Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії
Кафедра програмного забезпечення

Бакалаврська кваліфікаційна робота

на тему: Розробка програмного забезпечення мультимедійного бота для управління Discord каналом

Виконав: студент 4 курсу
групи 4ПІ-21Б
спеціальності
121 – Інженерія програмного забезпечення
(шифр і назва напряму підготовки, спеціальності)
Довгалюк Д. В.
(прізвище та ініціали)
Керівник: к.т.н., доц. каф. ПЗ Романюк О. В.
(прізвище та ініціали)
Рецензент: к.т.н., доц. каф. ЗІ Куперштейн Л. М.
(прізвище та ініціали)

Допущено до захисту

Зав. кафедри ПЗ
д.т.н., проф. Романюк О. Н.
«09» 06 2025 р.

ВНТУ – 2025

Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії
Кафедра програмного забезпечення
Рівень вищої освіти перший (бакалаврський)
Галузь знань 12 «Інформаційні технології»
Спеціальність 121 «Інженерія програмного забезпечення»
Освітньо-професійна програма «Інженерія програмного забезпечення»

ЗАТВЕРДЖУЮ
Завідувач кафедри ПЗ
_____О. Н. Романюк
« 21 » березня 2025 р.

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

Довгалюку Дмитрію Валентиновичу

1. Тема роботи – розробка програмного забезпечення мультимедійного бота для управління Discord каналом.

Керівник роботи: Романюк Оксана Володимирівна, к.т.н., доцент кафедри ПЗ, затверджені наказом вищого навчального закладу від 20 березня 2025 р. №97.

2. Строк подання студентом роботи 2 червня 2025.

3. Вихідні дані до роботи: дії бота – відтворення аудіо у голосовому каналі; додавання треків до черги відтворення; призупинення, відновлення або пропуск треків; вихід бота з голосового каналу; текстові повідомлення – підтвердження виконання команд, повідомлення про помилки або стан черги; інтерактивні елементи – реакції на повідомлення, вбудовані повідомлення з інформацією про трек або чергу; обробка команд – через Discord API; мультимедійна обробка – завантаження потоків через youtube_dl та yt-dlp; конвертація аудіо за допомогою FFmpeg; передача аудіо у голосовий канал через PyNaCl; контроль доступу – перевірка ролей і прав користувачів для обмеження доступу до певних команд; кінцевий результат – взаємодія користувача з ботом у Discord каналі.

4. Зміст текстової частини: вступ; аналіз програмного забезпечення для управління discord каналом; розробка методів роботи програмного забезпечення для управління discord каналом; розробка програмного забезпечення мультимедійного бота для управління discord каналом; тестування програмного забезпечення; висновки; список використаних джерел; додатки.

5. Перелік ілюстративної частини: титульний слайд; актуальність теми; мета, об'єкт дослідження та предмет дослідження; постановка задач дослідження; наукова новизна отриманих результатів; порівняльний аналіз аналогів; діаграма діяльності Discord бота; діаграма прецедентів; метод та блок-схема алгоритму репортів; метод та блок-схема алгоритму реалізації

динамічного пошуку та керування відтворенням аудіо контенту; метод та блок-схема алгоритму модерації контенту; порівняльний аналіз середовищ розробки; порівняльний аналіз мов програмування; функції надсилання та збереження репортів; функції керування аудіо контенту; функції фільтрації, обмеження та автоматичного видалення; тестування відтворення аудіо контенту; тестування модерації контенту та створення репортів; апробація та публікація матеріалів бакалаврської кваліфікаційної роботи; фінальний слайд.

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		Завдання видав	Завдання прийняв
1-4	Романюк О. В., к.т.н., доц. каф. ПЗ	24.03.25	01.06.25

7. Дата видачі завдання 24 березня 2025 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів бакалаврської кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Аналіз програмного забезпечення для управління Discord каналом	25.03.25 – 31.03.2025	Вик.
2	Розробка діаграми діяльності та структури роботи програмного забезпечення	31.03.25 – 05.04.2025	Вик.
3	Розробка методу репортів	05.03.2025 – 07.04.2025	Вик.
4	Розробка методу динамічного пошуку та керування відтворенням аудіо контенту	07.04.2025 – 12.04.2025	Вик.
5	Розробка методу модерації контенту	12.04.2025 – 23.04.2025	Вик.
6	Розробка програмного забезпечення мультимедійного бота для управління Discord каналом	23.04.2025 – 01.05.2025	Вик.
7	Тестування програмного забезпечення	01.05.2025 – 10.05.2025	Вик.
8	Оформлення матеріалів до захисту БКР	10.05.2025 – 30.05.25	Вик

Студент _____ Д. В. Довгалюк
(підпис) (ініціали та прізвище)

Керівник бакалаврської кваліфікаційної роботи _____ О. В. Романюк
(підпис) (ініціали та прізвище)

АНОТАЦІЯ

УДК 004.4

Довгалюк Д. В. Розробка програмного забезпечення мультимедійного бота для управління Discord каналом: бакалаврська кваліфікаційна робота зі спеціальності 121 – інженерія програмного забезпечення, освітня програма – інженерія програмного забезпечення. Вінниця: ВНТУ, 2025. 66 с.

На укр. мові. Бібліогр.: 28 назв; рис.: 32; табл.: 14.

У бакалаврській кваліфікаційній роботі обґрунтовано доцільність, практичну цінність та мету розробки програмного забезпечення мультимедійного бота для управління Discord каналом, що забезпечує відтворення медіаконтенту та підвищення ефективності модерації сервера.

Розроблено методи та алгоритми реалізації ключових функцій мультимедійного бота, зокрема створення репортів, динамічного пошуку і керування відтворенням аудіоконтенту, а також модерації контенту. Програмне забезпечення реалізовано із використанням середовища розробки Visual Studio Code та мови програмування Python. Проведено тестування програмного забезпечення, підготовлено інструкцію користувача та визначено системні вимоги.

Результати розробки програмного забезпечення мультимедійного бота для управління Discord каналом можуть бути використані для розширення можливостей управління сервером та взаємодії з користувачами.

Ключові слова: Discord бот, мультимедіа, автоматизація, голосові канали, Python, API, модерація.

ABSTRACT

UDC 004.4

Dovhaliuk D. V. Software Development of a Multimedia Bot for Discord Channel Management*:* bachelor's thesis in specialty 121 – Software Engineering, educational program - Software Engineering. Vinnytsia: VNTU, 2025. 66 p.

In Ukrainian. Bibliography: 28 titles; Figures: 32; Table 14.

The bachelor's qualification thesis substantiates the practical value and purpose of developing software for a multimedia bot designed to manage a Discord channel, enabling media content playback and improving server moderation efficiency.

Methods and algorithms for implementing the key functions of the multimedia bot have been developed, including a report generation, dynamic search and audio content playback control, as well as a content moderation. The software was implemented using the Visual Studio Code development environment and the Python programming language. Software testing was conducted, a user manual was prepared, and system requirements were defined.

The results of the development of the multimedia bot software for managing a Discord channel can be used to expand server management capabilities and enhance user interaction

Keywords: Discord bot, multimedia, automation, voice channels, Python, API, moderation.

ЗМІСТ

ВСТУП 4
1 АНАЛІЗ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ДЛЯ УПРАВЛІННЯ DISCORD КАНАЛОМ 8
1.1 Аналіз мультимедійних ботів для Discord 8
1.2 Порівняльний аналіз музичних ботів 9
1.3 Аналіз напрямків удосконалення програмного забезпечення мультимедійних ботів для Discord 13
1.4 Аналіз методів розв’язання поставленої задачі 15
1.5 Постановка задачі 17
1.6 Висновки 17
2 РОЗРОБКА МЕТОДІВ РОБОТИ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ДЛЯ УПРАВЛІННЯ DISCORD КАНАЛОМ 19
2.1 Розробка діаграми діяльності Discord бота 19
2.2 Розробка структури програмного забезпечення 20
2.3 Розробка методу репортів 28
2.4 Розробка методу динамічного пошуку та керування відтворенням аудіоконтенту 30
2.5 Розробка методу модерації контенту 32
2.6 Висновки 34
3 РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ МУЛЬТИМЕДІЙНОГО БОТА ДЛЯ УПРАВЛІННЯ DISCORD КАНАЛОМ 35
3.1 Варіантний аналіз і обґрунтування вибору засобів для реалізації програмного забезпечення 35
3.2 Аналіз вхідних та вихідних даних програмного забезпечення 38
3.3 Розробка функцій надсилання та збереження репортів 40
3.4 Розробка функцій керування аудіо контенту 42
3.5 Розробка функцій фільтрації, обмеження та автоматичного видалення 46
3.6 Висновки 49

4 ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ 51
4.1 Тестування функціоналу бота 51
4.2 Розробка інструкції користувача 60
4.3 Висновки 61
ВИСНОВКИ 63
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ 64
ДОДАТКИ 67
ДОДАТОК А. Технічне завдання 68
ДОДАТОК Б. Протокол перевірки на плагіат 72
ДОДАТОК В. Лістинг програмного забезпечення 73
ДОДАТОК Г. Ілюстративна частина 98

ВСТУП

Обґрунтування вибору теми дослідження.

Сучасний етап розвитку інформаційних технологій характеризується активним поширенням цифрових платформ для комунікації, серед яких особливу популярність здобула система голосового та текстового спілкування Discord. З початкового позиціонування як сервіс для геймерів, Discord перетворився на універсальний інструмент для організації спільнот різних напрямків. Зростання кількості користувачів та масштабів каналів створює потребу в автоматизації управління серверами, зокрема в частині контролю мультимедійного контенту, модерації та взаємодії з учасниками [1].

Існуючі інструменти автоматизації в Discord, зокрема боти, часто мають обмежену функціональність, або не відповідають специфічним потребам конкретних спільнот. Деякі з них мають складний процес налаштування, або не дозволяють керувати медіа контентом, таким як музика, відео або голосові повідомлення. У цьому контексті розробка власного мультимедійного бота є актуальною задачею, яка дозволяє створити інструмент з урахуванням індивідуальних вимог, адаптивний до потреб спільноти [2].

Мультимедійний бот здатний не лише покращити взаємодію користувачів у каналі, але й автоматизувати процеси відтворення музики, керування чергами відтворення, пошуку контенту за ключовими словами, а також інтеграції з зовнішніми API платформами для підвищення функціональності. Бот дозволяє зменшити навантаження на модераторів та забезпечити стабільне функціонування сервера навіть при великій кількості учасників [3].

Розробка програмного забезпечення мультимедійного бота також створює можливості для практичного застосування знань з програмування, роботи з API, обробки потокових даних, а також створення масштабованих архітектур з підтримкою багатозадачності.

Таким чином тема має практичну значимість та відповідає сучасним тенденціям сучасної трансформації комунікаційних платформ. Її реалізація

дозволяє підвищити дієвості управління Discord каналом, забезпечити взаємодію користувачів та розширити навички у сфері розробки програмних засобів.

Зв’язок роботи з науковими програмами, планами, темами. Робота виконувалася згідно з планом виконання наукових досліджень на кафедрі програмного забезпечення.

Мета та завдання дослідження. Метою дослідження є покращення процесу керування мультимедійним контентом на Discord каналі за рахунок розробки нових методів, що дозволить дозволить автоматизувати відтворення музики, обробку запитів користувачів і адміністрування контенту на сервері.

Відповідно до поставленої мети в бакалаврській кваліфікаційна роботі потрібно вирішити такі задачі:

– провести аналіз предметної області та існуючих методів для автоматизації керування Discord серверами;

– розробити діаграму діяльності та структуру роботи програмного забезпечення;

– розробити метод модерації контенту з автоматичним виявленням порушень на основі встановлених правил та шаблонів;

– розробити метод обробки репортів користувачів із подальшою класифікацією та передачею на розгляд модератору;

– розробити метод динамічного пошуку та керування відтворенням аудіоконтенту з урахуванням запитів користувача в режимі реального часу;

– розробити програмне забезпечення мультимедійного бота для управління Discord каналом;

– провести тестування програмного забезпечення;

– розробити інструкцію користувача та описати вимоги програмного забезпечення.

Об’єкт дослідження – процес управління мультимедійним контентом у середовищі Discord каналу.

Предмет дослідження – методи та засоби управління мультимедійним

контентом у середовищі Discord каналу.

Методи дослідження. У процесі дослідження застосовувалися: теорія системного програмування для реалізації автоматичного управління процесами в різних операційних системах; методи веб-інтеграції та REST API для взаємодії з Discord платформою та зовнішніми музичними сервісами; теорія алгоритмів для розробки алгоритми обробки аудіо-потоків та медіа контенту для відтворення музики з YouTube та інших джерел у голосових каналах Discord; методи тестування програмного забезпечення для перевірки стабільності роботи через комплексні перевірки середовища та автоматичне відновлення після збоїв; комп'ютерне моделювання для аналізу та перевірки отриманих теоретичних положень.

Наукова новизна отриманих результатів:

1. Розроблено новий метод динамічного пошуку та керування відтворенням аудіоконтенту, який поєднує API-запити до зовнішніх мультимедійних платформ із фільтрацією результатів за ключовими словами та релевантністю, що підвищує зручність використання бота, зменшує кількість помилок і забезпечує швидке та релевантне відтворення медіаконтенту.

2. Удосконалено метод модерації контенту, який на відміну від відомих забезпечує автоматичне виявлення та блокування шкідливих повідомлень у текстових каналах Discord, що дозволяє зменшити кількість хибних спрацювань і підвищити продуктивність модерації без втручання людини.

3. Розроблено новий метод репортів, який дозволяє повідомляти про порушення правил та помилки в чаті через спеціальний репорт-канал, що дало можливість полегшити роботу модераторів та пришвидшити реагування на інциденти.

Практичне значення отриманих результатів полягає в тому, що на основі отриманих в бакалаврській кваліфікаційній роботі теоретичних досліджень запропоновано алгоритми та розроблено програмне забезпечення мультимедійного Discord бота, яке забезпечує автоматизоване керування відтворенням аудіоконтенту, підтримку інтеграції з зовнішніми медіасервісами,

а також управління користувацькими командами у голосових і текстових каналах.

Особистий внесок здобувача. Усі наукові результати отримано здобувачем самостійно. У працях, опублікованих у співавторстві, студенту належать: аналіз напрямків удосконалення ботів для Discord з точки зору продуктивності та розширюваності [4]; алгоритм пошуку та відтворення аудіоконтенту, аналіз текстових повідомлень для модерації та обробка репортів [5]; реалізація обробки команд для Discord бота через слеш команди [6].

Апробація матеріалів бакалаврської кваліфікаційної роботи. Результати роботи доповідалися та обговорювалися на Міжнародних і Всеукраїнських конференціях, а саме:

- IV Всеукраїнській науково-технічній конференції молодих вчених, аспірантів і студентів «Комп'ютерні ігри і мультимедіа, як інноваційний підхід до комунікації - 2024» (Одеса, 2024);
- LIV Всеукраїнській науково-технічній конференції підрозділів Вінницького національного технічного університету (Вінниця, 2025);
- Міжнародній науково-практичній інтернет-конференції «Молодь в науці: дослідження, проблеми, перспективи (МН-2025)».

Публікації. За темою бакалаврської кваліфікаційної роботи опубліковано 3 наукові праці у збірниках матеріалів конференцій.

Структура та обсяг БКР. Бакалаврська кваліфікаційна робота складається зі вступу, чотирьох розділів, висновків, списку використаних джерел і додатків, у які винесено технічне завдання, протокол перевірки на плагіат, лістинг програмного забезпечення й ілюстративний матеріал до захисту роботи.

1 АНАЛІЗ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ДЛЯ УПРАВЛІННЯ DISCORD КАНАЛОМ

1.1 Аналіз мультимедійних ботів для Discord

З розвитком комунікаційних платформ, автоматизація управління медіаконтентом у спільнотах стає все більш актуальною. Discord є платформою для спілкування та координації користувачів, що об’єднують людей за інтересами. Впровадження мультимедійних ботів дозволяє значно підвищити управління спільнотами [7].

Однією з ключових переваг використання мультимедійних ботів у Discord є можливість управління аудіо та відео контентом. Представлена функція корисна для спільнот, пов’язаних з освітніми платформами та корпоративними комунікаціями. Такі боти можуть відтворювати музику, передавати повідомлення у голосових каналах, обробляти відеоконтент і інтегруватися з іншими цифровими сервісами для розширення функціональності.

Крім базового відтворення медіа, сучасні мультимедійні боти здатні аналізувати аудіо та відеопотоки в реальному часі, фільтрувати небажаний контент, розпізнавати мовлення та автоматично трансформувати його у текст, що є особливо корисним у навчальному та робочому середовищі.

Розробка програмних модулів для мультимедійних ботів у Discord потребує детального аналізу існуючих підходів та визначення ключових функціональних вимог. На ринку вже існують популярні мультимедійні боти, такі як Rythm, Groovy та Mee6. Вони надають базові можливості управління медіаконтентом. Проте більшість із них мають обмеження, пов’язані з ліцензуванням, можливих налаштувань та інтеграцією з іншими сервісами.

Багато Discord спільнот прагнуть мати власні мультимедійні рішення, адаптовані під їхні унікальні потреби. Саме тому розробка індивідуальних ботів із розширеним функціоналом стає все більш актуальною. Це дозволяє додавати унікальні можливості, такі як інтеграція для автоматичного модераторства, або персоналізовані рекомендації контенту.

Таким чином, розробка програмних методів мультимедійного бота для Discord є важливим напрямом у сфері автоматизації комунікацій. Інвестування інноваційні рішення дозволяє спільнотам автоматизувати управління контентом та підвищити рівень комфорту роботи з платформою. З урахуванням постійного розвитку технологій та зростаючих вимог користувачів, розробникам необхідно слідкувати за новітніми тенденціями та інтегрувати сучасні підходи до розробки таких систем.

1.2 Порівняльний аналіз музичних ботів

Зі зростанням популярності Discord, як платформи для комунікації значно збільшилася потреба в автоматизованих мультимедійних рішеннях. Боти для роботи з аудіо та відеоконтентом відіграють важливу роль у забезпеченні користувацького досвіду. Вони дозволяють користувачам керувати потоковим відтворенням музики, модерувати голосові чати та автоматизувати взаємодію з мультимедійним контентом.

На ринку існує декілька популярних підходів, які вже зарекомендували себе серед користувачів Discord:

- Rythm;
- Groovy;
- MEE6;
- Hydra;
- FredBoat.

Rythm [8] – це один із найпопулярніших музичних ботів для Discord, який використовувався мільйонами користувачів до моменту свого закриття. Його ключовою перевагою була стабільність у відтворенні аудіо та підтримка популярних платформ, таких як YouTube. Інтерфейс бота був інтуїтивно зрозумілим, що робило його доступним навіть для новачків. Проте він не мав функцій модерації та кастомізації, що обмежувало його функціональність у більших спільнотах (див. рисунок 1.1).

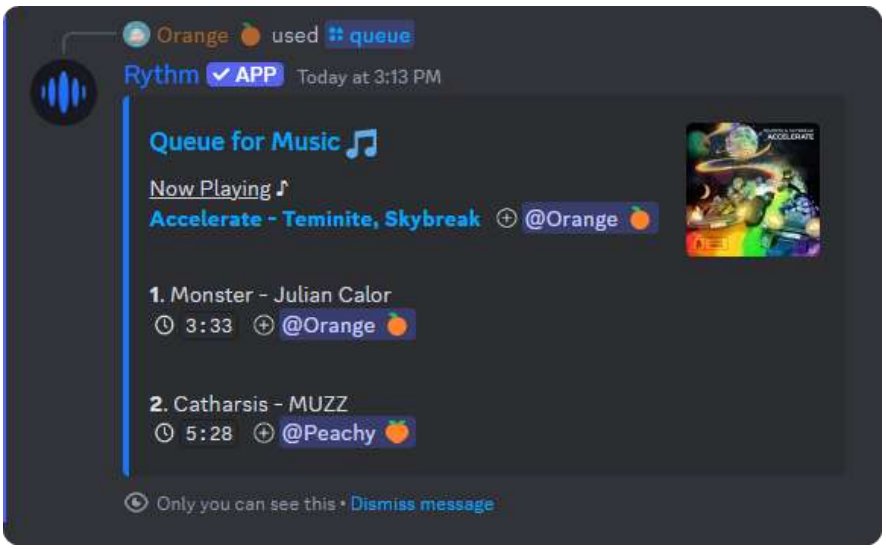

Рисунок 1.1 – Приклад роботи бота Rythm

Groovy (див. рисунок 1.2) музичний бот із можливістю налаштування швидкості відтворення, що вигідно відрізняло його від багатьох конкурентів. Він підтримував базові команди керування музикою та легко додавався на сервер. Через зручний інтерфейс користувачі могли швидко створювати плейлисти та черги треків. Незважаючи на обмежені можливості кастомізації, бот був зручним для щоденного використання [9]. Як і Rythm, Groovy припинив свою роботу через юридичні обмеження з боку музичних сервісів.

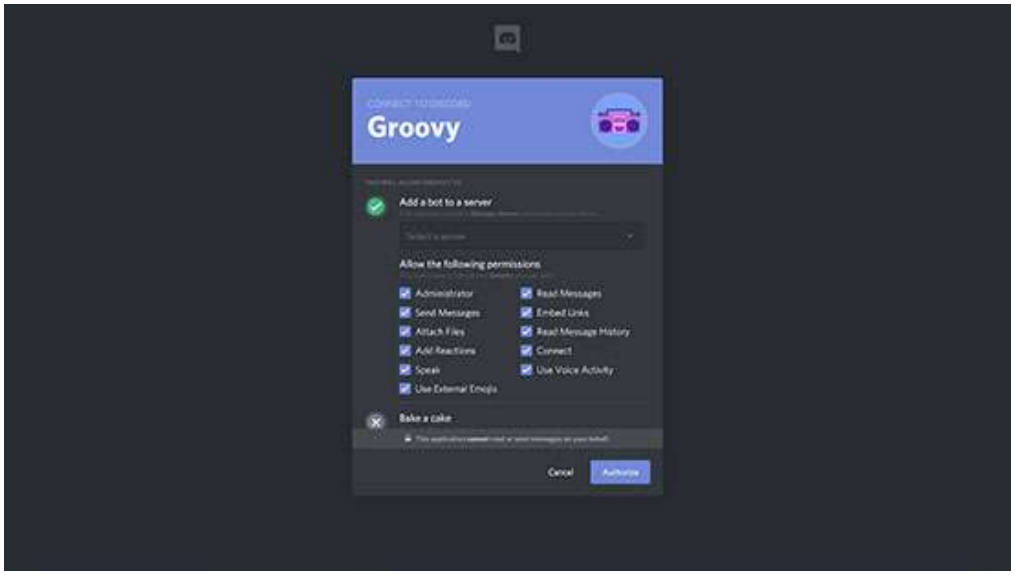

Рисунок 1.2 – Приклад роботи бота Groovy

MEE6 [10] – універсальний бот, який поєднує в собі функції модерації, ролей, сповіщень і медіа. Він має власну веб-панель для налаштувань, де можна створювати складні сценарії автоматизації. Бот підтримує кастомні команди, що дозволяє підлаштувати його під потреби конкретної спільноти. Він має базову підтримку музичних функцій, що робить його багатофункціональним рішенням. MEE6 активно оновлюється та залишається одним із найбільш використовуваних ботів на Discord (див. рисунок 1.3).

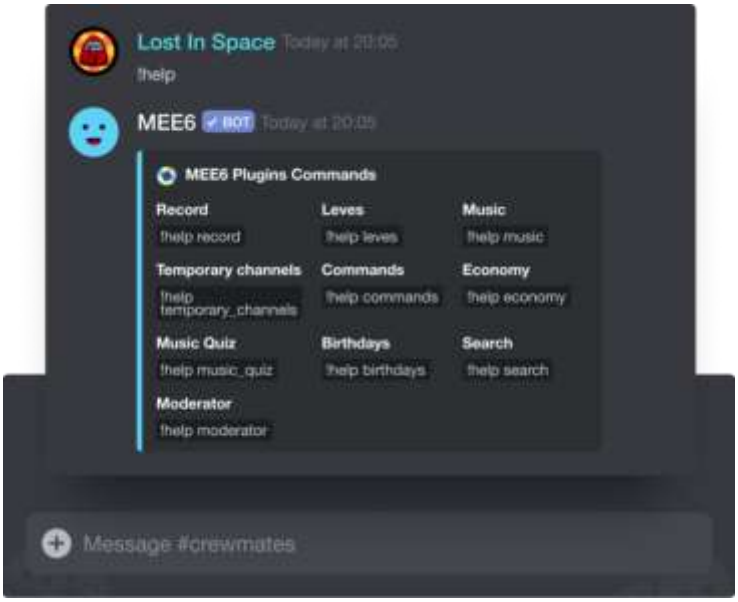

Рисунок 1.3 – Приклад роботи бота MEE6

Hydra [11] – це один із найпопулярніших сучасних музичних ботів з широкими можливостями керування аудіо. Він підтримує потокове відтворення з YouTube, Spotify, Deezer та SoundCloud, а також має графічний інтерфейс керування прямо у чаті. Бот підтримує кастомні команди, регулювання гучності, швидкості та навіть фільтри звуку. Hydra дозволяє користувачам переглядати тексти пісень у реальному часі, що додає інтерактивності. Завдяки стабільній роботі навіть при великому навантаженні, цей бот став вибором багатьох спільнот. Його можна легко локалізувати, що робить його зручним у міжнародному середовищі. На рисунку 1.4. продемонстровано приклад роботи бота.

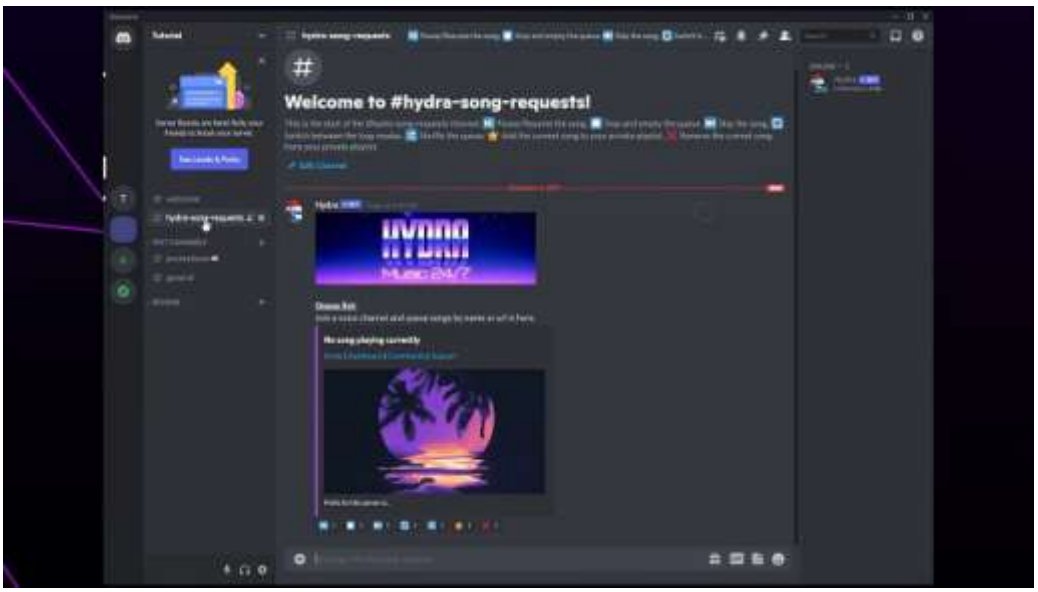

Рисунок 1.4 – Приклад роботи бота Hydra

FredBoat (див. рисунок 1.5) – це простий у використанні музичний бот із базовим функціоналом. Він підтримує додавання пісень у чергу, відтворення зі сторонніх платформ та автоматичне перемикання треків. Його головна перевага є стабільність та мінімалізм, що робить його чудовим вибором для невеликих серверів. Бот не має розширених функцій, таких як модерація. Однак він не потребує складної конфігурації [12].

