Центральноукраїнський національний технічний університет
Механіко-технологічний факультет
Кафедра кібербезпеки та програмного забезпечення

”Допущено до захисту”
Завідувач кафедри кібербезпеки
та програмного забезпечення
д.т.н., професор
_____ Олексій СМІРНОВ
“____” _____ 20__ р.

ВИПУСКНА КВАЛІФІКАЦІЙНА РОБОТА
за першим (бакалаврським) рівнем вищої освіти
на тему
“Програмне забезпечення чат-боту для платформи обміну повідомленнями Discord”

Виконав здобувач вищої освіти
IV курсу, групи КІ-21-2
ОПП «Комп’ютерна інженерія»
спеціальності 123 «Комп’ютерна інженерія»
_____ Графенюк В.О.
«____» _____ 20 р.

Керівник проекту
доктор технічних наук, професор
_____ Мелешко Є. В.
«____» _____ 20 р.
Рецензент _____
_____

м. Кропивницький

Центральноукраїнський національний технічний університет

Факультет *Механіко-технологічний*

Кафедра *Кібербезпеки та програмного забезпечення*

Освітній ступінь *бакалавр*

Галузь знань *12 "Інформаційні технології"*

Спеціальність *123 "Комп'ютерна інженерія"*

Освітньо-професійна (освітньо-наукова) програма *"Комп'ютерна інженерія"*

ЗАТВЕРДЖУЮ
Завідувач кафедри
д.т.н., проф.
_____О.А.Смірнов
« __ » _____ 20__ року

З А В Д А Н Н Я
НА КВАЛІФІКАЦІЙНУ БАКАЛАВРСЬКУ РОБОТУ СТУДЕНТУ

Графенюк Валерії Олександрівні
(прізвище, ім'я, по батькові)

1. Тема роботи *Програмне забезпечення чат-боту для платформи обміну повідомленнями Discord*

керівник роботи *Мелешко Єлизавета Владиславівна, д-р техн. наук, професор*
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом вищого навчального закладу *№47-02 від 17.01.2025 року*

2. Строк подання студентом роботи до захисту *24.05.2025 р.*

3. Мета та завдання кваліфікаційної бакалаврської роботи*: Метою розробки є програмне забезпечення чат-боту для платформи обміну повідомленнями Discord*

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)

1. Призначення та область використання.

2.Перегляд аналогічних існуючих систем.

3. Опис і обґрунтування проектних рішень.

4. Етапи програмування системи.

5. Впровадження системи в промислову експлуатацію.

6. Висновки

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

Структурна схема системи	*1 аркуш*
Функціональна схема системи	*1 аркуш*
Діаграма процесів	*1 аркуш*
Блок-схема алгоритму роботи додатку	*2 аркуша*

6. Дата видачі завдання 17.01.2025 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної бакалаврської роботи	Строк виконання етапів кваліфікаційної бакалаврської роботи	Примітка
1.	Аналіз існуючих систем	10.03.2025 р.	
2.	Постановка задачі, оформлення ТЗ	15.03.2025 р.	
3.	Розробка моделі компонента	20.03.2025 р.	
4.	Розробка структур даних	25.03.2025 р.	
5.	Розробка алгоритмів зв’язку та відображення	30.03.2025 р.	
6.	Програмування алгоритмів	10.04.2025 р.	
7.	Оформлення ПЗ	17.04.2025 р.	
8.	Попередній захист роботи	22.05.2025 р.	

Студент _____ _____
(підпис) (прізвище та ініціали)

Керівник роботи _____ _____
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Графенюк В. О. Програмне забезпечення чат-боту для платформи обміну повідомленнями Discord. 123 Комп'ютерна інженерія. Центральноукраїнський національний технічний університет. Кропивницький. 2025.

У даній кваліфікаційній бакалаврській роботі розроблено програмне забезпечення, яке призначено для реалізації чат-боту для платформи обміну повідомленнями Discord.

Метою роботи є створення програмного забезпечення чат-боту для платформи обміну повідомленнями Discord.

Результат роботи – програмна реалізація чат-боту для платформи обміну повідомленнями Discord.

В процесі роботи над реалізацією системи виконано дослідження існуючих методів, алгоритмів та програмних засобів. Розроблено та реалізовано власне програмне забезпечення, здійснено опис всіх його компонентів.

Розроблено зручний інтерфейс користувача. Наведені інструкції по роботі з програмними засобами.

Програма може використовуватися на ПЕОМ архітектури IBM PC з ОС Windows 10/11.

Програму розроблено на мові програмування Python.

Ключові слова: комп'ютерна інженерія, чат-бот, платформа обміну повідомленнями, Discord

ABSTRACT

Hrafeniuk V. O. Software for Chat-Bot of the Discord Messaging Platform. Central Ukrainian National Technical University. Kropyvnytskyi 2025.

This bachelor's qualification work presents software designed to implement a chat-bot for the Discord messaging platform.

The goal of the project is to create a chat-bot for Discord.

Project outcome – a fully functioning software implementation of a Discord chat-bot.

During development, existing methods, algorithms, and software tools were studied. Proprietary software was designed and implemented, and all of its components are described.

A user-friendly interface has been developed, and instructions for working with the software are provided.

The program can be used on IBM PC-compatible computers running Windows 10/11 and is written in the Python programming language.

Keywords: computer engineering, chat-bot, messaging platform, Discord

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ І ТЕРМІНІВ

API (Application Programming Interface – інтерфейс прикладного програмування) – узгоджений набір викликів і правил, через який одна програма безпечно користується функціями іншої.

CDN (Content Delivery Network – мережа доставки контенту) – розподілена по світу інфраструктура серверів, що кешує й швидко віддає статичні файли користувачам.

CI/CD (Continuous Integration / Continuous Deployment) – практика безперервного злиття коду та автоматичного його розгортання у продакшн-середовище.

CLI (Command-Line Interface – консольний інтерфейс) – спосіб керування програмою через текстові команди у терміналі.

Cog (discord.py) – клас-контейнер, що групує пов’язані команди та слухачі подій бота; забезпечує модульність коду.

DB / БД (DataBase – база даних) – структуроване сховище для довготривалого збереження й пошуку інформації.

Discord Gateway – постійне WebSocket-з’єднання, через яке клієнти та боти миттєво отримують усі події сервера (повідомлення, реакції, зміни статусу).

Docker-контейнер – ізольоване середовище, яке пакує застосунок разом із залежностями для передбачуваного запуску будь-де.

Embed – багате повідомлення Discord із заголовком, полями, кольоровою смугою й ескізом, яке бот формує через API.

FFmpeg (Fast Forward MPEG) – набір консольних утиліт для запису, конвертації та трансляції аудіо- та відеопотоків.

GPU (Graphics Processing Unit – графічний процесор) – паралельний співпроцесор, що пришвидшує обчислення, зокрема запуск нейронних мереж.

HTTP (HyperText Transfer Protocol) – основний протокол передачі даних у Всесвітній павутині.

Intent (Discord) – набір категорій подій, доступ до яких бот має право запитувати; вмикаються у панелі розробника.

JSON (JavaScript Object Notation) – легковаговий текстовий формат обміну даними «ключ – значення», популярний в API.

LLM (Large Language Model – велика мовна модель) – нейромережа, натренована на гігантських корпусах тексту (приклад – GPT-4o) і здатна генерувати зв'язну мову.

Open-Meteo API – безкоштовний веб-сервіс, який надає погодні прогнози у форматі JSON.

ORM (Object-Relational Mapping) – технологія, що відображає таблиці БД у вигляді об'єктів мови програмування.

PNG (Portable Network Graphics) – растровий графічний формат, який підтримує прозорість та безвтратну компресію.

Prometheus-метрика – числовий показник (латентність, кількість таймерів тощо), який збирається моніторинговою системою Prometheus.

Reaction Roles – механізм, за якого користувач сам «підписується» на роль, натискаючи кнопку або ставлячи реакцію.

REST (Representational State Transfer) – стиль побудови веб-API на основі стандартних методів HTTP і чітких URI-ресурсів.

RFC 5545 RRULE – формальна нотація стандарту iCalendar для опису повторюваності подій (аналог cron).

SaaS (Software as a Service) – модель, у якій програмне забезпечення надається користувачам як он-лайн-сервіс за підпискою.

SDK (Software Development Kit) – комплект бібліотек, інструментів і документації для створення програм під конкретну платформу.

Slash-команда – команда, що викликається символом «/» і має автодоповнення параметрів прямо в інтерфейсі Discord.

SQL (Structured Query Language) – універсальна мова запитів до реляційних баз даних.

SQLite – легка вбудована SQL-база, що працює як єдиний файл без окремого серверного процесу.

Streaming Presence – спеціальний статус Discord, що з'являється, коли користувач транслює відео; використовується ботами для авто-оголошень.

TTL (Time To Live – «час життя») – обмеження тривалості кешування даних або існування мережевих пакетів.

URL (Uniform Resource Locator) – унікальний текстовий рядок, який однозначно адресує ресурс в Інтернеті.

WebSocket – двосторонній протокол поверх TCP, який утримує постійне з'єднання між клієнтом і сервером без повторних HTTP-запитів.

yt-dlp (YouTube-download-plus) – активний форк youtube-dl, що дозволяє отримувати прямі медіапотоки й метадані з відеосервісів.

ВСТУП

Актуальність теми. У сучасному інформаційному просторі обсяг комунікацій у цифрових месенджерах зростає експоненційно. Однією з найбільш динамічних екосистем є платформа Discord, що за рахунок поєднання текстових, голосових і відеоканалів переросла з сервісу для геймерів у мультифункціональне середовище для освітніх, професійних та волонтерських спільнот. Станом на 2023 рік платформа налічувала близько 200 млн активних користувачів щомісяця [10], а експерти прогнозують понад 650 млн зареєстрованих облікових записів до кінця 2025 року [26]. Зростаюча аудиторія зумовлює потребу в автоматизованих інструментах для модерації, підтримки користувачів і інтеграції зовнішніх сервісів, що робить чат-боти ключовим компонентом сучасних серверів Discord. Паралельно стрімко розвивається ринок чат-ботів загалом – у 2024 році глобальний обсяг цієї індустрії оцінювався у 7,76 млрд дол. США з прогнозованим середньорічним темпом зростання 23 % у 2025-2030 рр. [7].

Широке впровадження ботів у комерційних та освітніх проєктах зумовлене переходом до цілодобового сервісу, мінімізації часу відповіді, а також можливостями штучного інтелекту для персоналізації спілкування.

Особливу актуальність розробка власного програмного забезпечення для Discord має в українському контексті. Після початку повномасштабної війни ця платформа стала одним із головних каналів координації IT-спільнот, волонтерських ініціатив і навчальних груп. Водночас більшість існуючих ботів орієнтовані на англомовну аудиторію та не враховують специфіку локального контенту й автоматизації. Створення гнучкого, україномовного чат-боту відкриває можливості:

– Автоматизована підтримка спільнот – миттєве надання довідок, розсилок та відповідей на типові запитання.

– Модерація та безпека – фільтрація неприйнятного контенту і протидія

фішинговим повідомленням.

– Інтеграція зовнішніх сервісів – підключення навчальних платформ, систем тестування або аналітики, що особливо актуально для університетів і дослідницьких груп.

– Локалізовані сценарії – робота з українськими API, сповіщення про національні події, адаптація до правопису та культурних норм.

Мета й завдання дослідження. Метою роботи є створення програмного забезпечення чат-боту для платформи обміну повідомленнями Discord.

Для досягнення поставленої мети визначена програма дослідження, що складається з наступних завдань:

– Огляд існуючих систем для реалізації чат-ботів для платформи обміну повідомленнями Discord.

– Проектування системи для реалізації чат-боту для платформи обміну повідомленнями Discord.

– Програмна реалізація системи для реалізації чат-боту для платформи обміну повідомленнями Discord.

Задачі, пов’язані зі створенням чат-ботів для месенджерів, постійно вимагають нових підходів до проектування та реалізації, тому що додатки обміну повідомленнями постійно змінюються та оновлюються.

Таким чином, розробка програмного забезпечення чат-боту для платформи обміну повідомленнями Discord є своєчасним і практично значущим завданням, що поєднує тренди розвитку глобального ринку чат-ботів, зростаючу популярність Discord та потреби українських користувачів у локалізованих цифрових інструментах. Реалізація такого проєкту сприятиме підвищенню якості сервісів, оптимізації ресурсів модераторів і розширенню можливостей комунікації в академічному та професійному середовищах України.

Практична цінність отриманих результатів полягає в тому, що розроблені алгоритми дозволяють успішно здійснити реалізацію чат-боту для платформи обміну повідомленнями Discord.

1 ПРИЗНАЧЕННЯ ТА ОБЛАСТЬ ВИКОРИСТАННЯ

1.1 Призначення системи

Розроблене програмне забезпечення призначене для автоматизації й удосконалення комунікаційних процесів у середовищі Discord-серверів.

Розроблений бот призначений стати «розумним помічником» Discord-спільноти, покриваючи щоденні потреби, що не стосуються музики чи суворої модерації. Він дозволяє учасникам планувати та організовувати свою роботу безпосередньо в чаті: створювати особисті й командні нагадування з гнучкими інтервалами, швидко запускати опитування і збирати зворотний зв’язок, автоматично формувати щоденні стендап-звіти та закріплювати їх для команди.

Додатково бот надає корисні інтеграції: миттєвий переклад повідомлень між українською й англійською, показ актуальної погоди за запитом, а також систему самовидачі ролей через інтерактивні кнопки.

Усе це реалізовано на сучасних slash-командах Discord, тож користувачам не потрібно запам’ятовувати складні префікси, а адміністратори отримують чистий лог дій без перевантаження головного модуля бота.

Бот також виконує роль «лаіт-дешборду» для сервера: усі події, пов’язані з нагадуваннями, опитуваннями та стендапами, фіксуються в окремому журналі, що полегшує аналіз активності й дає змогу відстежувати, як команда дотримується дедлайнів. Завдяки підтримці кнопок і випливних меню користувачі взаємодіють із ботом інтуїтивно – наприклад, одну й ту саму роль можна увімкнути або вимкнути простим натиском, без реакцій-емодзі чи складних команд.

1.2 Область застосування

Розроблене програмне забезпечення може використовуватися усюди, де Discord-сервер виконує роль «центру координації», а користувачам потрібні легкі інструменти самоорганізації:

– *Робочі та проєктні команди* – від невеликих стартапів до розподілених open-source-груп. Нагадування допомагають тримати дедлайни, стендапи фіксують щоденний поступ, а швидкі опитування спрощують прийняття рішень без довгих дискусій.

– *Навчальні курси й студентські гурти* – викладач створює строкові «дзвіночки» про здачу лабораторних, публікує опитування з теми наступної лекції, а бот акуратно закріплює відповіді та зберігає журнал активності для заліку.

– *Волонтерські й громадські ініціативи* – коли треба координувати чергування, збір ресурсів чи спільні виїзди. Самопризначувані ролі через кнопки дозволяють учасникам самостійно записуватися в зміни та отримувати лише релевантні пінги.

– *Геймерські клан-ком'юніті* – бот нагадує про рейди чи турніри, збирає реакції-реєстрації, відправляє прогноз погоди в ігровому світі (якщо інтегрувати API) й дає можливість миттєво перекладати тактичні повідомлення для міжнародного складу.

– *Онлайн-івенти, хакатони, конференції* – опитування визначають тему lightning talk, нагадування не дають пропустити доповідь, а ролі «Спікер», «Організатор», «Учасник» роздаються без втручання модераторів.

Скрізь, де потрібна дисципліна без «важкої артилерії» складних таск-менеджерів, Utility Bot забезпечує мінімально достатню функціональність прямо в чаті, зберігаючи звичний темп спілкування та не перевантажуючи головного бота сервера.

2 ПЕРЕГЛЯД АНАЛОГІЧНИХ ІСНУЮЧИХ СИСТЕМ

2.1 Огляд існуючих систем, технологій, архітектур, програмних рішень за профілем теми кваліфікаційної бакалаврської роботи

Було здійснено огляд популярних універсальних та утилітарних Discord-ботів, які покривають схожі потреби з чат-ботом, що треба розробити у даній кваліфікаційній роботі. Нижче наведено результати проведеного огляду.

MEE6

Один із наймасовіших SaaS-ботів – поєднує автоперевірку повідомлень, рівні активності, вручення ролей, реакційні опитування й базовий плеєр. Більшість продвинутих плагінів (музика з YouTube, запис стрімів, кастомний дашборд) доступні тільки в Premium-підписці, тож для великої спільноти безкоштовного функціоналу часто бракує.

Dyno

Класика «модераційного ножа» з 2017 р. – авто-мод, фільтри, автоматичні попередження, журнал дій із розгорнутими кейс-ID. Має web-дашборд та базові музичні команди, але музика вимкнена за замовчуванням і потребує окремого ввімкнення; за глибші лог-фільтри та AutoMod v2 також просять платну підписку.

Carl-bot

Фокус на Reaction Roles (до 250 ролей у різних режимах), кастомні теги, жорсткий авто-мод і ведення докладних логів. Саме Carl-bot найчастіше радять, коли потрібна самовидача ролей без коду. Однак нагадувачі в ньому примітивні (лише однократні) і немає внутрішнього музичного плеєра.

