**Міністерство освіти і науки України**

**ВСП «ЕКОНОМІКО-ПРАВНИЧИЙ ФАХОВИЙ КОЛЕДЖ**

**ЗАПОРІЗЬКОГО НАЦІОНАЛЬНОГО УНІВЕРСИТЕТУ»**

Циклова комісія математичних дисциплін та інформаційних технологій

**КВАЛІФІКАЦІЙНА РОБОТА**

нa тeмy: **«**РОЗРОБКА БОТУ НА ОСНОВІ NODE.JS ДЛЯ ІНТЕГРАЦІЇ З СЕРВІСОМ МУЗИЧНОГО КОНТЕНТУ**»**

|  |  |
| --- | --- |
| Викoнaв: | здобувач освіти 4 кyрсy, грyпи К121-20 |
| Спeцiaльність | 121 Iнженерiя |
|  | прoгрaмнoгo зaбeзпeчeння |
|  | (шифр i спeцiaльнiсть) |
|  | Дмитро СЕЛЯВІН |
| (ім’я та ПРIЗВИЩE) | |
| Кeрiвник | Катерина РЕШЕВСЬКА |
|  | (ім’я та ПРIЗВИЩE) |
|  | |
| Рeцeнзeнт | завідувач кафедри програмної інженерії ЗНУ, к.ф.-м.н., доцент Андрій ЛІСНЯК |
|  | (пoсaдa, вчeнe звaння, нayкoвий стyпiнь, ім’я та ПРIЗВИЩE) |
|  | |

Зaпoрiжжя

2024

|  |  |  |  |
| --- | --- | --- | --- |
| ЗAПOРIЗЬКИЙ НAЦIOНAЛЬНИЙ УНIВEРСИТEТ | | | |
|  | | | |
| ВСП «Eкoнoмiкo-прaвничий фаховий кoлeдж ЗНУ» | | | |
| Oсвiтньo–квaлiфiкaцiйний ступінь | | | фаховий молодший бакалавр |
| Спeцiaльнiсть | 121 Iнженерiя прoгрaмнoгo зaбeзпeчeння | | |
|  | | (шифр i спeцiaльнiсть) | |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **ЗAТВEРДЖУЮ**  Гoлoва циклoвoї кoмiсiї мaтeмaтичних  дисциплiн тa інформаційних технологій | | | | | |
|  | | | Тетяна СМОЛЯНКОВА | | |
| (пiдпис) | | |  | | |
|  | | | | | |
| “ | 14 | ” | | червня | 2024 р. |

# ЗАВДАННЯ НА КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТУ

|  |
| --- |
| СЕЛЯВІНУ Дмитру Сергійовичу |

(прiзвищe, iм’я тa пo– бaтькoвi)

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 1. Тeмa роботи | «Розробка боту на основі Node.js для інтеграції з сервісом музичного | | | | | | | | | | |
| контенту» | | | | | | | | | | | |
|  | | | | | | | | | | | |
| Кeрiвник роботи | к.т.н., доцент каф. комп’ютерних наук РЕШЕВСЬКА Катерина Сергіївна | | | | | | | | | | |
|  | (нayкoвий стyпiнь, вчeнe звaння, прiзвищe, iм’я тa пo– бaтькoвi) | | | | | | | | | | |
|  | | | | | | | | | | | |
| Зaтвeрджeнi нaкaзoм ВСП EПФК ЗНУ вiд | | | « | 30 | » | листoпaдa | 2023 р. № | 2004–с | | |
|  | | | | | | | | | | | |
| 2. Стрoк пoдaння стyдeнтoм роботи | | | | 01.06.2024 | | | | | | | |
|  | | | | | | | | | | | |
| 3. Вихiднi дaнi дo роботи | | 1. Пoстaнoвкa зaдaчi. | | | | | | | | | |
| 2. Пeрeлiк лiтeрaтyри. | | | | | | | | | | | |
|  | | | | | | | | | | | |
|  | | | | | | | | | | | |
| 4. Змiст рoзрaхyнкoвo– пoяснювaльнoї зaписки (пeрeлiк питaнь, якi пoтрiбнo рoзрoбити) | | | | | | | | | |  | |
| 1.Сучасні інформаційні системи | | | | | | | | | | | |
| 2. Розробка проєкту інформаційної системи | | | | | | | | | | | |
| 3. Програмна реалізація інформаційної системи | | | | | | | | | | | |
| 5. Пeрeлiк грaфiчнoгo мaтeрiaлy (з тoчним зaзнaчeнням oбoв’язкoвих крeслeнь) | | | | | | | | |  | | |
| прeзeнтaцiя дo зaхистy | | | | | | | | | | | |
|  | | | | | | | | | | | |