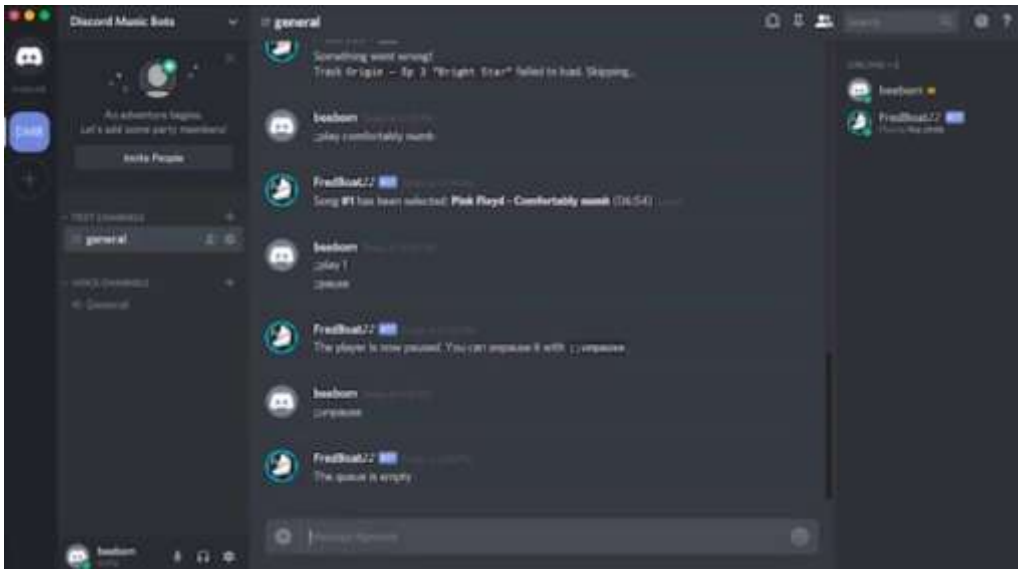

Рисунок 1.5 – Приклад роботи бота FredBoat

Для порівняння функціональних можливостей розглянутих аналогів та власної розробки, було сформовано таблицю 1.1.

Таблиця 1.1 – Порівняння мультимедійних ботів

Функція	Rythm	Groovy	MEE6	Hydra	FredBoat	Власна розробка
Підтримка музичних платформ	1	1	0	1	0	1
Регулювання швидкості відтворення	0	1	0	1	0	0
Автоматична модерація	0	0	1	0	1	1
Кастомні команди	0	0	1	1	1	1
Інтеграція з ШІ	0	0	0	0	0	1
Підтримка відеоконтенту	0	0	0	1	0	1
Сумарний коефіцієнт	1	2	2	4	2	5

Згідно з проведеним аналізом, власна розробка мультимедійного бота має значні переваги над існуючими рішеннями. Вона поєднує найкращі функціональні можливості інших ботів, додаючи інтеграцію зі штучним інтелектом, підтримку відеоконтенту та розширене керування медіа.

Таким чином, розробка унікального мультимедійного бота для Discord є перспективою, яка забезпечить вищу функціональність та зручність у порівняня з аналогами.

1.3 Аналіз напрямків удосконалення програмного забезпечення мультимедійних ботів для Discord

Аналіз напрямків удосконалення програмного забезпечення

мультимедійних Discord ботів полягає у виявленні недоліків і обмежень існуючих варіантів, визначенні можливостей розширення функціоналу та покращення зручності взаємодії користувача з системою. Такі напрямки охоплюють як технічні аспекти, так і функціональні розширення можливостей керування мультимедіа, автоматизація модерації, підтримка інтерактивних функцій для спільноти. Особливу увагу слід приділити реалізації інструментів, які забезпечують ефективну взаємодію користувачів із сервером, зокрема через голосові та текстові канали [4].

Однією з ключових функцій таких ботів є відтворення аудіоконтенту у голосових каналах. Сучасні боти дозволяють створювати чергу композицій, регулювати гучність, змінювати порядок треків та призупиняти відтворення. Проте часто зустрічаються обмеження у підтримці одночасного керування декількома голосовими каналами, що знижує зручність для великих серверів. Перспективним напрямком розвитку є модульна система керування чергами відтворення, яка дозволятиме кожному голосовому каналу працювати автономно з індивідуальним списком медіа та конфігурацією.

Ще одним важливим аспектом є точність та швидкість пошуку медіаконтенту. Існуючі боти часто покладаються на пошук через сторонні сервіси, однак результати можуть бути нерелевантними або дубльованими. Вдосконалення можливе через впровадження інтелектуального фільтрування пошукових результатів за релевантністю, жанром, тривалістю або популярністю, що дозволить покращити користувацький досвід.

Іншою важливою функцією є метод репортів, який дозволяє повідомляти про порушення правил у чаті або поведінку інших учасників. Сучасна реалізація зазвичай обмежується створенням повідомлення у визначеному каналі. Подальше вдосконалення можливе шляхом впровадження категоризації скарг, автоматичної пріоритизації та логування, що допоможе адміністраторам швидко реагувати на інциденти.

Також варто розглянути розширення системи керування правами доступу, яка дозволяє більш точно налаштовувати, хто саме має змогу керувати ботом,

використовувати окремі команди або функціонал. Перспективним є створення інтерфейсу керування правами з прив'язкою до ролей Discord та журналом дій користувачів.

Отже, функціональність мультимедійних Discord ботів має значний потенціал для розвитку. Удосконалення можуть бути спрямовані як на покращення базового медіафункціоналу, так і на розширення взаємодії користувачів через інструменти анонімного спілкування, автоматизованого модераторського контролю та розширеного управління правами доступу.

1.4 Аналіз методів розв’язання поставленої задачі

Після аналізу існуючих мультимедійних ботів для Discord каналів були визначені ключові Розробка програмного забезпечення мультимедійного бота для Discord включає використання низки підходів, технологій і бібліотек, які забезпечують ефективну інтеграцію з API Discord, стабільне відтворення мультимедіа та зручне управління ботом через текстові та голосові канали. Основними методами реалізації подібних систем є використання спеціалізованих бібліотек для Discord, асинхронної обробки запитів, а також застосування модульного підходу до структури проєкту. Розглянемо ключові підходи до розв’язання поставленої задачі.

Найпоширенішим середовищем для створення Discord ботів є мова Python у зв’язці з бібліотекою discord.py, яка забезпечує зручну інтеграцію з Discord API, підтримку подій, команд та роботи з каналами. Також актуальними є JavaScript з використанням бібліотеки discord.js, а для більш продуктивних методів є мови Go та Rust. У власному проєкті для зручності, простоти масштабування та активної спільноти було обрано мову Python з асинхронною обробкою подій [13].

Одним із ключових компонентів є обробка та трансляція аудіоконтенту у голосових каналах Discord. Для цього можуть використовуватись інструменти як FFmpeg для обробки аудіопотоків та youtube-dl, або його форки yt-dlp для завантаження медіа. Передача аудіо реалізується через discord.VoiceClient.

Покращення можливе за рахунок буферизації, удосконалення, а також переходу на WebRTC базовані рішення [14].

Ефективна архітектура бота охоплює розділення функцій на окремі модулі: команди, події, обробники помилок. Це дозволяє легко додавати новий функціонал до бота. Такий підхід сприяє масштабованості, повторному використанню коду та зручності тестування.

Реалізація логіки керування ботом здійснюється через систему текстових команд з префіксом або інтеграцію slash-команд Discord Interaction API. Останній підхід є більш сучасним, забезпечує автодоповнення та кращу інтеграцію з Discord UI. Для підвищення зручності користування у проєкті реалізовано підтримку обох варіантів.

Реалізація анонімного спілкування реалізується через окремий текстовий канал, у який користувачі надсилають повідомлення через спеціальну команду. Бот пересилає їх у канал без вказування автора. Додатково реалізована функція блокування користувачів у разі зловживань та автоматичне логування повідомлень для модерації.

Для забезпечення модерації користувачів реалізована функція надсилання скарг до окремого каналу. Користувачі можуть викликати команду !report, обрати порушника та коротко описати ситуацію. Повідомлення формуються у вигляді вбудованих повідомлень із таймштампами, що дозволяє адміністрації швидко реагувати.

Для зберігання конфігурацій бота, логів, репортів можуть використовуватись різні методи: від локальних JSON-файлів до повноцінних баз даних SQLite, PostgreSQL. У проєкті застосовується структура на основі JSON-файлів з можливістю подальшого переходу на реляційну базу [15].

Для забезпечення постійної доступності бота можуть використовуватись хмарні платформи, де забезпечено автоматичне розгортання оновлень та моніторинг працездатності. У рамках даного проєкту було реалізовано автоматичне перезавантаження при помилках і логування [16].

Таким чином, для реалізації мультимедійного Discord бота застосовується

низка перевірених методів, які забезпечують не лише базовий функціонал програвання медіа, але й інтеграцію з серверними процесами, зручне адміністрування та взаємодію користувачів через чат. Обрані методи відповідають сучасним вимогам до надійності, масштабованості та зручності використання програмного забезпечення в контексті Discord-серверів.

1.5 Постановка задачі

Після аналізу існуючих мультимедійних ботів для Discord каналів були визначені ключові завдання для розробки власного програмного забезпечення:

– провести аналіз предметної області та існуючих методів для автоматизації керування Discord серверами;

– розробити діаграму діяльності та структуру роботи програмного забезпечення;

– розробити метод модерації контенту з автоматичним виявленням порушень на основі встановлених правил та шаблонів;

– розробити метод обробки репортів користувачів із подальшою класифікацією та передачею на розгляд модератору;

– розробити метод динамічного пошуку та керування відтворенням аудіоконтенту з урахуванням запитів користувача в режимі реального часу;

– розробити програмне забезпечення мультимедійного бота для управління Discord каналом;

– провести тестування програмного забезпечення;

– розробити інструкцію користувача та описати вимоги програмного забезпечення.

Технічне завдання на розробку наведено в додатку А.

1.6 Висновки

У цьому розділі було проаналізовано сучасні мультимедійні боти для Discord, їхні можливості, переваги та недоліки. Найпопулярніші рішення Rythm, Groovy, MEE6, Hydra, FredBoat надають базові функції є відтворення

музики, створення черги треків, керування гучністю. Проте вони мають обмеження: відсутність модерації, інтеграції зі штучним інтелектом і підтримки відеоконтенту.

Порівняльний аналіз показав, що власна розробка може перевершити ці аналоги, поєднавши їхні сильні сторони і додавши нові функції: анонімний чат, систему скарг, інтелектуальну модерацію та підтримку відео.

У проєкті запропоновано використовувати Python та бібліотеку discord.py, а також інструменти для роботи з медіа FFmpeg, yt-dlp.

Отже, розробка власного мультимедійного Discord бота є перспективний напрям, який дозволяє створити зручне та ефективне рішення для автоматизації роботи спільнот.

2 РОЗРОБКА МЕТОДІВ РОБОТИ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ДЛЯ УПРАВЛІННЯ DISCORD КАНАЛОМ

2.1 Розробка діаграми діяльності Discord бота

Діаграми діяльності Discord бота відображає основні етапи обробки користувацьких команд та виконання відповідних дій. Послідовність включає: прийом запиту, перевірку дозволів, пошук і обробку мультимедійного контенту, а також подальше відтворення аудіо та реагування на додаткові команди [17].

При створені програмного забезпечення для мультимедійного бота, призначеного для управління Discord каналом, особливу увагу було приділено моделюванню взаємодії користувача з ботом. З цією метою розроблено діаграму діяльності, яка демонструє основні сценарії використання бота, що показані на рисунку 2.1 [5].

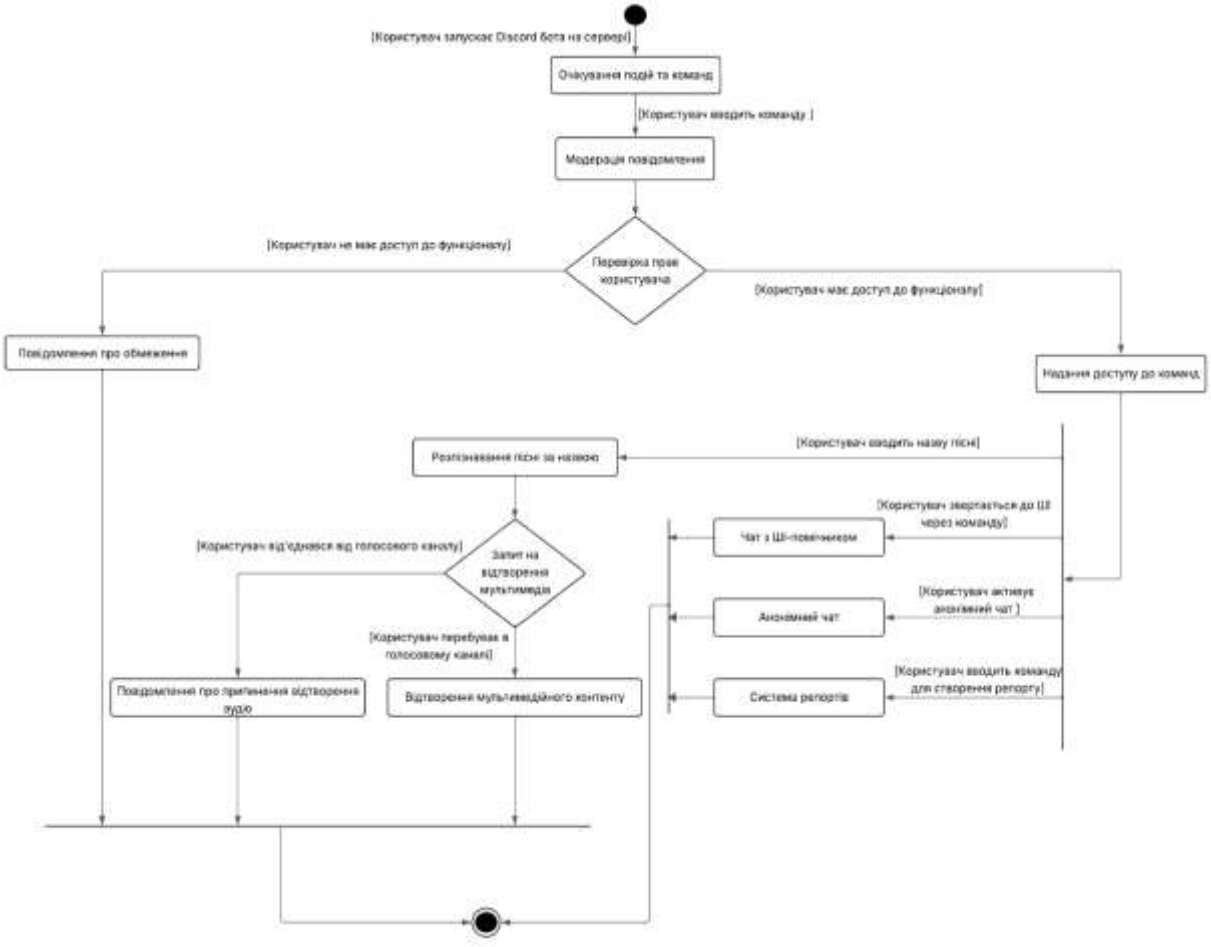

Рисунок 2.1 – Діаграма діяльності Discord бота

На діаграмі відображено повний цикл взаємодії користувача з Discord ботом. Процес починається із запуску бота на сервері та очікування на команди. Далі бот перевіряє права користувача на доступ до певного функціоналу. Якщо доступ надано, користувач може виконати низку дій: ввести назву пісні для її відтворення, звернутись до окремих методів бота, таких як чат, анонімний чат або система репортів. Після введення музичної команди бот виконує розпізнавання пісні, перевіряє, чи користувач перебуває в голосовому каналі. Якщо так, то запускає відтворення мультимедійного контенту. Якщо ж користувач залишає голосовий канал, відтворення припиняється автоматично.

Таким чином, діаграма діяльності охоплює основні шляхи користувацької активності та дозволяє ефективно реалізувати логіку роботи мультимедійного Discord бота з урахуванням прав доступу, дій користувача та реакції бота на події.

2.2 Розробка структури програмного забезпечення

Розробка структури програмного забезпечення мультимедійного бота містить планування та організацію всіх функціональних компонентів і їх взаємозв’язків для забезпечення стабільної та ефективної роботи в межах Discord. У випадку з ботом для управління Discord каналом, структура повинна включати всі необхідні методи, які дозволяють автоматизувати взаємодію з користувачами, керування медіаконтентом, виконання команд, обробку повідомлень і інтеграцію зі сторонніми сервісами.

Це дає змогу забезпечити зручне управління каналом, покращити взаємодію учасників спільноти та розширити функціональність Discord сервера. Завдяки чітко спроєктованій структурі бот може ефективно масштабуватись, доповнюватись новими функціями та адаптуватися до потреб конкретної спільноти [18].

Аналіз варіантів використання програмного забезпечення мультимедійного бота для управління Discord каналом показав наступні зв’язки, що показано на рисунку 2.2.

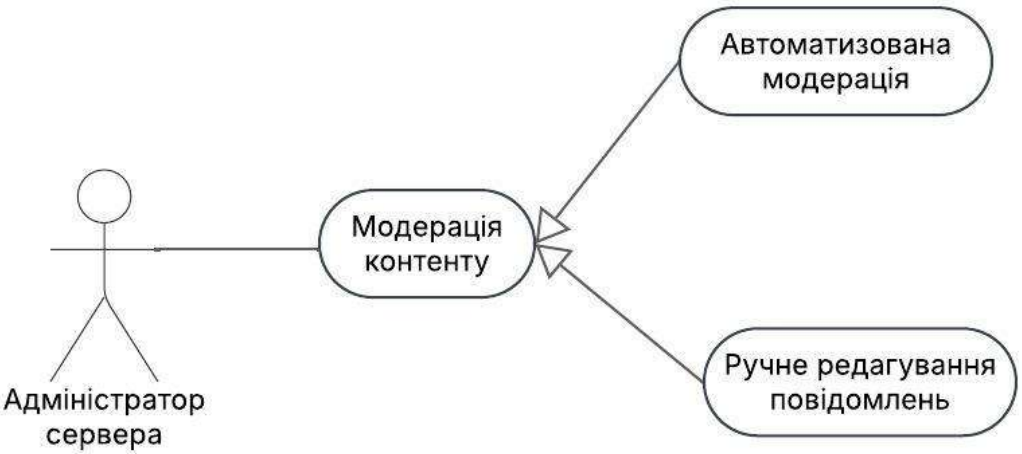

Рисунок 2.2 – Узагальнення варіантів використання щодо модерації контенту в Discord каналі

Варіанти використання «Модерація контенту» і «Ручне редагування повідомлень» мають схожий функціонал і відсутність принципових відмінностей, тому було прийнято рішення запровадити новий варіант використання «Автоматизована модерація». При цьому прецедент «Модерація контенту» залишається основним, узагальнюючи як нововведений прецедент, так і прецедент «Ручне редагування повідомлень». Було здійснено аналіз поточної структури прецедентів із метою оптимізації їхньої взаємодії. Виявлено доцільності функціоналу для зменшення дублювання.

Параметр «Автоматична модерація мультимедійного контенту» базується на основному прецеденті «Модерація контенту», але містить більш складну логіку обробки, зокрема перевірку зображень та відео.

У зв’язку з цим, прийнято рішення зв’язати зазначені прецеденти відношенням розширення. Крім того, прецедент «Автоматична модерація мультимедійного контенту» використовує логіку прецеденту «Перевірка повідомлень на відповідність правилам». У зв’язку з цим, прийнято рішення зв’язати зазначені прецеденти відношенням включення.

Аналіз зв’язків розширення і включення для варіантів використання управління контентом у Discord каналі відображено на рис. 2.3.

Рисунок 2.3 – Аналіз зв’язків розширення і включення для модерації контенту в Discord каналі

На основі аналізу, проведеного у розділі «Структурування варіантів використання», було прийнято рішення виключити два варіанти використання: «Модерація текстового контенту» та «Модерація мультимедійного контенту», оскільки їх функціональність значною мірою перетинається. Обидва варіанти об’єднано в єдиний варіант використання «Автоматизована модерація».

Отриманий список варіантів використання відображено у табл. 2.1.

Таблиця 2.1 – Реєстр варіантів використання

Код	Основний актор	Найменування	Формулювання
A1	Адміністратор серверу	Модерація контенту	Виконує перевірку та обробку текстових і мультимедійних повідомлень згідно з налаштованими правилами.
A2	Адміністратор серверу	Аналіз трендів порушень	Адміністратор отримує статистичні дані щодо найчастіших порушень та ефективності модерації.
M1	Модератор	Перевірка повідомлення на відповідність правилам	Дає змогу модератору аналізувати, коригувати або видаляти повідомлення, що можуть порушувати правила.
K1	Користувач	Надсилання повідомлення	Користувач створює та надсилає текстове або мультимедійне повідомлення у канал.

Продовження таблиці 2.1

Код	Основний актор	Найменування	Формулювання
В1	Бот	Перевірка повідомлень	Бот аналізує повідомлення та визначає його відповідність встановленим правилам.
В2	Бот	Реагування на порушення	Бот може автоматично видаляти, редагувати або надсилати попередження користувачу.
В3	Бот	Логування модераційних дій	Всі дії модерації фіксуються в журналі для подальшого аналізу.
В4	Бот	Взаємодія з користувачем через повідомлення	Бот надсилає користувачам повідомлення, щодо запиту відтворення аудіо контенту, або іших команд

На основі таблиці 2.1 було більш детально розглянуто можливі сценарії використання програмного забезпечення. Це дозволило конкретизувати функціональні можливості системи та краще зрозуміти потреби користувачів у реальних умовах.

Процес модерація контенту зображено на таблиці 2.2.

Таблиця 2.2 – Модерація контенту

Код	Основний актор	Найменування	Формулювання
А1	Адміністратор серверу	Модерація контенту	Виконує перевірку та обробку текстових і мультимедійних повідомлень згідно з налаштованими правилами.

Головний актор: Адміністратор серверу

Інші учасники: Модератор

Зв’язки з іншими варіантами використання: включає «Перевірка повідомлень», «Реагування на порушення»

Короткий опис: цей варіант використання дозволяє адміністратору налаштовувати правила модерації для текстових та мультимедійних

повідомлень. Бот аналізує повідомлення на відповідність забороненим словам, зображенням або іншим критеріям та здійснює відповідні дії.

Процес аналізу трендів порушень контенту зображено на таблиці 2.3.

Таблиця 2.3 – Аналіз трендів порушень

Код	Основний актор	Найменування	Формулювання
A2	Адміністратор серверу	Аналіз трендів порушень	Адміністратор отримує статистичні дані щодо найчастіших порушень та ефективності модерації.

Головний актор: Адміністратор серверу

Інші учасники: Бот

Зв’язки з іншими варіантами використання: використовує «Логування модераційних дій»

Короткий опис: бот надає адміністраторам аналітичні дані про найчастіші порушення, що дозволяє покращувати правила модерації та автоматизовані фільтри.

Процес перевірка повідомлення на відповідність правилам зображено на таблиці 2.4.

Таблиця 2.4 – Перевірка повідомлення на відповідність правилам

Код	Основний актор	Найменування	Формулювання
M1	Модератор	Перевірка повідомлення на відповідність правилам	Дає змогу модератору аналізувати, коригувати або видаляти повідомлення, що можуть порушувати правила.

Головний актор: Модератор

Інші учасники: Бот

Зв’язки з іншими варіантами використання: розширює «Модерація контенту»

Короткий опис: дозволяє модераторам переглядати, аналізувати та змінювати повідомлення, які потенційно порушують правила спільноти, якщо автоматична модерація не спрацювала або потребує уточнення.

Процес надсилання повідомлення зображено на таблиці 2.5.

Таблиця 2.5 – Надсилання повідомлення

Код	Основний актор	Найменування	Формулювання
K1	Користувач	Надсилання повідомлення	Користувач створює та надсилає текстове або мультимедійне повідомлення у канал.

Головний актор: Користувач

Інші учасники: Бот

Зв’язки з іншими варіантами використання: включає «Перевірка повідомлень»

Короткий опис: користувач створює текстове або мультимедійне повідомлення та надсилає його в канал. Повідомлення перевіряється ботом перед публікацією.

Процес перевірка повідомлень зображено на таблиці 2.6.

Таблиця 2.6 – Перевірка повідомлень

Код	Основний актор	Найменування	Формулювання
B1	Бот	Перевірка повідомлень	Бот аналізує повідомлення та визначає його відповідність встановленим правилам.

Головний актор: Бот

Інші учасники: Адміністратор серверу

Зв’язки з іншими варіантами використання: розширює «Надсилання повідомлення»

Короткий опис: бот здійснює аналіз повідомлення на відповідність встановленим правилам, таким як заборонені слова, образливий контент або несанкціоновані медіафайли.

Процес реагування на порушення зображено на таблиці 2.7.

Таблиця 2.7 – Реагування на порушення

Код	Основний актор	Найменування	Формулювання
B2	Бот	Реагування на порушення	Бот може автоматично видаляти, редагувати або надсилати попередження користувачу.

Головний актор: Бот

Інші учасники: Модератор

Зв’язки з іншими варіантами використання: включає «Перевірка повідомлень»

Короткий опис: якщо повідомлення не відповідає правилам, бот може його автоматично видалити, замінити або надіслати попередження користувачу.

Процес логування модераційних дій зображено на таблиці 2.8.

Таблиця 2.8 – Логування модераційних дій

Код	Основний актор	Найменування	Формулювання
B3	Бот	Логування модераційних дій	Всі дії модерації фіксуються в журналі для подальшого аналізу.

Головний актор: Бот

Інші учасники: Адміністратор серверу

Зв’язки з іншими варіантами використання: розширює «Модерація контенту»

Короткий опис: бот записує всі модераційні дії у лог-файл або базу даних, щоб адміністратор або модератор могли переглядати історію модерації.

Процес Взаємодія з користувачем через повідомлення зображено на таблиці 2.9.

Таблиця 2.9 – Взаємодія з користувачем через повідомлення

Код	Основний актор	Найменування	Формулювання
В4	Бот	Взаємодія з користувачем через повідомлення	Бот надсилає користувачам повідомлення, щодо запиту відтворення аудіо контенту, або іших команд

Головний актор: Бот

Інші учасники: Користувач

Зв’язки з іншими варіантами використання: розширює «Надсилання повідомлення»

Короткий опис: бот надсилає користувачам повідомлення у відповідь на запити на відтворення аудіо контенту та виконання інших команд, пояснюючи результат обробки запиту, надаючи інструкції для подальших дій.

На діаграмі прецедентів, зображеній на рисунку 2.4, представлено взаємодію між основними учасниками системи модерації контенту в рамках Discord сервера.

Бот здійснює модерацію контенту, що включає перевірку повідомлень на відповідність правилам, надсилання повідомлень, взаємодію з користувачем через повідомлення та реагування на порушення. Крім того, реалізовано логування модераційних дій та аналіз трендів порушень. Автоматична модерація мультимедійного контенту є окремим випадком автоматизованої модерації, яка

розширює загальну функцію модерації контенту. У разі потреби також можливе ручне редагування повідомлень.

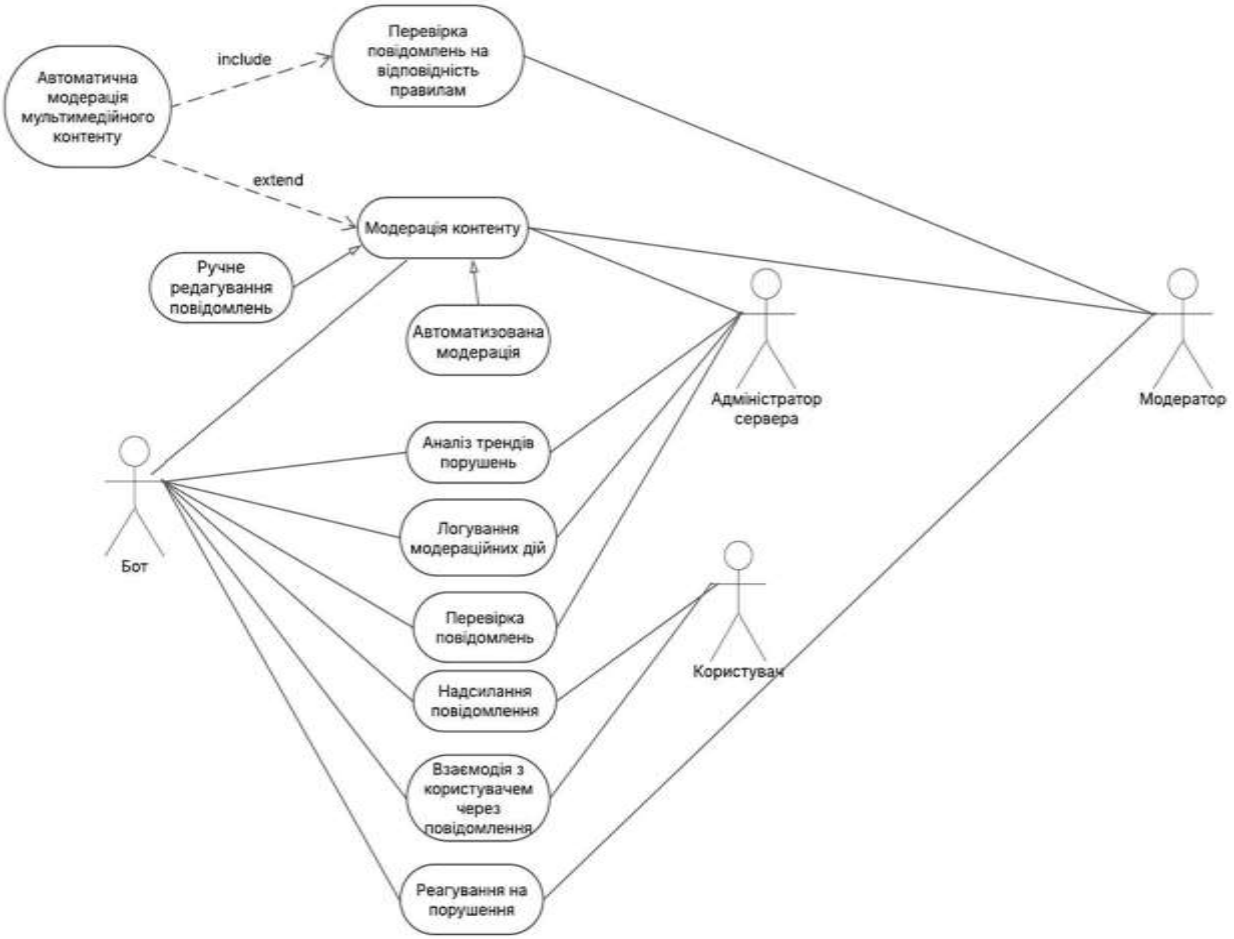

Рисунок 2.4 – Діаграма прецедентів програмного забезпечення

Діаграма прецедентів відображає основні функції програмного забезпечення модерації контенту за участі бота, користувачів та адміністраторів. Вона демонструє взаємодію між учасниками та охоплює як автоматизовані, так і ручні процеси модерації, що забезпечує управління повідомленнями бота.