Hydra

Спочатку позиціонувався як «ідеальний музичний бот», але після обмежень Discord 2023 тимчасово вимикав музику; нині повернув плеєр для

преміум-серверів і розширився дашбордом, рівнями й лідербордом. Для стрім-сповіщень чи карток статистики модулів немає.

FredBoat / Jockie Music / Hydra-Music-Fork

Нішові «чисті» музичні боти: підтримують YouTube, Spotify і SoundCloud, мають чергу та плейлисти, але не пропонують ані модерації, ані логів, ані нагадувань. Їх ставлять у парах із Dyno чи Carl-bot. (Джерело – спільнота Reddit 2025)

Reminder Bot / Recurring Reminders

Спеціалізуються винятково на одноразових і повторюваних «remind me»; мають зручні slash-команди й веб-дашборд, але не інтегруються з реакційними ролями та опитуваннями; пісочниця для стендапів відсутня. Доступ до повторюваних тригерів часто прив’язаний до Patreon-підписки.

Simple Poll / Poll-Bot Advanced

Дають красиві опитування-Embed з emoji-кнопками й таймером, але не мають системних нагадувань, ролей чи звітів; у великих серверах доводиться ставити додатковий бот для AutoMod і ще один – для музики.

2.2 Обґрунтування вибору засобів для побудови системи та мови програмування

Для розробки програмного забезпечення для моделювання роботи комп’ютерних мереж було обрано наступні засоби:

1. Мова програмування Python 3.12. Обґрунтування вибору:

– Швидкість розробки. Для чат-ботів критично важлива швидка ітерація: нові команди додаються за лічені хвилини, баги виправляються «на гарячу». У Python мінімальний «синтаксичний шум», тому читається та редагується навіть неавторами.

– Асинхронність «з коробки». Модуль asyncio і сучасний синтаксис async / await дозволяють обробляти тисячі подій Discord, не блокуючи цикл, а отже

тримати музику, нагадування й API-запити паралельно без окремих потоків.

– Велика екосистема. Усі потрібні інтеграції (FFmpeg, yt-dlp, Pillow, OpenAI, Prometheus-клієнт, Docker SDK) мають зрілі Python-біндінги, що усуває необхідність писати власні C++-розширення чи shell-скрипти.

– Крос-платформеність. Бот однаково працює на Windows-ПК (для локальної відладки) та на Linux-VPS / Docker-хостах у продакшені.

2. Discord-SDK – discord.py 2.4 (x API v10). Обґрунтування вибору:

– Офіційна сумісність із Discord 2025. Форк підтримує Slash-команди, інтерактивні компоненти, intent-v10 та новий формат дозволів, тому не доведеться колупати raw WebSocket-payload.

– Поширена спільнота. Доступні сотні прикладів, плагіни і готові Cog-и; при виникненні проблеми майже завжди є відповіді на GitHub / Stack Overflow.

– Підтримка типів. Проєкт post-pepified: pyright / mypy фіксують помилки ще до запуску.

3. yt-dlp + FFmpeg для відтворення мультимедіа. Обґрунтування вибору:

– yt-dlp – активний форк youtube-dl з частими патчами під зміни YouTube та Twitch, забезпечує стабільний забір аудіо-потоків.

– FFmpeg дає можливість перекодувати та повторно з’єднувати потік без затримок; це промисловий стандарт, тож виникнення «дивних» форматів мінімізується.

4. SQLite → PostgreSQL для зберігання даних. Обґрунтування вибору:

– SQLite ідеально підходить для одиночних інстансів: проста, не потребує демона, підключається одним файлом на VPS.

– PostgreSQL легко підставити замість SQLite через asyncpg і ORM-шару; це важливо при масштабуванні на кілька контейнерів або якщо знадобляться складні запити (наприклад, аналітика участі в опитуваннях).

5. Docker (+ docker-compose) для контейнеризації. Обґрунтування вибору:

– Чітке ізолювання залежностей: різні версії FFmpeg чи discord.py не конфліктують із системними пакетами.

– Спрощений CI/CD: образ збирається GitHub Actions, деплоїться однією командою на Fly.io або Railway.

– Локальна і продакшн-середа ідентичні, тому «працювало у мене» зникає.

6. Prometheus + Grafana для моніторингу. Обґрунтування вибору:

– Нативний клієнт на Python дає метрики latency, активні нагадування, розмір черги музики.

– Grafana-дашборд дозволяє адміністраторам візуально бачити стан обох ботів без підключення до консолі.

7. aiohttp + aiocache для кешу і HTTP. Обґрунтування вибору:

– aiohttp широко використовується в екосистемі discord.py; він же лежить в основі самого REST-клієнта бібліотеки.

– aiocache (або in-memory dict із TTL) зменшує навантаження на зовнішні API (Open-Meteo, OpenAI), що важливо при безключових тарифах.

8. Лінійка DevOps-інструментів. Обґрунтування вибору:

– GitHub Actions автоматизує lint → test → build → deploy.

– Ruff швидко перевіряє стиль і знаходить помилки без повільного flake8+pylint стека.

– pytest-asyncio дає можливість тестувати coroutines unit-тестами, гарантує, що нагадування або опитування не «висять» між івент-лупами.

9. Розділення бота на два модулі Brave та Utility. Обґрунтування вибору:

– Надійність. Голосові стріми (Brave) вимагають стабільного з’єднання і високого пріоритету CPU; таймери й API-запити (Utility) – ні. Рознесення процесів усуває взаємні лаги.

– Чиста відповідальність. Якщо завтра з’явиться ще один бот (наприклад, для ігор), ми додаємо третій контейнер, не ускладнюючи існуючі.

– Гнучкий доступ. Brave Bot може мати максимум прав (керування

каналами), Utility – обмежені до Manage Roles; це спрощує аудит безпеки.

Python і його сучасний async-стек забезпечують швидку розробку, а комбінація discord.py 2.4, yt-dlp, FFmpeg, Docker та Prometheus дає зрілу, добре підтримувану основу. Розподіл функцій між двома ботами підвищує стабільність і масштабованість, залишаючи кожен компонент легким для підтримки й подальшого розширення.

2.3 Розгорнута постановка завдання

Метою роботи є створення програмного забезпечення чат-боту для платформи обміну повідомленнями Discord.

Основні завдання:

1. Створити один Discord-сервер, який виконує ролі майданчика для спілкування, навчання, роботи та/або волонтерської координації.

2. Створити для користувачів як «атмосферні» функції (музика, привітання, статистика), так і серйозні інструменти самоорганізації (нагадування, опитування, ролі за підпискою).

3. Щоб уникнути монолітного коду та перевантаження процесу, систему слід розділити на два окремі боти, що працюють паралельно та спілкуються лише через API Discord і спільну БД.

Компоненти системи:

1. *Brave Bot*:

– Мультимедіа: відтворення аудіо/відео з YouTube через yt-dlp та FFmpeg; керування чергою; команди play/stop.

– Модерація: чистка повідомлень, кік/бан/анбан, мут/анмут, автоматична видача ролі новачкам.

– Журнали: видалення, редагування, зміни ролей, створення/видалення каналів – усе фіксується в лог-каналі.

– Картки-статистики: динамічна генерація PNG-карток користувача та

сервера (Pillow).

– Сповіщення про стрім: відстеження статусу «Streaming», автоматична роль «У прямому ефірі» + інформативні Embed-и.

2. *Utility Bot*:

– Нагадувач: одноразові та повторювані нагадування в DM або канал; збереження у SQLite / PostgreSQL.

– Опитування: Slash-команда, що генерує кнопки-варіанти та підбиває підсумок по таймеру.

– Щоденний стендап: збір коротких звітів, агрегування, пін у канал.

– Погода: інтеграція з Open-Meteo, кешування відповіді.

– Переклад: швидкий translate UA⇄EN через OpenAI / DeepL.

– Самовидача ролей: повідомлення з кнопками для підписки на теги-ролі (наприклад, «Анонси», «Бета-тест»).

– Моніторинг: метрики latency, кількість активних нагадувань, файл-та-канал логів.

Цілі та очікуваний результат:

– Надати користувачам повний спектр інструментів – від музики й «привабливої» статистики до тайм-менеджменту – без потреби покидати чат або ставити зовнішні SaaS.

– Розподілити навантаження. Brave Bot обробляє голосовий трафік і важкі зображення, Utility Bot – таймери та зовнішні API-запити.

– Забезпечити модульність. Кожна нова функція додається окремим Cog-ом, кожен бот може оновлюватися незалежно.

Функціональні вимоги:

– Команди Brave Bot – префікс #, сумісні з legacy-ботами; відповіді українською, емодзі-реакції.

– Команди Utility Bot – виключно Slash (/), автодоповнення параметрів; усі взаємодії через кнопки/меню.

– Стійкість нагадувачів – задачі зберігаються в БД, перезапускаються

після – крашу; максимум 5 ретраїв, потім сповіщення адміну.

– Безпечна робота з ролями – бот додає/знімає лише перелічені ролі; не може впливати на адміністративні.

– Логування – вивід у utility-log і brave-log, а також ротаційний файл із retention 30 днів.

Нефункціональні вимоги:

– Мови: інтерфейс українською; перекладач – UA ⇄ EN.

– Продуктивність: голосовий потік не переривається більш ніж на 2 с у пікових умовах; нагадування – відхилення ≤ ±30 с.

– Масштабованість: перехід з SQLite на Postgres мінімальним config-патчем; можливість запуску в окремих Docker-контейнерах.

– Безпека: токени та API-ключі лише у змінних середовища; боти мають мінімально необхідні права.

– Ліцензія проєкту: Creative Commons; вихідний код у приватному GitHub-репозиторії з CI / CD.

Інтеграція та взаємодія компонентів:

– Обидва боти слухають події одного сервера, але різні префікси/командні простори виключають конфлікти.

– Спільна таблиця scheduled_tasks у БД дозволяє за потреби мігрувати нагадування між ботами.

– Лог-канали розділені, проте критичні винятки обоє дублікують у #service-status.

– При необхідності Brave Bot може викликати Utility Bot через Discord HTTP API (наприклад, щоб створити опитування з інтро-Embed-у).

У підсумку реалізована система з двох доповнювальних Discord-ботів закриває повний спектр потреб спільноти, підсилює залученість учасників і не вимагає зовнішніх SaaS-сервісів для рутинних організаційних завдань.

3 ОПИС І ОБҐРУНТУВАННЯ ПРОЕКТНИХ РІШЕНЬ

3.1 Опис функціонування системи

Щоб уникнути монолітного коду та покращити можливості підтримки й масштабованість системи, було прийнято рішення розділити її на два окремі чат-боти, що працюють паралельно та спілкуються лише через API Discord і спільну БД. Таким чином розроблена система складається з наступних компонентів:

1. **Brave Bot** – концентрується на мультимедіа, модерації та «атмосфері» сервера.

2. **Utility Bot** – дає інструменти для самоорганізації, інтеграції й аналітики.

Разом обидва чат-боти покривають потреби від музичного фону й привітань до нагадувань, опитувань, перекладу й метрик, лишаючись при цьому розділеними процесами, які легко розширювати незалежно.

Brave Bot – реалізовані функції й особливості їхнього виконання:

1. Музика («Музика» Cog):

– play приймає або пряме посилання, або текст для YouTube-пошуку. yt-dlp витягає JSON-метадані без завантаження файлу, бере найкращий аудіо-потік, а discord.FFmpegPCMAudio з FFmpeg передає його у голосовий канал. Якщо автор уже слухає музику, бот лише додасть трек у чергу, не підключаючись повторно.

– stop перевіряє прапорець voice_connected; якщо бот у каналі, від’єднується й обнуляє чергу. Команда реагує емодзі ⏹ і зменшує спам у чаті.

– FFmpeg викликається без абсолютного шляху, щоб контейнер довіряв системному PATH, а reconnection-flags дозволяють безшовний перезапуск потоку, коли YouTube міняє CDN-URL.

2. Автоматична роль («АвтоРоль» Cog):

– Подія on_member_join дістає AUTO_ROLE_ID і додає роль разом із

вітальним Embed-ом. Вітальний канал береться як guild.system_channel; якщо адміністратор замінить системний канал, код не треба міняти.

– Додавання ролі відбувається з обґрунтуванням у полі reason – це видно в Audit Log Discord, що спрощує аудит безпеки.

3. Журнали («Журнали» Cog):

– Відслідковуються on_message_delete, on_message_edit, on_member_update, on_guild_channel_create/delete.

– У кожному Embed дублюється аватар автора, час і детальний опис дії. Усі логи сходяться в один канал, визначений константою LOGS_CHANNEL_ID, тому не потрібно заводити окремі канали «delete-log» та «edit-log».

– Збереження повідомлень у файл не робиться для економії диска, проте лог-канал можна експортувати стандартним Discord-архіватором.

4. Картки статистики («Статистика» Cog):

– Команди card_user та card_server беруть аватар/іконку, ресайзять до 100 × 100 px, накладають на фон #232529 і підписують дані шрифтом arial.ttf.

– Використаний Pillow-алгоритм LANCZOS дає чіткі аватари навіть при масштабуванні, а збереження у PNG з прозорістю утримує колірну схему Discord.

– Команди одразу видаляють власний виклик (ctx.channel.purge(limit=1)), щоби картка залишалася єдиним повідомленням у стрічці.

5. Стрім-сповіщення («СтрімСповіщення» Cog):

– Слухається on_presence_update. Якщо статус учасника переходить із не-Streaming у Streaming, бот видає роль LIVE й публікує Embed із кольором Colour.brand_red().

– Час початку стріму фіксується у глобальній змінній time1; коли стрім завершується, Embed «🏁 Стрім завершено» підраховує тривалість із точністю до секунди та видаляє початковий анонс, аби логіка не засмічувала канал.

– Для безпеки роль знімається одразу після втрати статусу Streaming, щоб учасники не зловживали видимістю.

6. Модерація («Модерація» Cog):

– clear – масове видалення; бот додає «🗑 Видалено N» і сам прибирає це повідомлення через 5 секунд.

– kick, ban, unban, mute, unmute – функції перевіряють права через декоратор @has_permissions, а результат підверджують Embed-ом з аватаром порушника і автором дії.

– Для бану та кіку в Embed прописується reason, аби зрозуміти контекст у SicAudit.

7. Реакції на ключові слова («Реакції» Cog):

– Головний фільтр видаляє повідомлення з «забороненим» словом і робить попередження автору; окрема гілка вітає користувача при «hello/hi/привіт».

– Виклики bot.process_commands(message) лишаються в кінці, аби фільтр не блокував інші команди.

Utility Bot – функціональні модулі й особливості реалізації:

1. Нагадувач («Нагадувач» Cog):

– Slash-команда /remind приймає природній вираз часу (15m, 2h30m, tomorrow 09:00) або RFC RRULE. Вхід парсить dateutil.rrule та parsedatetime.

– Завдання зберігаються у таблиці reminders (id, user_id, channel_id, message, next_fire, rrule). Фоновий asyncio.Task раз на 30 с опитує найближчі нагадування й відправляє їх, додаючи автоматичний «🔔».

– Після рестарту бот читає таблицю й відтворює задачі; при несправності доставки робить до 5 спроб, потім пише адміну.

2. Опитування («Опитування» Cog):

– /poll "Запитання" вар1 вар2 … --duration 30m створює Embed і динамічно додає discord.ui.Button для кожного варіанта.

– Користувач може голосувати один раз; повторне натискання замінює попередній вибір. Голоси кешуються у dict[poll_id][user_id], дублюються у БД poll_votes, тож при перезапуску бот відновлює стан.

– Після дедлайну кнопки дизейбляться, бот рахує голоси й редагує оригінальний Embed, показуючи переможця жирним.

3. Стендап («Стендап» Cog):

– /standup start – бот надсилає форму із трьома кнопками («Вчора», «Сьогодні», «Блокери»). Натискання відкриває модальне вікно Discord 2025 (discord.ui.Modal).

– Відповіді зберігаються у таблиці standups; після дедлайну бот формує зведення Markdown-списком і авто-пінить повідомлення.

– При повторному запуску стендап-потік з тим же тегом видаляє старий пін, щоб не накопичувати спаму.

41. Погода («Погода» Cog):

– /weather [місто] робить GET до api.open-meteo.com, вибираючи geo-координати через Nominatim-кеш; дані кешуються в aiocache на 30 хв за ключем city_name.

– Бот форматує Embed із денною мін/макс-t°, іконкою, шансом опадів, виводить час оновлення. При помилці API повертає дружнє повідомлення «не можу дістати прогноз».

5. Переклад («Переклад» Cog):

– /translate ua|en <текст> або reply – визначає мову джерела й цілі параметром, робить запит до OpenAI /v1/chat/completions (модель gpt-4o-mini), додає system-prompt «You are a concise translator».

– Якщо ліміт вичерпано, бот переключається на безкоштовний DeepL-free; у відповідь додає примітку «(via DeepL)».