6. Кoнсyльтaнти рoздiлiв роботи

|  |  |  |  |
| --- | --- | --- | --- |
| **Рoздiл** | **Прiзвищe, iнiцiaли тa пoсaдa кoнсyльтaнтa** | **Пiдпис, дaтa** | |
| **Зaвдaння видaв** | **Зaвдaння прийняв** |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

|  |  |
| --- | --- |
| 7. Дaтa видaчi зaвдaння | 01.12.2023 |

**КAЛEНДAРНИЙ ПЛAН**

|  |  |  |  |
| --- | --- | --- | --- |
| **№** | **Нaзвa eтaпiв кваліфікаційної роботи** | **Стрoк викoнaння eтaпiв роботи** | **Примiткa** |
| 1. | Рoзрoбкa плaнy рoбoти: |  |  |
|  | Пoстaнoвкa зaдaчi | Грудень 2023 | викoнaнo |
| 2. | Збiр вихiдних дaних, oбрoбкa мeтoдичних |  |  |
|  | тa тeoрeтичних джeрeл | Сiчень 2024 | викoнaнo |
| 3. | Рoзрoбкa пeршoгo рoздiлy: |  |  |
|  | Огляд інформаційних систем та технологій для їхнього створення | Лютий 2024 | викoнaнo |
| 4. | Рoзрoбкa дрyгoгo рoздiлy: |  |  |
|  | Розробка проєкту інформаційної системи та вибір технологій | Квiтeнь 2024 | викoнaнo |
| 5. | Рoзрoбкa трeтьoгo рoздiлy: |  |  |
|  | Рoзрoбкa застосунку | Трaвeнь 2024 | викoнaнo |
| 6. | Oфoрмлeння i нoрмoкoнтрoль кваліфікаційної роботи тa пeрeвiркa нa плaгiaт | Червень 2024 | викoнaнo |
| 7. | Зaхист кваліфікаційної роботи | 21.06.2024 | викoнaнo |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Здобувач освіти | |  |  | Дмитро СЕЛЯВІН |
|  |  | (пiдпис) |  | (ім’я ПРIЗВИЩE) |
|  |  | | | |
| Кeрiвник роботи | |  |  | Катерина РЕШЕВСЬКА |
|  |  | (пiдпис) |  | (ім’я ПРIЗВИЩE) |

**Нoрмoкoнтрoль прoйдeнo**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Нoрмoкoнтрoлeр | |  |  | Юлія БOРИСOВСЬКA |
|  |  | (пiдпис) |  | (ім’я ПРIЗВИЩE) |

# 

# РЕФЕРАТ

Кваліфікаційна робота: 49 сторінка, 1 таблиця, 7 ілюстрацій, 9 джерел, 3 додаток.

Об’єкт дослідження – основні методи та засоби розробки чат-ботів з інтеграцією сторонніх сервісів.

Предмет дослідження – бот для інтеграції з музичними сервісами, такими як YouTube та YouTube Music.

Мета дослідження – розробка боту на основі Node.js для інтеграції з сервісом музичного контенту, що дозволяє покращити користувацький досвід і автоматизувати взаємодію з музичними платформами.

У кваліфікаційному проєкті викладено теоретичні відомості про види та призначення чат-ботів, а також розглянуто приклади сучасних популярних ботів для інтеграції з музичними сервісами. На основі теоретичних відомостей розроблено проєкт боту, який використовує платформу Node.js та інтегрується з музичними сервісами через їх API. Проєкт включає розробку серверної частини, інтеграцію з YouTube та YouTube Music, а також створення інтерфейсу для взаємодії з користувачами через текстові повідомлення.

БОТ, API, NODE.JS, MEE6, ProBot, Hydra, @DeezerMusicBot, @mephbot, @LyBot.

# 

# 

# SUMMARY

Qualification work: 49 pages, 1 table, 7 illustrations, 9 sources, 3 appendices.

The object of research is the main methods and tools for developing chatbots with integration of third-party services.

The subject of the research is a bot for integration with music services such as YouTube and YouTube Music.

The purpose of the research is to develop a Node.js-based bot for integration with a music content service that will improve user experience and automate interaction with music platforms.

The qualification project provides theoretical information about the types and purpose of chatbots, as well as examples of modern popular bots for integration with music services. Based on the theoretical information, a bot project has been developed that uses the Node.js platform and integrates with music services via their API. The project includes the development of the server side, integration with YouTube and YouTube Music, and the creation of an interface for interacting with users via text messages.

БОТ, API, NODE.JS, MEE6, ProBot, Hydra, @DeezerMusicBot, @mephbot, @LyBot.

**ЗМІСТ**

[Завдання на кваліфікаційну роботу студенту 2](#_Toc168415647)

[Реферат 4](#_Toc168415648)

[Summary 5](#_Toc168415649)

[Вступ 7](#_Toc168415650)

[1 Сучасні інформаційні системи 9](#_Toc168415651)

[1.1 Поняття бот 9](#_Toc168415652)

[1.2 Класифікація боту 9](#_Toc168415653)

[1.3 Аналіз музичних ботів 10](#_Toc168415654)

[1.4 Висновок з розділом 1 19](#_Toc168415655)

[2 Розробка проєкту інформаційної системи 20](#_Toc168415656)

[2.1 Технічне завдання 20](#_Toc168415657)

[2.2 Етапи створення інформаційної системи обліку 21](#_Toc168415658)

[2.3 Діаграма прецедентів 22](#_Toc168415659)

[2.4 Проєктування інтерфейсу бота 26](#_Toc168415660)

[2.5 Висновки за розділом 2 27](#_Toc168415661)

[3 Програмна реалізація інформаційної системи 29](#_Toc168415662)

[3.1 Створення REST-full сервера 29](#_Toc168415663)

[3.2 Реалізація команд бота 30](#_Toc168415664)

[3.3 Інструкція з використання музичного бота 35](#_Toc168415665)

[3.4 Висновки за розділом 3 35](#_Toc168415666)

[Перелік використаних джерел 38](#_Toc168415667)

[Додаток А 39](#_Toc168415668)

[Додаток Б 41](#_Toc168415669)

# ВСТУП

У сучасному світі інформаційних технологій музикальні боти стають все більш важливим інструментом для покращення взаємодії бізнесу з клієнтами та забезпечення їхнього задоволення від використання продуктів або послуг. Особливо важливою є їхня роль в сфері розваг та розвитку, де музикальні боти мають значний потенціал для створення унікального користувацького досвіду.