2.3 Розробка методу репортів

Розроблено метод репортів для повідомлення про порушення правил чи помилок в чаті, що реалізується через спеціальний репорт-канал. Інноваційність підходу полягає в автоматичній обробці звернень, фільтрації та можливості

інтеграції з системою логування дій користувачів, що полегшує роботу модераторів та пришвидшує реагування на інциденти [5].

Таким чином, блок-схема алгоритму репортів показує послідовність дій від моменту перевірки прав користувача до фінального розгляду скарги та винесення рішення модерацією (див. рисунок 2.5).

Рисунок 2.5 – Блок-схема алгоритму репортів

Створення репорту починається з перевірки прав користувача на подання скарги. Бот визначає, чи має користувач дозвіл на створення репорту. Якщо відповідних прав немає, процес завершується без подальших дій. У випадку, якщо користувач має необхідні повноваження, відбувається перехід до наступного етапу. Користувач формує репорт, вказуючи суть скарги, причину та докази порушення. Після цього репорт надсилається для подальшої обробки.

Після збереження, скарга передається на перевірку модераторам. Бот визначає, чи підтвердила модерація поданий репорт. Якщо скаргу не схвалено, то процес завершено. У разі підтвердження порушення, бот переходить до етапу винесення покарання.

2.4 Розробка методу динамічного пошуку та керування відтворенням аудіоконтенту

Розроблено новий метод динамічного пошуку та керування відтворенням аудіоконтенту, який поєднує API-запити до зовнішніх мультимедійних платформ із фільтрацією результатів за ключовими словами та релевантністю. Він також адаптивну логіку взаємодії з користувачами в голосових каналах Discord. Метод враховує контекст попередніх запитів для точнішого пошуку й інтерпретації команд, що підвищує зручність використання бота, зменшує кількість помилок і забезпечує швидке та релевантне відтворення медіа контенту [5].

Метод починається з етапу очікування введення команди від користувача. Бот перевіряє, чи перебуває користувач у голосовому каналі. Якщо ні, то надсилається сповіщення про неможливість відтворення аудіо без присутності користувача в голосовому каналі і бот повертається до режиму очікування. Якщо користувач перебуває в голосовому каналі, бот перевіряє, чи було використано команду відтворення аудіо. Якщо команда не була введена, то бот автоматично запускає відтворення випадкового мультимедійного контенту з доступного списку. Якщо команда була введена, бот перевіряє її на коретне введення. Неправильно введена команда ігнорується, при цьому користувачу надсилається відповідне повідомлення про помилку. Після чого бот повертається до очікування нових команд.

На рисунку 2.6 зображено блок-схему алгоритму реалізації динамічного пошуку та керування відтворенням аудіо контенту. Метод починається з етапу очікування введення команди від користувача. Бот перевіряє, чи перебуває користувач у голосовому каналі.

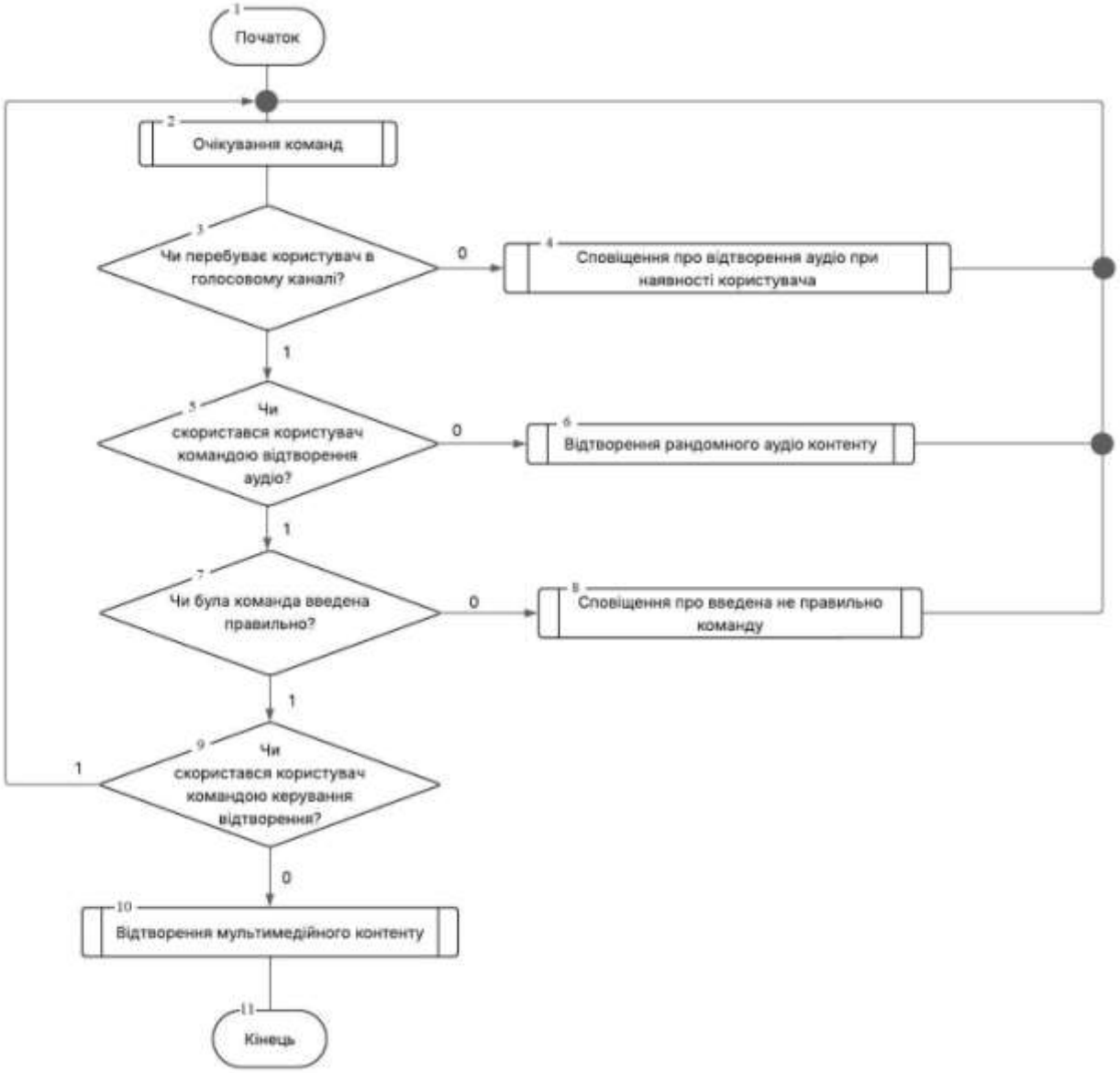

Рисунок 2.6 – Блок-схема алгоритму реалізації динамічного пошуку та керування відтворенням аудіоконтенту

Якщо ні, то надсилається сповіщення про неможливість відтворення аудіо без присутності користувача в голосовому каналі і бот повертається до режиму очікування. Якщо користувач перебуває в голосовому каналі, бот перевіряє, чи було використано команду відтворення аудіо. Якщо команда не була введена, то бот автоматично запускає відтворення випадкового мультимедійного контенту з доступного списку. Якщо команда була введена, бот перевіряє її на коретне введення. Неправильно введена команда ігнорується, при цьому

користувачу надсилається відповідне повідомлення про помилку. Після чого бот повертається до очікування нових команд.

Правильно введена команда обробляється далі, бот перевіряє, чи містить вона команду паузи, перемотки, або зміни треку. Якщо така команда присутня, то відбувається кероване відтворення мультимедійного контенту згідно з командами користувача. Робота завершується після повного відтворення контенту, або при виникненні помилок, які зупиняють виконання методу. Таким чином, дана блок-схема відображає підхід до управління аудіоконтентом, дозволяючи забезпечити інтерактивну взаємодію з користувачем.

Функціонал методу містить інтеграцію з сервісом «YouTube» для завантаження й обробки аудіопотоків у реальному часі. Для цього використовується бібліотека «yt_dlp», як ефективне рішення для отримання метаданих і прямого посилання на аудіо. Відтворення здійснюється за допомогою «discord.FFmpegOpusAudio», що забезпечує підтримку потокового аудіо у форматі «Opus» [19].

2.5 Розробка методу модерації контенту

Удосконалено метод модерації контенту, який забезпечує автоматичне виявлення та блокування шкідливих повідомлень у текстових каналах Discord. Метод базується на поєднанні словникових фільтрів, алгоритмів обробки природної мови та адаптивних шаблонів поведінки користувачів. Інноваційність підходу полягає в оцінці повідомлень із врахуванням історії діалогу, що дозволяє зменшити кількість хибних спрацювань і підвищити продуктивність модерації без втручання людини [5].

Проаналізовано метод модерації контенту, реалізований у вигляді блок-схеми алгоритму, що зображена на рисунку 2.7. Цей метод призначений для обробки повідомлень та подій з метою забезпечення безпечного та контрольованого інформаційного простору. Робота бота починається з очікування нових повідомлень, або подій.

Рисунок 2.7 – Блок-схема алгоритму модерації контенту

Після чого бот виконує перевірку на наявність заборонених слів. Якщо повідомлення не містить таких слів, воно допускається до подальшого оброблення. У разі виявлення заборонених слів бот активує механізм фільтрації небажаних ключових слів, що дозволяє попередити поширення небажаного, або шкідливого контенту. Перевіряється, чи є повідомлення спамом. Якщо повідомлення не вважається спамом, воно зберігається в системі без змін. У

випадку виявлення спаму застосовуються обмеження щодо кількості запитів, з подальшим видаленням повідомлень, які порушують правила. Усі такі випадки автоматично фіксуються в журналі аудиту, що забезпечує контроль за діяльністю користувачів.

Алгоритм підтримує як автоматичну, так і частково ручну модерацію, дозволяючи масштабувати процес на великі об’єми даних. Такий підхід є ефективним інструментом у боротьбі з порушеннями правил платформи, сприяє підвищенню довіри користувачів та підтримці якісного середовища для спілкування.

2.6 Висновки

На основі зібраної інформації побудовано UML-діаграму діяльності, яка відображає обробку команд, перевірку прав доступу, ініціацію мультимедіа та роботу з репортами. Створена структура забезпечує інтеграцію основних функцій бота, зокрема автоматизованої модерації та взаємодії з користувачем. Структура забезпечує інтеграцію функцій у межах програмного забезпечення мультимедійного бота з урахуванням потреб користувачів, адміністраторів та модераторів Discord каналу.

Розроблено методи для управління Discord каналом за допомогою мультимедійного бота. Розроблений метод репортів дозволяє автоматизувати процес фіксації та обробки скарг, що суттєво знижує навантаження на модераторів і підвищує оперативність реагування на інциденти. Метод динамічного пошуку та керування аудіоконтентом забезпечує інтеграцію з зовнішніми мультимедійними сервісами та дозволяє реалізувати відтворення, яка адаптується до дій користувача. Метод модерації контенту виконує фільтрування шкідливих повідомлень та обмежень активності.

Усі розглянуті методи забезпечують комплексний підхід до управління сервером Discord та комфортного середовища для користувачів.

3 РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ МУЛЬТИМЕДІЙНОГО БОТА ДЛЯ УПРАВЛІННЯ DISCORD КАНАЛОМ

3.1 Варіантний аналіз і обґрунтування вибору засобів для реалізації програмного забезпечення

Інтегроване середовище розробки – це програмний інструмент, який поєднує в собі основні засоби, необхідні для створення програмного забезпечення. До таких засобів належать редактор коду, компілятор або інтерпретатор, налагоджувач, візуальні інструменти для збирання проєкту та тестування, а також засоби керування залежностями і контролю версій. Використання IDE значно підвищує продуктивність роботи розробника завдяки централізованому доступу до всіх необхідних інструментів [20].

Visual Studio [21] – потужне середовище розробки від Microsoft. Воно підтримує велику кількість мов програмування, зокрема C#, C++, Python, JavaScript. Visual Studio має потужний редактор, інтеграцію з Git та GitHub, візуальні засоби налагодження та відлагодження, а також розвинуту екосистему плагінів. Це середовище добре підходить для створення десктопних та веб-додатків, зокрема ботів, включно з Discord ботами на Python.

NetBeans [22] – IDE з відкритим вихідним кодом, яка підтримує Java, PHP, JavaScript та деякі інші мови. Хоча вона пропонує базовий функціонал для розробки, її інтерфейс має менше функціоналу порівняно з іншими сучасними IDE, а підтримка плагінів обмежена. NetBeans рідко використовується в проектах, пов’язаних із Discord ботами, через обмежену інтеграцію з популярними фреймворками.

Xcode [23] – офіційне середовище розробки для macOS та iOS від Apple. Хоча воно є чудовим для розробки під iOS, його використання в контексті Discord ботів обмежене через орієнтацію на Swift та Objective-C. Xcode не призначене для кросплатформеної бекенд-розробки.

Порівняльний аналіз на основі можливостей згаданих середовищ розробки, наведено в таблиці 3.1.

Таблиця 3.1 – Порівняльний аналіз середовищ розробки

Функція	Visual Studio	NetBeans	Xcode
Підтримка Python	1	0	0
Вбудований емулятор тестування	1	0	1
Інтеграція з Git	1	1	0
Інструменти профілювання	1	0	1
Підтримка бот фреймворків	1	0	0
Екосистема плагінів	1	0	0
Сумарний коефіцієнт	6	1	2

Для розробки мультимедійного бота для Discord найкраще підходить Visual Studio, оскільки воно показало найбільший сумарний коефіцієнт. А саме: підтримку мов програмування, інтеграцію з системами контролю версій, функціонал для налагодження та підтримує популярні бібліотеки для роботи з Discord. Натомість середовища на зразок NetBeans та Xcode менш дієві для такої задачі. Недоліком є обмежена підтримка відповідних технологій.

Для розробки та реалізації програмного забезпечення мультимедійного бота для управління Discord каналом важливим завданням є вибір оптимальної мови програмування, яка забезпечить стабільну взаємодію з Discord API, зручну обробку медіаконтенту, легкість у розробці та підтримці. Для аналізу було розглянуто три популярні мови програмування: Python, JavaScript та C++.

Python [24] – це високорівнева інтерпретована мова програмування з простим і зрозумілим синтаксисом, що робить її ідеальною для швидкої розробки та прототипування. Завдяки бібліотеці discord.py, яка забезпечує повну підтримку API Discord, Python активно використовується для створення Discord-ботів з текстовою, голосовою та мультимедійною взаємодією. Крім того, мова має велику кількість сторонніх бібліотек для роботи з медіа, зокрема

youtube_dl, FFmpeg, PyNaCl тощо. До переваг Python належать швидкість розробки, велика екосистема готових підходів для Discord, активна спільнота та доступність навчальних матеріалів. Водночас, його недоліками є нижча продуктивність у порівнянні з компільованими мовами та обмежена підтримка мобільних і вебфреймворків.

JavaScript у середовищі виконання Node.js також широко використовується для створення Discord ботів, переважно завдяки бібліотеці discord.js. Node.js забезпечує високу продуктивність завдяки подієвій моделі обробки, що робить його ефективним для масштабованих проєктів, які потребують швидкої реакції на події або інтеграції з вебсервісами. Серед переваг є висока швидкодія, широка підтримка вебтехнологій, велика кількість npm-пакетів, а також зручність поєднання фронтенду й бекенду. Недоліками є складніша структура коду у великих системах та вища крива навчання для асинхронного програмування [25].

C++ [26] – це компільована мова з високою продуктивністю, яка зазвичай застосовується у системному програмуванні або в задачах, де потрібна максимальна продуктивність. Для створення Discord-ботів використання C++ є менш поширеним, через обмежену підтримку, складнішу реалізацію та меншу кількість прикладів і документації. Проте переваги C++ полягають у високій продуктивності, статичній типізації та можливості точного контролю над ресур.

Для кращого порівняння було прийнято рішення порівняльний аналіз мов програмування, було сформовано таблицю 3.2.

Таблиця 3.2 – Порівняльний аналіз мов програмування

Функція	Python	C++	JavaScript
Підтримка бібліотек для інтеграції з Discord API	1	0	1
Розширена підтримка AI	1	0	0

Продовження таблиці 3.2

Вбудовані інструменти для тестування та налагодження	1	0	1
Підтримка обробки даних в реальному часі	1	0	1
Підтримка масштабованості для великих проєктів	1	1	1
Інтеграція з хмарними платформами та сервісами	1	0	1
Сумарний коефіцієнт	6	1	5

Після порівняння трьох мов програмування за критеріями зручності розробки, наявності бібліотек, підтримки Discord API та можливостей роботи з мультимедійним контентом, оптимальним вибором для реалізації проєкту є Python.

3.2 Аналіз вхідних та вихідних даних програмного забезпечення

Для реалізації мультимедійного бота, орієнтованого на ефективне управління Discord каналом, будуть використані такі основні технології: мова програмування Python, бібліотека discord.py для взаємодії з Discord API, а також youtube_dl, FFmpeg і PyNaCl для роботи з мультимедійним контентом [27].

Вхідними даними для мультимедійного бота є команди, які користувачі надсилають у текстові канали Discord. Команди мають певну структуру, що зазвичай починається з префікса «!» і містить назву команди та можливі параметри.

Основні приклади вхідних команд [6]:

- !play <url або назва треку> – команда для відтворення музики з вказаного джерела;
- !pause, !resume, !skip – в команди керування поточним відтворенням;

- !queue – відображення черги треків;
- !leave – вихід бота з голосового каналу;
- !help – виведення довідки про доступні команди.

Бот обробляє команди, отримані через Discord API, і відповідно виконує необхідні дії. Для обробки аудіо та відеоконтенту використовуються такі інструменти, як:

- youtube_dl та yt-dlp – для завантаження та обробки мультимедійних потоків з онлайн-ресурсів;
- FFmpeg – для конвертації аудіо у формат, що підтримується Discord;
- PyNaCl – для шифрування голосових даних і взаємодії з голосовими каналами Discord.

Кожна команда користувача запускає відповідний обробник у коді бота. Команди можуть мати параметри, які передаються до функцій. Наприклад, посилання на відео, ключові слова для пошуку, або часова мітка для відтворення з певного моменту.

Також враховується ідентифікація користувача, ролі та права доступу на сервері Discord, щоб запобігти використанню певних команд некваліфікованими учасниками. Наприклад, команда !clear для очищення черги може бути обмежена лише адміністраторами або користувачами з відповідними дозволами.

На виході бот формує відповідну реакцію в текстовому або голосовому каналі: надсилає повідомлення про результат виконання команди, додає трек до черги або починає його відтворення. Для кращої взаємодії з користувачем бот може також використовувати інтерактивні елементи, такі як реакції на повідомлення або вбудовані повідомлення.

Таким чином, вхідні дані в системі представлені у вигляді структурованих команд, а вихідними є дії бота в межах каналу Discord. Чіткий синтаксис команд, налаштування ролей та обробка виняткових ситуацій дозволяють створити стабільну й ефективну систему мультимедійної взаємодії у спільноті.

3.3 Розробка функцій надсилання та збереження репортів

У процесі розробки мультимедійного Discord бота важливу роль відіграє реалізація функціоналу, що забезпечує контроль за поведінкою користувачів на сервері. Одним із ключових можливостей є надсилання та збереження скарг. Це дозволяє учасникам повідомляти про порушення, а адміністраторам оперативно реагувати на них.

На рисунку 3.1 показано функцію, яка реалізує базову команду репорт для надсилання скарги на користувача в межах Discord сервера.

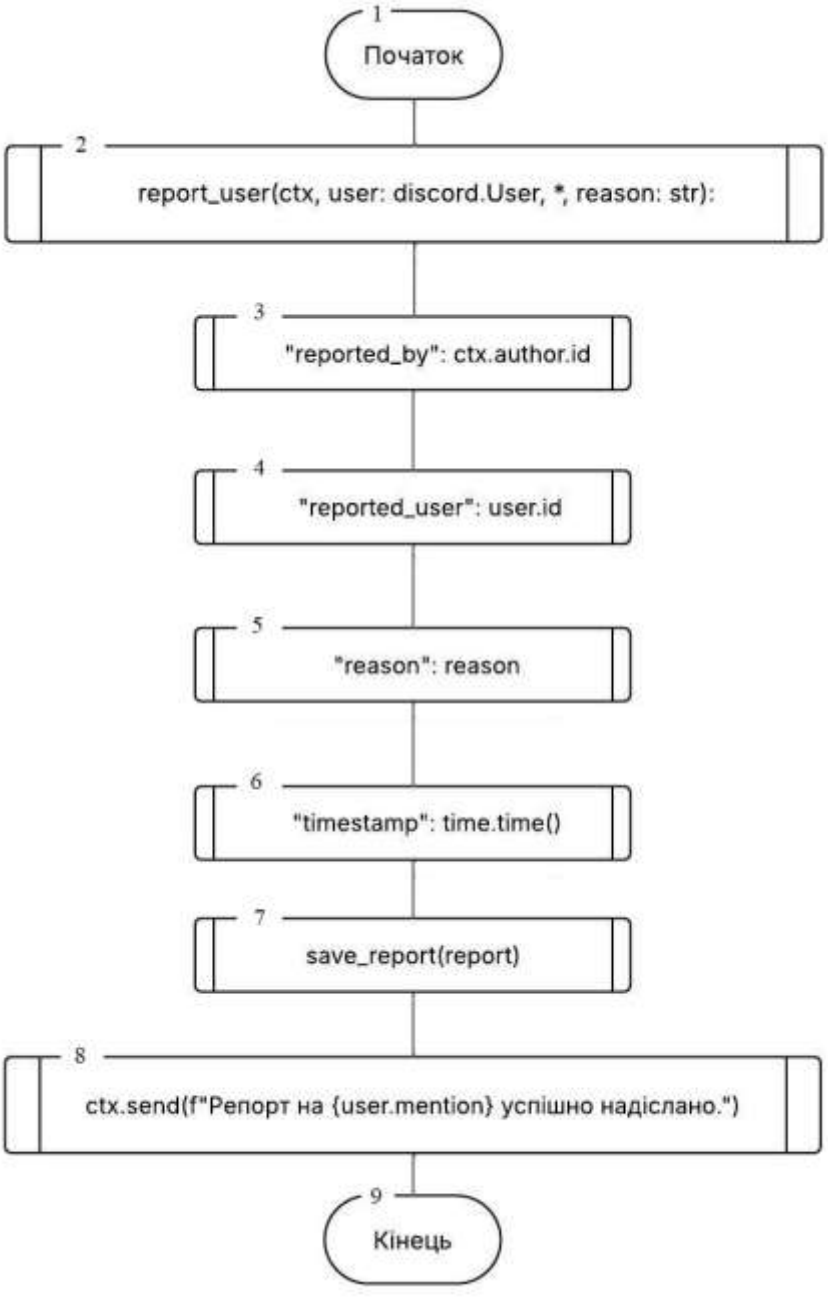

Рисунок 3.1 – Функція надсилання репорту

Створюється асинхронна команда репорту, яка приймає два параметри, а саме користувача на якого подається скарга та текст причини. Після виклику команди формується словник репорт, що містить номер відправника, номер порушника, причину скарги та часову мітку. Викликається функція збереження репорту, яка зберігає інформацію про репорт для подальшого аналізу. Користувач отримує підтвердження про успішне відправлення скарги.

На наступному рисунку 3.2 подано фрагмент функції для збереження репортів у локальному сховищі.

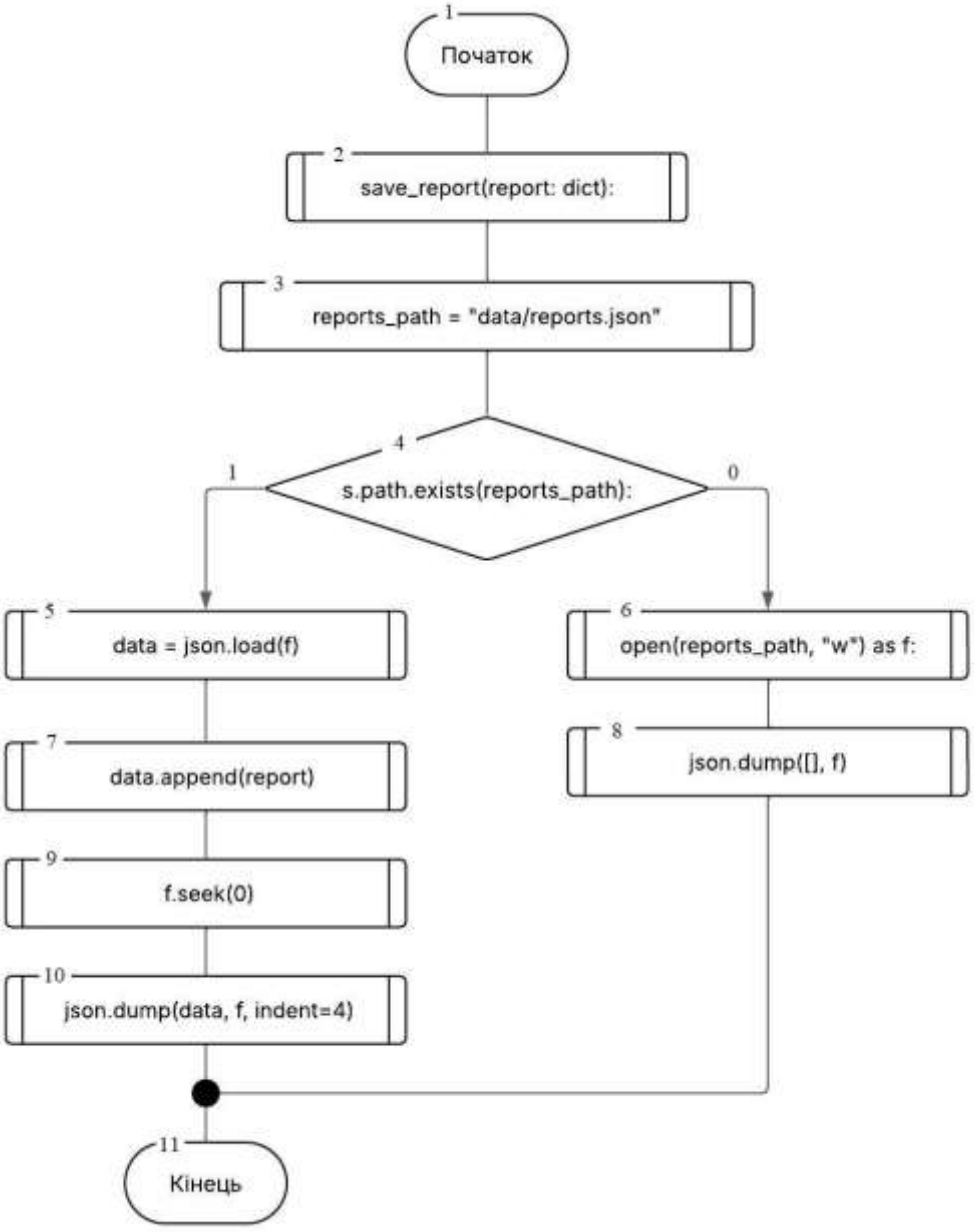

Рисунок 3.2 – Функція збереження репортів

Функція збереження репортів перевіряє наявність файлу reports.json, де зберігаються усі скарги. Якщо файл не існує, створюється новий порожній масив у форматі JSON. Після цього відкривається файл для читання і запису. Новий репорт додається до масиву і весь масив перезаписується з оновленими даними. Таким чином, забезпечується накопичення історії репортів без втрати попередніх записів.