6. Самовидача ролей («СамоРолі» Cog):

– /roles create створює Embed з кнопками-ролями (discord.ui.Button(style=2)). ID кнопки містить role_id, тож обробник натискання читає interaction.data.custom_id, додає/знімає роль і відповідає ephemeral-повідомленням «Роль додано/знято».

– Кожен клік логують у канал #utility-log із указанням користувача й ролі; це допомагає простежити масові підписки перед великими івентами.

7. Метрики («Метрики» Cog):

– Використовує prometheus_client і aiohttp-web; ендпоінт /metrics віддає latency web-socketa, кількість активних нагадувань, хітів кешу погоди.

– Якщо порт 8000 зайнятий, бот піднімає метрики на наступному вільному й лог-файлі записує URL.

8. Сервіс-лог («Сервіс-Лог» Cog):

– Перехоплює всі on_error та on_command_error, пише traceback у файл utility_bot.log (ротація 10 × 1 МБ) і дублює стиснене повідомлення в #service-status.

– При старті та зупинці бота публікує Embed «Bot online/offline» із версією git-коміту й тривалістю аптайму.

3.2 Розробка структурної схеми

На рисунку 3.1 зображено структурну схему роботи клієнт-серверної системи, у центрі якої розміщено чат-бот на платформі Discord.

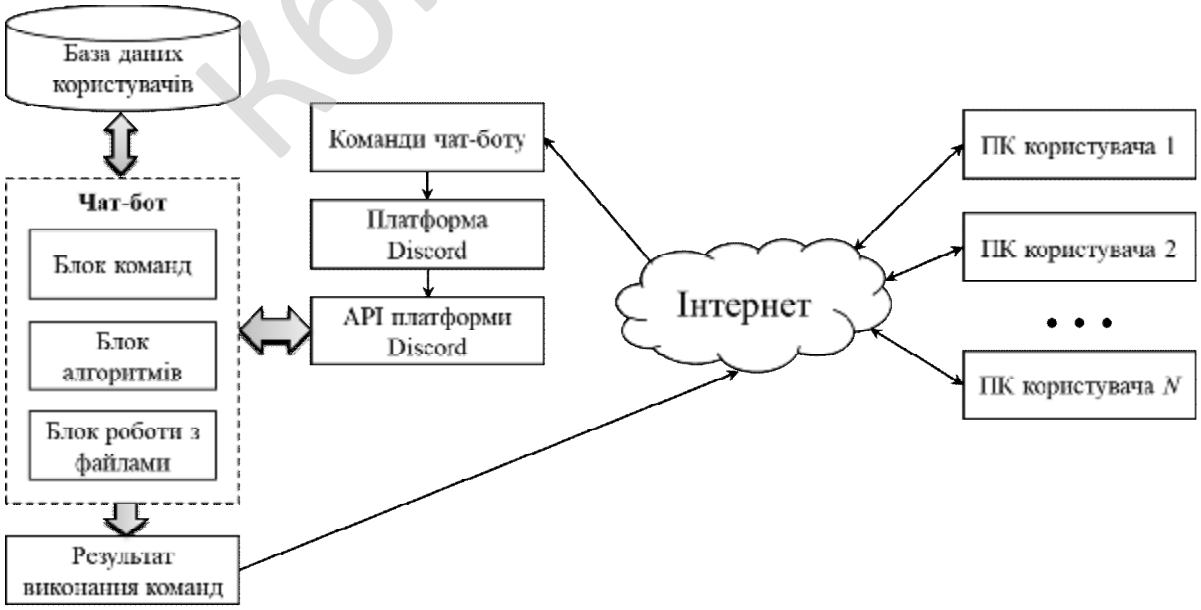

Рисунок 3.1 – Структурна схема системи

ПК користувачів 1…N – це клієнтські пристрої, з яких надходять запити (команди) до бота. Кожен клієнт використовує офіційний застосунок або веб-клієнт Discord для встановлення з’єднання через Інтернет.

Інтернет виступає транспортним середовищем, що забезпечує двосторонню комунікацію між користувацькими ПК та хмарною інфраструктурою Discord.

Платформа Discord приймає команди від користувачів і за допомогою API платформи Discord (Gateway + REST) пересилає їх безпосередньо до чат-бота. API також відправляє боту системні події (вхід/вихід учасників, реакції, зміни статусів тощо).

Чат-бот реалізовано як окремий серверний застосунок. Його внутрішня архітектура складається з трьох логічних блоків:

– Блок команд розпізнає вхідні інструкції, парсить їхні параметри та маршрутизує до відповідних обробників.

– Блок алгоритмів виконує основну прикладну логіку: обробку нагадувань, генерацію опитувань, модераційні дії, музичний стрім тощо.

– Блок роботи з файлами відповідає за створення і збереження зображень, аудіопотоків та інших ресурсів, необхідних для відповіді бота.

Для зберігання персистентних даних (нагадування, голоси опитувань, статистика) бот звертається до бази даних користувачів. Обмін із БД відбувається асинхронно, щоб не блокувати основний цикл подій.

Після опрацювання запиту формується результат виконання команд – це може бути текстове повідомлення, візуальна картка, реакція-емодзі, музичний потік чи зміна ролей. Бот передає цей результат назад у Discord через API, а платформа доставляє його користувацьким клієнтам тим самим каналом зв’язку.

Таким чином схема демонструє повний ланцюг: від ініціації запиту на ПК користувача, через мережеву інфраструктуру Discord і обробку всередині чат-бота, до повернення відповіді, що стає доступною всім учасникам сервера.

3.3 Розробка функціональної схеми

На рисунку 3.2 наведена функціональна схема системи.

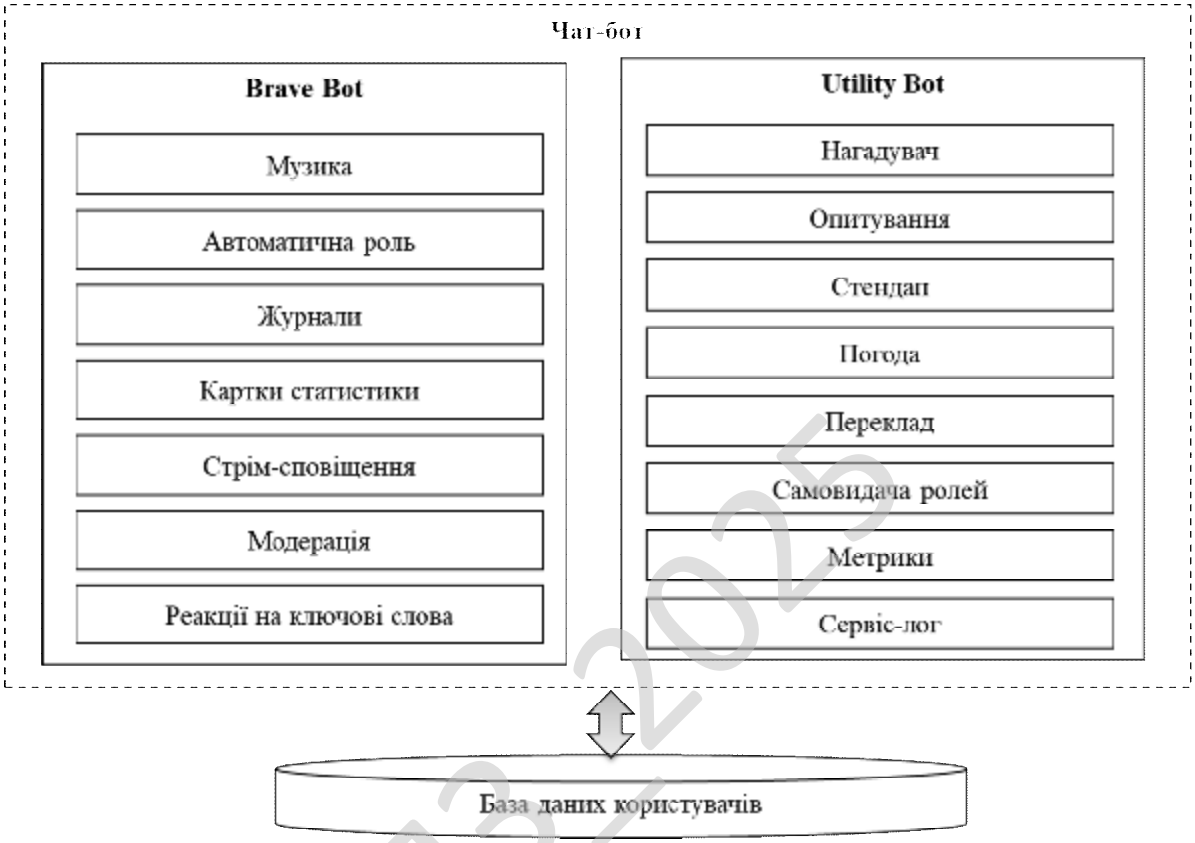

Рисунок 3.2 – Функціональна схема системи

Розроблювана система складається з наступних модулів:

1. Чат-бот Brave Bot – концентрується на мультимедіа, модерації та «атмосфері» сервера.

2. Чат-бот Utility Bot – дає інструменти для самоорганізації, інтеграції й аналітики.

3. База даних користувачів.

Чат-бот Brave Bot реалізує наступні функції:

– Музика – відтворює аудіо / відео-потоки через yt-dlp + FFmpeg; команди play і stop.

– АвтоРоль – автоматично видає роль «Новачок» усім, хто щойно приєднався.

– Журнали – фіксує видалення та редагування повідомлень, зміни ролей, створення / видалення каналів у лог-каналі.

– Картки статистики – генерує PNG-картку користувача або сервера з аватаром і ключовими даними.

– Стрім-сповіщення – при старті трансляції дає роль LIVE й публікує Embed-анонс; по завершенні знімає роль і показує тривалість.

– Модерація – швидкі команди clear, kick, ban, mute тощо з підтверджувальними Embed-ами.

– Фільтр і привітання – видаляє повідомлення з «забороненими» словами та реагує дружнім «Привіт!» на hello/hi/привіт.

Чат-бот Utility Bot реалізує наступні функції:

– Нагадувач – /remind створює одноразові або повторювані (RRULE) нагадування в DM чи канал, збережені у БД.

– Опитування – /poll формує кнопкове голосування з таймером і автоматичним підбиттям підсумків.

– Стендап – збирає щоденні звіти (вчора / сьогодні / блокери) й закріплює зведений пост.

– Погода – /weather показує 24-годинний прогноз Open-Meteo з кешем на 30 хв.

– Переклад – /translate ua|en миттєво перекладає повідомлення через OpenAI або DeepL-free.

– СамоРолі – повідомлення з кнопками для самостійної підписки / відписки від ролей (наприклад, «Анонси»).

– Метрики – експортує latency бота, кількість активних нагадувань та інші показники у формат Prometheus.

– Сервіс-лог – централізує винятки, старти / зупинки й важливі події у #service-status та в ротаційний лог-файл.

3.4 Розробка діаграми процесів

Діаграма процесів (рис. 3.3) ілюструє життєвий цикл двох взаємопов'язаних ботів – Brave Bot і Utility Bot – а також логічні переходи між їхніми ключовими підсистемами.

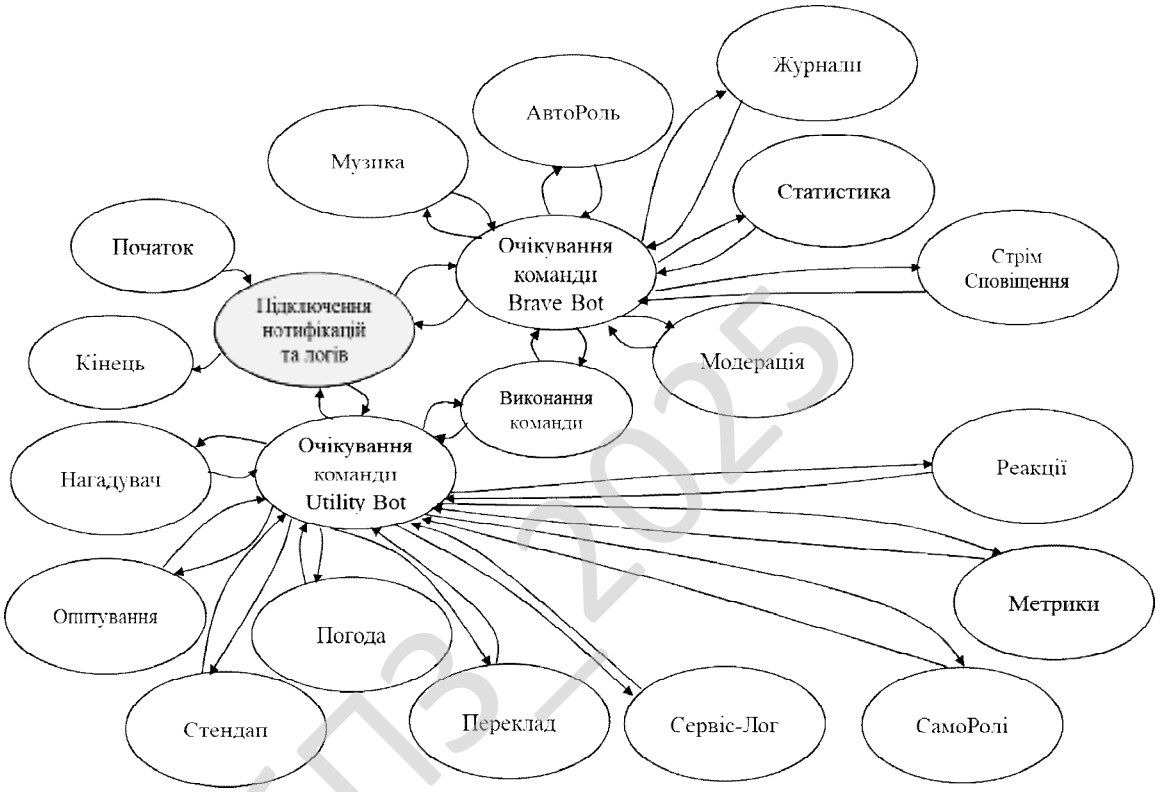

Рисунок 3.3 – Діаграма процесів системи

Ініціалізаційний етап

Початок → Підключення нотифікацій та логів

Після запуску застосунок встановлює з'єднання з Discord-Gateway, ініціює канали службових сповіщень і відкриває файлові/стрім-логи. Далі керування передається у стан очікування команд для кожного бота.

Контур Brave Bot

Очікування команди Brave Bot – основний цикл обробки подій Gateway (слухає префіксні #-команди).

Звідси можливі переходи до функціональних модулів:

– Музика – запуск/зупинка потоку, керування чергою.

– АвтоРоль – авто-видання ролей новачкам.

– Журнали – фіксація дій у лог-каналі.

– Статистика – генерація карток користувача/сервера.

– Стрім Сповіщення – видача ролі LIVE та анонси.

– Модерація – kick/ban/mute тощо.

– Реакції – авто-відповіді або фільтрація слів.

Після виконання підмодуля керування повертається у Очікування команди, формуючи замкнене коло.

Контур Utility Bot

Очікування команди Utility Bot – асинхронний цикл Slash-команд та фонових задач (нагадування, стендапи).

Доступні переходи до підсистем:

– Нагадувач – створення/тригери одноразових і повторюваних нагадувань.

– Опитування – генерація кнопкових голосувань з таймером.

– Стендап – збір та публікація щоденних звітів.

– Погода – запит до Open-Meteo й кешування прогнозу.

– Переклад – виклики OpenAI / DeepL для миттєвого перекладу.

– СамоРолі – інтерактивна самовидача ролей через кнопки.

– Метрики – експорт Prometheus-показників.

– Сервіс-Лог – централізовані повідомлення про помилки та перезапуски.

– Реакції – (спільний з Brave) швидкі відповіді на тригер-слова.

Кожен модуль завершивши роботу повертає систему знову у Очікування команди.

Завершення роботи

Перехід Очікування → Кінець активується лише при явному вимкненні сервісу або критичній помилці; бот закриває WebSocket-сесію та БД-з’єднання.

Взаємозв’язок ботів

Попри окремі цикли, обидва боти використовують спільний канал «Підключення нотифікацій та логів» – це уніфікована точка фіксації стану, що дає адміністраторам цілісну картину й спрощує моніторинг. Логічні стрілки між блоками вказують на можливі переходи подій без очікування зовнішнього втручання; циклічні стрілки символізують повернення до режиму слухання нових команд після завершення обробки конкретної задачі.

4 РЕАЛІЗАЦІЯ РОБОТИ. РОЗРАХУНКИ І ЕКСПЕРИМЕНТАЛЬНІ ДАНІ, ЩО ПІДТВЕРДЖУЮТЬ ВІРНІСТЬ ПРОЕКТНИХ ТА ПРОГРАМНИХ РІШЕНЬ

4.1 Блок-схеми та опис алгоритмів функціонування системи

На рис. 4.1 наведена блок-схема основної програми.