Ціль розробленого боту полягатиме у забезпеченні користувачам можливості швидкого та зручного пошуку музичного контенту, який відповідає їхнім потребам та вподобанням. Задачі для досягнення цієї мети будуть включати реалізацію зручного інтерфейсу пошуку, інтеграцію з відповідним сервісом музичного контенту та оптимізацію швидкодії боту.

У рамках цього проєкту буде використано мову програмування Node.js, яка є популярною серед розробників через свою ефективність та гнучкість. Крім того, використання Node.js дозволить забезпечити асинхронне програмування, що є важливим для реалізації реактивного інтерфейсу боту.

Одним із ключових етапів реалізації боту буде інтеграція з сервісом музичного контенту, яка дозволить користувачам отримати доступ до широкого асортименту музики. Для цього буде використано відповідні API та сервіси, які надають доступ до музичного контенту.

Очікується, що розроблений бот стане корисним інструментом для користувачів, які шукають якісний музичний контент та хочуть отримати доступ до нього за мінімальними зусиллями. Крім того, розробка боту на основі Node.js також відкриє нові можливості для подальшого розвитку та вдосконалення функціоналу.

Задачі розробленого боту полягатимуть у забезпеченні користувачам можливості швидкого та зручного пошуку музичного контенту, який відповідає їхнім потребам та вподобанням. Для досягнення цієї мети необхідно виконати ряд технічних завдань, серед яких реалізація зручного інтерфейсу пошуку, інтеграція з відповідним сервісом музичного контенту та оптимізація швидкодії боту.

Предмет дослідження: Розробка музичного бота з використанням мови програмування Node.js та інтеграцією з відповідним сервісом музичного контенту.

Об'єкт дослідження: Технічні аспекти реалізації зручного інтерфейсу пошуку музичного контенту, оптимізація швидкодії боту, асинхронне програмування для реалізації реактивного інтерфейсу, використання відповідних API та сервісів для отримання доступу до музичного контенту.

Структура і обсяг роботи. Бакалаврська кваліфікаційна робота складається зі вступу, трьох розділів, висновків, списку використаних джерел. Загальний обсяг бакалаврської роботи – сторінки, рисунків. Список використаних джерел нараховує найменувань.

# 1 СУЧАСНІ ІНФОРМАЦІЙНІ СИСТЕМИ

## 1.1 Поняття бот

Бот (від англ. bot, скорочення від robot) – це програма, яка імітує поведінку користувача в Інтернеті. Боти можуть використовуватись для різних цілей, наприклад:

Автоматизація завдань: боти можуть автоматизувати повторювані завдання, такі як публікація повідомлень у соціальних мережах, відстеження цін на продукти або обробка замовлень.

Спілкування з користувачами: боти можуть використовуватися для спілкування з користувачами, наприклад, для надання підтримки клієнтам або для проведення опитувань.

Збір даних: боти можуть використовуватися для збору даних про користувачів або про Інтернет [1].

## 1.2 Класифікація боту

Існує багато способів класифікації ботів. Деякі поширені категорії:

1. За функціональністю:

Інформаційні боти: надають інформацію, наприклад, новини, прогноз погоди, спортивні результати.

Служби підтримки: відповідають на запитання та надають допомогу користувачам.

Транзакційні боти: дозволяють користувачам здійснювати покупки, бронювати квитки, переказувати гроші тощо.

Розважальні боти: грають з користувачами, розповідають анекдоти, генерують контент.

1. За платформою:

Веб-боти: працюють у веб-браузерах.

Мобільні боти: працюють у мобільних додатках.

Месенджери: працюють у месенджерах, таких як Telegram, Facebook Messenger, WhatsApp.

1. За технологією:

На основі правил: ґрунтуються на наборі правил, щоб визначити, як відповідати на вхідні дані.

На основі машинного навчання: використовує машинне навчання для розуміння вхідних даних та генерування відповіді [2].

## 1.3 Аналіз музичних ботів

Основні функції:

Пошук за назвою пісні, виконавцем, альбомом, жанром та URL-посиланням.

Відтворення музики з різних джерел такі як Youtube Music, Youtube, Spotify тощо.

Створення та управління плейлистами.

Основні платформи для відтворення музики є Discord, а для пошуку є Telegram.

Бізнес-моделі можуть включати в себе: підписки, рекламу та партнерські програми.

Також відрізняються команди в ботах. Наприклад, якщо в Discord користувач захоче увімкнути музику, то треба вводити команди play, вставляти посилання або назву, щоб бот знайшов потрібну музику і відтворював її, або якщо треба додати музику в чергу queue, то бот буде додавати музику в чергу. А для пошукового бота в Telegram треба вводити команду search і назву або якщо користувач не пам'ятає назву, то ввести рядок із музики.

