковлевич
(підпис) (прізвище та ініціали) (студент)

Дата 02.06.2025 _____ Скрипник Ірина Анатоліївна
(підпис) (прізвище та ініціали) (керівник)