Функція репортів допомагає адміністраторам отримувати інформацію про інциденти, аналізувати їх та вживати необхідних заходів на сервері.

3.4 Розробка функцій керування аудіо контенту

Для забезпечення повноцінної взаємодії користувачів з мультимедійним Discord ботом реалізовано набір функцій для керування аудіо контентом. Функції охоплюють підключення до голосового каналу, відтворення медіа, призупинення, поновлення відтворення та вихід із каналу.

Обробка команди відтворення охоплює перевірку наявності підключення до голосового каналу, підключення за потреби, обробку посилання та ініціацію аудіо потоку. Викликом команди «!play» активує наступний фрагмент. Спочатку перевіряється, чи перебуває користувач у голосовому каналі. У разі підтвердження визначається, до якого саме каналу потрібно здійснити підключення. Далі перевіряється, чи вже має бот активне з’єднання з голосовим каналом. Якщо з’єднання відсутнє, виконується підключення до каналу користувача. Якщо ж користувач не приєднаний до жодного голосового каналу, надсилається повідомлення з інструкцією про необхідність підключитися до такого каналу для використання цієї функції. Після встановлення з’єднання із голосовим каналом виконується обробка вказаного посилання на медіа, що вимагає отримання відповідної інформації про аудіо. На основі цієї інформації створюється джерело звуку, придатне для відтворення. Отримане джерело передається для програвання в голосовому каналі, після чого розпочинається саме відтворення (див. рисунок 3.3).

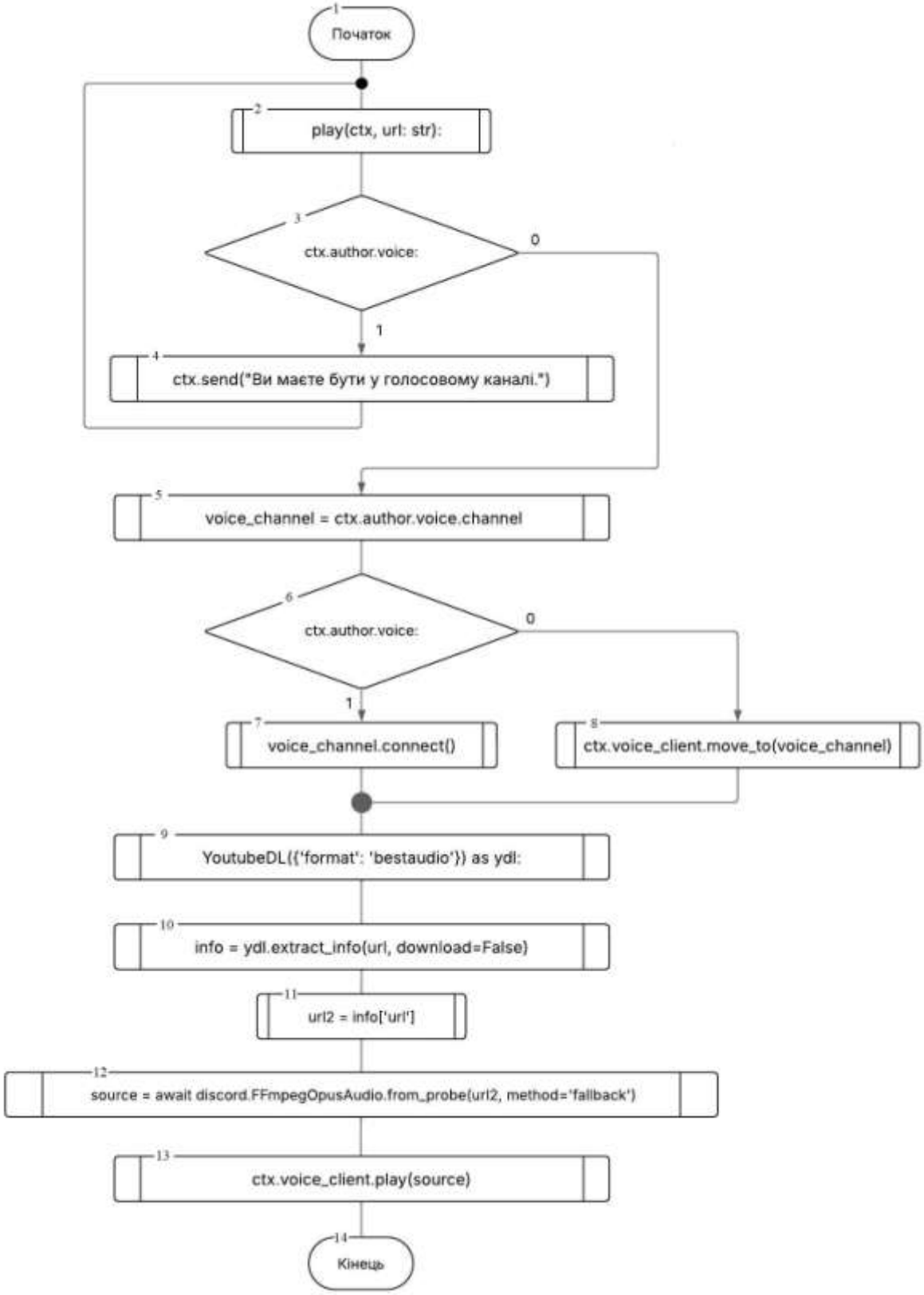

Рисунок 3.3 – Функція відтворення аудіо контенту

Функція керування відтворенням «А» відображений на рисунку 3.4, розпочинається з перевірки, чи виконується відтворення аудіо в голосовому

каналі. Якщо відтворення активне, воно тимчасово зупиняється. Після цього користувачу надсилається повідомлення про призупинення відтворення. Якщо відтворення не відбувається, то функція завершується.

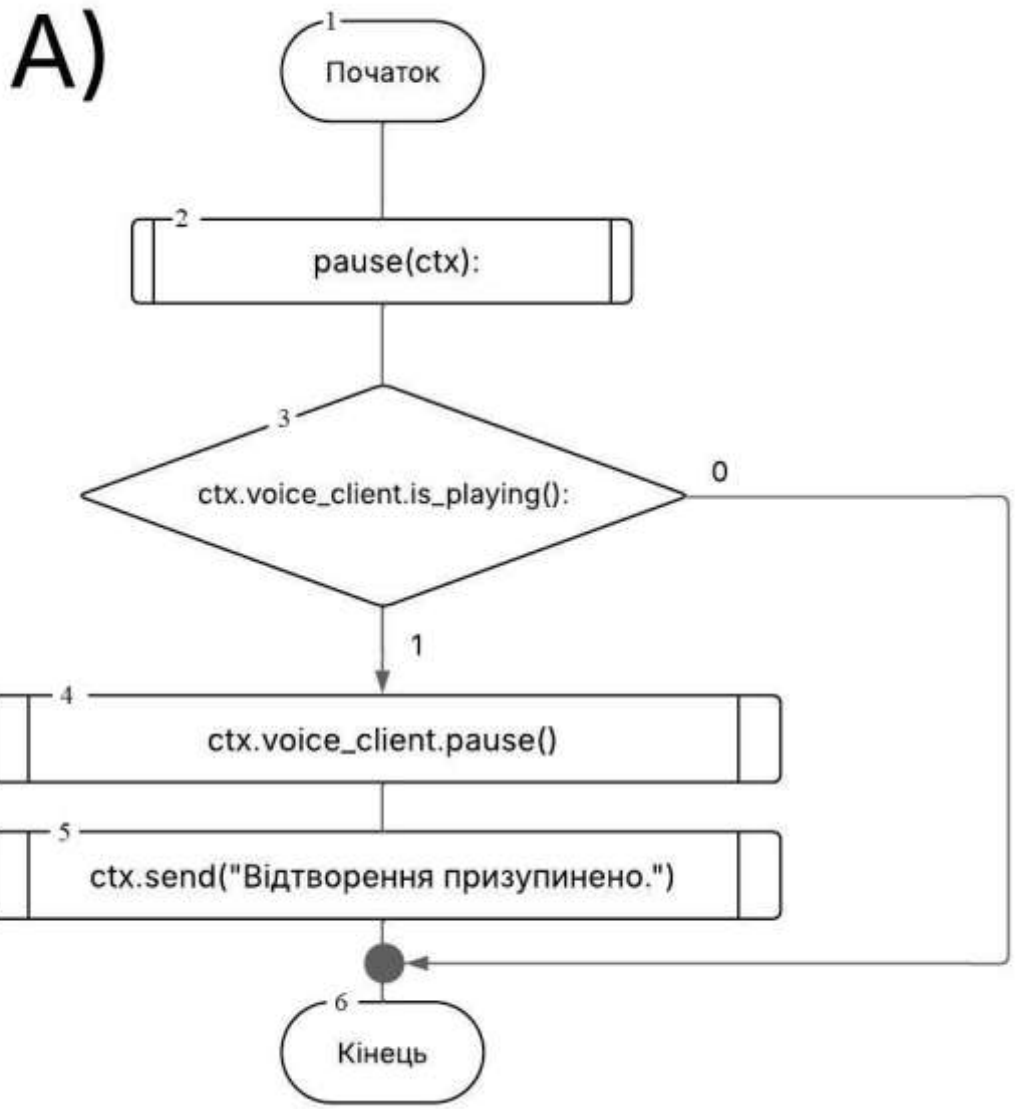

Рисунок 3.4 – Функція керування відтворенням «А»

Функція керування відтворенням «Б» показана на рисунку 3.5. Перш за все перевіряється, чи знаходиться відтворення на паузі. Якщо так, відтворення поновлюється, і користувач отримує відповідне повідомлення про відновлення. Якщо ж пауза не була активною, функція завершується без жодної дії. Завдяки перевірці поточного стану відтворення перед виконанням дій, бот запобігає випадковому, або повторному запуску вже активної дії.

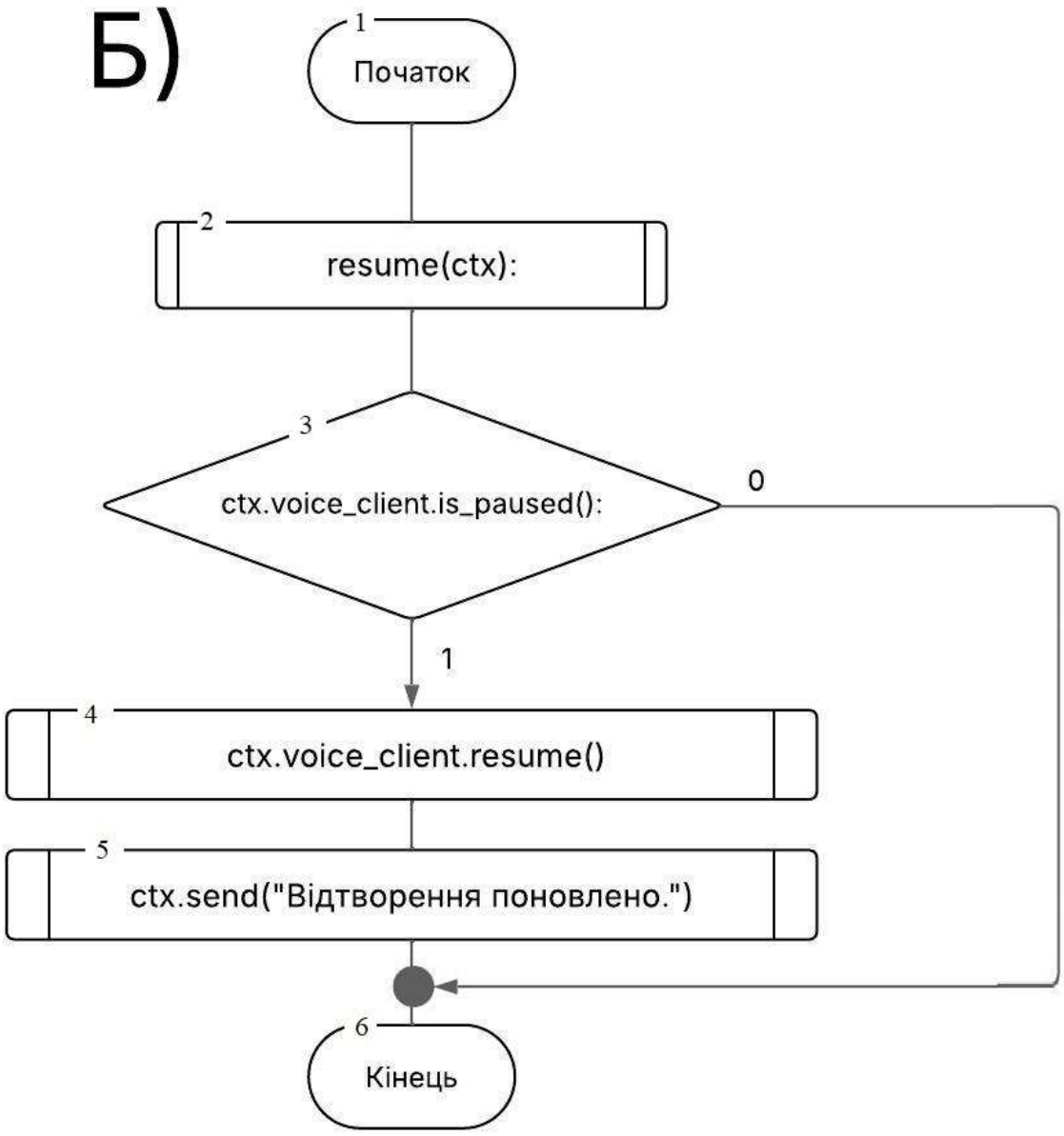

Рисунок 3.5 – Функція керування відтворенням «Б»

Функція вихід з голосового каналу (див. рисунок 3.6). Після активації функції перевіряється, чи підключений бот до голосового каналу. Якщо бот підключений до голосового каналу, то здійснює вихід із каналу. Після цього надсилається повідомлення з підтвердженням того, що бот покинув голосовий канал. Якщо підключення не було встановлено, жодних дій не відбувається і виконання функції завершується. Це дозволяє уникнути помилок у випадку, коли бот уже не перебуває в жодному голосовому каналі.

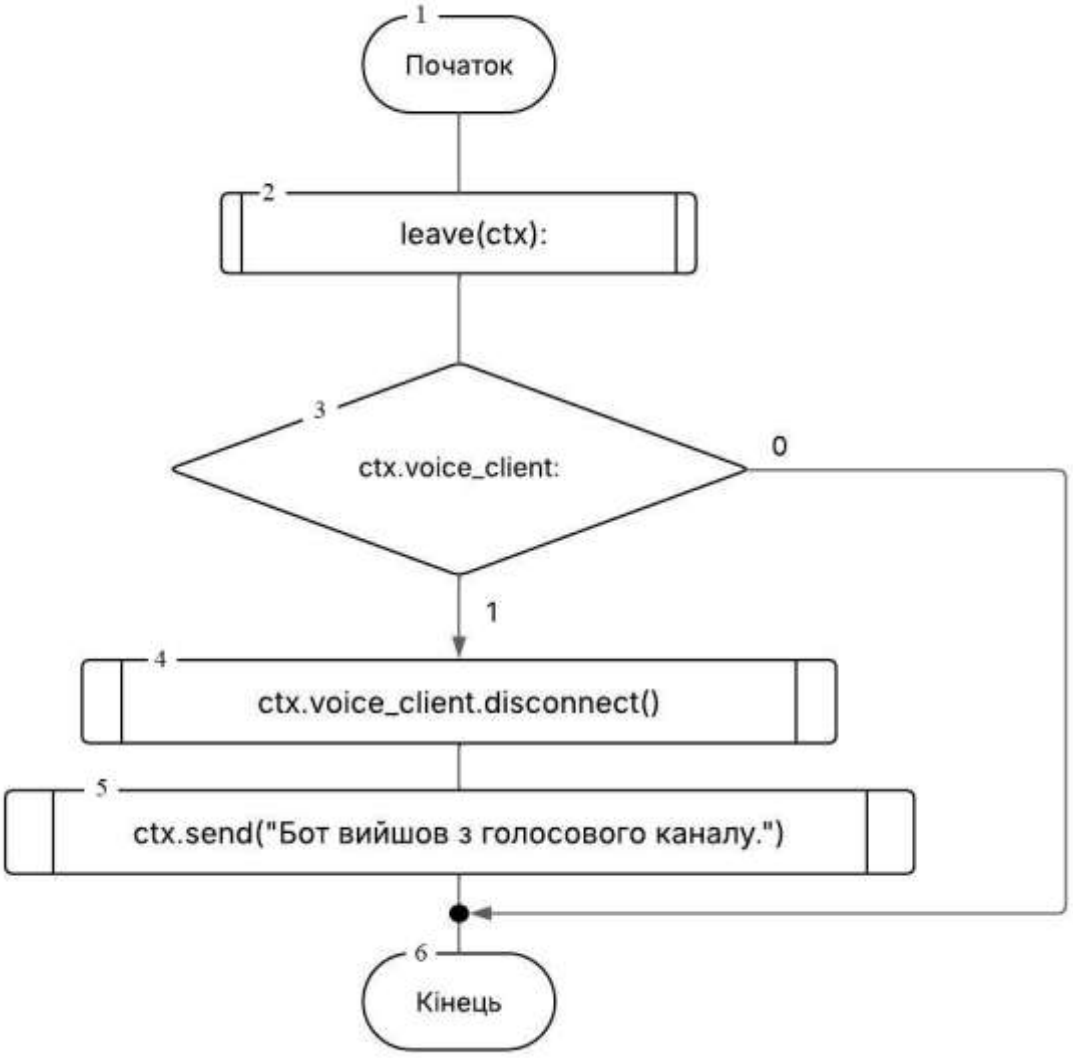

Рисунок 3.6 – Функція вихід з голосового каналу

Функція реалізації управління відтворенням у голосовому каналі забезпечує повноцінну взаємодію з ботом: від підключення й відтворення аудіо до призупинення, поновлення та виходу. Такий підхід гарантує зручність користувача та ефективну організацію роботи голосових команд.

3.5 Розробка функцій фільтрації, обмеження та автоматичного видалення

Для підтримання безпечного та впорядкованого користування ботом на Discord сервері реалізовано функції фільтрації, обмеження та автоматичного видалення. Їхня розробка запобігає появі небажаного контенту, зловживанню

чергою запитів окремими користувачами та забезпечує застосування зауважень за порушення встановлених правил.

Фільтрації небажаних ключових слів визначає, чи дозволено використовувати введений текстовий запит, перевіряючи його на наявність заборонених слів (див. рисунок 3.7).

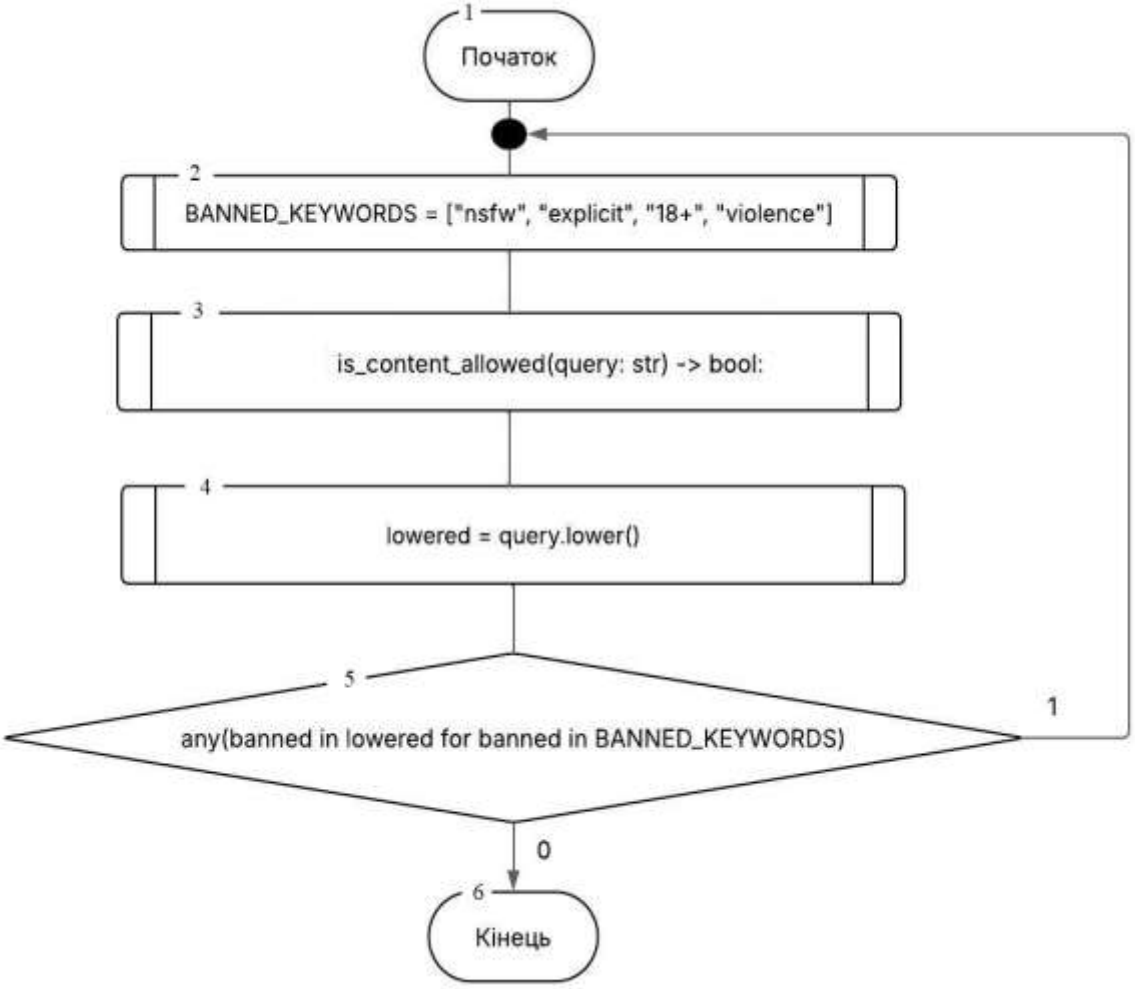

Рисунок 3.7 – Функція фільтрації небажаних ключових слів

Вона забезпечує автоматичну перевірку змісту на відповідність встановленим правилам. Якщо виявлено хоча б одне небажане слово, запит блокується ще до обробки. Таким чином, бот підтримує безпечне інформаційне середовище та запобігає поширенню небажаного контенту.

Обмеження на кількість активних запитів від однієї особидопоможе, щоб уникнути домінування одного користувача у черзі (див. рисунок 3.8).

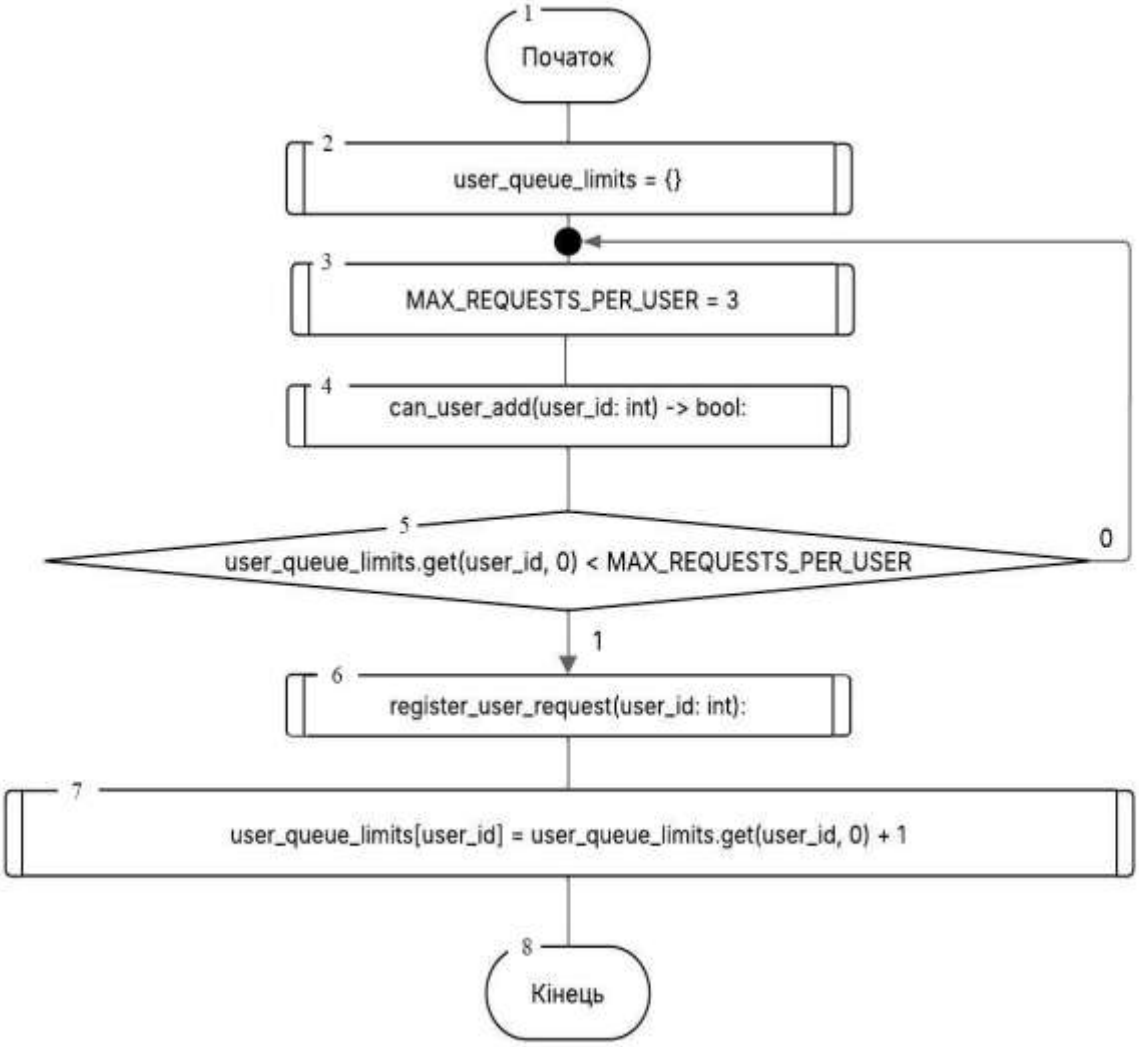

Рисунок 3.8 – Функція обмеження за кількістю запитів

Це дозволяє розподілити доступ до черги рівномірно серед усіх користувачів. Якщо ця межа перевищена, нові запити тимчасово блокуються до моменту завершення. Таким чином, бот регулює рівномірне використання команд між учасниками сервера та сприяє створенню комфортного середовища для спільного прослуховування.

Функція призначена для відстеження кількості порушень, скоєних кожним користувачем. Після кожного нового порушення бот збільшує лічильник. Якщо кількість перевищує допустиму норму, користувача тимчасово блокують. Якщо ж ліміт не досягнуто, надсилається попередження із зазначенням кількості порушень. (див. рисунок 3.9).