Найпопулярнішими ботами, які відтворюють музику є: MEE6 [3], ProBot [4], Hydra [5] тощо.

MEE6: один з найпопулярніших ботів Discord, який пропонує широкий спектр функцій, включаючи відтворення музики з YouTube, Spotify, SoundCloud та інших джерел. MEE6 також має систему рівнів, яка дозволяє користувачам отримувати доступ до додаткових функцій, а також можливість модерації чату та інші корисні інструменти (див. рис. 1.1).

Перелік команд, які використовуються для бота:

play: відтворює музику з YouTube, Spotify, SoundCloud або URL-адреси.

pause: призупиняє відтворення музики.

resume: продовжує відтворення музики.

skip: пропускає поточну пісню.

queue: показує чергу пісень.

clearqueue: очищає чергу пісень.

shuffle: перемішує чергу пісень.

repeat: повторює поточну пісню або весь плейлист.

volume: встановлює гучність.

join: приєднується до голосового каналу.

leave: покидає голосовий канал.

help: показує список команд.

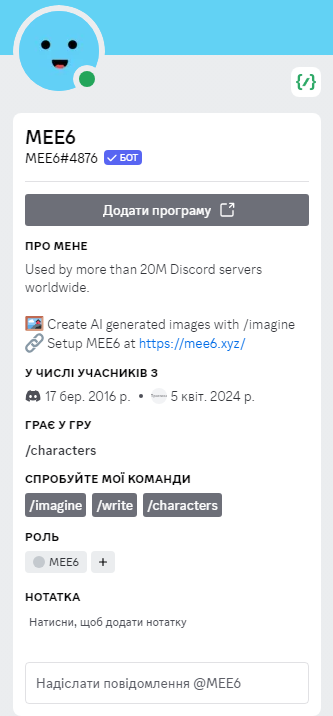


Рисунок 1.1 – Візуальний складник бота MEE6

ProBot: ще один популярний музичний бот, який пропонує багато тих же функцій, що й MEE6, а також можливість створювати власні плейлисти, трансляції з радіо та багато іншого. ProBot також має систему авто-DJ, яка може автоматично відтворювати музику на основі ваших уподобань (див. рис. 1.2).

Перелік команд, які використовуються для бота:

play: відтворює музику з YouTube, Spotify, SoundCloud або URL-адреси.

pause: призупиняє відтворення музики.

resume: продовжує відтворення музики.

skip: пропускає поточну пісню.

queue: показує чергу пісень.

clearqueue: очищає чергу пісень.

shuffle: перемішує чергу пісень.

repeat: повторює поточну пісню або весь плейлист.

volume: встановлює гучність.

join: приєднується до голосового каналу.

leave: покидає голосовий канал.

help: показує список команд.

playlist: створює, редагує та видаляє плейлисти.

radio: запускає радіостанцію.

autodj: автоматично відтворює музику на основі ваших уподобань.

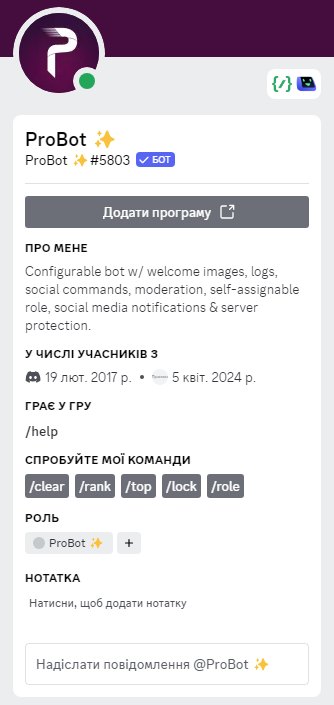


Рисунок 1.2 – Візуальний складник бота ProBot

Hydra: потужний музичний бот, який може відтворювати музику з YouTube, Twitch, SoundCloud, Bandcamp та інших джерел. Hydra також має багато додаткових функцій, таких як трансляції з радіо, караоке, голосові команди та багато іншого (див. рис. 1.3).

Перелік команд, які використовуються для бота:

play: відтворює музику з YouTube, Twitch, SoundCloud, Bandcamp або URL-адреси.

pause: призупиняє відтворення музики.

resume: продовжує відтворення музики.

skip: пропускає поточну пісню.

queue: показує чергу пісень.

clearqueue: очищає чергу пісень.

shuffle: перемішує чергу пісень.

repeat: повторює поточну пісню або весь плейлист.

volume: встановлює гучність.

join: приєднується до голосового каналу.

leave: покидає голосовий канал.

help: показує список команд.

playlist: створює, редагує та видаляє плейлисти.

radio: запускає радіостанцію.

karaoke: запускає караоке.

voice: використовує голосові команди для керування ботом.

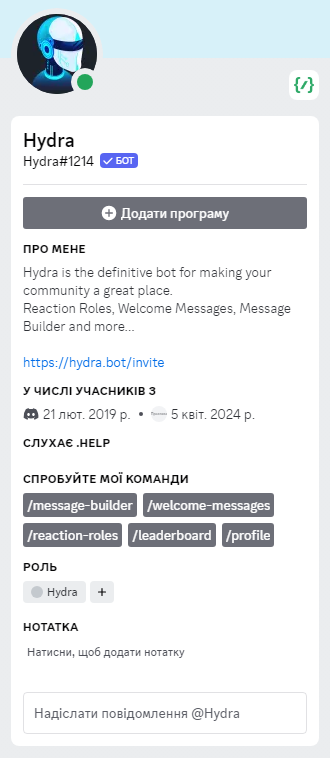


Рисунок 1.3 – Візуальний складник бота Hydra

Найпопулярнішими ботами, які находять музику є: @DeezerMusicBot [6], @mephbot [7],@LyBot [8] тощо.