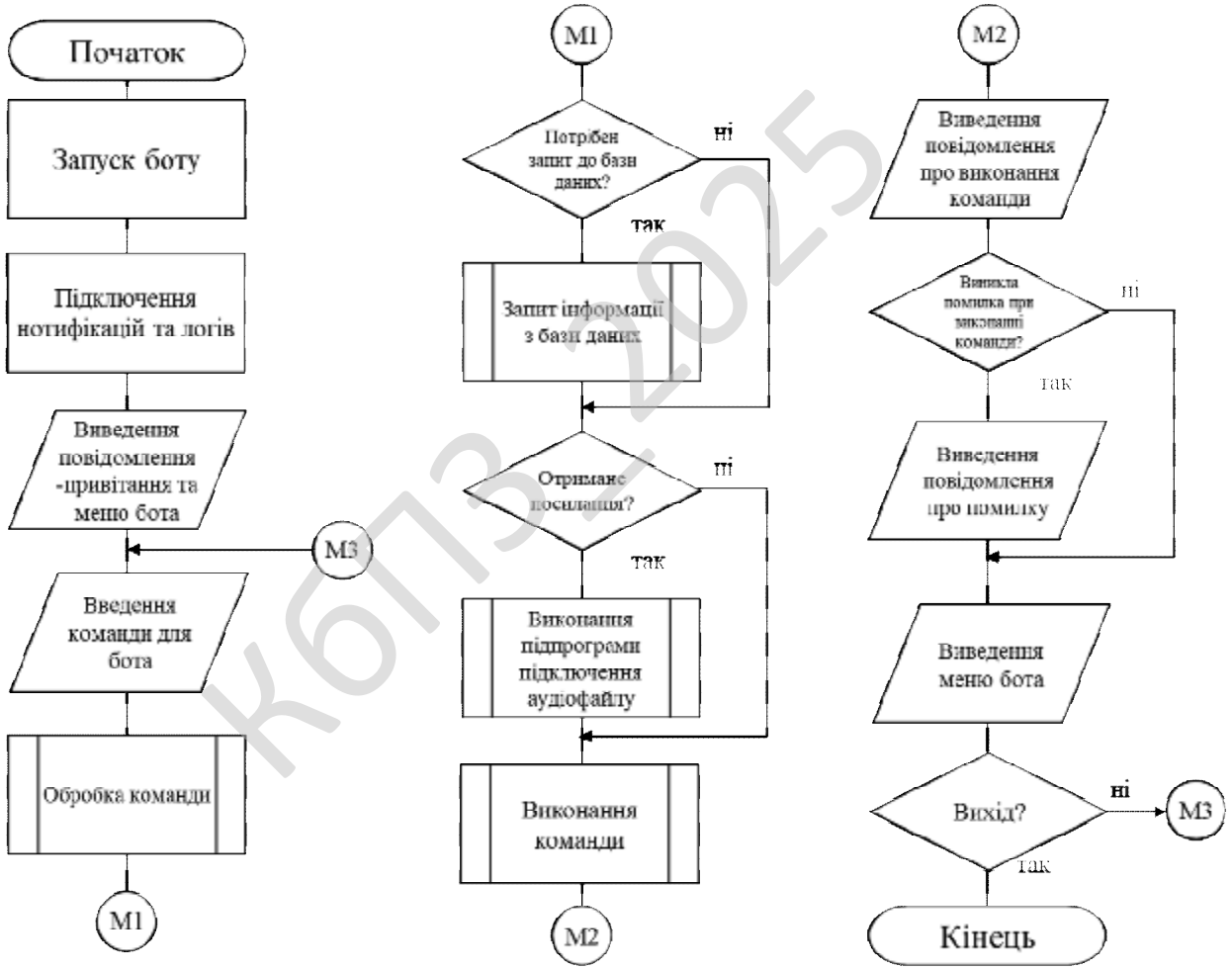

Рисунок 4.1 – Блок-схема роботи основної програми

На даній блок-схемі проілюстрована логіка роботи основної програми розробленого чат-бота.

Розглянемо детальніше кроки алгоритму.

Крок 1. Початок. Запускається процес, ініціюється інтерпретатор Python та середовище виконання.

Крок 2. Запуск боту. Читаються конфігураційні файли, токен Discord, піднімається асинхронний цикл подій.

Крок 3. Підключення нотифікацій та логів. Бот установлює WebSocket-з'єднання з Discord Gateway, відкриває файловий лог і «службовий» канал для внутрішніх повідомлень.

Крок 4. Виведення привітання та меню бота. Після успішного on_ready бот надсилає вказаний Embed-повідомлення з коротким довідником доступних команд. Звідси можливий повторний перехід до введення команд, коли користувач вирішить «повернутися до меню».

Крок 5. Введення команди для бота. Користувач надсилає у чат текстову або slash-команду.

Крок 6. Обробка команди. Блок перевіряє, який саме модуль має обробити запит і передає керування на відповідний підпроцес.

Крок 7. Потрібен запит до бази даних? Якщо команда використовує персистентні дані (нагадування, опитування, статистику) – так; інакше одразу переходить нижче.

Крок 8. Запит інформації з бази даних. Асинхронно читаються потрібні записи; при помилці повертається виняток.

Крок 9. Отримане посилання? Перевірка, чи команда вимагає сторонній ресурс (наприклад, URL аудіо). Якщо ні – переходить до «Виконання команди». Якщо так – викликається підпрограма підключення аудіофайлу (ініціалізація FFmpeg-потоку).

Крок 10. Виконання команди. Безпосереднє застосування логіки: відтворення музики, видача ролі, створення нагадування тощо.

Крок 11. Формування відповіді й обробка помилок.

Крок 12. Виведення повідомлення про виконання. Бот публікує

підтвердження (Embed, реакція, текст).

Крок 13. Виникла помилка при виконанні? Якщо так – генерується користувацьке повідомлення про помилку й лог записується у service-log. Якщо ні – переходить далі.

Крок 14. Виведення меню бота. Для зручності користувачу знову показується скорочений список доступних дій.

Крок 15. Вихід? Перевіряється, чи це була команда «вийти» / «зупинити». Якщо ні – повернення на мітку М3 (очікування нової команди). Якщо так – переходимо до завершальної стадії.

Крок 16. Кінець. Закриваються WebSocket та HTTP-сесії, скидаються таймери, коректно завершується робота програми.

На рисунку 4.2 показана блок-схема підпрограми чат-боту «Музика».

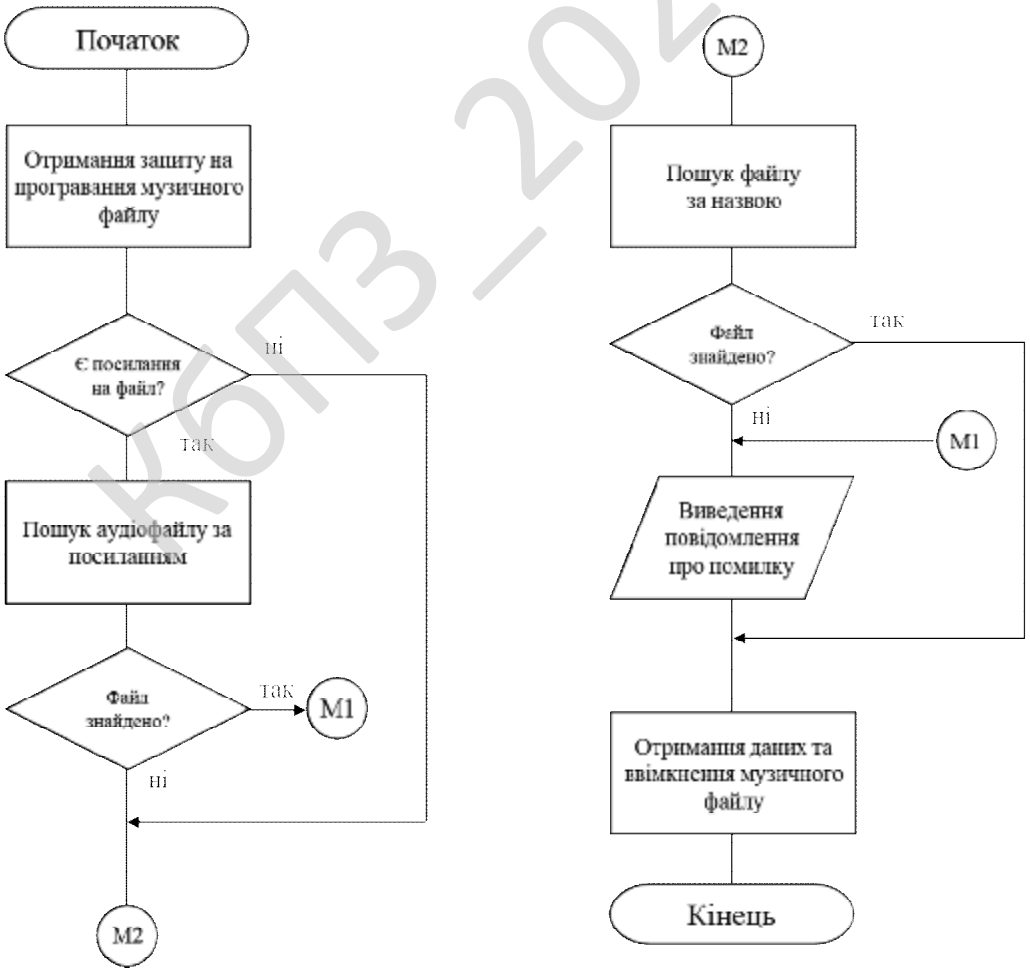

Рисунок 4.2 – Блок-схема підпрограми чат-боту «Музика»

Блок-схема 4.2 описує логіку підпрограми «Музика», яка обробляє запит користувача на відтворення треку.

Крок 1. Отримання запиту на програвання музичного файлу. Бот отримує команду play … від користувача.

Крок 2. Перевірка: чи передано посилання? Так – бот сприймає аргумент як URL (YouTube, SoundCloud, прямий mp3). Ні – переходить до підпроцесу M2 (пошук за назвою).

Крок 3. Пошук аудіофайлу за посиланням. Викликається yt-dlp: завантажується лише JSON-метадані й дістається прямий аудіострім.

Крок 4. Файл знайдено? Так – переходимо до мітки (передаємо дані у плеєр FFmpeg). Ні – переходить на мітку M2, що повертає керування в головну програму з повідомленням: «Файл не знайдено».

Крок 5. Пошук файлу за назвою. yt-dlp запускається з режимом ytsearch:<текст> і повертає перший знайдений результат.

Крок 6. Файл знайдено? Так – переходить до мітки M1 (стандартне відтворення). Ні – виводиться повідомлення про помилку, після чого повернення у головний цикл (указано стрілкою мітки до мM1, яка фактично веде назад до очікування нової команди).

Крок 7. Отримання даних та ввімкнення музичного файлу. Для знайденого треку формується FFmpeg-потік; бот надсилає реакцію «▶» або Embed-підтвердження й починає відтворення у голосовому каналі.

Крок 8. Кінець. Підпрограма завершується, керування повертається основному циклу бота, який тепер підтримує трансляцію й слухає наступні команди (stop, skip, додавання в чергу тощо).

Таким чином схема демонструє дві рівноцінні стратегії пошуку аудіо – за прямим URL або за текстовим запитом – із перевіркою на успіх і обробкою помилки, перш ніж передати потокові дані в модуль відтворення.

Розглянемо деякі реалізовані підпрограми.

Скрипт відтворення аудіо та музики:

```
def user_voice(ctx: commands.Context) -> Optional[discord.VoiceState]:
    """Повертає voice-стан користувача або None, якщо він не у voice."""
    return ctx.author.voice

@bot.command(name="play", brief="Відтворити музику за URL або пошуком")
async def cmd_play(ctx: commands.Context, *, query: str) -> None:
    """Під'єднання до voice-каналу та відтворення аудіо."""
    global voice_connected

    voice_state = user_voice(ctx)
    if voice_state is None:
        await ctx.reply("Спочатку приєднайтеся до голосового каналу.")
        return

    if not voice_connected:
        vc = await voice_state.channel.connect()
        voice_connected = True
    else:
        vc = ctx.voice_client  # type: ignore
```

Скрипт завантаження аудіо-потік через yt-dlp:

```
  with yt_dlp.YoutubeDL(YDL_OPTS) as ydl:
        try:
            info = (
                ydl.extract_info(query, download=False)
                if query.startswith("http")
                else ydl.extract_info(f"ytsearch:{query}",
download=False)["entries"][0]
            )
        except Exception as exc:
            await ctx.reply(f"Помилка yt-dlp: {exc}")
            return

    stream_url = info["url"]
    vc.play(
        discord.FFmpegPCMAudio(
            executable="ffmpeg",  # сучасний Discord дозволяє без шляху, якщо
ffmpeg у PATH
            source=stream_url,
            **FFMPEG_OPTS,
        )
    )
    await ctx.message.add_reaction("▶")

@bot.command(name="stop", brief="Зупинити музику та вийти з voice")
async def cmd_stop(ctx: commands.Context) -> None:
    """Вихід із voice-каналу та скидання прапорця."""
    global voice_connected
    if not voice_connected:
        await ctx.reply("Я поки не в голосовому каналі.")
        return
    await ctx.voice_client.disconnect()  # type: ignore
    voice_connected = False
```

```
    await ctx.message.add_reaction("▣")
```

Скрипт статистичних даних:

```
def _draw_card_base(avatar_bytes: bytes) -> Image.Image:
    """Створює базове зображення й вставляє аватар."""
    card = Image.new("RGBA", STAT_CARD_SIZE, THEME_BG)
    avatar =
Image.open(io.BytesIO(avatar_bytes)).convert("RGBA").resize(AVATAR_SIZE,
Image.LANCZOS)
    card.paste(avatar, (15, 15, 115, 115))
    return card

def _font(size: int) -> ImageFont.FreeTypeFont:
    return ImageFont.truetype(DEFAULT_FONT, size=size)

@bot.command(aliases=["я", "stats", "me"], brief="Ваша особиста статистика")
async def card_user(ctx: commands.Context, member: Optional[discord.Member] =
None) -> None:
    """Генерує та надсилає картку користувача."""
    member = member or ctx.author
    response = requests.get(str(member.display_avatar.replace(size=128)))
    card = _draw_card_base(response.content)
    draw = ImageDraw.Draw(card)

    draw.text((145, 15), f"{member} ", font=_font(20), fill="white")
    draw.text((145, 50), f"Найвища роль: {member.top_role.name}",
font=_font(12), fill="white")
    draw.text((145, 80), f"Приєднався: {member.joined_at.strftime(DATE_FMT)}",
font=_font(12), fill="white")
    draw.text((145, 110), f"Зареєструвався:
{member.created_at.strftime(DATE_FMT)}", font=_font(12), fill="white")
    draw.text((145, 140), f"Статус: {member.status}", font=_font(12),
fill="white")

    path = Path("user_card.png")
    card.save(path)
    await ctx.send(file=discord.File(path))

@bot.command(aliases=["server", "сервер"], brief="Інформація про сервер")
async def card_server(ctx: commands.Context) -> None:
    """Генерує картку сервера."""
    guild = ctx.guild
    if guild is None:
        return
    response = requests.get(str(guild.icon.replace(size=128)))
    card = _draw_card_base(response.content)
    draw = ImageDraw.Draw(card)

    draw.text((145, 15), guild.name, font=_font(20), fill="white")
    draw.text((145, 50), f"Власник: {guild.owner}", font=_font(12),
fill="white")
    draw.text((145, 80), f"Учасників: {guild.member_count}", font=_font(12),
fill="white")
    draw.text((145, 110), f"Каналів: {len(guild.channels)}", font=_font(12),
fill="white")
    draw.text((145, 140), f"Текстових: {len(guild.text_channels)}",
font=_font(12), fill="white")
```

```
    draw.text((145, 170), f"Голосових: {len(guild.voice_channels)}",
font=_font(12), fill="white")

    path = Path("server_card.png")
    card.save(path)
    await ctx.send(file=discord.File(path))
```

Скрипт Стрим-сповіщень:

```
@bot.event
async def on_presence_update(before: discord.Member, after: discord.Member) ->
None:
    """Видає/знімає роль STREAM_ROLE_ID та сповіщає про початок/кінець
стріму."""
    if after.guild and after.guild.id != GUILD_ID:
        return

    guild = after.guild
    stream_role = guild.get_role(STREAM_ROLE_ID) if guild else None
    channel = bot.get_channel(STREAM_NOTIFY_CH)

    async def _post_start():
        if stream_role:
            await after.add_roles(stream_role, reason="Почався стрім")
        emb = discord.Embed(
            title=f"🔴 {after.display_name} зараз у прямому ефірі!",
            url=after.activity.url,
            colour=discord.Colour.brand_red(),
            timestamp=dt.datetime.now(),
        )
        emb.add_field(name="Назва", value=after.activity.name)
        if after.activity.game:
            emb.add_field(name="Категорія", value=after.activity.game,
inline=False)
        await channel.send(embed=emb)

    async def _post_end(prev_activity: Streaming):
        if stream_role:
            await after.remove_roles(stream_role, reason="Стрім завершено")
        emb = discord.Embed(
            title=f"🏁 Стрім {prev_activity.twitch_name} завершено",
            colour=discord.Colour.orange(),
            timestamp=dt.datetime.now(),
        )
        emb.add_field(name="Тема", value=prev_activity.name)
        await channel.send(embed=emb)
```