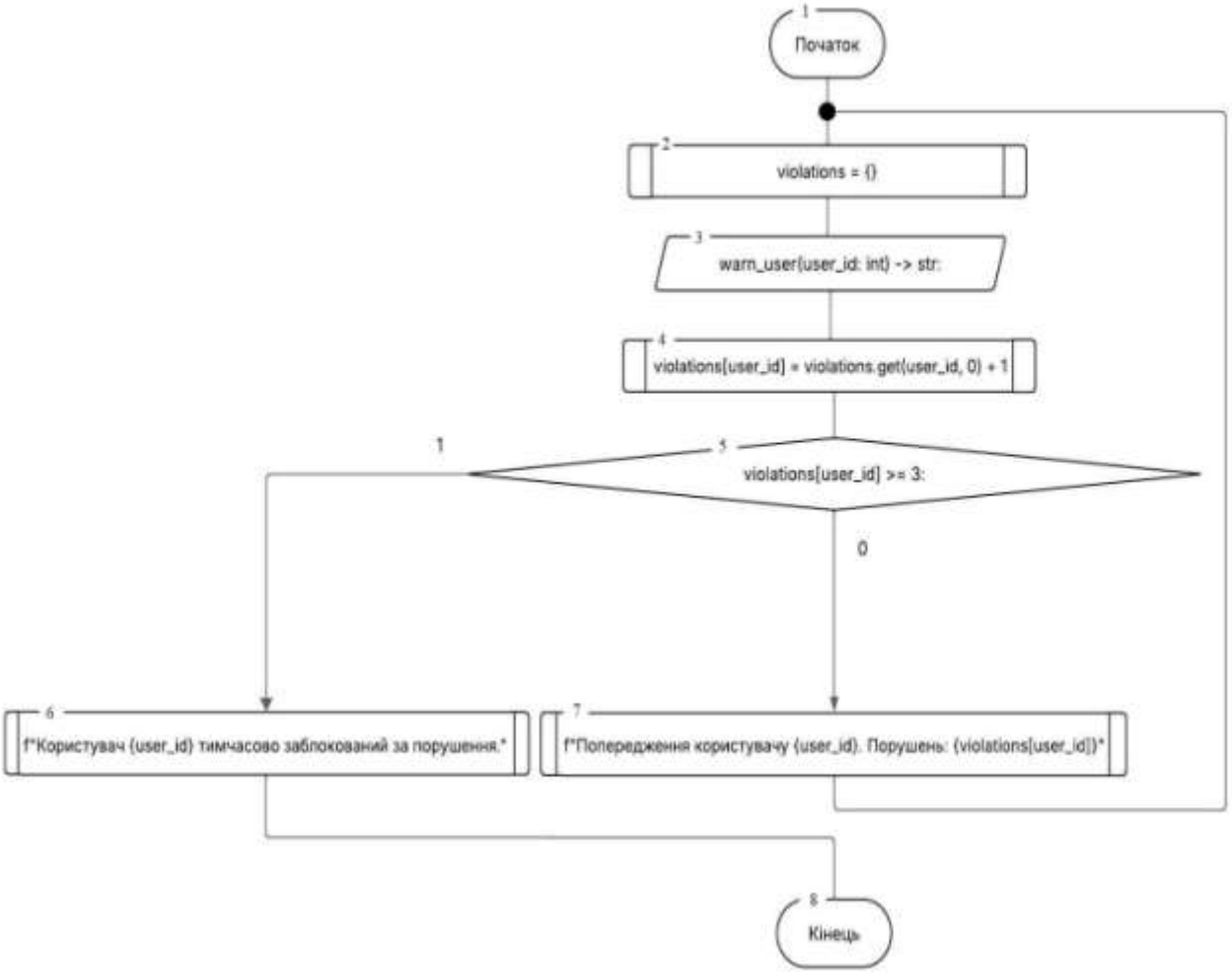

Рисунок 3.9 – Функція автоматичного видалення порушень

У разі порушень можливе довготривале, або повне обмеження доступу до функцій бота. Такий підхід мотивує дотримуватись встановлених правил та зберігає стабільність і справедливість у спільному користуванні ресурсом.

3.6 Висновки

В цьому розділі було здійснено варіантний аналіз і обґрунтування вибору мов програмування. Visual Studio обрано, як середовище для програмування, а мовою програмування обрано Python, як інструмент для реалізації бота. Вибір зумовлено його зручним синтаксисом, підтримкою бібліотек для Discord, а також можливостями для роботи з мультимедійним контентом.

Розроблено основні функціональні компоненти: обробку команд управління аудіо, підключення до голосових каналів, фільтрацію небажаного

контенту, обмеження доступу до функцій, а також систему надсилання та збереження скарг користувачів.

Забезпечено обробку вхідних та вихідних даних з урахуванням ролей і дозволів, що підвищує безпеку використання бота. Усі функції спрямовано на забезпечення інтерактивної та контрольованої роботи мультимедійного бота в Discord каналі.

4 ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

4.1 Тестування функціоналу бота

Тестування бота – це системний процес, спрямований на перевірку його стабільності, коректності роботи та зручності для користувачів. Воно включає низку етапів, методів і інструментів, які допомагають виявити помилки, оптимізувати продуктивність і забезпечити відповідність вимогам [28].

Під час тестування на сервері Discord користувач виконав команду додавання треку за посиланням на YouTube. Бот успішно опрацював команду, додав композицію «FUNK DO BOUNCE (Slowed)» у чергу для відтворення, про що було повідомлено у відповідь. Повідомлення також містило орієнтовний час очікування та місце у черзі (див. рисунок 4.1).

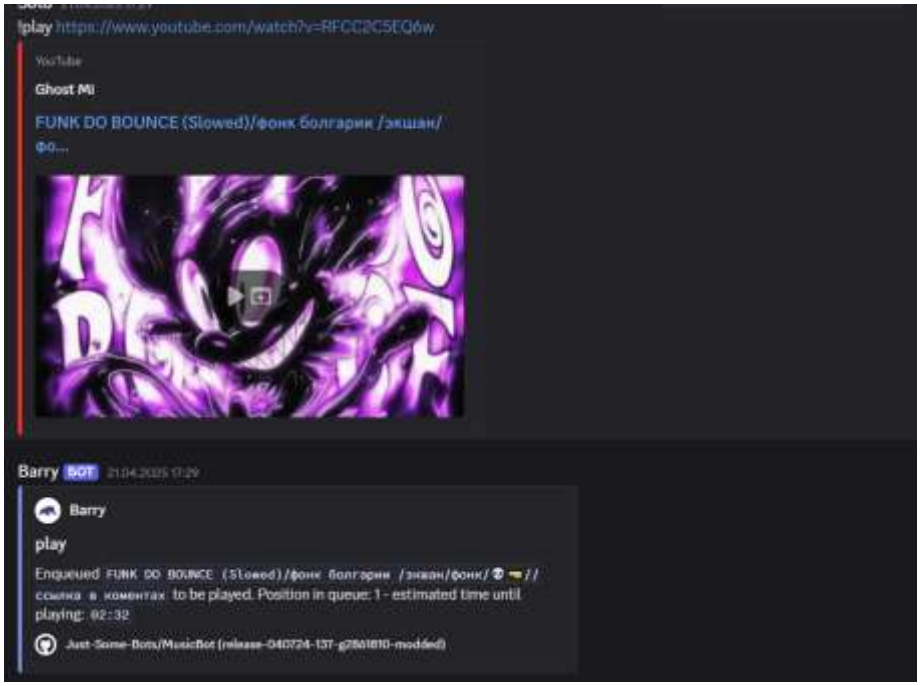

Рисунок 4.1 – Результат введення команди «!play»

Це тестування підтверджує здатність бота коректно сприймати введені команди, виконувати пошук і витягування медіаконтенту з зовнішнього джерела, а також зберігати інформацію про чергу та тривалість очікування.

Ще однією важливою функцією мультимедійного Discord бота є здатність до автоматичного додавання треків до черги відтворення, що значно підвищує інтерактивність і комфорт користувачів у голосових каналах. Бот має підтримувати додавання музики без прямого втручання користувача, реагуючи на певні внутрішні події чи попередньо задані налаштування.

На рисунку 4.2 продемонстровано результат тестування автоматичного відтворення треку «Filous – Better Off (ft. Josh Roa & Bishop)», який був доданий у канал «Загальний» без явної команди користувача. Бот повідомив про запуск композиції, відобразивши обкладинку треку та супроводжуючу інформацію в чаті.

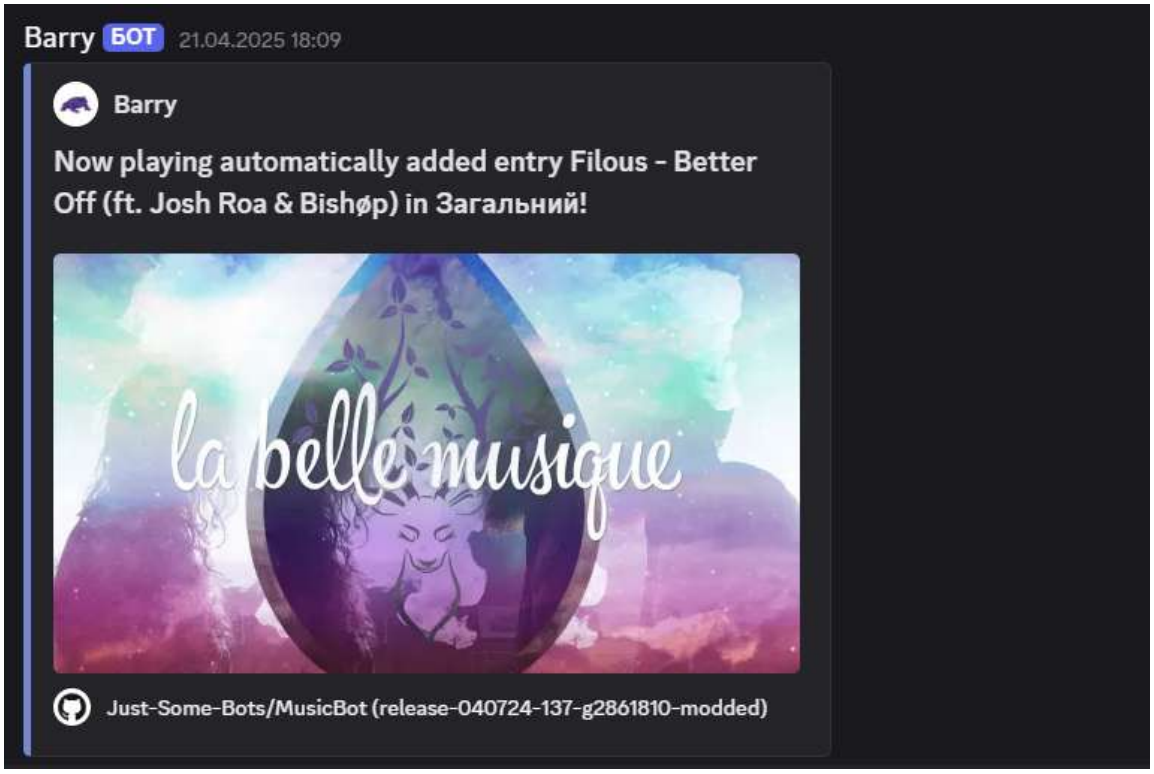

Рисунок 4.2 – Результат тестування автоматичного відтворення треку

Сценарій тестування показав, що бот здатен самостійно обирати та запускати треки відповідно до заданих умов, забезпечуючи безперервне звучання музики. Подібна функціональність є корисною, наприклад, у випадках створення атмосферного фону під час ігор, або спілкування в групових чатах.

Для забезпечення управління музичним контентом у голосовому каналі Discord бот має реалізовану функцію голосування за пропуск треку. Це дає змогу учасникам взаємодіяти зі списком відтворення в режимі реального часу та впливати на звучання музики без необхідності доступу до панелі керування.

На рисунку 4.3 зображено результат тестування команди «!skip», введеної користувачем. Бот Barry успішно зареєстрував голос користувача за пропуск композиції «Fallin Water – Monty Lane Allen», після чого підтвердив, що голосування завершилося успішно і трек було пропущено.

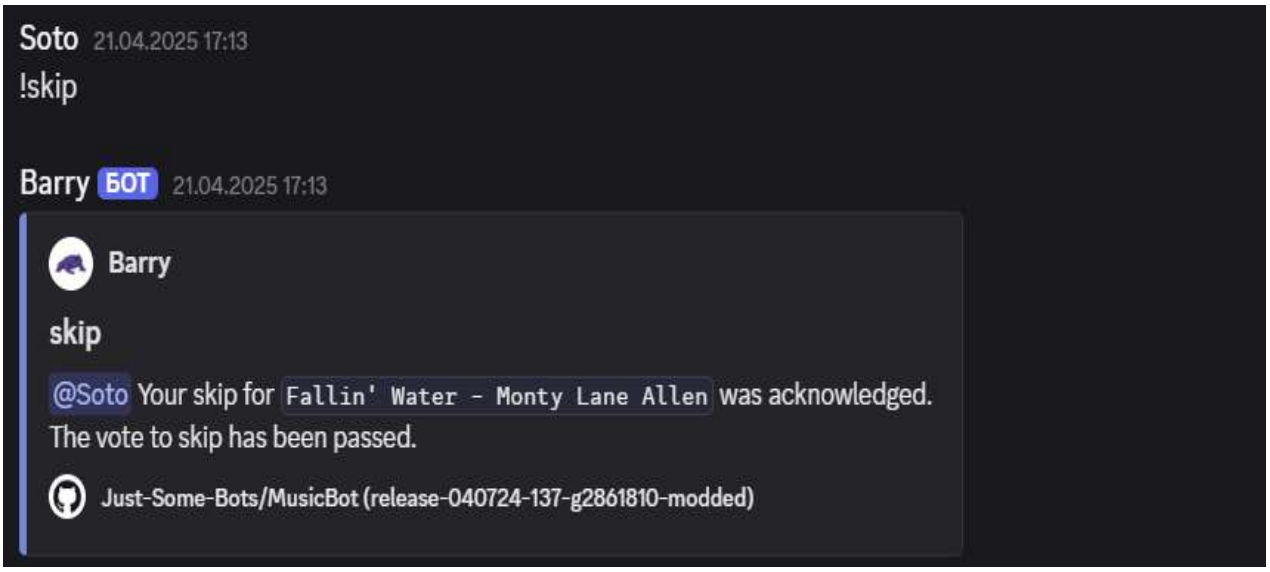

Рисунок 4.3 – Результат тестування команди «!skip»

Цей тест демонструє, що бот коректно обробляє голосування, фіксує імена користувачів, які ініціюють зміну, і надає чіткий зворотний зв'язок у чаті. Такий підхід до взаємодії підвищує динаміку та контроль користувача над відтворенням.

У процесі взаємодії з Discord ботом користувачі потребують доступу до переліку доступних команд для зручного керування мультимедійним відтворенням. Команда «!help» є базовим інструментом, який забезпечує швидкий огляд функціоналу бота.

На рисунку 4.4 зображено результат виконання команди «!help» користувачем у текстовому каналі Discord сервера. Бот із назвою Barry надає перелік доступних команд, серед яких «!clean», «!help», «!id», «!np», «!perms», «!play», «!queue», «!search», «!skip».

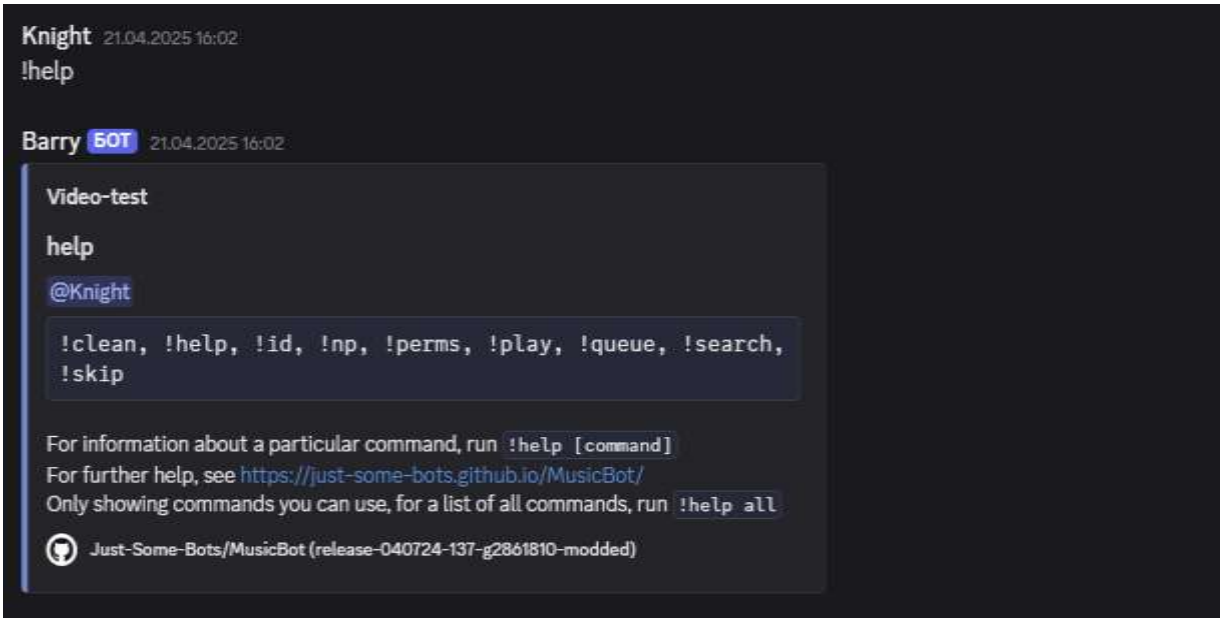

Рисунок 4.4 – Результат тестування команди «!help»

Цей тест демонструє, що бот коректно обробляє запити користувача, забезпечує належний зворотний зв'язок та сприяє інтуїтивному ознайомленню з функціоналом. Це важливо для покращення досвіду користувача та ефективного управління медіаконтентом.

Однією з основних функцій мультимедійного Discord бота є можливість перегляду поточної черги відтворення, що дає змогу користувачам бачити, які треки заплановані до програвання. Це сприяє прозорому керуванню списком композицій та взаємодії учасників у голосовому каналі.

На рисунку 4.5 зображено результат тестування команди «!queue», введеної користувачем. У відповідь бот повідомляє про відсутність активної черги треків. Це вказує на те, що список відтворення порожній, і користувачеві пропонується додати композицію за допомогою команди «!play».

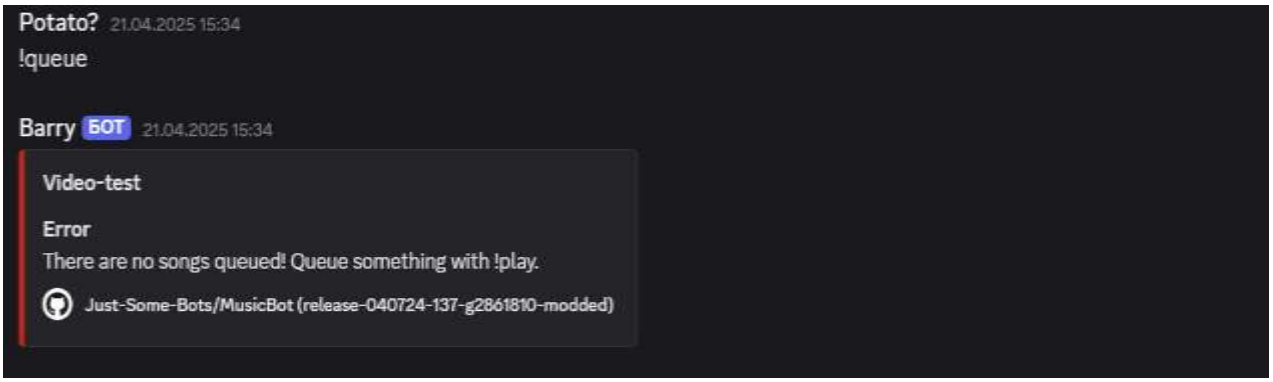

Рисунок 4.5 – Результат тестування команди «!queue»

Тест підтверджує, що бот правильно обробляє ситуації з порожньою чергою та надає інструкції для подальших дій, що покращує зручність користування та орієнтацію в функціоналі сервісу.

Функція «!autoplaylist» у мультимедійному Discord боті зазвичай використовується для автоматичного запуску заздалегідь визначеного списку треків при відсутності активної черги.

На рисунку 4.6 зображено результат спроби користувача скористатися командою «!autoplaylist». Бот Barry повернув повідомлення про помилку з поясненням. Це означає, що команда не дозволена для поточної групи користувача, через відсутність відповідної ролі на сервері.

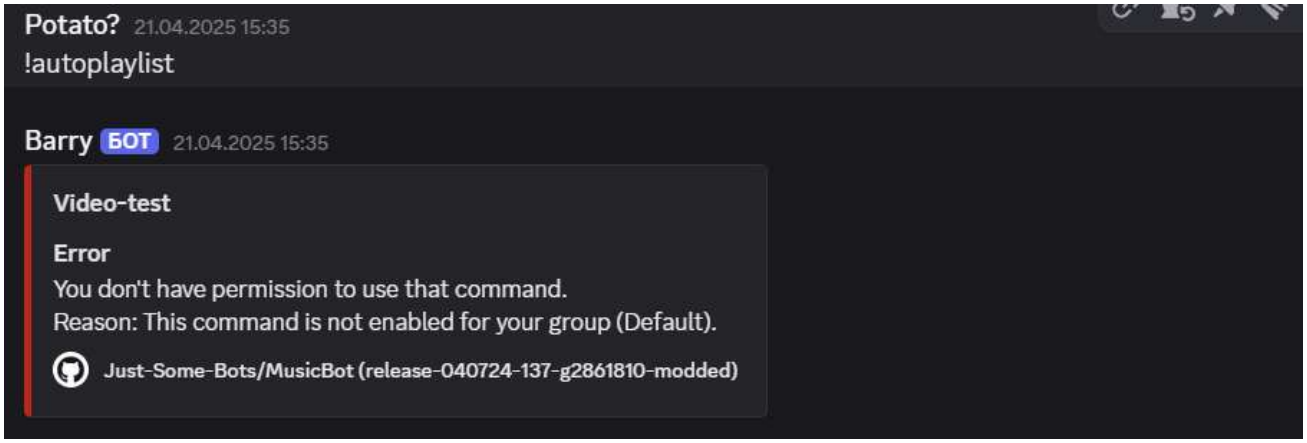

Рисунок 4.6 – Результат тестування команди при відсутній ролі на сервері

Цей тест показує важливість правильного налаштування прав доступу в боті. Подібні обмеження запобігають несанкціонованому використанню функцій та дають змогу адміністраторам серверів контролювати поведінку бота відповідно до власної політики.

Якість взаємодії з чат-ботом значною мірою залежить від чіткості та коректності сформульованих користувачем запитів. У цьому підрозділі проаналізовано два приклади комунікації користувача з ботом, які демонструють різні сценарії: некоректне і коректне формулювання звернення.

На рисунку 4.7 зображено ситуацію, коли користувач вводить фрагменти тексту без зрозумілого запиту. У відповідь бот намагається з'ясувати, що саме мав на увазі користувач, і просить надати більше контексту, або переформулювати питання. Це свідчить про обмеження у розпізнаванні ботом неконкретних запитів, та його здатність вічливо спрямувати користувача на конструктивне спілкування.

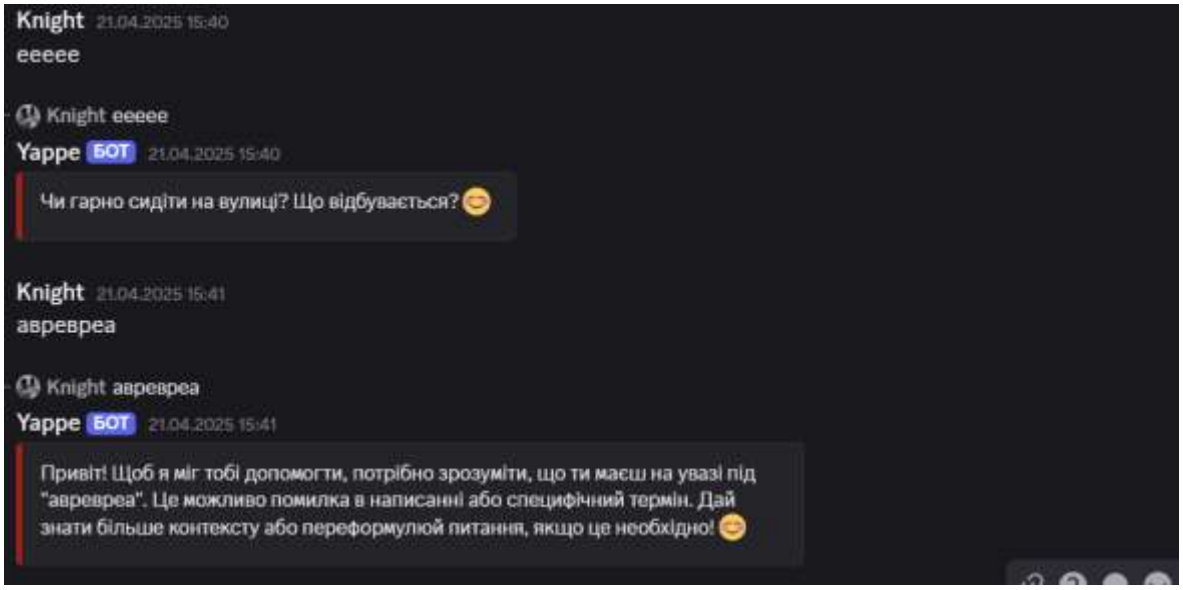

Рисунок 4.7 – Відповідь бота на незрозуміле повідомлення

Натомість на рисунку 4.8 продемонстровано приклад вдалого звернення до бота, де користувач чітко формулює запит. Така реакція свідчить про адаптивність системи та здатність коректно інтерпретувати зрозумілі запити.

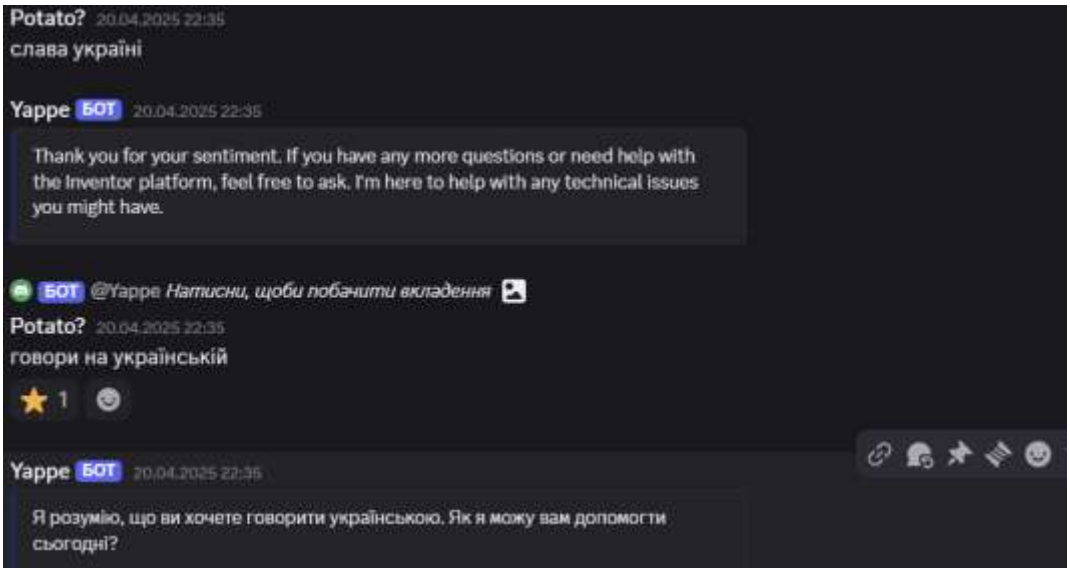

Рисунок 4.8 – Правильне формулювання запиту

Ці приклади підкреслюють важливість грамотного формулювання звернень до ботів у цифрових середовищах. Чим чіткіше поставлено питання, тим вища ймовірність отримання корисної та релевантної відповіді.

Для забезпечення зручності спілкування при перевантаженні наявних голосових каналів, мультимедійний Discord бот має функціонал автоматичного створення тимчасових голосових каналів.

На рисунку 4.9 зображено процес створення нового голосового каналу. Користувач обирає опцію «Створити канал», після чого відкривається меню налаштування, де можна задати назву нового каналу та вказати його як приватний, якщо потрібно.

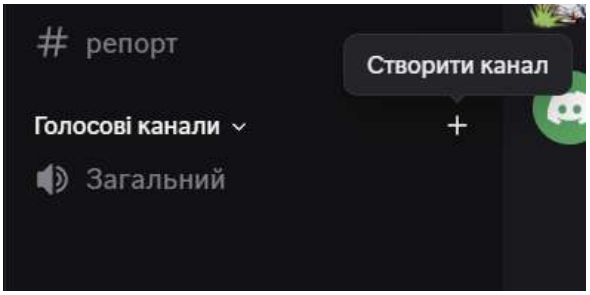

Рисунок 4.9 – Інтерфейс створення тимчасово голосового каналу

На рисунку 4.10 зображено вікно створення нового каналу, де користувач може задати назву каналу та обрати опцію «Приватний канал», щоб обмежити доступ до нього лише певним учасникам. У випадку автоматичного створення, ці параметри налаштовуються ботом відповідно до заданої конфігурації

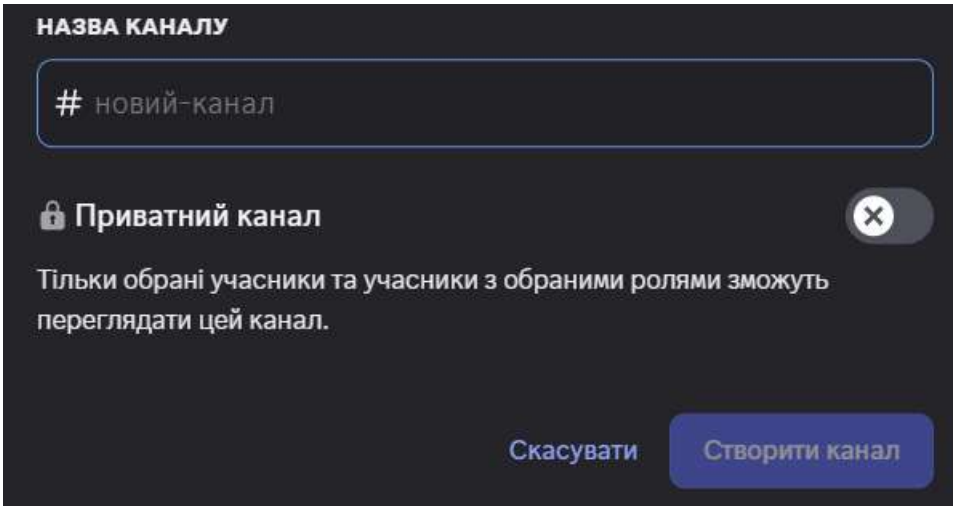

Рисунок 4.10 – Інтерфейс створення нового голосового каналу

Бот автоматично видаляє тимчасовий голосовий канал після того, як усі учасники залишають його, що дозволяє уникати накопичення зайвих каналів. Такий підхід забезпечує динамічне масштабування серверу відповідно до кількості активних користувачів та підвищує зручність комунікації в режимі реального часу.