@DeezerMusicBot: бот, який інтегрується з Deezer, щоб надати вам доступ до мільйонів пісень. Ви можете шукати пісні за назвою, виконавцем, альбомом або жанром, а також створювати власні плейлисти (див. рис. 1.4).

Перелік команд, які використовуються для бота:

/start Start

/donate Donate

/quality Set a quality of Deezer songs (donated users only)

/links enable/disable links with songs

/spotify\_now Info about currently playing Spotify song (donated users only)

/spotify\_dl\_now Download currently playing Spotify song (donated users only)

/lastfm\_scrobble Scrobble on Last.fm

/lastfm\_love Love on Last.fm

/spotify\_logout Logout from Spotify

/lastfm\_logout Logout from Last.fm

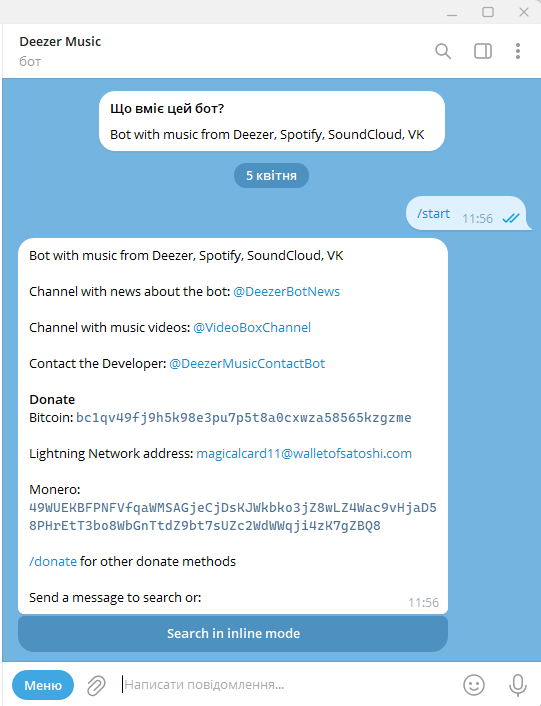


Рисунок 1.4 – Візуальний складник бота @DeezerMusicBot

@mephbot: бот, який може шукати пісні на YouTube, Spotify, SoundCloud та інших джерелах. Ви можете шукати пісні за назвою, виконавцем, альбомом або жанром, а також прослухати уривок пісні перед тим, як додати її до свого плейлиста (див. рис. 1.5).

Перелік команд, які використовуються для бота:

/song Search by title or artist

/artist Search by artist

/settings Settings

/help Information

/donate — support authors

/premium — premium info

/my — list your playlist

/top — popular songs

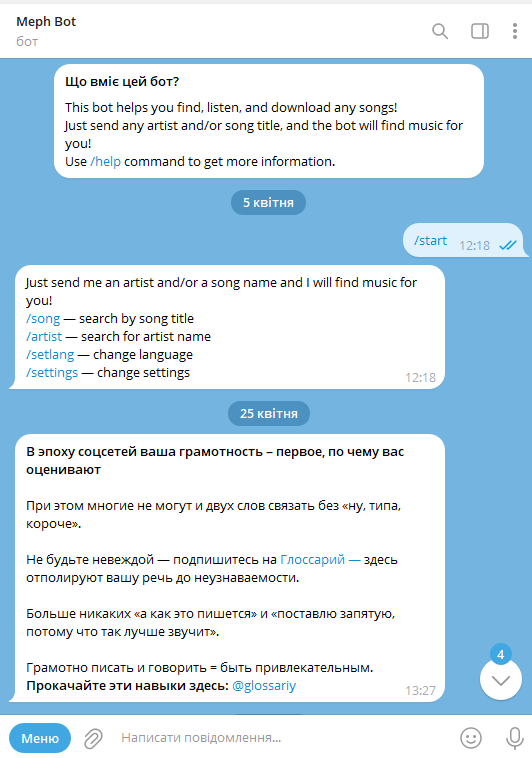


Рисунок 1.5 – Візуальний складник бота @mephbot

@LyBot: багатофункціональний бот для пошуку та прослуховування музики у YouTube Music. Він має великий каталог музики, зручний пошук, якісний звук і багато інших корисних функцій. Бот є безкоштовним, але в ньому присутня реклама (див. рис. 1.6).