Скрипт Модерації:

```
def is_admin():
    return commands.has_permissions(administrator=True)

@bot.command(brief="Очистити повідомлення")
@is_admin()
async def clear(ctx: commands.Context, amount: int = 50) -> None:
```

```
    deleted = await ctx.channel.purge(limit=amount + 1)
    await ctx.send(f"🧺 Видалено {len(deleted)-1} повідомлень.",
delete_after=5)

@bot.command(brief="Вигнати користувача")
@is_admin()
async def kick(ctx: commands.Context, member: discord.Member, *, reason: str =
"Без пояснення") -> None:
    await member.kick(reason=reason)
    await ctx.send(f"🥾 {member.mention} вигнано. Причина: {reason}")

@bot.command(brief="Забанити користувача")
@is_admin()
async def ban(ctx: commands.Context, member: discord.Member, *, reason: str =
"Без пояснення") -> None:
    await member.ban(reason=reason)
    await ctx.send(f"⛔ {member.mention} забанено. Причина: {reason}")

@bot.command(brief="Розбанити користувача")
@is_admin()
async def unban(ctx: commands.Context, user: discord.User) -> None:
    await ctx.guild.unban(user)
    await ctx.send(f"✅ {user.mention} розбанено.")
```

Скрипт команд нагадувачів:

```
@bot.command(name="remind", brief="Нагадування через час")
async def remind(ctx: commands.Context, timespec: str, *, text: str) -> None:
    """!remind 10m Піти за кавою"""
    delay = parse_timespec(timespec)
    if delay is None:
        await ctx.reply("⏰ Формат часу: 10m | 2h30m | 1d…")
        return

    await ctx.reply(f"⏳ Добре, нагадаю за {timespec}…")

    async def _later():
        await asyncio.sleep(delay)
        await ctx.author.send(f"🔔 Нагадування: {text}")

    bot.loop.create_task(_later())
```

Скрипт опитувань/голосувань:

```
POLL_EMOJIS = ("1️⃣", "2️⃣", "3️⃣", "4️⃣", "5️⃣", "6️⃣", "7️⃣", "8️⃣", "9️⃣", "🔟")

@bot.command(name="poll", brief="Створити опитування")
async def poll(ctx: commands.Context, question: str, *options: str) -> None:
    """!poll "Що робимо?" Граємо Читаємо Спимо"""
    if len(options) < 2:
        await ctx.reply("Потрібно щонайменше 2 варіанти.")
        return
    if len(options) > len(POLL_EMOJIS):
        await ctx.reply("Максимум 10 варіантів.")
```

```
        return

    desc = "\n".join(f"{POLL_EMOJIS[i]}  {opt}" for i, opt in
enumerate(options))
    embed = discord.Embed(title=question, description=desc,
colour=discord.Colour.gold())
    message = await ctx.send(embed=embed)
    for i in range(len(options)):
        await message.add_reaction(POLL_EMOJIS[i])
```

Скрипт перегляду погоди:

```
async def fetch_weather(lat: float, lon: float) -> str:
    url = (
        "https://api.open-
meteo.com/v1/forecast?latitude={lat}&longitude={lon}&hourly=temperature_2m"
        .format(lat=lat, lon=lon)
    )
    async with aiohttp.ClientSession() as session:
        async with session.get(url) as resp:
            data = await resp.json()
    temp = data["hourly"]["temperature_2m"][0]
    return f"🌡 Температура зараз: {temp}°C"

@bot.command(name="weather", brief="Погода по координатах")
async def weather(ctx: commands.Context, lat: float, lon: float):
    msg = await fetch_weather(lat, lon)
    await ctx.send(msg)
```

Скрипт Reaction-roles:

```
REACTION_ROLE_MESSAGE_ID: Optional[int] = None  # ← встановіть вручну або
створіть через команду
ROLE_BY_EMOJI: dict[str, int] = {
    "🔴": 123456789012345678,  # id ролей
    "🔵": 234567890123456789,
}

@bot.command(name="rrsetup", hidden=True)
@commands.has_permissions(manage_roles=True)
async def rrsetup(ctx: commands.Context):
    """Показує повідомлення для реакцій-ролей і запам'ятовує його ID."""
    global REACTION_ROLE_MESSAGE_ID
    lines = [f"{emoji}  <@&{role_id}>" for emoji, role_id in
ROLE_BY_EMOJI.items()]
    msg = await ctx.send("**Оберіть роль:**\n" + "\n".join(lines))
    REACTION_ROLE_MESSAGE_ID = msg.id
    for emoji in ROLE_BY_EMOJI:
        await msg.add_reaction(emoji)


@bot.event
async def on_raw_reaction_add(payload: discord.RawReactionActionEvent):
    if payload.message_id != REACTION_ROLE_MESSAGE_ID or payload.guild_id is
None:
        return
```

```
        role_id = ROLE_BY_EMOJI.get(str(payload.emoji))
        if role_id is None:
            return
        guild = bot.get_guild(payload.guild_id)
        if guild is None:
            return
        role = guild.get_role(role_id)
        member = guild.get_member(payload.user_id)
        if role and member:
            await member.add_roles(role)

@bot.event
async def on_raw_reaction_remove(payload: discord.RawReactionActionEvent):
    if payload.message_id != REACTION_ROLE_MESSAGE_ID or payload.guild_id is
None:
        return
    role_id = ROLE_BY_EMOJI.get(str(payload.emoji))
    if role_id is None:
        return
    guild = bot.get_guild(payload.guild_id)
    if guild is None:
        return
    role = guild.get_role(role_id)
    member = guild.get_member(payload.user_id)
    if role and member:
        await member.remove_roles(role)
```

4.2 Захист розробленого програмного забезпечення

Для захисту розробленого програмного забезпечення було використано ліцензію Creative Commons. Її було обрано з наступних причин:

– Відкрите поширення з контрольованими обмеженнями. Ліцензії CC надають автору гнучкий набір умов (атрибуція, некомерційність, share-alike тощо), дозволяючи відкрито поширювати продукт і водночас залишати за собою ключові права.

– Міжнародна юрисдикція. CC-ліцензії діють у 150+ країнах, включаючи Україну, що спрощує використання й правозастосування у глобальних спільнотах Discord.

– Розпізнаваність у навчальному й некомерційному середовищі. Більшість академічних та волонтерських проєктів вже орієнтуються на CC, тому обирати таку модель для бот-проекту, розробленого у рамках дипломної роботи, природно.

Було обрано наступну версію цієї ліцензії – **CC BY-NC-SA 4.0**:

– **BY (Attribution).** Кожен, хто використовує код або матеріали, має згадати авторство. У README.md або команді !about необхідно вказати ім’я розробника та посилання на репозиторій.

– **NC (Non-Commercial).** Заборонено пряме комерційне використання без окремої угоди. Хостинг бота на платній SaaS-платформі чи продаж кастомізованої версії вимагатиме ліцензійного дозволу автора.

– **SA (Share-Alike).** Похідні роботи повинні поширюватися на тих самих умовах. Якщо хтось форкає репозиторій та додає нові модулі, він зобов’язаний опублікувати їх теж під CC BY-NC-SA 4.0.

Таким чином, чат-бот залишається вільним для навчання та волонтерських ініціатив, але захищений від несанкціонованої комерціалізації.

Практичні кроки впровадження:

1. Додати файл LICENSE у репозиторій з повним текстом CC BY-NC-SA 4.0 (українською або англійською).

2. Вказати шапку-дисклеймер в основних файлах коду:

```
# © 2025, Графенюк В.О., ЦНТУ.
# Licensed under CC BY-NC-SA 4.0
```

3. Описати умови у документації (README, Wiki): мета ліцензії, порядок атрибуції, спосіб отримання комерційної ліцензії.

4. Вказати контакт для ліцензійних питань. E-mail автора для бажаючих отримати інші права (наприклад, CC BY або комерційний ресел).

Переваги такої ліцензії для проєкту:

- Прозорість для користувачів Discord, які одразу розуміють свої права й обов’язки.

- Спрощений обмін знаннями між університетськими командами, будь-хто може форкнути бот, адаптувати для власного курсу й повернути поліпшення upstream.

- Легітимний захист авторських прав з мінімальними витратами на юридичний супровід, достатньо згадки ліцензії та правильної атрибуції.

Застосування ліцензії Creative Commons BY-NC-SA 4.0 забезпечує

збалансований захист інтелектуальної власності, відкриває шлях до співпраці та сприяє сталому розвитку проєкту в академічному й волонтерському середовищах.

5 МЕТОДИКА ВПРОВАДЖЕННЯ СИСТЕМИ В ПРОМИСЛОВУ ЕКСПЛУАТАЦІЮ

Для впровадження в експлуатацію розробленого програмного забезпечення треба встановити наступні сторонні бібліотеки для мови програмування Python:

– discord – головний SDK для взаємодії з Discord API: робота з Gateway-подіями, Slash-командами, UI-компонентами.

– aiohttp – HTTP-клієнт для виклику зовнішніх API (Open-Meteo, OpenAI, DeepL) та завантаження зображень/аватарів.

– python-dateutil – розбір і генерація RFC-5545 RRULE, маніпуляції з датами та часовими зонами.

– prometheus_client – експорт технічних метрик (/metrics) для моніторингу в Grafana/Prometheus.

– logging + log-rotation – форматування логів, ротація файлів, відправлення службових сповіщень у Discord.

– yt-dlp – отримання прямого аудіо-URL з YouTube/Spotify/SoundCloud без завантаження файлу.

– FFmpeg (CLI-бінарі) – транскодування та стрім аудіо у voice-канал (викликається через discord.FFmpegPCMAudio).

– Pillow (PIL) – генерація PNG-карток користувача і сервера (ресайз аватара, малювання тексту).

– requests – синхронне завантаження аватарів/іконок (не критичні до блокування, бо окремий тред у Pillow).

– parsedatetime – «людяний» розбір виразів часу – “завтра о 9:00”, “15m”, “2h30m”.

– aiocache – in-memory/Redis кеш для прогнозу погоди й інших API-відповідей (TTL 30 хв).

– openai – виклик LLM-моделі для миттєвого перекладу повідомлень UA

⇄ EN.

– discord.ui / app_commands – побудова інтерактивних кнопок та модальних вікон (SelfRoles, Poll, Standup).

Для впровадження системи в промислову експлуатацію необхідно здійснити наступні дії.

Щоб запустити і підтримувати Brave Bot та Utility Bot на одному сервері, адміністратору знадобиться передусім сучасне оточення – встановлена Python 3.12 із pip, FFmpeg, а також Git для отримання вихідного коду. Для продакшн-розгортання бажано мати Docker з docker-compose, хоча локальний запуск теж можливий. Спершу треба клонувати репозиторій проекту з GitHub у робочу директорію. Далі у корені створити файл *.env*, де прописати токени двох ботів, ідентифікатори каналів і ролей Discord, шлях до бази нагадувань та, за потреби, ключі зовнішніх API.

Після цього слід перейти до встановлення залежностей. У разі «чистого» запуску без контейнерів слід створити віртуальне середовище, активувати його і виконати pip install за файлом *requirements.txt*. Якщо використовується Docker, усі необхідні пакети буде зібрано усередині контейнера. У режимі відладки запустити ботів можна окремими командами python brave_bot.py та python utility_bot.py і поспостерігати, як у консолі з’являються повідомлення «Connected», після чого слід перевірити роботу команд у тестовому сервері Discord.

Для продакшн-варіанта треба запустити docker-compose up у фоновому режимі; у системі з’являються два контейнери – brave_bot і utility_bot. Живі логи кожного можна переглядати командою docker logs, а в Discord-сервері вся службова інформація дублюється у відповідні канали – *brave-log* та *utility-log*. Якщо потрібно перезапустити або оновити ботів, досить стягнути свіжі зміни git pull, перебілдити образи і знову підняти docker-compose.

Адміністратор у щоденній роботі стежить за логами, стежить за Prometheus-метриками, що віддаються на порт, зазначений у *.env*, і при необхідності виправляє типові збої. Наприклад, якщо Brave Bot скаржиться на

відсутній ffmpeg, встановлюють пакунки системного менеджера; якщо SelfRoles не працюють, перевіряють, щоб роль бота була вище керованих ролей. Усі нагадування зберігаються у SQLite-файлі або у PostgreSQL, тому резервні копії роблять простим копіюванням файлу чи pg_dump.

Оновлення Python-залежностей виконують через pip install -U потрібних пакетів або повний перебілд Docker-образу. Коли виникає потреба у новому функціональному модулі, розробник додає окремий Cog-файл до відповідної директорії, реєструє його в setup-функції, перезапускає контейнер – і нова можливість одразу стає доступною користувачам сервера. Таким чином адміністратор отримує керовану систему з двох легких, незалежно оновлюваних ботів, які закривають і розважальні, і організаційні потреби спільноти без сторонніх SaaS-сервісів.

Для додавання чат-боду до Discord треба зайти у developers на сайті Discord, для створення аплікації зображено на рис. 5.1.

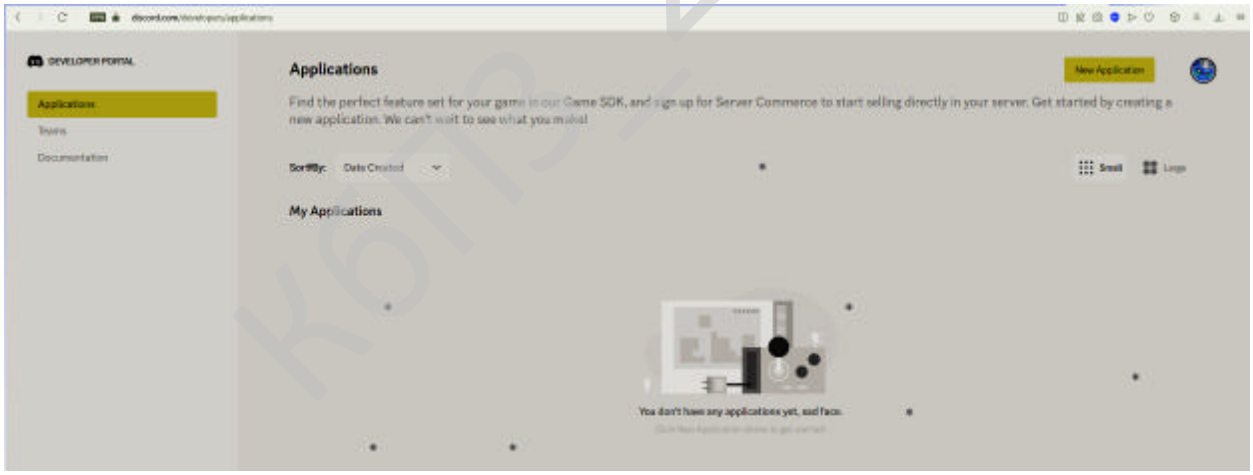

Рисунок 5.1 – Developers сайт Discord

Після цього потрібно створити аплікацію, як зображено на рис. 5.2.

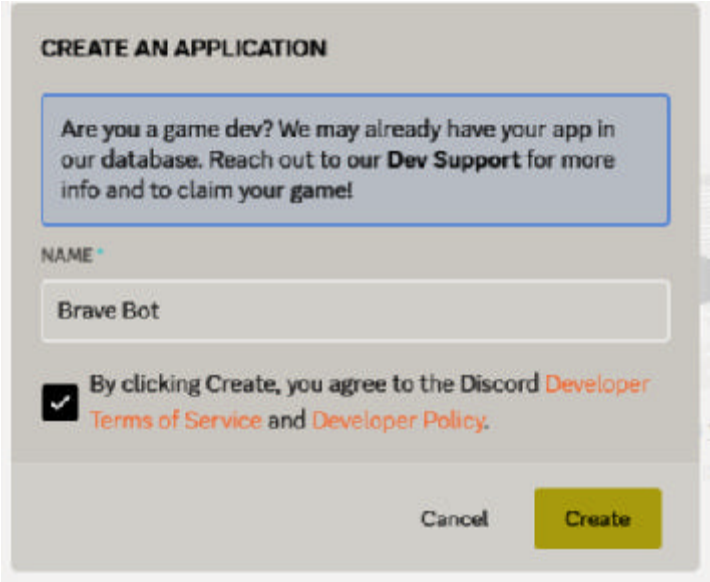

Рисунок 5.2 – Створення аплікації

Для того щоб можна було додати бота та писати/додати для нього код потрібно його створити у аплікації (рис. 5.3) та дати йому ті дозволи, які потрібно для обслуговування серверу, так як бот для адміністрування, то слід додати йому функцію адміністратора (рис. 5.4) та додати його на свій сервер для розробки.