Для забезпечення модерації та оперативного реагування на небажані повідомлення Discord бот має функцію репортування. Ця функція дозволяє користувачам відправляти скарги на повідомлення, які порушують правила спільноти, до спеціально відведеного каналу для модерації.

Користувач надіслав повідомлення, на яке інший користувач відреагував, використовуючи команду репорту. Бот успішно переслав це повідомлення до текстового каналу, призначеного для обробки скарг. У відповідь бот підтвердив, що повідомлення було зареєстровано, як репорт і повідомив про це

користувача. На рисунку 4.11 зображено результат тестування команди репорту.

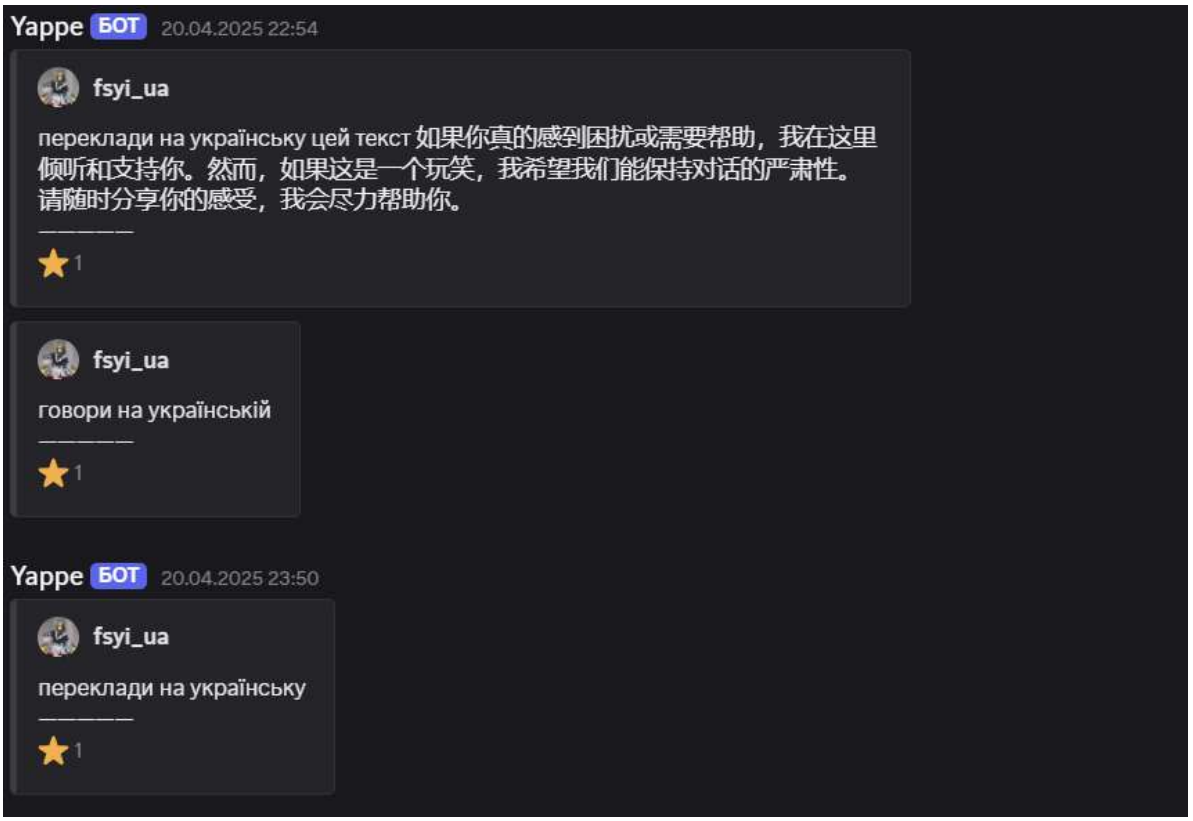

Рисунок 4.11 – Підтвердження скарги на повідомлення користувача

Цей тест демонструє, що бот коректно обробляє запити на репортування, фіксує інформацію про повідомлення та його автора, а також забезпечує прозорий зворотний зв’язок. Такий механізм сприяє підвищенню безпеки та комфорту в спільноті.

Ці етапи тестування спрямовані на забезпечення високої якості та надійності програмних модулів мультимедійного бота для управління Discord каналом. Вони дозволяють виявити та усунути можливі помилки, перевірити стабільність роботи функціоналу, а також гарантувати зручність взаємодії з ботом для кінцевих користувачів у різних сценаріях використання. Тестування охоплює функціональні компоненти та їхню взаємодію в межах системи, Це дозволяє забезпечити комплексну перевірку всіх аспектів роботи бота.

4.2 Розробка інструкції користувача

Інструкція користувача зумовлює визначення технічних вимог для запуску програмного продукту мультимедійного бота для Discord.

Деталі щодо мінімальної конфігурації пристрою, необхідного для хостингу бота, подано в таблиці 4.1.

Таблиця 4.1 – Мінімальна конфігурація

Параметр	**Значення**
Тип процесора	2-ядерний, 2.0 GHz
Об’єм оперативної пам’яті	2 ГБ для 64-разрядної системи
Інтернет-з’єднання	Стабільне з'єднання, 1 Мбіт/с і вище
Місце на жорсткому диску	1 ГБ
Операційна система	Windows 7,10

Деталі щодо рекомендованої конфігурації пристрою, необхідного для хостингу бота, подано на таблиці 4.2.

Таблиця 4.2 – Рекомендована конфігурація

Параметр	**Значення**
Тип процесора	4-ядерний, 2.5 GHz
Об’єм оперативної пам’яті	2 ГБ для 64-разрядної системи
Інтернет-з’єднання	Стабільне з'єднання, 10 Мбіт/с і вище
Місце на жорсткому диску	1 ГБ
Операційна система	Windows 10

Після встановлення бота та запуску його на сервері, користувач повинен налаштувати файл конфігурації з унікальним Discord токеном, що дозволить здійснити авторизацію через Discord API. Після успішного запуску бот автоматично приєднується до вказаного каналу на Discord сервері та готовий до взаємодії з користувачами.

Для використання мультимедійного функціоналу користувачі можуть застосовувати стандартні команди. Наприклад: !play для відтворення музики, !pause – для призупинення, !skip – для переходу до наступного треку. Адміністративні команди доступні тільки користувачам з відповідними правами, що забезпечує контроль доступу до функціоналу.

Крім того, реалізовано функціонал створення форм для репортів. Користувачі можуть викликати форму через команду !report, після чого з’являється інтерфейс для опису помилки та порушення. Надіслані дані автоматично передаються до закритого каналу, доступного лише адміністраторам.

У разі першого використання бот відображає підказки щодо доступних команд та їх функцій. Це дозволяє швидко ознайомитись з можливостями бота, не вдаючись до сторонніх інструкцій.

Додатково передбачено механізм кастомізації команд і повідомлень, що дозволяє адаптувати бота під потреби конкретної спільноти. Всі взаємодії із ботом логуються, а основні події. Наприклад: відтворення, додавання в чергу, модераційні дії, автоматично відображаються в текстовому каналі.

Таким чином, розроблені програмні модулі забезпечують простий у використанні, стабільний та функціональний інструмент для управління Discord каналом, розширюючи можливості взаємодії між учасниками спільноти.

4.3 Висновки

У четвертому розділі було проведено тестування мультимедійного бота для управління Discord каналом. Проведено перевірку коректності обробки

команд користувачів, зокрема взаємодії з аудіофункціоналом, права доступу та реагуванням бота на некоректні запити.

Була розроблена інструкція користувача. Вона пояснює процес встановлення бота, його налаштування та використання основних функцій для керування Discord сервером.

ВИСНОВКИ

У межах бакалаврської кваліфікаційної роботи було реалізовано програмне забезпечення мультимедійного бота для управління Discord каналом. Основна увага була приділена автоматизації процесу керування медіаконтентом у голосових та текстових каналах.

Проведено аналіз сучасних методів у сфері мультимедійних Discord ботів. Після чого виявлено їхні переваги та недоліки. На основі порівняльного аналізу обґрунтовано причину створення власного програмного забезпечення, здатного поєднувати найкращі функції наявних ботів і доповнювати їх новим функціоналом.

Розроблено новий метод динамічного пошуку та керування відтворенням аудіоконтенту, який поєднує API-запити до зовнішніх мультимедійних платформ із фільтрацією результатів за ключовими словами та релевантністю, що підвищує зручність використання бота, зменшує кількість помилок і забезпечує швидке та релевантне відтворення медіаконтенту.

Удосконалено метод модерації контенту, який на відміну від відомих забезпечує автоматичне виявлення та блокування шкідливих повідомлень у текстових каналах Discord, що дозволяє зменшити кількість хибних спрацювань і підвищити продуктивність модерації без втручання людини.

Розроблено новий метод репортів, який дозволяє повідомляти про порушення правил та помилки в чаті через спеціальний репорт-канал, що дало можливість полегшити роботу модераторів та пришвидшити реагування на інциденти.

Проведено тестування основних модулів програмного забезпечення, що підтвердило його працездатність та відповідність функціональним вимогам. Розроблено інструкцію користувача і визначено системні вимоги для розміщення бота.

Таким чином, розроблене програмне забезпечення є ефективним інструментом для автоматизації керування Discord каналом.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Mirrashed M. «Turbocharging Your Discord Server with Bots: Improving Member Experience», IEEE Potentials, vol. 41, no. 4, pp. 22–25, Jul. – Aug. 2022.

2. Tkachenko O., Shevchenko A. «Some Aspects of Developing a Universal Server Discord-bot», Digital Platform Information Technologies in Sociocultural Sphere, vol. 4, no. 2, pp. 173–186, Dec. 2021.

3. Smith M. API Design Patterns. Shelter Island, NY, USA: Manning Publications, 2020. 480 p.

4. Романюк О. В., Довгалюк Д. В. Мультимедійний бот для управління Discord каналом. IV Всеукраїнської науково-технічної конференції молодих вчених, аспірантів і студентів «Комп'ютерні ігри і мультимедіа, як інноваційний підхід до комунікації - 2024» м. Одеса, 2024. URL: https://www.ontu.edu.ua/download/konfi/2024/Abstracts_Computer_games_and_multimedia_innovative_approach_electronic_communication_2024.pdf (дата звернення: 08.04.2025).

5. Романюк О. В., Довгалюк Д. В. Методи відтворення та модерації мультимедійного контенту в Discord каналі. Міжнародна науково-практична інтернет-конференція «Молодь в науці: дослідження, проблеми, перспективи (МН-2025)». Вінниця, 2025. URL: https://conferences.vntu.edu.ua/index.php/mn/mn2025/paper/viewFile/25101/20722 (дата звернення: 14.04.2025).

6. Романюк О. В., Довгалюк Д. В. Розробка програмного забезпечення мультимедійного бота для управління Discord каналом. LIV Всеукраїнська науково-технічна конференція підрозділів Вінницького національного технічного університету (2025). Вінниця, 2025. URL: https://conferences.vntu.edu.ua/index.php/all-fitki/all-fitki-2025/paper/view/24357 (дата звернення: 26.04.2025).

7. Усе в одному місці: як програма Discord допоможе організувати дистанційне навчання. URL: https://nus.org.ua/2020/05/04/use-v-odnomu-mistsi-

yak-programa-discord-dopomozhe-organizuvaty-dystantsijne-navchannya/ (дата звернення: 28.03.2025).

8. Rythm - Discord Music Bot Made for Listening Together. URL: https://discord.com/invite/ZtH6rZpa8y (Last accessed: 01.04.2025).

9. Groovy Discord Bot - Invite, Vote & Status. URL: https://alternative.me/discord/bots/groovy (Last accessed: 02.04.2025).

10. MEE6 / X. URL: https://help.mee6.xyz/support/home (Last accessed: 06.04.2025).

11. Hydra - The Perfect Discord Bot. URL: https://hydra.bot/ (Last accessed: 06.04.2025).

12. FredBoat/ URL: https://fredboat.com/docs/commands (Last accessed: 07.04.2025).

13. Horstmann Cay S. Core Java Volume I – Fundamentals (13th Edition), Boston: Pearson, 2024. 944 p.

14. Streamer P. WebRTC Integrator’s Guide, London: WebTech Press, 2021. 382 p.

15. Данилюк А. М. JSON: Структура, обробка та застосування у сучасному програмуванні. ТехноКнига: Харків, 2023. 245 с.

16. Erl T., Barcelo Monroy E. Cloud Computing: Concepts, Technology, Security, and Architecture (2nd Edition), Boston: Pearson, 2023. 944 p.

17. API Reference. URL: https://discordpy.readthedocs.io/en/stable/api.html

18. Sommerville I. Engineering Software Products: An Introduction to Modern Software Engineering. Boston: Pearson, 2020. 352 p.

19. Roger S., Maxim B. Software Engineering: A Practitioner’s Approach, 9th ed., McGraw-Hill Education, 2020. 912 p.

20. Sommerville I. Engineering Software Products: An Introduction to Modern Software Engineering. Boston: Pearson, 2020. 352 p.

21. Microsoft Corporation, Visual Studio Documentation. URL: https://learn.microsoft.com/en-us/visualstudio/?view=vs-2022 (Last accessed: 15.04.2025).

22. Welcome to Apache NetBeans. URL: https://netbeans.apache.org/front/main/index.html (Last accessed: 20.04.2025).

23. Apple Inc., Xcode Overview. URL: https://developer.apple.com/xcode/ (Last accessed: 20.04.2025).

24. Ramalho L. Fluent Python: Clear, Concise, and Effective Programming, 2nd ed. Sebastopol, CA: O'Reilly Media, 2022. 1016 p.

25. Zakas N. Professional JavaScript for Web Developers, 4th ed. Hoboken, NJ: Wiley, 2020. 960 p.

26. Bancila M. Modern C++ Programming Cookbook, 3rd ed. Birmingham, UK: Packt Publishing, 2024. 816 p.

27. Discord Inc., Discord Developer Portal: API Reference. URL: https://discord.com/developers/docs/intro (дата звернення: 25.04.2022).

28. Kinsbruner E., et al. Continuous Testing for DevOps Professionals, 1st ed. Berkeley, CA: Apress, 2020. 414 p.

ДОДАТКИ

68

ДОДАТОК А
Технічне завдання

Міністерство освіти і науки України

Вінницький національний технічний університет

Факультет інформаційних технологій та комп'ютерної інженерії

ЗАТВЕРДЖУЮ
Завідувач кафедри ПЗ
д.т.н., проф.
_____ О. Н. Романюк
"25" 03 2025 р.

Технічне завдання
на бакалаврську кваліфікаційну роботу «Розробка програмного забезпечення мультимедійного бота для управління Discord каналом»
за спеціальністю 121 Інженерія програмного забезпечення

Керівник бакалаврської кваліфікаційної роботи:
_____ к.т.н., доц. каф. ПЗ О.В. Романюк
"24" 03 2025 р.
Виконав:
_____ студент гр.4ПІ-21б Д. В. Довгалюк
"24" 03 2025 р.

Вінниця – 2025 року

1. **Найменування та галузь застосування**

Бакалаврська кваліфікаційна робота: «Розробка програмного забезпечення мультимедійного бота для управління Discord каналом».

Галузь застосування – інформаційні технології.

2. Підстава для розробки.

Підставою для виконання бакалаврської кваліфікаційної роботи (БКР) є індивідуальне завдання на БКР та наказ від 20 березня 2025 р. №97 ректора по ВНТУ про закріплення тем БКР.

3. Мета та призначення розробки.

Метою дослідження є покращення процесу керування мультимедійним контентом на Discord каналі за рахунок розробки нових методів, що дозволить дозволить автоматизувати відтворення музики, обробку запитів користувачів і адміністрування контенту на сервері.

Призначення роботи – розробка методів і програмного забезпечення мультимедійного бота для управління аудіовмістом та взаємодією користувачів у Discord каналі.

4. Вихідні дані для проведення НДР

Перелік основних літературних джерел, на основі яких буде виконуватись БКР.

1. Ramalho L. Fluent Python: Clear, Concise, and Effective Programming, 2nd ed. Sebastopol, CA: O’Reilly Media, 2022. 1016 p.

2. Smith M. API Design Patterns. Shelter Island, NY, USA: Manning Publications, 2020. 480 p.

3. Данилюк А. М. JSON: Структура, обробка та застосування у сучасному програмуванні. ТехноКнига: Харків, 2023. 245 с.

4. Kinsbruner E., et al. Continuous Testing for DevOps Professionals, 1st ed. Berkeley, CA: Apress, 2020. 414 p.

5. Технічні вимоги

Дії бота – відтворення аудіо у голосовому каналі; додавання треків до черги відтворення; призупинення, відновлення або пропуск треків; вихід бота з голосового каналу; текстові повідомлення – підтвердження виконання команд, повідомлення про помилки або стан черги; інтерактивні елементи – реакції на повідомлення, вбудовані повідомлення з інформацією про трек або чергу; обробка команд – через Discord API; мультимедійна обробка – завантаження потоків через youtube_dl та yt-dlp; конвертація аудіо за допомогою FFmpeg; передача аудіо у голосовий канал через PyNaCl; контроль доступу – перевірка ролей і прав користувачів для обмеження доступу до певних команд; кінцевий результат – взаємодія користувача з ботом у Discord каналі.

6. Конструктивні вимоги.

Графічна та текстова документація повинна відповідати діючим стандартам України.

7. Перелік технічної документації, що пред'являється по закінченню робіт:

– пояснювальна записка до БКР;
– технічне завдання;
– лістинги програми.

8. Вимоги до рівня уніфікації та стандартизації

При розробці програмних засобів слід дотримуватися уніфікації і ДСТУ.

9. Стадії та етапи розробки:

№ з/п	Назва етапів бакалаврської кваліфікаційної роботи	Строк виконання етапів роботи
1	Аналіз програмного забезпечення для управління Discord каналом	25.03.25 – 31.03.2025
2	Розробка діаграми діяльності та структури роботи програмного забезпечення	31.03.25 – 05.04.2025
3	Розробка методу репортів	05.03.2025 – 07.04.2025
4	Розробка методу динамічного пошуку та керування відтворенням аудіо контенту	07.04.2025 – 12.04.2025
5	Розробка методу модерації контенту	12.04.2025 – 23.04.2025
6	Розробка програмного забезпечення мультимедійного бота для управління Discord каналом	23.04.2025 – 01.05.2025
7	Тестування програмного забезпечення	01.05.2025 – 10.05.2025
8	Оформлення матеріалів до захисту БКР	10.05.2025 – 30.05.25

10. Порядок контролю та прийняття.

Виконання етапів бакалаврської кваліфікаційної роботи контролюється керівником згідно з графіком виконання роботи. Захист бакалаврської кваліфікаційної роботи здійснюється ДЕК, затвердженою зав. кафедрою згідно з графіком**.**

72

ДОДАТОК Б
Протокол перевірки на плагіат

ПРОТОКОЛ ПЕРЕВІРКИ КВАЛІФІКАЦІЙНОЇ РОБОТИ

Назва роботи: Розробка програмного забезпечення мультимедійного бота для управління Discord каналом.

Тип роботи: бакалаврська кваліфікаційна робота
(бакалаврська кваліфікаційна робота / магістерська кваліфікаційна робота)

Підрозділ кафедра програмного забезпечення, ФІТКІ, 4ПІ-21б
(кафедра, факультет, навчальна група)

Коефіцієнт подібності текстових запозичень, виявлених у роботі системою StrikePlagiarism 4,5 %

Висновок щодо перевірки кваліфікаційної роботи (відмітити потрібне)

■ Запозичення, виявлені у роботі, є законними і не містять ознак плагіату, фабрикації, фальсифікації. Роботу прийняти до захисту

☐ У роботі не виявлено ознак плагіату, фабрикації, фальсифікації, але надмірна кількість текстових запозичень та/або наявність типових розрахунків не дозволяють прийняти рішення про оригінальність та самостійність її виконання. Роботу направити на доопрацювання.

☐ У роботі виявлено ознаки плагіату та/або текстових маніпуляцій як спроб укриття плагіату, фабрикації, фальсифікації, що суперечить вимогам законодавства та нормам академічної доброчесності. Робота до захисту не приймається.

Експертна комісія:

_____ _____
(прізвище, ініціали, посада) (підпис)

_____ _____
(прізвище, ініціали, посада) (підпис)

Особа, відповідальна за перевірку _____ Черноволик Г. О.

З висновком експертної комісії ознайомлений(-на)

Керівник _____ Романюк О. В., к.т.н., доц. каф. ПЗ
(підпис) (прізвище, ініціали, посада)

Здобувач _____ Довгалюк Д. В.
(підпис) (прізвище, ініціали)

ДОДАТОК Б
Протокол перевірки на плагіат

ПРОТОКОЛ ПЕРЕВІРКИ КВАЛІФІКАЦІЙНОЇ РОБОТИ

Назва роботи: Розробка програмного забезпечення мультимедійного бота для управління Discord каналом.

Тип роботи: бакалаврська кваліфікаційна робота
(бакалаврська кваліфікаційна робота / магістерська кваліфікаційна робота)

Підрозділ кафедра програмного забезпечення, ФІТКІ, 4ПІ-21б
(кафедра, факультет, навчальна група)

Коефіцієнт подібності текстових запозичень, виявлених у роботі системою StrikePlagiarism _____%

Висновок щодо перевірки кваліфікаційної роботи (відмітити потрібне)

■ Запозичення, виявлені у роботі, є законними і не містять ознак плагіату, фабрикації, фальсифікації. Роботу прийняти до захисту

☐ У роботі не виявлено ознак плагіату, фабрикації, фальсифікації, але надмірна кількість текстових запозичень та/або наявність типових розрахунків не дозволяють прийняти рішення про оригінальність та самостійність її виконання. Роботу направити на доопрацювання.

☐ У роботі виявлено ознаки плагіату та/або текстових маніпуляцій як спроб укриття плагіату, фабрикації, фальсифікації, що суперечить вимогам законодавства та нормам академічної доброчесності. Робота до захисту не приймається.

Експертна комісія:

_____ _____
(прізвище, ініціали, посада) (підпис)

_____ _____
(прізвище, ініціали, посада) (підпис)

Особа, відповідальна за перевірку _____ Черноволик Г. О.

З висновком експертної комісії ознайомлений(-на)

Керівник _____ Романюк О. В., к.т.н., доц. каф. ПЗ
(підпис) (прізвище, ініціали, посада)

Здобувач _____ Довгалюк Д. В.
(підпис) (прізвище, ініціали)