Перелік команд, які використовуються для бота:

/help Help

/language Language

/legal Legal info

/stop Clear queue

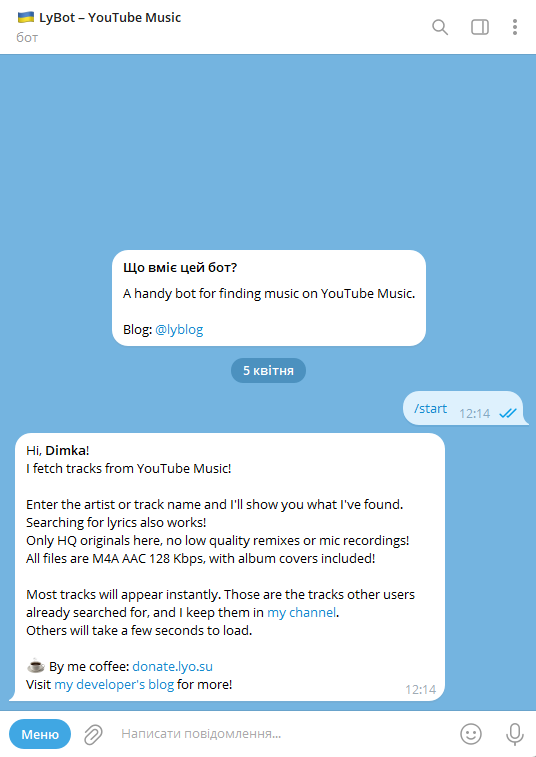


Рисунок 1.6 – Візуальний складник бота @LyBot

## 1.4 Висновок з розділом 1

Аналіз музичних ботів показав, що ця технологія вже широко застосовується для відтворення та пошуку музики через платформи, такі як: Discord, Telegram тощо. Основні функції таких ботів включають пошук за різними критеріями, відтворення музики з різних джерел і створення плейлистів.

Бізнес-моделі цих ботів можуть включати в себе різноманітні стратегії, такі як: підписки, реклама та партнерські програми. Це дозволяє розробникам заробляти гроші на своїй роботі і забезпечує подальший розвиток та підтримку продукту.

Важливою особливістю таких ботів є їхня різноманітність у командах та інтерфейсі. Від користувачів вимагається розуміння того, як взаємодіяти з кожним конкретним ботом, щоб виконати потрібні дії, такі як відтворення музики чи пошук конкретного треку.

Найпопулярніші музичні боти для відтворення музики на Discord включають: MEE6, ProBot та Hydra, тоді як для пошуку музики в Telegram використовуються боти, такі як: @DeezerMusicBot, @mephbot та @LyBot.

У цілому, музичні боти стають все більш важливими інструментами для спілкування та розваги у віртуальних спільнотах, надаючи користувачам доступ до широкого спектру музичного контенту простим і зручним способом.

# 2 РОЗРОБКА ПРОЄКТУ ІНФОРМАЦІЙНОЇ СИСТЕМИ

## 2.1 Технічне завдання

**2.1.1 Найменування і область застосування**

Цей бот призначений для полегшення доступу до музичного контенту. Він надасть користувачам можливість шукати музику за URL-посиланням з YouTube та YouTube Music і керувати відтворенням, таким як пауза, відновлення, пропуск та зупинка. Цей бот буде інтегровано в чат Discord, щоб користувачі могли керувати музикою, не залишаючи чат.

**2.1.2 Підстава для розробки**

Цей бот розробляється з наступних причин:

Зростання популярності музичних сервісів: музичні сервіси, такі як YouTube Music та YouTube, стають все більш популярними. Цей бот полегшить користувачам доступ до цих сервісів та керування ними.

Потреба в більш зручному способі доступу до музичного контенту: традиційні способи доступу до музики, такі як пошук в Інтернеті або використання мобільних додатків, можуть бути громіздкими та незручними. Цей бот пропонує більш зручний та інтуїтивно зрозумілий спосіб доступу до музики.

Можливість використання ботів для автоматизації завдань: боти можуть використовуватися для автоматизації завдань, таких як пошук музики та створення списків відтворення. Це може заощадити час користувачам та зробити їх досвід користування музикою більш приємним.

**2.1.3 Призначення розробки**

Призначенням розробки цього бота є створення зручного, функціонального інструменту для доступу до сервісу музичного контенту та без підписок. Бот має бути простим у використанні та надавати користувачам широкий спектр можливостей.

**2.1.4 Технічні вимоги до програмного продукту**

Програмне забезпечення: реалізація бота буде здійснюватися з використанням Node.js.

Сервіс музичного контенту: для доступу до музичного контенту будуть використовуватися YouTube та YouTube Music.

Інтерфейс: бот буде інтегрований у чатовий інтерфейс Discord.

## 2.2 Етапи створення інформаційної системи обліку

Основними етапами створення інформаційної системи обліку є:

1. аналіз вимог, який складається з визначення цілей і завдань бота, аналізу цільової аудиторії та її потреб, визначення функціональних і нефункціональних вимог, вибору сервісу музичного контенту для інтеграції (наприклад, Spotify, Apple Music);
2. проєктування системи, яке складається з розробки архітектури системи, музичних плейлистів, історії запитів, визначення API для взаємодії з музичним сервісом, розробки інтерфейсу користувача (якщо бот матиме веб або мобільний інтерфейс);
3. розробка бота, яка включає в себе налаштування середовища розробки (встановлення Node.js, необхідних пакетів, налаштування серверу), реалізацію серверної частини бота на Node.js, інтеграцію з API музичного сервісу (отримання доступу, реалізація запитів);
4. тестування, яке полягає в перевірці роботи з різними типами запитів, коректності обробки даних, правильності отримання даних, стійкості системи при високому навантаженні та обробці помилок;
5. впровадження, яке складається з розгортання бота на сервері (вибір хостингу, налаштування серверного середовища), з налаштування безпеки (SSL-сертифікати, захист від DDoS-атак, безпека API ключів), з моніторингу роботи бота (налаштування логування, систем моніторингу та оповіщення);
6. підтримка та вдосконалення полягає в постійному відстеженні роботи бота, аналізі журналів роботи, виправленні помилок і оптимізації коду, додаванні нових функцій та оновленні відповідно до змін у вимогах користувачів та сервісу музичного контенту, випуску регулярних оновлень і підтримці сумісності з новими версіями API музичного сервісу.