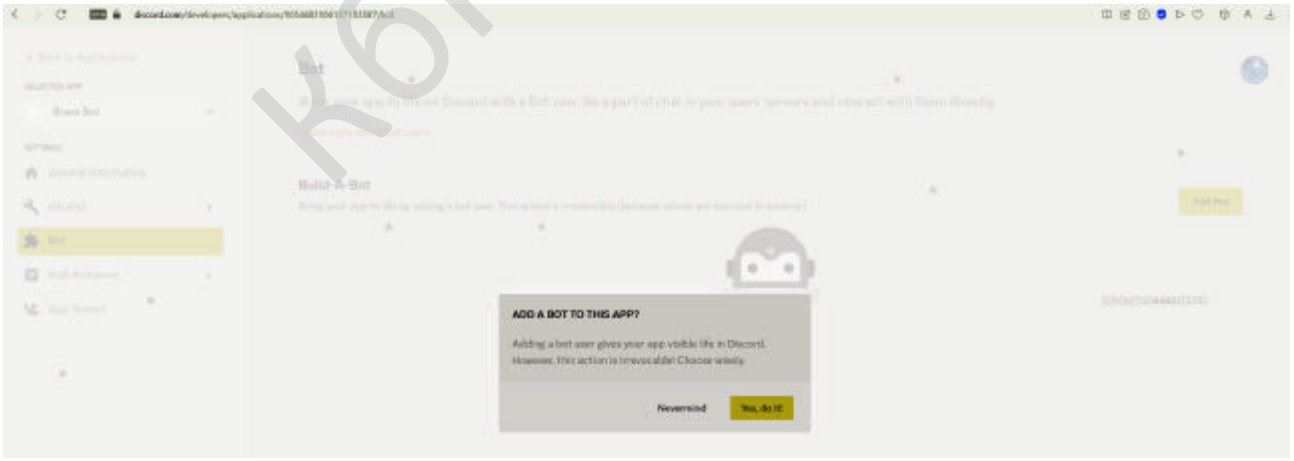

Рисунок 5.3 – Створення бота в аплікації

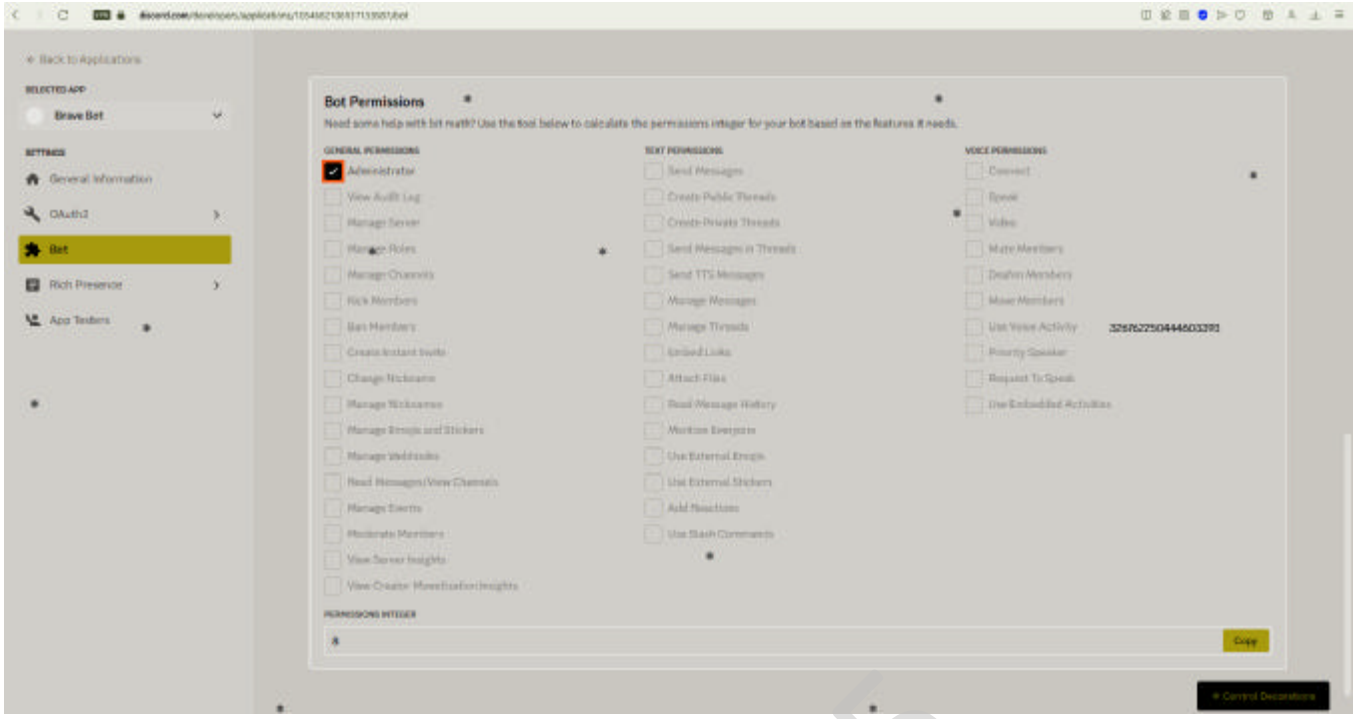

Рисунок 5.4 – Надання прав боту

Після цього потрібно додати бота на сервер за допомогою створення URL, як показано на рис. 5.5-5.6.

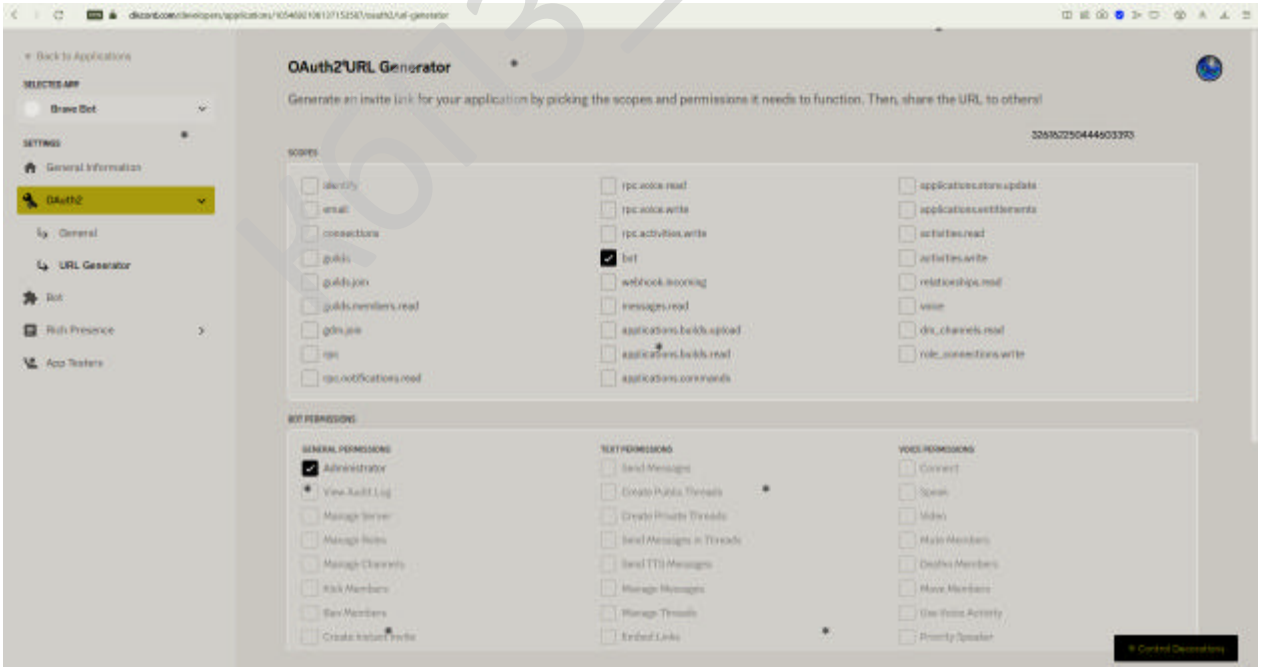

Рисунок 5.5 – Надання прав та створення посилання

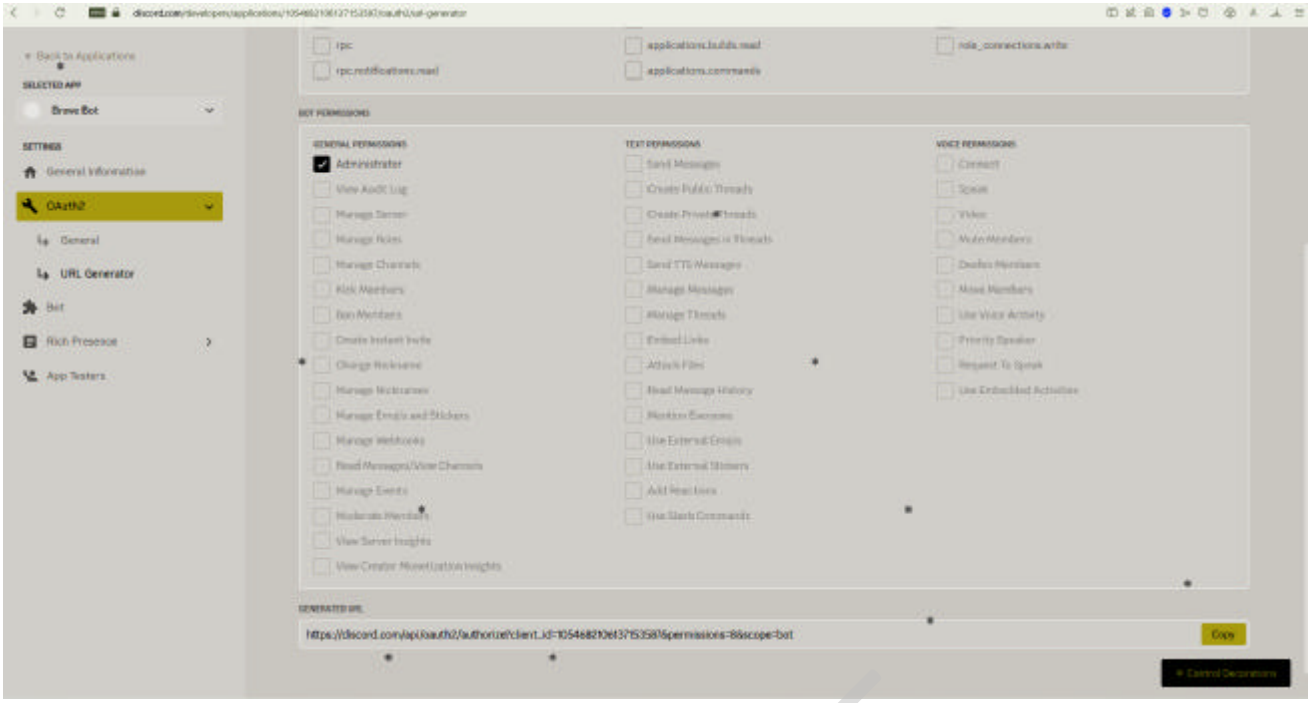

Рисунок 5.6 – Отримання URL посилання на додавання боту

Скопіювавши посилання вставляємо його в браузер та додаємо на сервер.

Для того щоб тепер працювати з ботом потрібно отримати його токен та додати його підключення.

Приклад роботи чат-боту у показано на рис. 5.7.

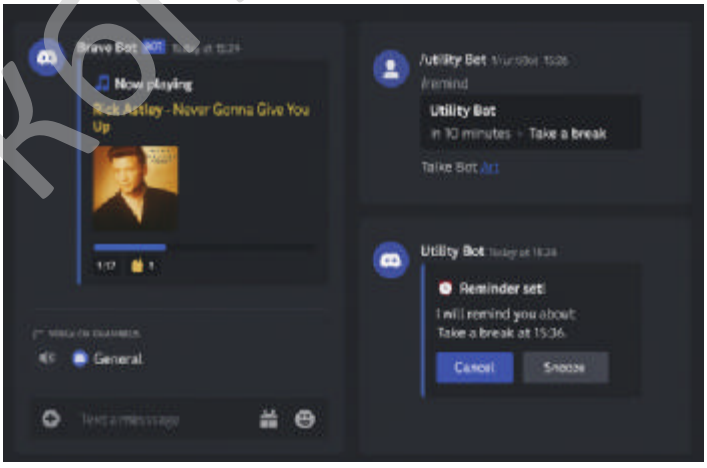

Рисунок 5.7 – Приклад роботи чат-боту

6 ОСНОВНІ ВИСНОВКИ

Програмне забезпечення, створене в результаті виконання кваліфікаційної роботи, призначено для реалізації чат-боту платформи обміну повідомленнями Discord.

Для вирішення поставленої мети було проведено:

– Огляд існуючих систем для реалізації чат-ботів для платформи обміну повідомленнями Discord.

– Проектування системи для реалізації чат-боту для платформи обміну повідомленнями Discord.

– Програмна реалізація системи для реалізації чат-боту для платформи обміну повідомленнями Discord.

Реалізовані під час виконання кваліфікаційної роботи алгоритми дозволяють успішно вирішувати завдання реалізації чат-ботів для платформи обміну повідомленнями Discord.

Розроблене програмне забезпечення представляє собою додаток, що можна використовувати практично на будь-якому сучасному комп’ютері.

Програмне забезпечення розроблялося у об’єктно-орієнтованій парадигмі, що робить його гнучким та зручним для підтримки та масштабування.

Для розробки програмного забезпечення чат-ботів для платформи обміну повідомленнями Discord використовувалася мова програмування Python. Дана мова програмування дозволила ефективно вирішити поставлену мету та задачі кваліфікаційної роботи.

У п’ятому розділі пояснювальної записки надаються усі необхідні рекомендації з встановлення програмного забезпечення та інструкція для його використання.

Для захисту розробленого програмного забезпечення було обрано вільну ліцензію Creative Commons.

Програмне забезпечення чат-боту для платформи обміну повідомленнями Discord є важливим інструментом при проектуванні комп’ютерних мереж. Враховуючи все більшу актуальність месенджерів й важливість їх ролі в організації командної роботи, використання такого роду програмного забезпечення стає не просто рекомендацією, а необхідністю. Тож, розроблене у цій кваліфікаційній роботі програмне забезпечення має важливе призначення.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. A Survey on Conversational Agents/Chatbots: Classification and Design Techniques / Nuruzzaman M., Khaled A. – 2024 // International Journal of Computer Applications. – Vol. 186, № 8. – С. 1–12.

2. Advanced Command Creation // discord.js Guide [Електронний ресурс]. – Режим доступу: https://discordjs.guide/slash-commands/advanced-creation (дата звернення: 16.05.2025).

3. Aho A.V., Hopcroft J.E., Ullman J.D. Data Structures and Algorithms. – Pearson, 2001. – 620 p.13. Кормен Т., Лейзерсон Ч., Ривест Р., Штайн К. Алгоритмы: построение и анализ, 3-е издание – М.: Диалектика, 2019. – 1328 p.

4. asyncio – Asynchronous I/O // Python Standard Library [Електронний ресурс]. – Режим доступу: https://docs.python.org/3/library/asyncio.html (дата звернення: 16.05.2025).

5. Beazley D., Jones B. Python Cookbook. – 3rd ed. – O'Reilly Media, 2013. – 706 p.

6. Bird S., Klein E., Loper E. Natural Language Processing with Python. – Sebastopol : O'Reilly Media, 2009. – 506 p.

7. Chatbot Market Size, Share & Trends Analysis Report By Offering (Solution, Services), By Type, By Medium, By Business Function, By Application, By Vertical By Region, And Segment Forecasts, 2025-2030– URL: https://www.grandviewresearch.com/industry-analysis/chatbot-market

8. Commands // discord.py Documentation [Електронний ресурс]. – Режим доступу: https://discordpy.readthedocs.io/en/stable/ext/commands/commands.html (дата звернення: 16.05.2025).

9. Creating Slash Commands // discord.js Guide [Електронний ресурс]. – Режим доступу: https://discordjs.guide/creating-your-bot/slash-commands (дата звернення: 16.05.2025).

10. Curry D. Discord Revenue and Usage Statistics (2025) – URL:

https://www.businessofapps.com/data/discord-statistics/?utm_source=chatgpt.com

11. Discord API Docs. Rate_Limits.md : GitHub-репозиторій [Електронний ресурс]. – Режим доступу: https://github.com/discord/discord-api-docs/blob/main/docs/topics/Rate_Limits.md (дата звернення: 16.05.2025).

12. Discord Bots Unleashed: Maximizing Community Engagement with Custom Bots [Електронний ресурс]. Instabooks, 2024. – Режим доступу: https://instabooks.ai/products/2baf5af0-2599-9d0c-003d-1b1a69df0dae (дата звернення: 16.05.2025).

13. Discord developer portal [Електронний ресурс] Режим доступу до ресурсу: https://discord.com/developers/docs/intro

14. Discord Developer Portal [Електронний ресурс]. – Режим доступу: https://discord.com/developers (дата звернення: 16.05.2025).

15. Discord Developer Portal. OAuth 2.0 : [Електронний ресурс]. – Режим доступу: https://discord.com/developers/docs/topics/oauth2 (дата звернення: 16.05.2025).

16. Discord Developer Portal. Rate Limits : [Електронний ресурс]. – Режим доступу: https://discord.com/developers/docs/topics/rate-limits (дата звернення: 16.05.2025).

17. Discord Embedded App SDK Documentation [Електронний ресурс]. – Режим доступу: https://discord.com/developers/build (дата звернення: 16.05.2025).