ДОДАТОК В

Лістинг програмного забезпечення

```
import argparse
import asyncio
import importlib.util
import json
import logging
import os
import pathlib
import shutil
import signal
import ssl
import subprocess
import sys
import textwrap
import time
from base64 import b64decode
from typing import Any, Callable, Coroutine, Dict, List, Optional, Tuple, Union

from musicbot.constants import (
    DEFAULT_LOGS_KEPT,
    DEFAULT_LOGS_ROTATE_FORMAT,
    MAXIMUM_LOGS_LIMIT,
)
from musicbot.constants import VERSION as BOTVERSION
from musicbot.exceptions import (
    HelpfulError,
    RestartCode,
    RestartSignal,
    TerminateSignal,
)
from musicbot.utils import (
    rotate_log_files,
    set_logging_level,
    set_logging_max_kept_logs,
    set_logging_rotate_date_format,
    setup_loggers,
    shutdown_loggers,
)
try:
    from aiohttp.client_exceptions import ClientConnectorCertificateError
```

```
except ImportError:
    class ClientConnectorCertificateError(Exception):  # type: ignore[no-redef]
        pass
log = logging.getLogger("musicbot.launcher")
class GIT:
    @classmethod
    def works(cls, raise_instead: bool = False) -> bool:
        """
        Checks for output from git --version to verify git can be run.
        :param: raise_instead:  Return True on success but raise Runtime error
otherwise.
        :raises:  RuntimeError  if `raise_instead` is set True.
        """
        try:
            git_bin = shutil.which("git")
            if not git_bin:
                if raise_instead:
                    raise RuntimeError(
                        "Cannot locate `git` executable in environment path."
                    )
                return False
            return bool(subprocess.check_output([git_bin, "--version"]))
        except (
            OSError,
            ValueError,
            PermissionError,
            FileNotFoundError,
            subprocess.CalledProcessError,
        ) as e:
            if raise_instead:
                raise RuntimeError(
                    f"Cannot execute `git` commands due to an error:  {str(e)}"
                ) from e
            return False

    @classmethod
    def show_branch(cls) -> str:
        """
        Runs `git rev-parse --abbrev-ref HEAD` to get the current branch name.
        Will return an empty string if running the command fails.
        """
        try:
            git_bin = shutil.which("git")
            if not git_bin:
                log.warning("Could not find git executable.")
```

```
            return ""

        gitbytes = subprocess.check_output(
            [git_bin, "rev-parse", "--abbrev-ref", "HEAD"],
            stderr=subprocess.STDOUT,
        )
        branch = gitbytes.decode("utf8").strip()

        return branch
    except (OSError, ValueError, subprocess.CalledProcessError):
        return ""

@classmethod
def check_updates(cls) -> Tuple[str, str]:
    """
    Runs `git fetch --dry-run` and extracts the commit IDs.
    If the command fails or no commit IDs are found, this
    will return empty strings rather than raise errors.
    """
    branch = cls.show_branch()
    if not branch:
        return ("", "")

    try:
        commit_at = ""
        commit_to = ""
        git_bin = shutil.which("git")
        if not git_bin:
            return ("", "")

        gitbytes = subprocess.check_output([git_bin, "fetch", "--dry-run"])
        lines = gitbytes.decode("utf8").split("\n")
        for line in lines:
            parts = line.split()
            if branch in parts:
                commits = line.strip().split(" ", maxsplit=1)[0]
                commit_at, commit_to = commits.split("..")
                break

        return (commit_at, commit_to)
    except (OSError, ValueError, subprocess.CalledProcessError):
        return ("", "")

@classmethod
def run_upgrade_pull(cls) -> None:
```

```
        """Runs `git pull` in the current working directory."""
        cls.works(raise_instead=True)

        log.info("Attempting to upgrade with `git pull` on current path.")
        try:
            git_bin = shutil.which("git")
            if not git_bin:
                raise FileNotFoundError("Could not locate `git` executable on path.")
            raw_data = subprocess.check_output([git_bin, "pull"])
            git_data = raw_data.decode("utf8").strip()
            log.info("Result of git pull:  %s", git_data)
        except (
            OSError,
            UnicodeError,
            PermissionError,
            FileNotFoundError,
            subprocess.CalledProcessError,
        ):
            log.exception("Upgrade failed, you need to run `git pull` manually.")


class PIP:
    @classmethod
    def run(cls, command: str, check_output: bool = False) -> Union[bytes, int]:
        """Runs a pip command using `sys.exectutable -m pip` through subprocess.
        Given `command` is split before it is passed, so quoted items will not work.
        """
        if not cls.works():
            raise RuntimeError("Cannot execute pip.")

        try:
            return cls.run_python_m(command.split(), check_output=check_output)
        except subprocess.CalledProcessError as e:
            return e.returncode
        except (OSError, PermissionError, FileNotFoundError):
            log.exception("Error using -m method")
        return 0

    @classmethod
    def run_python_m(
        cls, args: List[str], check_output: bool = False, quiet: bool = True
    ) -> Union[bytes, int]:
        """
        Use subprocess check_call or check_output to run a pip module
        command using the `args` as additional arguments to pip.
```

```
    The returned value of the call is returned from this method.

    :param: check_output:  Use check_output rather than check_call.
    """
    cmd_args = [sys.executable, "-m", "pip"] + args
    if check_output:
        if quiet:
            return subprocess.check_output(cmd_args, stderr=subprocess.DEVNULL)
        return subprocess.check_output(cmd_args)
    if quiet:
        return subprocess.check_call(cmd_args, stdout=subprocess.DEVNULL)
    return subprocess.check_call(cmd_args)

@classmethod
def run_install(
    cls, cmd: str, quiet: bool = False, check_output: bool = False
) -> Union[bytes, int]:
    """
    Runs pip install command and returns the command exist status.

    :param: cmd:  a string of arguments passed to `pip install`.
    :param: quiet:  attempt to silence output using -q command flag.
    :param: check_output:  return command output instead of exit code.
    """
    q_flag = "-q " if quiet else ""
    return cls.run(f"install {q_flag}{cmd}", check_output)

@classmethod
def works(cls) -> bool:
    """Checks for output from pip --version to verify pip can be run."""
    try:
        rcode = cls.run_python_m(["--version"])
        if rcode == 0:
            return True
        return False
    except subprocess.CalledProcessError:
        log.exception("PIP failed while calling sub-process.")
        return False
    except PermissionError:
        log.exception("PIP failed due to Permissions Error.")
        return False
    except FileNotFoundError:
        log.exception(
            "PIP failed due to missing Python executable?  (%s)",
            sys.executable,
```

```
            )
            return False
        except OSError:
            log.exception("PIP failed due to OSError.")
            return False

    @classmethod
    def check_updates(cls) -> List[Dict[str, Any]]:
        """
        Runs `pip install -U -r ./requirements.txt --quiet --dry-run --report -`
        and returns the number of packages that could be updated.
        """
        updata = cls.run_install(
            "-U -r ./requirements.txt --quiet --dry-run --report -",
            check_output=True,
        )
        try:
            if isinstance(updata, bytes):
                pip_data = json.loads(updata)
                ilist = pip_data.get("install", [])
                if not isinstance(ilist, list):
                    return []
                return ilist
        except json.JSONDecodeError:
            log.warning("Could not decode pip update report JSON.")

        return []

    @classmethod
    def run_upgrade_requirements(
        cls,
        get_output: bool = False,
        quiet: bool = True,
    ) -> Union[str, int]:
        """
        Uses a subprocess call to run python using sys.executable.
        Runs `pip install --no-warn-script-location --no-input -U -r ./requirements.txt`
        This method attempts to catch all exceptions and ensure a return value.

        :param: get_output:  Return the process output rather than its exit code.
            If set True, and an exception is caught, this will return the string
"[[ProcessException]]"
            If set False and an exception is caught, this will return int -255

        :returns:  process exit code, where 0 is assumed success.
```

```
        """
        if not cls.works():
            raise RuntimeError("Cannot locate or execute python -m pip")

        log.info(
            "Attempting to upgrade with `pip install --upgrade -r requirements.txt` on
current path..."
        )
        try:
            raw_data = cls.run_python_m(
                [
                    "install",
                    "--no-warn-script-location",
                    "--no-input",
                    "-U",
                    "-r",
                    "requirements.txt",
                ],
                check_output=get_output,
                quiet=quiet,
            )
            if isinstance(raw_data, bytes):
                pip_data = raw_data.decode("utf8").strip()
                log.info("Result of pip upgrade:\n%s", pip_data)
                if get_output:
                    return pip_data

            if isinstance(raw_data, int):
                log.info("Result exit code from pip upgrade: %s", raw_data)
                return raw_data

            # if somehow raw_data is not int or bytes.
            if get_output:
                return "[[OutputTypeException]]"
            return -255
        except (
            PermissionError,
            FileNotFoundError,
            OSError,
            UnicodeError,
            subprocess.CalledProcessError,
        ):
            log.exception(
                "Upgrade failed to execute or we could not understand the output"
            )
```

```
        log.warning(
            "You may need to run `pip install --upgrade -r requirements.txt` manually."
        )

        if get_output:
            return "[[ProcessException]]"
        return -255


def bugger_off(msg: str = "Press enter to continue . . .", code: int = 1) -> None:
    """Make the console wait for the user to press enter/return."""
    input(msg)
    sys.exit(code)


def sanity_checks(args: argparse.Namespace) -> None:
    """
    Run a collection of pre-startup checks to either automatically correct
    issues or inform the user of how to correct them.

    :param: optional:  Toggle optional start up checks.
    """
    log.info("Starting sanity checks")
    """Required Checks"""
    # Make sure we're on Python 3.8+
    req_ensure_py3()

    # Make sure we're in a writable env
    req_ensure_env()

    # Make our folders if needed
    pathlib.Path("data").mkdir(exist_ok=True)

    # For rewrite only
    req_check_deps()

    log.info("Required checks passed.")

    """Optional Checks"""
    if not args.do_start_checks:
        return

    # Check disk usage
    opt_check_disk_space()
```

```
    # Display an update check, if enabled.
    if not args.no_update_check:
        opt_check_updates()
    else:
        log.info("Skipped checking for updates.")

    log.info("Optional checks passed.")


def req_ensure_py3() -> None:
    """
    Verify the current running version of Python and attempt to find a
    suitable minimum version in the system if the running version is too old.
    """
    log.info("Checking for Python 3.8+")

    if sys.version_info < (3, 8):
        log.warning(
            "Python 3.8+ is required. This version is %s", sys.version.split()[0]
        )
        log.warning("Attempting to locate Python 3.8...")
        # Should we look for other versions than min-ver?

        pycom = None

        if sys.platform.startswith("win"):
            pycom = shutil.which("py.exe")
            if not pycom:
                log.warning("Could not locate py.exe")

            try:
                subprocess.check_output([pycom, "-3.8", '-c "exit()"'])
                pycom = f"{pycom} -3.8"
            except (
                OSError,
                PermissionError,
                FileNotFoundError,
                subprocess.CalledProcessError,
            ):
                log.warning("Could not execute `py.exe -3.8` ")
                pycom = None

            if pycom:
                log.info("Python 3 found.  Launching bot...")
                os.system(f"start cmd /k {pycom} run.py")
```

```
            sys.exit(0)

        else:
            log.info('Trying "python3.8"')
            pycom = shutil.which("python3.8")
            if not pycom:
                log.warning("Could not locate python3.8 on path.")

            try:
                subprocess.check_output([pycom, '-c "exit()"'])
            except (
                OSError,
                PermissionError,
                FileNotFoundError,
                subprocess.CalledProcessError,
            ):
                pycom = None

            if pycom:
                log.info(
                    "\nPython 3.8 found.  Re-launching bot using: %s run.py\n", pycom
                )
                os.execlp(pycom, pycom, "run.py")

        log.critical(
            "Could not find Python 3.8 or higher.  Please run the bot using Python 3.8"
        )
        bugger_off()
    else:
        log.info("Python version:  %s", sys.version)


def req_check_deps() -> None:
    """
    Check that we have the required dependency modules at the right versions.
    """
    try:
        import discord  # pylint: disable=import-outside-toplevel

        if discord.version_info.major < 2:
            log.critical(
                "This version of MusicBot requires a newer version of discord.py. "
                "Your version is %s. Try running update.py.",
                discord.__version__,
            )
```

```
            bugger_off()
    except ImportError:
        # if we can't import discord.py, an error will be thrown later down the line
anyway
        pass
    except AttributeError:
        # if the user has a library like dpytest installed but discord.py is missing
somehow.
        pass


def req_ensure_env() -> None:
    """
    Inspect the environment variables, validating and updating values where needed.
    """
    log.info("Ensuring we're in the right environment")

    if os.environ.get("APP_ENV") != "docker" and not os.path.isdir(
        b64decode("LmdpdA==").decode("utf-8")
    ):
        log.critical(
            b64decode(

"Qm90IHdhc24ndCBpbnN0YWxsZWQgdXNpbmcgR2l0LiBSZWluc3RhbGwgdXN
pbmcgaHR0cDovL2JpdC5seS9tdXNpY2JvdGRvY3Mu"
            ).decode("utf-8")
        )
        bugger_off()

    try:
        if not os.path.isdir("config"):
            raise RuntimeError('folder "config" not found')

        if not os.path.isdir("musicbot"):
            raise RuntimeError('folder "musicbot" not found')

        if not os.path.isfile("musicbot/__init__.py"):
            raise RuntimeError("musicbot folder is not a Python module")

        if not importlib.util.find_spec("musicbot"):
            raise RuntimeError("musicbot module is not importable")
    except RuntimeError as e:
        log.critical("Failed environment check, %s", e)
        bugger_off()
```

```
    try:
        os.mkdir("musicbot-test-folder")
    except (
        OSError,
        FileExistsError,
        PermissionError,
        IsADirectoryError,
    ):
        log.critical("Current working directory does not seem to be writable")
        log.critical("Please move the bot to a folder that is writable")
        bugger_off()
    finally:
        shutil.rmtree("musicbot-test-folder", True)

    # this actually does an access check as well.
    ffmpeg_bin = shutil.which("ffmpeg")
    if sys.platform.startswith("win"):
        if ffmpeg_bin:
            log.info("Detected FFmpeg is installed at:  %s", ffmpeg_bin)
        else:
            log.info("Adding local bins/ folder environment PATH for bundled
ffmpeg...")
            os.environ["PATH"] += ";" + os.path.abspath("bin/")
            sys.path.append(os.path.abspath("bin/"))  # might as well
            ffmpeg_bin = shutil.which("ffmpeg")
    if not ffmpeg_bin:
        log.critical(
            "MusicBot could not locate FFmpeg binary in your environment.\n"
            "Please install FFmpeg so it is available in your environment PATH variable."
        )
        if sys.platform.startswith("win"):
            log.info(
                "On Windows, you can add a pre-compiled EXE to the MusicBot `bin`
folder,\n"
                "or you can install FFmpeg system-wide using WinGet or by running the
install.bat file."
            )
        elif sys.platform.startswith("darwin"):
            log.info(
                "On MacOS, you may be able to install FFmpeg via homebrew.\n"
                "Otherwise, check the official FFmpeg site for build or install steps."
            )
        else:
            log.info(
                "On Linux, many distros make FFmpeg available via system package
```

```
managers.\n"
            "Check for ffmpeg with your system package manager or build from
sources."
        )
        bugger_off()


def opt_check_disk_space(warnlimit_mb: int = 200) -> None:
    """
    Performs and optional check of system disk storage space to warn the
    user if the bot might gobble that remaining space with downloads later.
    """
    if shutil.disk_usage(".").free < warnlimit_mb * 1024 * 2:
        log.warning(
            "Less than %sMB of free space remains on this device",
            warnlimit_mb,
        )


def opt_check_updates() -> None:
    """
    Runs a collection of git and pip commands and logs if updates are available.
    """
    log.info("\nChecking for updates to MusicBot or dependencies...")
    needs_update = False
    if GIT.works():
        git_branch = GIT.show_branch()
        commit_at, commit_to = GIT.check_updates()
        if commit_at and commit_to:
            log.warning(
                "MusicBot updates are available through `git` command.\n"
                "Your current branch is:  %s\n"
                "The latest commit ID is:  %s",
                git_branch,
                commit_to,
            )
            needs_update = True
        else:
            log.info("No MusicBot updates available via `git` command.")
    else:
        log.warning(
            "Could not check for updates using `git` commands.  You should check
manually."
        )
```

```
    if PIP.works():
        install_pkgs = PIP.check_updates()
        package_count = len(install_pkgs)
        if package_count:
            pkg_list = "The following packages can be updated:\n"
            for pkg in install_pkgs:
                pkg_meta = pkg.get("metadata", {})
                pkg_name = pkg_meta.get("name", "")
                pkg_ver = pkg_meta.get("version", "")
                if pkg_name:
                    pkg_list += f"  {pkg_name}  to version:  {pkg_ver}\n"
            log.warning(
                "There may be updates for dependency packages. "
                "PIP reports %s package(s) could be installed.\n%s",
                package_count,
                pkg_list,
            )
            needs_update = True
        else:
            log.info("No dependency updates available via `pip` command.")
    else:
        log.warning(
            "Could not check for updates using `pip` commands.  You should check
manually."
        )
    if needs_update:
        log.info(
            "You can run a guided update by using the command:\n    %s ./update.py",
            sys.executable,
        )


def parse_cli_args() -> argparse.Namespace:
    """
    Parse command line arguments and do reasonable checks and assignments.

    :returns:  Command line arguments parsed via argparse.
    """

    def kept_logs_int(value: str) -> int:
        """Validator for log rotation limits."""
        try:
            val = int(value)
            if val > MAXIMUM_LOGS_LIMIT:
                raise ValueError("Value is above the maximum limit.")
```

```
            if val <= -1:
                raise ValueError("Value must not be negative.")
            return val
        except (TypeError, ValueError) as e:
            raise argparse.ArgumentTypeError(
                f"Value for Max Logs Kept must be a number from 0 to
{MAXIMUM_LOGS_LIMIT}",
            ) from e

    def log_levels_int(level_name: str) -> int:
        """Validator for log level name to existing level int."""
        level_name = level_name.upper()
        try:
            val = getattr(logging, level_name, None)
            if not isinstance(val, int):
                raise TypeError(f"Log level '{level_name}' is not available.")
            return val
        except (TypeError, ValueError) as e:
            raise argparse.ArgumentTypeError(
                "Log Level must be one of:  CRITICAL, ERROR, WARNING, INFO,
DEBUG, "
                "VOICEDEBUG, FFMPEG, NOISY, or EVERYTHING",
            ) from e

    ap = argparse.ArgumentParser(
        formatter_class=argparse.RawDescriptionHelpFormatter,
        description=textwrap.dedent(
            """\
        Launch a music playing discord bot built using discord.py, youtubeDL, and
ffmpeg.
        Available via Github:
          https://github.com/Just-Some-Bots/MusicBot
        """
        ),
        epilog=textwrap.dedent(
            """\
        For more help and support with this bot, join our discord:
          https://discord.gg/bots

        This software is provided under the MIT License.
        See the `LICENSE` text file for complete details.
        """
        ),
    )
```

```
    ap.add_argument(
        "-V",
        "--version",
        dest="show_version",
        action="store_true",
        help="Print the MusicBot version information and exit.",
    )
    ap.add_argument(
        "--no-checks",
        dest="do_start_checks",
        action="store_false",
        help="Skip all optional startup checks, including the update check.",
    )

    ap.add_argument(
        "--no-disk-check",
        dest="no_disk_check",
        action="store_true",
        help="Skip only the disk space check at startup.",
    )

    ap.add_argument(
        "--no-update-check",
        dest="no_update_check",
        action="store_true",
        help="Skip only the update check at startup.",
    )

    ap.add_argument(
        "--no-install-deps",
        dest="no_install_deps",
        action="store_true",
        help="Disable MusicBot from trying to install dependencies when it cannot
import them.",
    )

    # Log related options.
    ap.add_argument(
        "--logs-kept",
        dest="keep_n_logs",
        default=DEFAULT_LOGS_KEPT,
        type=kept_logs_int,
        help=f"Specify how many log files to keep, between 0 and
{MAXIMUM_LOGS_LIMIT} inclusive."
        f" (Default: {DEFAULT_LOGS_KEPT})",
```

```
    )
    ap.add_argument(
        "--log-level",
        dest="log_level",
        default="NOTSET",
        type=log_levels_int,
        help="Override the log level settings set in config. Must be one of: "
        "CRITICAL, ERROR, WARNING, INFO, DEBUG, VOICEDEBUG,
FFMPEG, "
        "NOISY, or EVERYTHING   (Default: NOTSET)",
    )
    ap.add_argument(
        "--log-rotate-fmt",
        dest="old_log_fmt",
        default=DEFAULT_LOGS_ROTATE_FORMAT,
        type=str,
        help="Override the default date format used when rotating log files. "
        "This should contain values compatible with strftime().  "
        f"(Default:  '{DEFAULT_LOGS_ROTATE_FORMAT.replace('%', '%%')}')",
    )

    # TODO: maybe more arguments for other things:
    # --config-dir      force this directory for config data (all files)
    # --config-file     load config from this file, but default for other configs.

    args = ap.parse_args()

    # Show version and exit.
    if args.show_version:
        print(f"Just-Some-Bots/MusicBot\nVersion:  {BOTVERSION}\n")
        sys.exit(0)

    if -1 < args.keep_n_logs <= MAXIMUM_LOGS_LIMIT:
        set_logging_max_kept_logs(args.keep_n_logs)

    if args.log_level != logging.NOTSET:
        set_logging_level(args.log_level, override=True)

    if args.old_log_fmt != DEFAULT_LOGS_ROTATE_FORMAT:
        set_logging_rotate_date_format(args.old_log_fmt)

    return args


def setup_signal_handlers(
```

```
    loop: asyncio.AbstractEventLoop,
    callback: Callable[
        [signal.Signals, asyncio.AbstractEventLoop], Coroutine[Any, Any, None]
    ],
) -> None:
    """
    This function registers signal handlers with the event loop to help it close
    with more grace when various OS signals are sent to this process.
    """
    if os.name == "nt":
        return

    def handle_signal(sig: signal.Signals, loop: asyncio.AbstractEventLoop) -> None:
        """Creates and asyncio task to handle the signal on the event loop."""
        asyncio.create_task(callback(sig, loop), name=f"Signal_{sig.name}")
    for sig in sigs:
        loop.add_signal_handler(sig, handle_signal, sig, loop)
    setattr(loop, "_sig_handler_set", True)
def respawn_bot_process() -> None:
    """
    Use a platform dependent method to restart the bot process, without
    an external process/service manager.
    This uses the sys.executable and sys.argv to restart the bot.

    This function attempts to make sure all buffers are flushed and logging
    is shut down before restarting the new process.

    On Linux/Unix-style OS this will use sys.execlp to replace the process
    while keeping the existing PID.

    On Windows OS this will use subprocess.Popen to create a new console
    where the new bot is started, with a new PID, and exit this instance.
    """
    exec_args = [sys.executable] + sys.argv

    shutdown_loggers()
    rotate_log_files()

    sys.stdout.flush()
    sys.stderr.flush()
    logging.shutdown()

    if os.name == "nt":
        subprocess.Popen(  # pylint: disable=consider-using-with
            exec_args,
```

```
            creationflags=subprocess.CREATE_NEW_CONSOLE,  # type: ignore[attr-
defined]
        )
        print("Opened a new MusicBot instance. This terminal can be safely closed!")
        sys.exit(0)
    else:
        os.execlp(exec_args[0], *exec_args)


def set_console_title() -> None:
    """
    Attempts to set the console window title using the current version string.
    On windows, this method will try to enable ANSI Escape codes by enabling
    virtual terminal processing or by calling an empty sub-shell.
    """
    if os.name == "nt":
        try:
            import colorama  # type: ignore[import-untyped]

            # this is only available in colorama version 0.4.6+
            # which as it happens isn't required by colorlog.
            colorama.just_fix_windows_console()
        except (ImportError, AttributeError):
            # This might only work for Win 10+
            from ctypes import windll  # type: ignore[attr-defined]

            k32 = windll.kernel32
            try:
                k32.SetConsoleMode(k32.GetStdHandle(-11), 7)
            except Exception:  # pylint: disable=broad-exception-caught
                try:
                    os.system("")
                except Exception:  # pylint: disable=broad-exception-caught
                    pass
    try:
        sys.stdout.write(f"\x1b]2;MusicBot {BOTVERSION}\x07")
    except (TypeError, OSError):
        pass


def main() -> None:
    """
    All of the MusicBot starts here.
    """
    set_console_title()
```

```
    setup_loggers()

    cli_args = parse_cli_args()
    log.info("Loading MusicBot version:  %s", BOTVERSION)
    log.info("Log opened:  %s", time.ctime())

    # Check if run.py is in the current working directory.
    run_py_dir = os.path.dirname(os.path.realpath(__file__))
    if run_py_dir != os.getcwd():
        # if not, verify musicbot and .git folders exists and change directory.
        run_mb_dir = pathlib.Path(run_py_dir).joinpath("musicbot")
        run_git_dir = pathlib.Path(run_py_dir).joinpath(".git")
        if run_mb_dir.is_dir() and run_git_dir.is_dir():
            log.warning("Changing working directory to:  %s", run_py_dir)
            os.chdir(run_py_dir)
        else:
            log.critical(
                "Cannot start the bot!  You started `run.py` in the wrong directory"
                " and we could not locate `musicbot` and `.git` folders to verify"
                " a new directory location."
            )
            log.error(
                "For best results, start `run.py` from the same folder you cloned MusicBot
into.\n"
                "If you did not use git to clone the repository, you are strongly urged to."
            )
            time.sleep(3)  # make sure they see the message.
            sys.exit(127)
    sanity_checks(cli_args)
    exit_signal: Union[RestartSignal, TerminateSignal, None] = None
    event_loop: Optional[asyncio.AbstractEventLoop] = None
    tried_requirementstxt: bool = False
    use_certifi: bool = False
    retries: int = 0
    max_wait_time: int = 60

    while True:
        m = None
        try:
            if "MusicBot" not in dir():
                from musicbot.bot import (  # pylint: disable=import-outside-toplevel
                    MusicBot,
                )
            # py3.8 made ProactorEventLoop default on windows.
```

```
            # py3.12 deprecated using get_event_loop(), we need new_event_loop().
            event_loop = asyncio.new_event_loop()
            asyncio.set_event_loop(event_loop)

            # init some of bot, but don't run it yet.
            m = MusicBot(use_certifi=use_certifi)

            # register system signal handlers with the event loop.
            if not getattr(event_loop, "_sig_handler_set", False):
                setup_signal_handlers(event_loop, m.on_os_signal)

            # let the MusicBot run free!
            event_loop.run_until_complete(m.run_musicbot())
            retries = 0

        except (ssl.SSLCertVerificationError, ClientConnectorCertificateError) as e:
            if isinstance(e, ClientConnectorCertificateError) and isinstance(
                e.__cause__, ssl.SSLCertVerificationError
            ):
                e = e.__cause__
            else:
                log.critical(
                    "Certificate error is not a verification error, not trying certifi and
exiting."
                )
                log.exception("Here is the exact error, it could be a bug.")
                break

            # In case the local trust store does not have the cert locally, we can try certifi.
            # We don't want to patch working systems with a third-party trust chain
outright.
            # These verify_code values come from OpenSSL:
https://www.openssl.org/docs/man1.0.2/man1/verify.html
            if e.verify_code == 20:  #
X509_V_ERR_UNABLE_TO_GET_ISSUER_CERT_LOCALLY
                if use_certifi:  # already tried it.
                    log.exception(
                        "Could not get Issuer Certificate even with certifi!\n"
                        "Try running:  %s -m pip install --upgrade certifi ",
                        sys.executable,
                    )
                    log.warning(
                        "To easily add a certificate to Windows trust store, \n"
                        "you can open the failing site in Microsoft Edge or IE...\n"
                    )
```

```
                break

            log.warning(
                "Could not get Issuer Certificate from default trust store, trying certifi
instead."
            )
            use_certifi = True
            retries += 1
            continue

        except SyntaxError:
            if "-modded" in BOTVERSION:
                log.exception("Syntax error (version is dirty, did you edit the code?)")
            else:
                log.exception("Syntax error (this is a bug, not your fault)")
            break

        except (AttributeError, ImportError, ModuleNotFoundError) as e:
            # In case a discord extension is installed but discord.py isn't.
            if isinstance(e, AttributeError):
                if "module 'discord'" not in str(e):
                    raise

            if cli_args.no_install_deps:
                helpfulerr = HelpfulError(
                    preface="Cannot start MusicBot due to an error!",
                    issue=(
                        f"Error: {str(e)}\n"
                        "This is an error importing MusicBot or a dependency package."
                    ),
                    solution=(
                        "You need to manually install dependency packages via pip.\n"
                        "Or launch without `--no-install-deps` and MusicBot will try to install
them for you."
                    ),
                    footnote=(
                        "You have the `--no-install-deps` option set."
                        "Normally MusicBot attempts "
                    ),
                )
                log.error(str(helpfulerr))
                break

            if not PIP.works():
                log.critical(
```

```
                "MusicBot could not import dependency modules and we cannot run
`pip` automatically!\n"
                "You will need to manually install `pip` package for your version of
python.\n"
            )
            log.warning(
                "If you already installed `pip` but still get this error:\n"
                " - Check that you installed it for this python version: %s\n"
                " - Check installed packages are accessible to the user running
MusicBot",
                sys.version.split(maxsplit=1)[0],
            )
            break

        if not tried_requirementstxt:
            tried_requirementstxt = True

            log.info(
                "Attempting to install MusicBot dependency packages
automatically...\n"
            )
            pip_exit_code = PIP.run_upgrade_requirements(quiet=False)

            # If pip ran without issue, it should return 0 status code.
            if pip_exit_code:
                print()
                dep_error = HelpfulError(
                    preface="MusicBot dependencies may not be installed!",
                    issue="We didn't get a clean exit code from `pip` install.",
                    solution=(
                        "You will need to manually install dependency packages.\n"
                        "MusicBot tries to use the following command, so modify as
needed:\n"
                        "  pip install -U -r ./requirements.txt"
                    ),
                    footnote="You can also ask for help in MusicBot support server:
https://discord.gg/bots",
                )
                log.critical(str(dep_error))
                break

            print()
            log.info("OK, lets hope that worked!")
            print()
```

```
                retries += 1
                continue

            if tried_requirementstxt and retries >= 1:
                exit_signal = RestartSignal(RestartCode.RESTART_FULL)
                retries += 1
                break

            log.error(
                "MusicBot got an ImportError after trying to install packages. MusicBot
must exit..."
            )
            log.exception("The exception which caused the above error: ")
            retries = 0
            exit_signal = TerminateSignal(exit_code=1)
            break

        except HelpfulError as e:
            log.info(e.message)
            break

        except TerminateSignal as e:
            exit_signal = e
            retries = 0
            break

        except RestartSignal as e:
            if e.get_name() == "RESTART_SOFT":
                log.info("MusicBot is doing a soft restart...")
                retries = 1
                continue

            log.info("MusicBot is doing a full process restart...")
            exit_signal = e
            retries = 1
            break
        except Exception:  # pylint: disable=broad-exception-caught
            log.exception("Error starting bot")
            break
        finally:
            if event_loop:
                log.debug("Closing event loop.")
                event_loop.close()
            sleeptime = min(retries * 2, max_wait_time)
            if sleeptime:
```

```
            log.info("Restarting in %s seconds...", sleeptime)
            time.sleep(sleeptime)
    print()
    log.info("All done.")
    shutdown_loggers()
    rotate_log_files()
    print()
    if exit_signal:
        if isinstance(exit_signal, RestartSignal):
            if exit_signal.get_name() == "RESTART_FULL":
                respawn_bot_process()
            elif exit_signal.get_name() == "RESTART_UPGRADE_ALL":
                PIP.run_upgrade_requirements()
                GIT.run_upgrade_pull()
                respawn_bot_process()
            elif exit_signal.get_name() == "RESTART_UPGRADE_PIP":
                PIP.run_upgrade_requirements()
                respawn_bot_process()
            elif exit_signal.get_name() == "RESTART_UPGRADE_GIT":
                GIT.run_upgrade_pull()
                respawn_bot_process()
        elif isinstance(exit_signal, TerminateSignal):
            sys.exit(exit_signal.exit_code)
if __name__ == "__main__":
    try:
        main()
    except KeyboardInterrupt:
        print("OK, we're closing!")
        shutdown_loggers()
        rotate_log_files()

    sys.exit(0)
```