## 2.3 Діаграма прецедентів

Діаграма прецедентів (або діаграма варіантів використання) (англ. Use case diagram) – в UML, діаграма, на якій зображено відношення між акторами та прецедентами в системі [9].

Діаграма прецедентів (див. табл. 2.1) показує різні варіанти використання та різні типи користувачів системи і часто супроводжується іншими типами діаграм.

Таблиця 2.1 – Діаграма прецендентів

|  |  |
| --- | --- |
| **Актор** | **Прецедент** |
| Користувач | Відтворити музику за URL  Пауза музики  Відновити відтворення музики  Пропустити поточну пісню  Зупинити відтворення музики  Додати пісню в чергу  Покинути канал |

Варіанти використання представлені колами або еліпсами (див. рис. 2.1). Актори (дійові особи) часто зображуються у вигляді паличок [9].

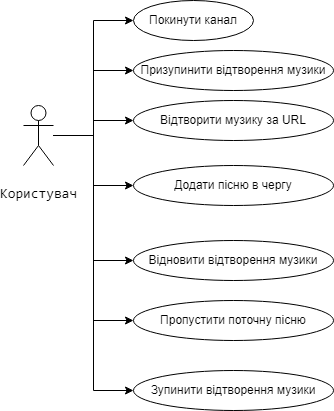
****

Рисунок 2.1 – Діаграма прецендентів

Прецедент «Відтворити музику за URL»

Призначення: даний варіант використання надає можливість користувачу відтворити музику за вказаним URL.

Основний потік подій:

Прецедент починає виконуватися, коли користувач надсилає команду «відтворити музику за URL» разом з посиланням на пісню.

Система перевіряє валідність URL.

Якщо URL валідний, система починає відтворення пісні за вказаним URL.

Виняткова ситуація 1:

Якщо URL невалідний або пісня недоступна, система відправляє повідомлення про помилку.

Прецедент «Пауза музики»

Призначення: даний варіант використання надає можливість користувачу призупинити відтворення музики.

Основний потік подій:

Прецедент починає виконуватися, коли користувач надсилає команду «пауза музики».

Система призупиняє відтворення поточної пісні.

Виняткова ситуація 1:

Якщо музика вже на паузі, система відправляє повідомлення про те, що музика вже призупинена.

Прецедент «Відновити відтворення музики»

Призначення: даний варіант використання надає можливість користувачу відновити відтворення музики після паузи.

Основний потік подій:

Прецедент починає виконуватися, коли користувач надсилає команду «відновити відтворення музики».

Система відновлює відтворення музики з того місця, де вона була призупинена.

Виняткова ситуація 1:

Якщо музика не була на паузі, система відправляє повідомлення про те, що музика вже відтворюється.

Прецедент «Пропустити поточну пісню»

Призначення: даний варіант використання надає можливість користувачу пропустити поточну пісню та перейти до наступної.

Основний потік подій:

Прецедент починає виконуватися, коли користувач надсилає команду «пропустити поточну пісню».

Система переходить до відтворення наступної пісні в черзі.

Альтернативний потік:

Якщо черга порожня, система відправляє повідомлення про відсутність пісень для відтворення.

Прецедент «Зупинити відтворення музики»

Призначення: даний варіант використання надає можливість користувачу повністю зупинити відтворення музики.

Основний потік подій:

Прецедент починає виконуватися, коли користувач надсилає команду «зупинити відтворення музики».

Система зупиняє відтворення поточної пісні та очищає чергу.

Прецедент «Додати пісню в чергу»

Призначення: даний варіант використання надає можливість користувачу додати нову пісню в чергу для відтворення.

Основний потік подій:

Прецедент починає виконуватися, коли користувач надсилає команду «додати пісню в чергу» разом з інформацією про пісню.

Система додає вказану пісню в чергу.

Виняткова ситуація 1:

Якщо інформація про пісню некоректна, система відправляє повідомлення про помилку та пропонує повторити спробу.

Прецедент «Покинути канал»

Призначення: даний варіант використання надає можливість користувачу змусити бота покинути голосовий канал.

Основний потік подій:

Прецедент починає виконуватися, коли користувач надсилає команду «покинути канал».

Система відключає бота від голосового каналу.

Виняткова ситуація 1:

Якщо бот вже не підключений до каналу, система відправляє повідомлення про це.

## 2.4 Проєктування інтерфейсу бота

Для проєктування інтерфейсу першим кроком є розуміння вимог до бота. Бот повинен виконувати такі функції: пошук музики, створення плейлистів, рекомендації, управління відтворенням.

Під час проєктування інтерфейсу будуть інтегровані такі сервіси музичного контенту: YouTube Music, YouTube.