18. discord.js Guide: Introduction [Електронний ресурс]. – Режим доступу: https://discordjs.guide/ (дата звернення: 16.05.2025).

19. Docker Docs. Manuals : офіційна документація Docker [Електронний ресурс]. – Режим доступу: https://docs.docker.com/manuals/ (дата звернення: 16.05.2025).

20. Fowler M. Refactoring: Improving the Design of Existing Code. – 2nd ed. – Addison-Wesley, 2019. – 448 p.

21. Gamma E. та ін. Design Patterns: Elements of Reusable Object-Oriented Software. – Boston : Addison-Wesley, 1995. – 395 p.

22. GitHub. Discord API Documentation [Електронний ресурс]. – Режим доступу: https://github.com/discord/discord-api-docs (дата звернення: 16.05.2025).

23. Grinberg M. Flask Web Development: Developing Web Applications with Python. – 2nd ed. – O'Reilly Media, 2018. – 392 p.

24. How to make a discord bot in python [Електронний ресурс] Режим доступу до ресурсу: https://realpython.com/how-to-make-a-discord-bot-python/

25. Jurafsky D., Martin J. Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition. – 4th ed. – Pearson, 2023. – 1 280 p.

26. Kumar N. Discord Users & Market Share Statistics 2025 (Updated) – URL: https://www.demandsage.com/discord-statistics/?utm_source=chatgpt.com

27. Lambert K. A. Fundamentals of Python: First Programs, 2nd Edition. – Cengage, 2019.

28. Li F., Yang Y. Impact of Artificial Intelligence–Generated Content Labels on Perceived Accuracy, Message Credibility, and Sharing Intentions for Misinformation: Web-Based, Randomized, Controlled Experiment // JMIR Form. Res. – 2024. – Vol. 8. – e60024 [Електронний ресурс]. – Режим доступу: https://www.jmir.org/themes/763-chatbots-and-conversational-agents/2024 (дата звернення: 16.05.2025).

29. Linda H. Create & host a Discord bot with Heroku in 5 min : інструкція / Medium [Електронний ресурс]. – 2024. – Режим доступу: https://medium.com/@linda0511ny/create-host-a-discord-bot-with-heroku-in-5-min-5cb0830d0ff2 (дата звернення: 16.05.2025).

30. Lutz M. Learning Python, 5th Edition Fifth Edition. - O'Reilly Media, 2016. - 1643 p.

31. Lutz M. Python: Pocket Reference Fourth Edition. - O'Reilly Media, 2016. - 210 p.

32. Norambuena I. N., Bergel A. Building a bot for automatic expert retrieval on discord. In: Proceedings of the 5th international workshop on machine learning

techniques for software quality evolution. 2021. p. 25-30.

33. OAuth 2.0 Authorization Framework (RFC 6749) [Електронний ресурс]. – Режим доступу: https://datatracker.ietf.org/doc/html/rfc6749 (дата звернення: 16.05.2025).

34. Parsing Options // discord.js Guide [Електронний ресурс]. – Режим доступу: https://discordjs.guide/slash-commands/parsing-options (дата звернення: 16.05.2025).

35. PEP 3156 – Asynchronous IO Support Rebooted: the “asyncio” Framework [Електронний ресурс]. – Режим доступу: https://peps.python.org/pep-3156/ (дата звернення: 16.05.2025).

36. PEP 492 – Coroutines with async and await syntax [Електронний ресурс]. – Режим доступу: https://peps.python.org/pep-0492/ (дата звернення: 16.05.2025).

37. PEP 8 – Style Guide for Python Code [Електронний ресурс]. – Режим доступу: https://peps.python.org/pep-0008/ (дата звернення: 16.05.2025).

38. PostgreSQL Global Development Group. PostgreSQL 16 Documentation. – Ver. 16.9. – 2025. – 3680 с. : [Електронний ресурс]. – Режим доступу: https://www.postgresql.org/files/documentation/pdf/16/postgresql-16-A4.pdf (дата звернення: 16.05.2025).

39. Python requests module [Електронний ресурс] Режим доступу до ресурсу: https://www.w3schools.com/python/module_requests.asp

40. Python Software Foundation. Python 3.12 Documentation [Електронний ресурс]. – Режим доступу: https://docs.python.org/3.12/ (дата звернення: 16.05.2025).

41. Python: створення і використання бібліотек [Електронний ресурс] Режим доступу: http://www.kievoit.ippo.kubg.edu.ua/kievoit/2016/66_Python/index.html

42. Quickstart // discord.py Documentation [Електронний ресурс]. – Режим доступу: https://discordpy.readthedocs.io/en/stable/quickstart.html (дата звернення: 16.05.2025).

43. Richardson C. Microservices Patterns: With Examples in Java. – Shelter Island : Manning Publications, 2019. – 520 с.

44. Subbotin S. O. Нейронні мережі: теорія та практика : навч. посіб. / С. О. Субботін. – Житомир : Вид. О.О. Євенок, 2020. – 184 с.

45. The impact of chatbots based on large language models on second-language vocabulary acquisition // Heliyon. – 2024. – Vol. 10, № 4. – Article e14014 : [Електронний ресурс]. – Режим доступу: https://www.sciencedirect.com/science/article/pii/S2405844024014014 (дата звернення: 16.05.2025).

46. The WebSocket Protocol (RFC 6455) [Електронний ресурс]. – Режим доступу: https://datatracker.ietf.org/doc/html/rfc6455 (дата звернення: 16.05.2025).

47. Unveiling Trends of Chatbot and Conversational Agents: A Bibliometric Review // Advances in Computer Science and Systems. – 2024. – № 1. – С. 25–50. [Електронний ресурс]. – Режим доступу: https://sciendo.com/article/10.2478/acss-2024-0019 (дата звернення: 16.05.2025).

48. Verma A., Tyagi S., Mathur G. A comprehensive review on bot-discord bot. International Journal of Scientific Research in Computer Science, Engineering and Information Technology, 2021, 7.2: 532-536.

49. Welcome to discord.py : документація [Електронний ресурс]. – Режим доступу: https://discordpy.readthedocs.io/ (дата звернення: 16.05.2025).

50. What are Privileged Intents? : довідка для розробників Discord [Електронний ресурс]. – 08.04.2025. – Режим доступу: https://support-dev.discord.com/hc/en-us/articles/6207308062871-What-are-Privileged-Intents (дата звернення: 16.05.2025).

51. WriteBots. How To Make a Discord Bot in 2023: From the Ground Up [Електронний ресурс]. – Режим доступу: https://www.writebots.com/how-to-make-a-discord-bot/ (дата звернення: 16.05.2025).

52. Руденко О. Г., Бодянський Є. В. Штучні нейронні мережі : навч. посібник. – Харків : ТОВ «Компанія СМІТ», 2006. – 404 с.

Додаток А
(обов'язковий)

Технічне завдання

Зміст

1 Найменування та область застосування

Це технічне завдання розповсюджується на розробку програмного забезпечення чат-боту для платформи обміну повідомленнями Discord.

2 Підстава для розробки

Підставою для розробки служить завдання на кваліфікаційну бакалаврську роботу №47-02 від 17.01.2025 року, видане на кафедрі кібербезпеки та програмного забезпечення.

3 Мета та призначення розробки

Метою кваліфікаційної бакалаврської роботи є розробка програмного забезпечення чат-боту для платформи обміну повідомленнями Discord.

4 Джерела розробки

Джерелом цієї кваліфікаційної бакалаврської роботи є стосовна до теми література і існуючі аналоги.

5 Технічні вимоги

5.1 Склад продукції

Складниками розробки є:

– вибір і обґрунтування методів реалізації проекту;

– розробка програмної частин системи, а також розробка взаємодії системи з ОС та з користувачем;

– розробка програми, що реалізує чат-бот для платформи обміну повідомленнями Discord.

5.2 Показники призначення

Система повинна забезпечувати:

– систему для реалізації чат-боту для платформи обміну повідомленнями Discord;

– цілісність даних у процесі роботи та при зберіганні;

– простий, інтуїтивно зрозумілий інтерфейс.

5.3 Вимоги до функціональних характеристик

Розроблене програмне забезпечення не повинно мати обмежень на версію драйверів та операційної системи.

5.4 Вимоги до архітектури

Компонент, що розробляється повинен використовувати системні засоби та апаратні засоби, що на даному етапі розвитку обчислювальної техніки найбільше поширені.

5.5 Вимоги до надійності

Програмні модулі написані по всім правилам, які стосуються стандартних викликів процедур, функцій, методів і форм, визначених технічною документацією на середовище розробки.

5.6 Умови експлуатації

Робочі місця користувачів ПЗ повинні задовольняти наступним умовам експлуатації:

– температура повітря: 19-20 град. по Цельсію;

– відносна вологість повітря до 80%;

– атмосферний тиск 107 кПа.

5.7 Вимоги до складу та параметрів технічних засобів

Програмне забезпечення повинно бути реалізоване на ПЕОМ архітектури IBM PC, працювати в ОС Windows 10/11 і з сумісними з цією платформою пристроями і прикладним програмним забезпеченням.

5.8 Вимоги до інформаційної і програмної сумісності

Переносність програмного забезпечення повинна бути забезпечена за рахунок його реалізації стандартного інтерфейсу взаємодії з ОС, що працюють під управлінням ОС Windows 10/11.

5.8.1 Обладнання

Комп’ютер Intel Core i7/8 ГБ /1 Tb/ GeForce GT 1030 2GB або сумісні з ним.

5.8.2 Мова програмування

Програму розроблено на мовах програмування Python.

5.8.3 Вхідні дані

Опис алгоритму роботи запропонованої системи.

5.8.4 Вихідні дані

Робоча програма.

6 Вимоги до програмної документації

Програмна продукція повинна бути представлена у виді опису структури даних, схем та опису алгоритму, а також текстів вихідних модулів програмного забезпечення згідно ЄСПД.

7 Перелік документів, що розробляються

– Структурна схема системи.
– Функціональна схема системи.
– Діаграма процесів.
– Блок-схема алгоритму роботи програми.
– Пояснювальна записка.

8 Етапи розробки

8.1 Збір і обробка інформації по темі кваліфікаційної бакалаврської роботи. Постановка задачі на виконання кваліфікаційної бакалаврської роботи (складання ТЗ).

8.2 Проведення досліджень або експериментальних робіт для уточнення основних положень кваліфікаційної бакалаврської роботи.

8.3 Розробка функціональних схем, блок схем алгоритмів роботи програмного забезпечення.

8.4 Побудова схем взаємодії даних.

8.5 Створення прототипу ПЗ.

8.6 Віднаходження ПЗ, аналіз отриманих результатів.

8.7 Оформлення пояснювальної записки і виконання робіт по графічній частині.

11 Порядок контролю та приймання

11.1 Подання кваліфікаційної бакалаврської роботи на попередній захист 24.05.2025 р.

11.2 Подання кваліфікаційної бакалаврської роботи на захист __.06.2025 р.

Додаток Б

(обов’язковий)