ДОДАТОК Г

Ілюстративна частина

ІЛЮСТРАТИВНА ЧАСТИНА

РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ МУЛЬТИМЕДІЙНОГО БОТА ДЛЯ УПРАВЛІННЯ DISCORD КАНАЛОМ

Рисунок Г.1 – Титульний слайд

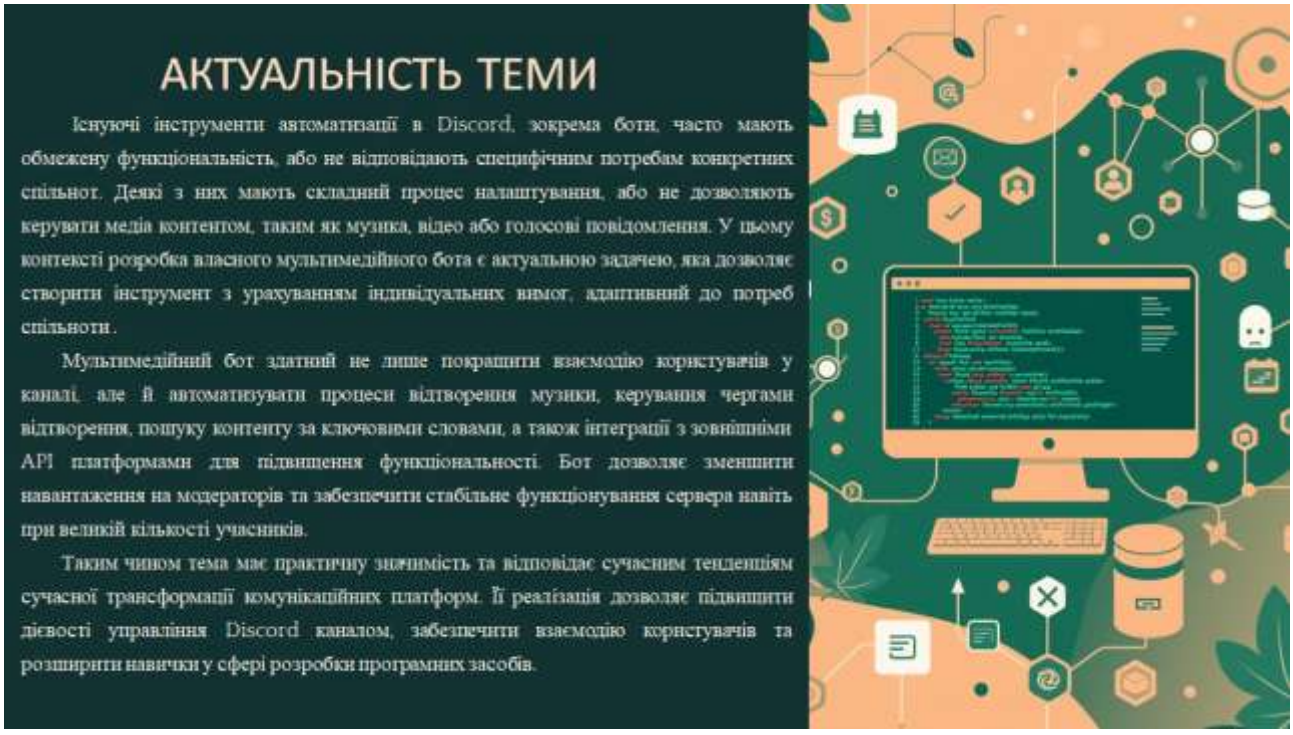

Рисунок Г.2 – Актуальність теми

Рисунок Г.3 – Мета, об’єкт дослідження та предмет дослідження

Рисунок Г.4 – Постановка задач дослідження

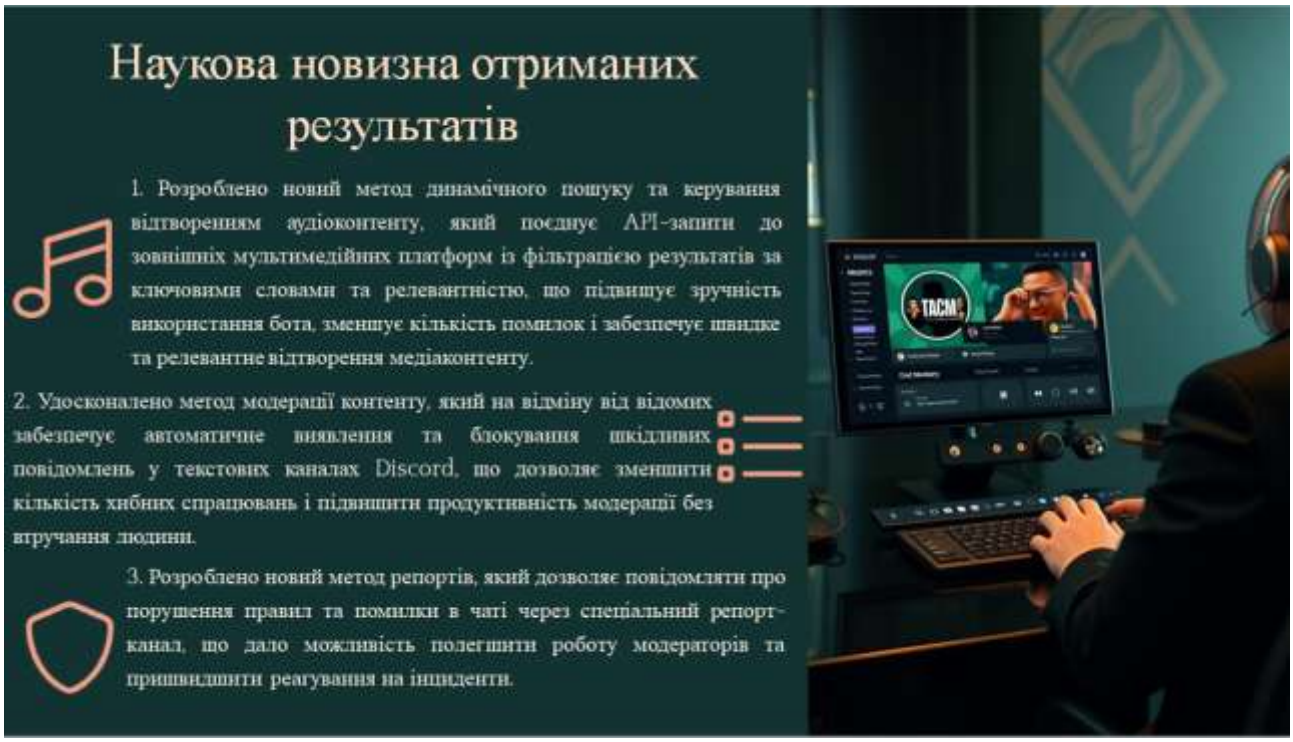

Рисунок Г.5 – Наукова новизна отриманих результатів

Функція	Rythm	Groovy	MEE6	Hydra	FredBoat	Власна розробка
Підтримка музичних платформ	1	1	0	1	0	1
Регулювання швидкості відтворення	0	1	0	1	0	0
Автоматична модерація	0	0	1	0	1	1
Кастомні команди	0	0	1	1	1	1
Інтеграція з ШІ	0	0	0	0	0	1
Підтримка відеоконтенту	0	0	0	1	0	1
Сумарний коефіцієнт	1	2	2	4	2	5

Рисунок Г.6 – Порівняльний аналіз аналогів

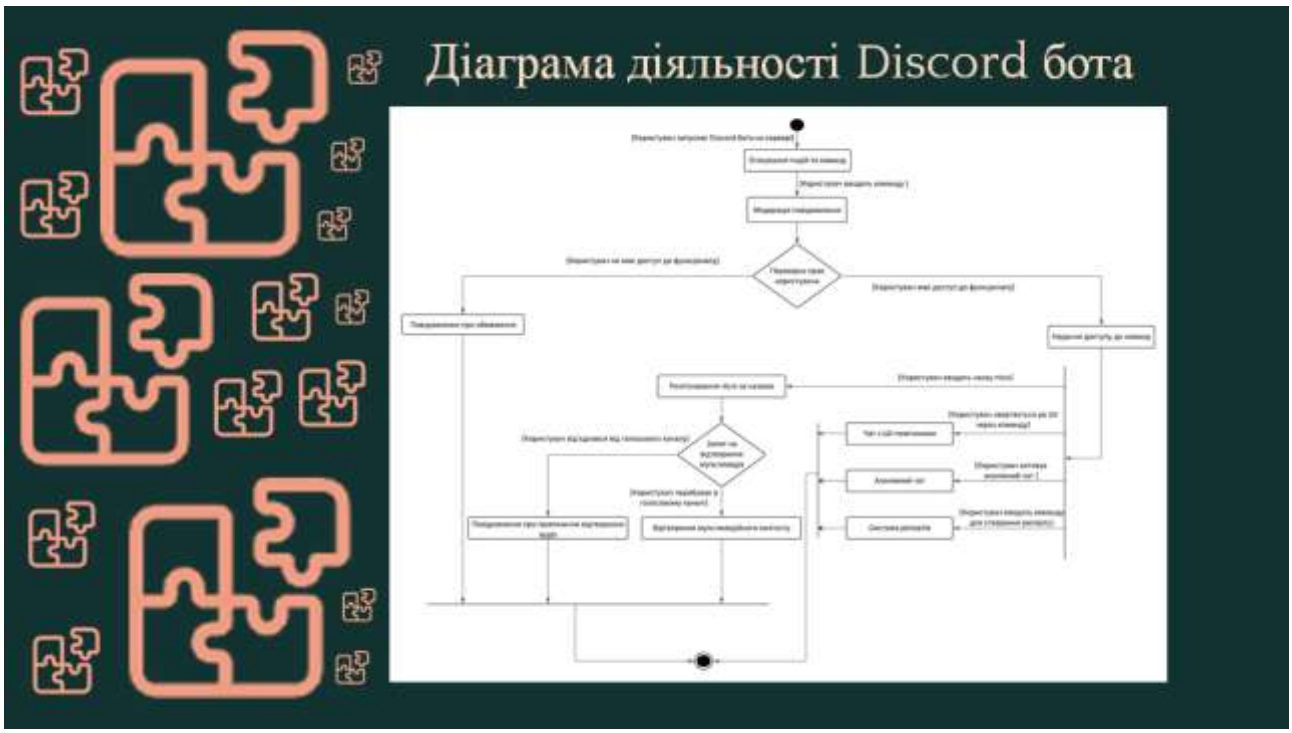

Рисунок Г.7 – Діаграма діяльності Discord бота

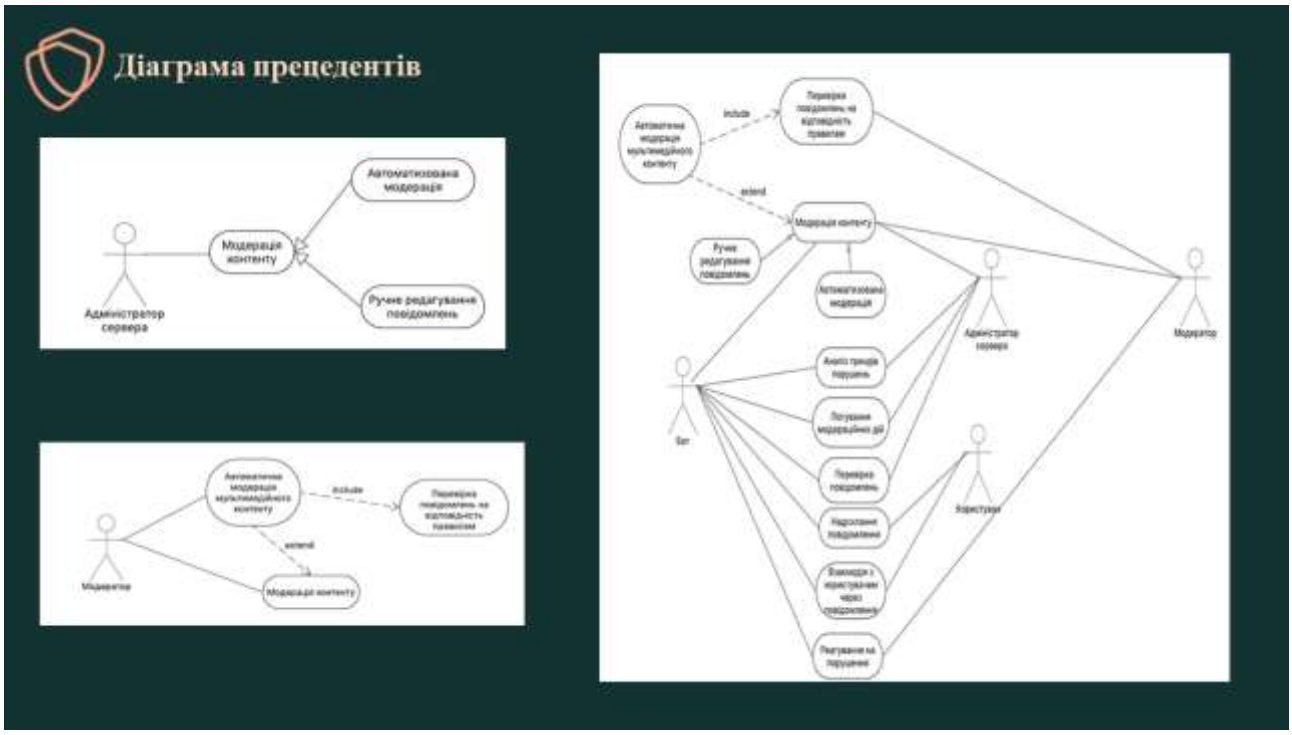

Рисунок Г.8 – Діаграма прецедентів

Рисунок Г.9 – Метод та блок-схема алгоритму репортів

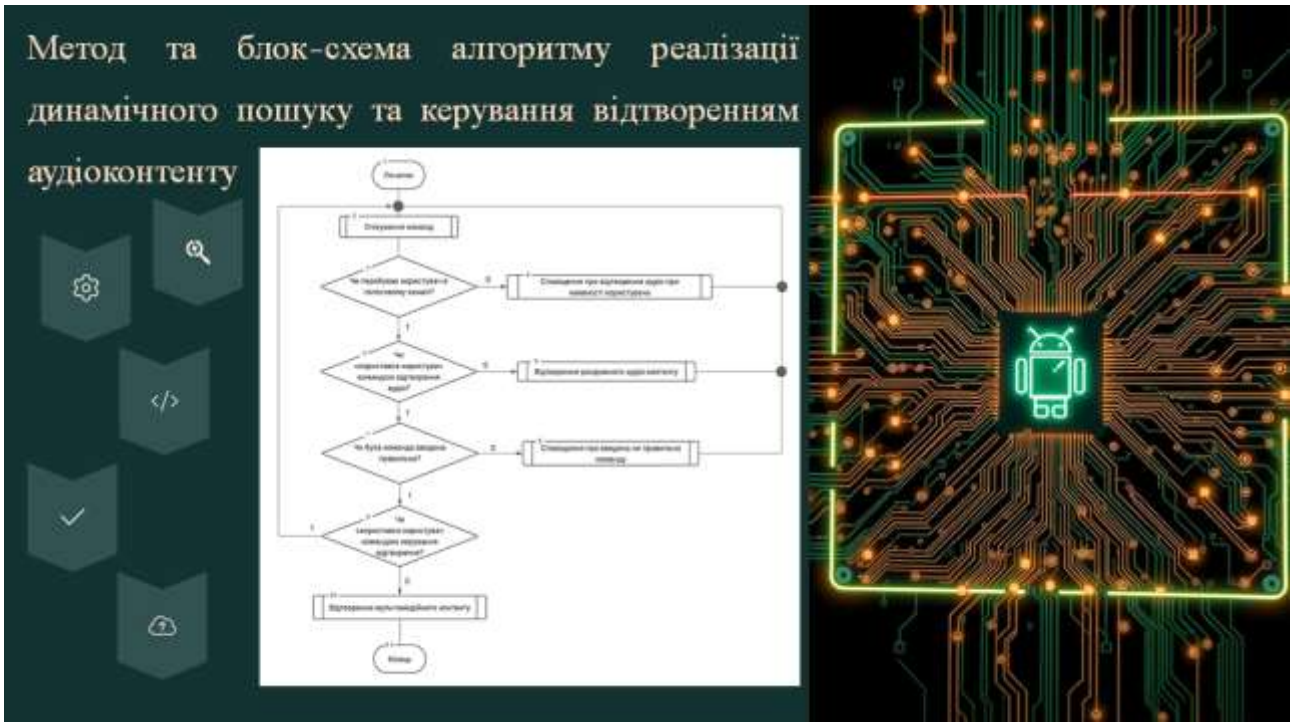

Рисунок Г.10 – Метод та блок-схема алгоритму реалізації динамічного пошуку та керування відтворенням аудіоконтенту

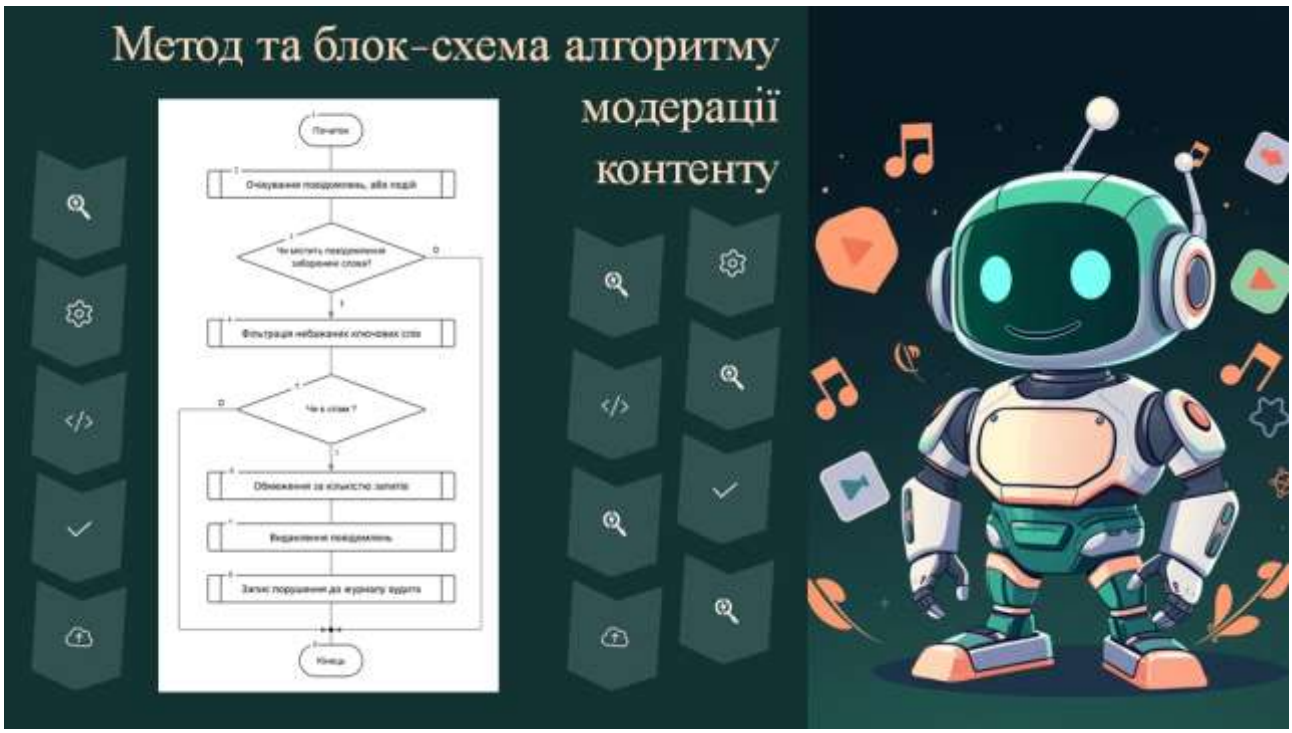

Рисунок Г.11 – Метод та блок-схема алгоритму модерації контенту

Функція	Visual Studio	NetBeans	Xcode
Підтримка Python	1	0	0
Вбудований емулятор тестування	1	0	1
Інтеграція з Git	1	1	0
Інструменти профілювання	1	0	1
Підтримка сучасних бот фреймворків	1	0	0
Екосистема плагінів	1	0	0
Сумарний коефіцієнт	6	1	2

Рисунок Г.12 – Порівняльний аналіз середовищ розробки

Функція	Python	C++	JavaScript
Підтримка бібліотек для інтеграції з Discord API	1	0	1
Розширена підтримка AI	1	0	0
Вбудовані інструменти для тестування та налагодження	1	0	1
Підтримка обробки даних в реальному часі	1	0	1
Підтримка масштабованості для великих проєктів	1	1	1
Інтеграція з хмарними платформами та сервісами	1	0	1
Сумарний коефіцієнт	6	1	5

Рисунок Г.13 – Порівняльний аналіз мов програмування

Рисунок Г.14 – Функції надсилання та збереження репортів

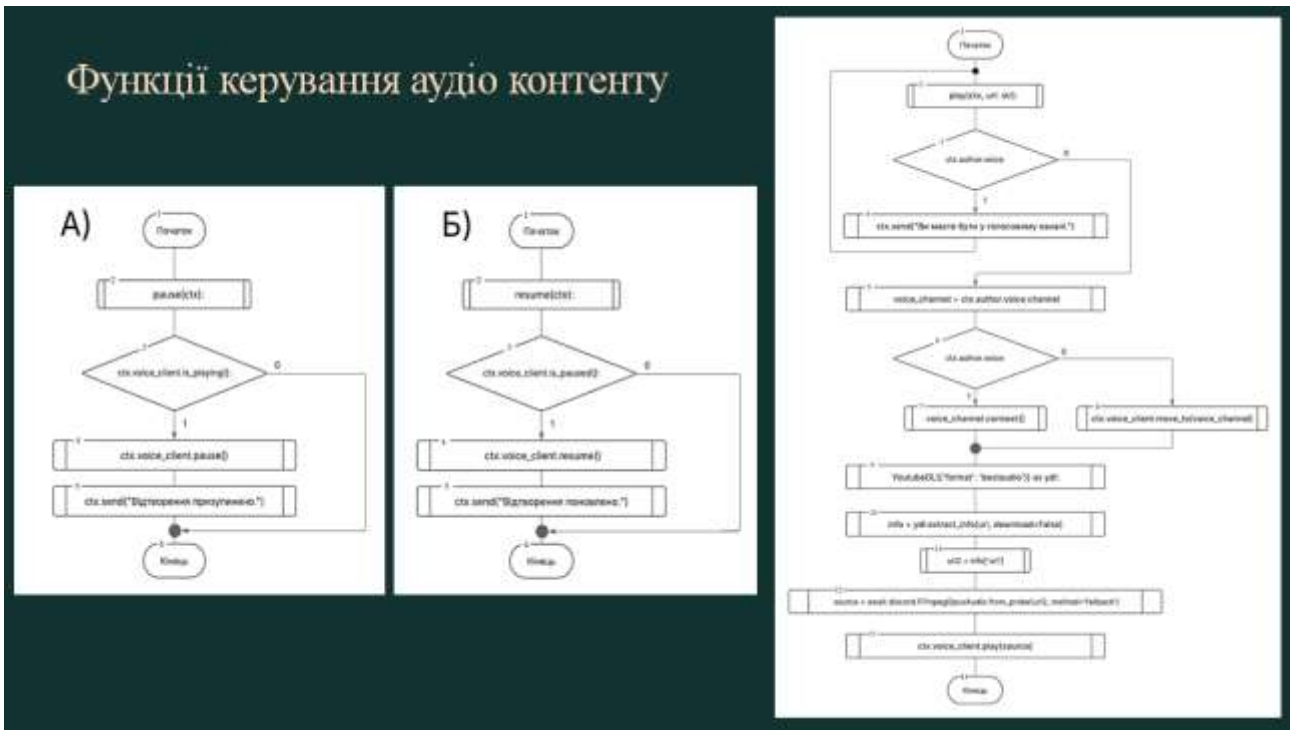

Рисунок Г.15 – Функції керування аудіо контенту

Рисунок Г.16 – Функції фільтрації, обмеження та автоматичного видалення

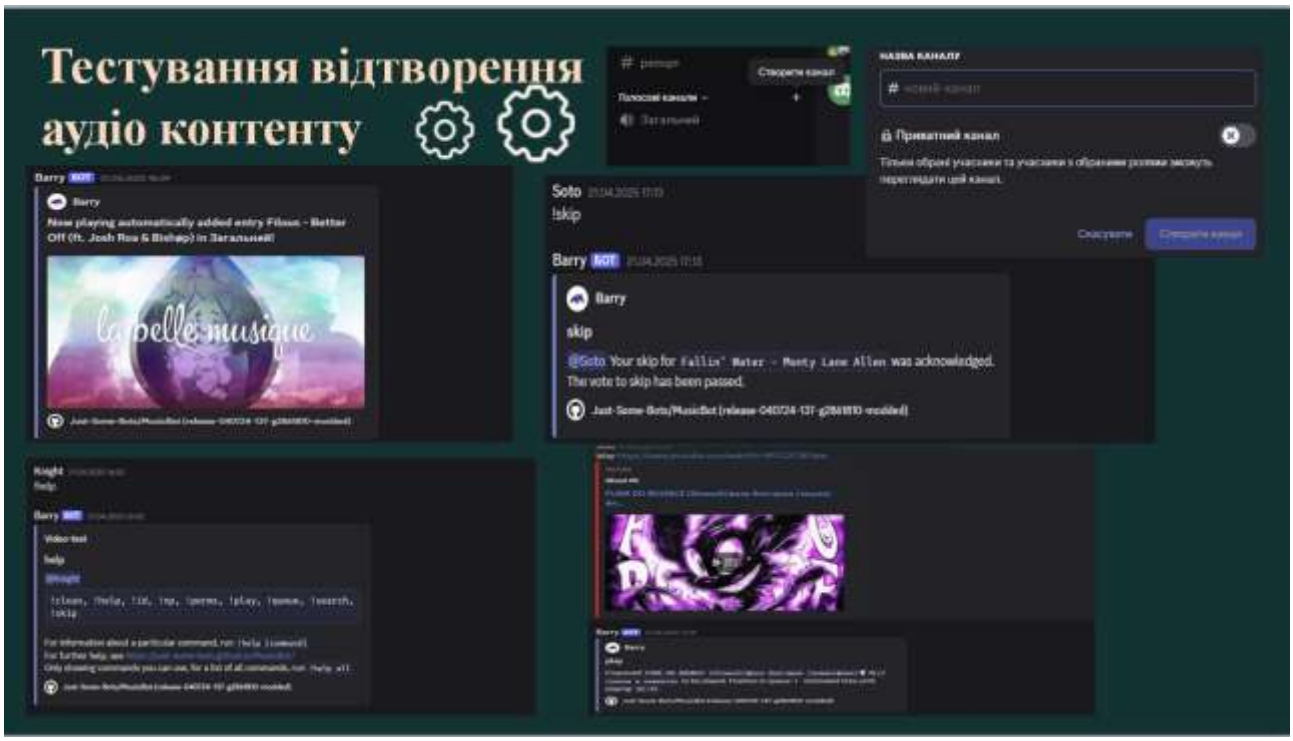

Рисунок Г.17 – Тестування відтворення аудіо контенту

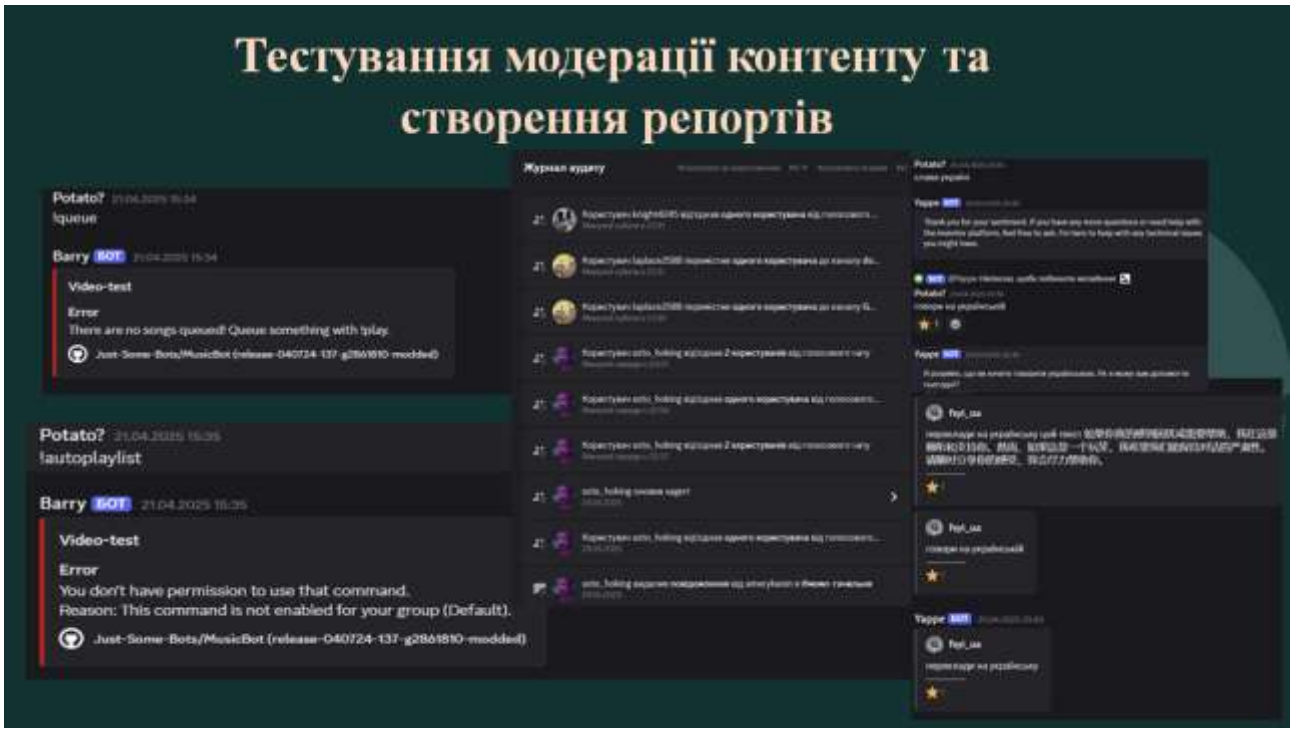

Рисунок Г.18 – Тестування модерації контенту та створення репортів

Рисунок Г.19 – Апробація та публікація матеріалів бакалаврської кваліфікаційної роботи

Рисунок Г.20 – Фінальний слайд