Міністерство освіти і науки України

Центральноукраїнський національний технічний університет

ЗАТВЕРДЖУЮ

Керівник випускної кваліфікаційної роботи

за першим (бакалаврським) рівнем вищої освіти

_____ Є.В. Мелешко

Програмне забезпечення чат-боту для платформи обміну повідомленнями Discord

Лістинг програми

Код документу 12

Носій: CD/DVD-диск

Загальна кількість аркушів: 12

Літера: РП

Кропивницький – 2025 року

Файл Bot_Discord.py основної програми

```
# -*- coding: utf-8 -*-
"""
Bot Discord
==============
Бот для Discord
Автор: Графенюк Валерія
Ліцензія: CC BY-NC-SA 4.0
"""


# ===== Імпорти ================================================================

from __future__ import annotations

import asyncio
import datetime as dt
import io
import logging
import textwrap
from pathlib import Path
from typing import Final, Optional

import requests
import yt_dlp
from bs4 import BeautifulSoup as BS
from PIL import Image, ImageDraw, ImageFont

import discord
from discord import Intents, Streaming
from discord.ext import commands
from discord.utils import get


# ===== Налаштування логування ================================================

LOG_FILE = Path("brave_bot.log")
logging.basicConfig(
    level=logging.INFO,
    format="%(asctime)s | %(levelname)8s | %(name)s » %(message)s",
    handlers=[
        logging.FileHandler(LOG_FILE, encoding="utf-8"),
        logging.StreamHandler(),
    ],
)
log = logging.getLogger("brave_bot")


# ===== Константи ==============================================================

COMMAND_PREFIX: Final[str] = "#"
DATE_FMT:       Final[str] = "%a, %d %b %Y"
YDL_OPTS: Final[dict] = {
    "format": "bestaudio/best",
    "noplaylist": False,
    "simulate": True,
    "key": "FFmpegExtractAudio",
}
FFMPEG_OPTS: Final[dict] = {
    "before_options": "-reconnect 1 -reconnect_streamed 1 -reconnect_delay_max
5",
    "options": "-vn",
}
```

```
HELLO_WORDS         = {"hello", "hi", "привіт"}
RESTRICTED_WORDS    = {"restricted"}
DEFAULT_FONT        = "arial.ttf"
STAT_CARD_SIZE      = (400, 200)
AVATAR_SIZE         = (100, 100)
THEME_BG            = "#232529"
LOGS_CHANNEL_ID     = 1042181518254149714  # канал для логів
AUTO_ROLE_ID        = 1042188049355911249  # роль, яку видаємо новачкам
STREAM_ROLE_ID      = 1041826271178543205  # роль «У прямому ефірі»
GUILD_ID            = 361970912058408960   # основний сервер
STREAM_NOTIFY_CH    = 1044229509412573294  # канал для сповіщень про стріми


# ===== Ініціалізація клієнта ================================================

intents = Intents.all()
bot = commands.Bot(
    command_prefix=COMMAND_PREFIX,
    intents=intents,
    help_command=None,        # вимикаємо класичну !help
)
voice_connected = False       # прапорець підключення до голосового каналу


# ===== Події-учасники ======================================================

@bot.event
async def on_ready() -> None:
    log.info("Бот успішно під'єднаний як %s", bot.user)
    await bot.change_presence(
        status=discord.Status.online,
        activity=discord.Game(name="Brave Bot 2025"),
    )


# ---------- Автоматична видача ролі ----------------------------------

@bot.event
async def on_member_join(member: discord.Member) -> None:
    """Видаємо роль новачку та кидаємо повідомлення у вітальний канал."""
    guild = bot.get_guild(GUILD_ID)
    if guild is None:
        return

    role = guild.get_role(AUTO_ROLE_ID)
    channel = guild.system_channel  # або статичний ID, якщо потрібно
    if role and channel:
        await member.add_roles(role, reason="Автовидача ролі при вході")
        await channel.send(
            embed=discord.Embed(
                description=f"Користувач **{member.mention}** приєднався до
команди!",
                colour=discord.Colour.green(),
            )
        )


# ======= Логи видалених / змінюваних повідомлень ============================

async def send_log(embed: discord.Embed) -> None:
    """Допоміжна функція, щоб не дублювати код."""
    chan = bot.get_channel(LOGS_CHANNEL_ID)
    if chan:
        await chan.send(embed=embed)
```

```
@bot.event
async def on_message_delete(message: discord.Message) -> None:
    if message.guild is None:  # DM
        return
    emb = discord.Embed(
        title=f"Видалене повідомлення від {message.author}",
        description=f"**Текст:** {message.content}\n**Канал:**
{message.channel.mention}",
        timestamp=dt.datetime.now(),
        colour=discord.Colour.red(),
    )
    emb.set_author(name=message.author.name,
icon_url=message.author.display_avatar)
    await send_log(emb)

@bot.event
async def on_message_edit(before: discord.Message, after: discord.Message) ->
None:
    if before.content == after.content:
        return
    emb = discord.Embed(
        title=f"Повідомлення від {before.author} змінено",
        description=(
            f"**Було:** {before.content}\n"
            f"**Стало:** {after.content}\n"
            f"**Канал:** {before.channel.mention}"
        ),
        timestamp=dt.datetime.now(),
        colour=discord.Colour.blue(),
    )
    emb.set_author(name=after.author.name, icon_url=after.author.display_avatar)
    await send_log(emb)

@bot.event
async def on_member_update(before: discord.Member, after: discord.Member) ->
None:
    if before.roles != after.roles:
        # Визначаємо, яку роль додали / зняли
        added  = set(after.roles) - set(before.roles)
        removed = set(before.roles) - set(after.roles)
        if added:
            role = next(iter(added))
            emb = discord.Embed(
                title="Нова роль",
                description=f"{after.mention} отримав роль **{role.name}**",
                colour=discord.Colour.green(),
                timestamp=dt.datetime.now(),
            )
        else:
            role = next(iter(removed))
            emb = discord.Embed(
                title="Роль прибрана",
                description=f"У {after.mention} забрали роль **{role.name}**",
                colour=discord.Colour.red(),
                timestamp=dt.datetime.now(),
            )
        emb.set_author(name=after.name, icon_url=after.display_avatar)
        await send_log(emb)
    elif before.nick != after.nick:
        emb = discord.Embed(
            title="Зміна ніка",
            description=f"**Було:** {before.nick}\n**Стало:** {after.nick}",
            colour=discord.Colour.orange(),
            timestamp=dt.datetime.now(),
        )
        emb.set_author(name=after.name, icon_url=after.display_avatar)
        await send_log(emb)
```

```
# ======= Голос та музика ==================================================

def user_voice(ctx: commands.Context) -> Optional[discord.VoiceState]:
    """Повертає voice-стан користувача або None, якщо він не у voice."""
    return ctx.author.voice

@bot.command(name="play", brief="Відтворити музику за URL або пошуком")
async def cmd_play(ctx: commands.Context, *, query: str) -> None:
    """Під'єднання до voice-каналу та відтворення аудіо."""
    global voice_connected

    voice_state = user_voice(ctx)
    if voice_state is None:
        await ctx.reply("Спочатку приєднайтеся до голосового каналу.")
        return

    if not voice_connected:
        vc = await voice_state.channel.connect()
        voice_connected = True
    else:
        vc = ctx.voice_client  # type: ignore


  # ----- Завантажуємо аудіо-потік через yt-dlp ---------------------------

  with yt_dlp.YoutubeDL(YDL_OPTS) as ydl:
        try:
            info = (
                ydl.extract_info(query, download=False)
                if query.startswith("http")
                else ydl.extract_info(f"ytsearch:{query}",
download=False)["entries"][0]
            )
        except Exception as exc:
            await ctx.reply(f"Помилка yt-dlp: {exc}")
            return

    stream_url = info["url"]
    vc.play(
        discord.FFmpegPCMAudio(
            executable="ffmpeg",  # сучасний Discord дозволяє без шляху, якщо
ffmpeg у PATH
            source=stream_url,
            **FFMPEG_OPTS,
        )
    )
    await ctx.message.add_reaction("▶")

@bot.command(name="stop", brief="Зупинити музику та вийти з voice")
async def cmd_stop(ctx: commands.Context) -> None:
    """Вихід із voice-каналу та скидання прапорця."""
    global voice_connected
    if not voice_connected:
        await ctx.reply("Я поки не в голосовому каналі.")
        return
    await ctx.voice_client.disconnect()  # type: ignore
    voice_connected = False
    await ctx.message.add_reaction("■")


# ======= Картки статистики ================================================

def _draw_card_base(avatar_bytes: bytes) -> Image.Image:
    """Створює базове зображення й вставляє аватар."""
    card = Image.new("RGBA", STAT_CARD_SIZE, THEME_BG)
    avatar =
Image.open(io.BytesIO(avatar_bytes)).convert("RGBA").resize(AVATAR_SIZE,
Image.LANCZOS)
```

```
    card.paste(avatar, (15, 15, 115, 115))
    return card

def _font(size: int) -> ImageFont.FreeTypeFont:
    return ImageFont.truetype(DEFAULT_FONT, size=size)

@bot.command(aliases=["я", "stats", "me"], brief="Ваша особиста статистика")
async def card_user(ctx: commands.Context, member: Optional[discord.Member] =
None) -> None:
    """Генерує та надсилає картку користувача."""
    member = member or ctx.author
    response = requests.get(str(member.display_avatar.replace(size=128)))
    card = _draw_card_base(response.content)
    draw = ImageDraw.Draw(card)

    draw.text((145, 15), f"{member} ", font=_font(20), fill="white")
    draw.text((145, 50), f"Найвища роль: {member.top_role.name}",
font=_font(12), fill="white")
    draw.text((145, 80), f"Приєднався: {member.joined_at.strftime(DATE_FMT)}",
font=_font(12), fill="white")
    draw.text((145, 110), f"Зареєструвався:
{member.created_at.strftime(DATE_FMT)}", font=_font(12), fill="white")
    draw.text((145, 140), f"Статус: {member.status}", font=_font(12),
fill="white")

    path = Path("user_card.png")
    card.save(path)
    await ctx.send(file=discord.File(path))

@bot.command(aliases=["server", "сервер"], brief="Інформація про сервер")
async def card_server(ctx: commands.Context) -> None:
    """Генерує картку сервера."""
    guild = ctx.guild
    if guild is None:
        return
    response = requests.get(str(guild.icon.replace(size=128)))
    card = _draw_card_base(response.content)
    draw = ImageDraw.Draw(card)

    draw.text((145, 15), guild.name, font=_font(20), fill="white")
    draw.text((145, 50), f"Власник: {guild.owner}", font=_font(12),
fill="white")
    draw.text((145, 80), f"Учасників: {guild.member_count}", font=_font(12),
fill="white")
    draw.text((145, 110), f"Каналів: {len(guild.channels)}", font=_font(12),
fill="white")
    draw.text((145, 140), f"Текстових: {len(guild.text_channels)}",
font=_font(12), fill="white")
    draw.text((145, 170), f"Голосових: {len(guild.voice_channels)}",
font=_font(12), fill="white")

    path = Path("server_card.png")
    card.save(path)
    await ctx.send(file=discord.File(path))


# ======= Стрим-сповіщення =================================================

@bot.event
async def on_presence_update(before: discord.Member, after: discord.Member) ->
None:
    """Видає/знімає роль STREAM_ROLE_ID та сповіщає про початок/кінець
стріму."""
    if after.guild and after.guild.id != GUILD_ID:
        return

    guild = after.guild
    stream_role = guild.get_role(STREAM_ROLE_ID) if guild else None
    channel = bot.get_channel(STREAM_NOTIFY_CH)
```

```
    async def _post_start():
        if stream_role:
            await after.add_roles(stream_role, reason="Почався стрім")
        emb = discord.Embed(
            title=f"🔴 {after.display_name} зараз у прямому ефірі!",
            url=after.activity.url,
            colour=discord.Colour.brand_red(),
            timestamp=dt.datetime.now(),
        )
        emb.add_field(name="Назва", value=after.activity.name)
        if after.activity.game:
            emb.add_field(name="Категорія", value=after.activity.game,
inline=False)
        await channel.send(embed=emb)

    async def _post_end(prev_activity: Streaming):
        if stream_role:
            await after.remove_roles(stream_role, reason="Стрім завершено")
        emb = discord.Embed(
            title=f"🏁 Стрім {prev_activity.twitch_name} завершено",
            colour=discord.Colour.orange(),
            timestamp=dt.datetime.now(),
        )
        emb.add_field(name="Тема", value=prev_activity.name)
        await channel.send(embed=emb)


  # --- Логіка початку/кінця ------------------------------------------------

 if isinstance(after.activity, Streaming) and not isinstance(before.activity,
Streaming):
        await _post_start()
    elif isinstance(before.activity, Streaming) and not
isinstance(after.activity, Streaming):
        await _post_end(before.activity)


# ======= Модерація ==========================================================

def is_admin():
    return commands.has_permissions(administrator=True)

@bot.command(brief="Очистити повідомлення")
@is_admin()
async def clear(ctx: commands.Context, amount: int = 50) -> None:
    deleted = await ctx.channel.purge(limit=amount + 1)
    await ctx.send(f"🧺 Видалено {len(deleted)-1} повідомлень.", delete_after=5)

@bot.command(brief="Вигнати користувача")
@is_admin()
async def kick(ctx: commands.Context, member: discord.Member, *, reason: str =
"Без пояснення") -> None:
    await member.kick(reason=reason)
    await ctx.send(f"👢 {member.mention} вигнано. Причина: {reason}")

@bot.command(brief="Забанити користувача")
@is_admin()
async def ban(ctx: commands.Context, member: discord.Member, *, reason: str =
"Без пояснення") -> None:
    await member.ban(reason=reason)
    await ctx.send(f"⛔ {member.mention} забанено. Причина: {reason}")

@bot.command(brief="Розбанити користувача")
@is_admin()
async def unban(ctx: commands.Context, user: discord.User) -> None:
    await ctx.guild.unban(user)
    await ctx.send(f"✅ {user.mention} розбанено.")
```

```
# ======= Реакції на ключові слова ========================================

@bot.event
async def on_message(message: discord.Message) -> None:
    if message.author.bot:
        return

    lower = message.content.lower()
    if lower in RESTRICTED_WORDS:
        await message.delete()
        await message.channel.send(f"{message.author.mention}, це слово
заборонене!")

    elif lower in HELLO_WORDS:
        await message.channel.send(f"Привіт, {message.author.mention}! 👋")

    await bot.process_commands(message)


# ======= Запуск ==========================================================

def main() -> None:
    token = Path("token.txt").read_text().strip()
    bot.run(token)

if __name__ == "__main__":
    main()
```

Файл Bot_Discord.py основної програми

```
# -*- coding: utf-8 -*-
"""
UtilityBot Discord
==============
Бот для Discord
Автор: Графенюк Валерія
Ліцензія: CC BY-NC-SA 4.0
"""

Додаткові модулі:
-------------------
1. **Нагадувачі / планувальник** — `!remind 15m Зробити перерву`;
2. **Опитування / голосування** — `!poll «Що сьогодні п’ємо?» Кава Чай`;
3. **Reaction-roles** — роздача ролей через емодзі;
4. **Миттєвий перекладач** — `!translate en Іспити скоро?`;
5. **Погода та новини** (API open-source, без ключів — JSON з open-meteo);
6. **GitHub notifier** — стрим-вебхук, пише про нові PR у канал;
7. **Stand-up wizard** — щоденне питання «Що зробив учора / сьогодні?»
   з автоматичним підсумком у thread.

"""

from __future__ import annotations

import asyncio
import datetime as dt
import re
from typing import Final, Optional

import aiohttp
import discord
from discord.ext import commands, tasks

# ---------------------------------------------------------------------------
#  Налаштування
# ---------------------------------------------------------------------------
COMMAND_PREFIX: Final[str] = "!"   # не перетинаємось із BraveBot («#»)
INTENTS = discord.Intents.default()
INTENTS.message_content = True      # потрібен для ремайндера / опитувань

bot = commands.Bot(
    command_prefix=COMMAND_PREFIX,
    intents=INTENTS,
    help_command=None,
    description="UtilityBot 2025 — нагадування, опитування, реакції та інше",
)

# ---------------------------------------------------------------------------
#  Допоміжні утиліти
# ---------------------------------------------------------------------------

def parse_timespec(spec: str) -> Optional[int]:
    """Перетворює рядок «10m», «2h30m», «1d» на секунди."""
    pattern =
re.compile(r"(?:(?P<d>\d+)d)?(?:(?P<h>\d+)h)?(?:(?P<m>\d+)m)?(?:(?P<s>\d+)s)?")
    match = pattern.fullmatch(spec)
    if not match or all(v is None for v in match.groupdict().values()):
        return None
    seconds = (
        int(match.group("d") or 0) * 86400 +
        int(match.group("h") or 0) * 3600 +
        int(match.group("m") or 0) * 60 +
        int(match.group("s") or 0)
    )
    return seconds or None
```

```
# ---------------------------------------------------------------------------
#  Команди: Нагадувачі
# ---------------------------------------------------------------------------

@bot.command(name="remind", brief="Нагадування через час")
async def remind(ctx: commands.Context, timespec: str, *, text: str) -> None:
    """!remind 10m Піти за кавою"""
    delay = parse_timespec(timespec)
    if delay is None:
        await ctx.reply("⏰ Формат часу: 10m | 2h30m | 1d…")
        return

    await ctx.reply(f"⏳ Добре, нагадаю за {timespec}…")

    async def _later():
        await asyncio.sleep(delay)
        await ctx.author.send(f"🔔 Нагадування: {text}")

    bot.loop.create_task(_later())

# ---------------------------------------------------------------------------
#  Команди: Опитування / голосування
# ---------------------------------------------------------------------------

POLL_EMOJIS = ("1️⃣", "2️⃣", "3️⃣", "4️⃣", "5️⃣", "6️⃣", "7️⃣", "8️⃣", "9️⃣", "🔟")

@bot.command(name="poll", brief="Створити опитування")
async def poll(ctx: commands.Context, question: str, *options: str) -> None:
    """!poll "Що робимо?" Граємо Читаємо Спимо"""
    if len(options) < 2:
        await ctx.reply("Потрібно щонайменше 2 варіанти.")
        return
    if len(options) > len(POLL_EMOJIS):
        await ctx.reply("Максимум 10 варіантів.")
        return

    desc = "\n".join(f"{POLL_EMOJIS[i]}  {opt}" for i, opt in
enumerate(options))
    embed = discord.Embed(title=question, description=desc,
colour=discord.Colour.gold())
    message = await ctx.send(embed=embed)
    for i in range(len(options)):
        await message.add_reaction(POLL_EMOJIS[i])

# ---------------------------------------------------------------------------
#  Команди: Reaction-roles (спрощена версія)
# ---------------------------------------------------------------------------

REACTION_ROLE_MESSAGE_ID: Optional[int] = None  # ← встановіть вручну або
створіть через команду
ROLE_BY_EMOJI: dict[str, int] = {
    "🎮": 123456789012345678,  # id ролей
    "🎨": 234567890123456789,
}

@bot.command(name="rrsetup", hidden=True)
@commands.has_permissions(manage_roles=True)
async def rrsetup(ctx: commands.Context):
    """Показує повідомлення для реакцій-ролей і запам'ятовує його ID."""
    global REACTION_ROLE_MESSAGE_ID
    lines = [f"{emoji}  <@&{role_id}>" for emoji, role_id in
ROLE_BY_EMOJI.items()]
    msg = await ctx.send("**Оберіть роль:**\n" + "\n".join(lines))
    REACTION_ROLE_MESSAGE_ID = msg.id
    for emoji in ROLE_BY_EMOJI:
        await msg.add_reaction(emoji)
```

```
@bot.event
async def on_raw_reaction_add(payload: discord.RawReactionActionEvent):
    if payload.message_id != REACTION_ROLE_MESSAGE_ID or payload.guild_id is
None:
        return
    role_id = ROLE_BY_EMOJI.get(str(payload.emoji))
    if role_id is None:
        return
    guild = bot.get_guild(payload.guild_id)
    if guild is None:
        return
    role = guild.get_role(role_id)
    member = guild.get_member(payload.user_id)
    if role and member:
        await member.add_roles(role)

@bot.event
async def on_raw_reaction_remove(payload: discord.RawReactionActionEvent):
    if payload.message_id != REACTION_ROLE_MESSAGE_ID or payload.guild_id is
None:
        return
    role_id = ROLE_BY_EMOJI.get(str(payload.emoji))
    if role_id is None:
        return
    guild = bot.get_guild(payload.guild_id)
    if guild is None:
        return
    role = guild.get_role(role_id)
    member = guild.get_member(payload.user_id)
    if role and member:
        await member.remove_roles(role)

# ---------------------------------------------------------------------------
#  Погода (open-meteo.com, без ключа)
# ---------------------------------------------------------------------------

async def fetch_weather(lat: float, lon: float) -> str:
    url = (
        "https://api.open-
meteo.com/v1/forecast?latitude={lat}&longitude={lon}&hourly=temperature_2m"
        .format(lat=lat, lon=lon)
    )
    async with aiohttp.ClientSession() as session:
        async with session.get(url) as resp:
            data = await resp.json()
    temp = data["hourly"]["temperature_2m"][0]
    return f"🌡 Температура зараз: {temp}°C"

@bot.command(name="weather", brief="Погода по координатах")
async def weather(ctx: commands.Context, lat: float, lon: float):
    msg = await fetch_weather(lat, lon)
    await ctx.send(msg)

# ---------------------------------------------------------------------------
#  Стенд-ап воркфлоу (щодня о 10:00 за Києвом)
# ---------------------------------------------------------------------------

STANDUP_CHANNEL_ID: Final[int] = 987654321098765432
TIMEZONE = dt.timezone(dt.timedelta(hours=3))  # Europe/Kyiv, без pytz

@tasks.loop(time=dt.time(hour=10, minute=0, tzinfo=TIMEZONE))
async def standup_daily():
    ch = bot.get_channel(STANDUP_CHANNEL_ID)
    if isinstance(ch, discord.TextChannel):
        thread = await ch.create_thread(name=f"Stand-up
{dt.date.today().isoformat()}")
```

```
        await thread.send("1️⃣ Що зробили вчора?\n2️⃣ Що плануємо сьогодні?\n3️⃣ Чи є
блокери?")

@standup_daily.before_loop
async def before_standup():
    await bot.wait_until_ready()

standup_daily.start()

# ---------------------------------------------------------------------------
#  Help
# ---------------------------------------------------------------------------

@bot.command(name="help")
async def help_cmd(ctx: commands.Context):
    cmds = {
        "remind": "Нагадування через час: !remind 30m Зробити каву",
        "poll": "Опитування: !poll \"Піцца?\" Маргарита Гавайська",
        "weather": "Погода: !weather <lat> <lon>",
    }
    embed = discord.Embed(title="UtilityBot — команди",
colour=discord.Colour.green())
    for name, desc in cmds.items():
        embed.add_field(name=f"!{name}", value=desc, inline=False)
    await ctx.send(embed=embed)

# ---------------------------------------------------------------------------
#  Запуск
# ---------------------------------------------------------------------------

def main() -> None:
    with open("token.txt", "r", encoding="utf-8") as fh:
        token = fh.readline().strip()
    bot.run(token)

if __name__ == "__main__":
    main()
```