Архітектура проєкту бота включає в себе основні компоненти системи, такі як: бекенд на Node.js (основний сервер, що обробляє запити користувачів), інтеграційні модулі (окремі модулі для роботи з API музичних сервісів), комунікаційні канали, через які користувачі взаємодіють з ботом.

Щоб бот працював, йому потрібно задати спеціальні команди, такі як:

* відключення бота від голосового каналу: /leave,
* пауза музики:/pause,
* відтворення музики: /play <URL-адреса>,
* додати до черги:/queue <URL-адреса>,
* відновити відтворення музики:/resume,
* пропустити пісню:/skip,
* зупинити відтворення музики:/stop.

Взаємодія бота з користувачем повинна бути зрозумілою та зручною. Вона полягає у використанні команд для Discord-бота та відправці повідомлень з чіткими інструкціями і прикладами команд.

Основними кроками розробки бота є: створення нового проєкту Node.js і встановлення необхідних залежностей (discord.js,@discordjs/voice,play-dl); створення сервера для обробки запитів; обробка команд бота; реалізація бота на Node.js.

## 2.5 Висновки за розділом 2

Потреба в боті для доступу до музичного контенту: зростання популярності музичних сервісів: вивчення ринку музичного контенту показало, що платформи, такі як YouTube Music та YouTube, набирають на популярності. Це свідчить про потребу у зручному доступі до музичного контенту.

Необхідність зручного інтерфейсу: традиційні методи пошуку та відтворення музики можуть бути не ефективними та незручними для користувачів. Розробка бота, який інтегрується з чатовим інтерфейсом Discord, має на меті полегшити цей процес.

Потенціал автоматизації завдань: ефективне використання часу користувачів: Боти можуть автоматизувати завдання, пов'язані з пошуком музики та керуванням відтворенням. Це дозволяє користувачам зосередитися на відтворенні музики, не витрачаючи час на пошук та налаштування.

Важливість інтерфейсу та зручності використання: інтуїтивно зрозумілі команди: розроблення набору команд, які легко розуміють користувачі, допомагає забезпечити зручність взаємодії з ботом.

Лаконічні та інформативні відповіді: важливо, щоб бот надавав користувачам зрозумілі та інформативні відповіді на їхні запити, спрощуючи процес взаємодії.

Збір зворотного зв'язку: важливо збирати відгуки користувачів та вносити відповідні зміни в інтерфейс для поліпшення його зручності та ефективності.

Процес тестування та впровадження:

Ретельне тестування перед впровадженням: перед введенням бота в експлуатацію важливо провести ретельне тестування, щоб виявити та виправити всі можливі помилки.

Постійне вдосконалення: після впровадження бота важливо збирати зворотний зв'язок від користувачів та вносити відповідні зміни для поліпшення його функціональності та інтерфейсу.

Отже, розробка бота для інтеграції з музичними сервісами має на меті забезпечити зручний та ефективний доступ до музичного контенту через чатовий інтерфейс, а також забезпечити можливість автоматизації завдань для полегшення життя користувачів.

# 3 ПРОГРАМНА РЕАЛІЗАЦІЯ ІНФОРМАЦІЙНОЇ СИСТЕМИ

На етапі проєктування інформаційної системи було вирішено створити бота на основі NODE.JS із застосуванням бібліотек Discord.js та play-dl. Для демонстрації роботи бота було знайдено хостинг на безкоштовній основі.

## 3.1 Створення REST-full сервера

У цьому розділі розглянемо процес розробки бота на Node.js для інтеграції з музичним сервісом. Основною метою є створення функціонального бота, який може взаємодіяти з користувачами через платформу Discord, забезпечуючи можливість відтворення музики з різних джерел. Використання бібліотек Discord.js та play-dl дозволить реалізувати необхідний функціонал для роботи з музичним контентом.

**3.1.1 Вибір технологій**

Node.js - популярна платформа для розробки серверних застосунків на основі мови JavaScript.

Discord.js - потужна бібліотека для роботи з API Discord.

play-dl - бібліотека для роботи з музичним контентом, яка підтримує різні музичні сервіси та дозволяє легко завантажувати та відтворювати музику.

**3.1.2 Створення бота**

1. Ініціалізація проєкту:

Створення нового проєкту на Node.js за допомогою npm (Node Package Manager). Встановлення необхідних залежностей: discord.js та play-dl.

|  |
| --- |
| npm init -y  npm install discord.js play-dl |

**3.1.3 Налаштування бота**

Лістинг створення основного файлу (index.js) для бота та імпорт необхідних модулів представлено у Додатку А.

Лістинг створення файлу з командами (commands.js) для бота та імпортуємо необхідні модулі представлено у Додатку Б.

Лістинг створення файлу з токеном (config.js) для бота має наступний вигляд:

|  |
| --- |
| export const DISCORD\_TOKEN = "MTEzMDE3MTkxNzA1MjE0OTg4MQ.G1OPxt.kSUcbdgRwkEn8u-xFyDxy67ZvyZ251MPKv7\_pM"; |

## 3.2 Реалізація команд бота

Реалізація необхідних команд для керування ботом:

Відтворити музику за допомогою URL:

|  |
| --- |
| const handlePlayCommand = async (interaction, options) => {  try {  const voiceChannel = interaction.member?.voice.channel;  if (!voiceChannel) {  interaction.reply('Ви повинні знаходитись у голосовому каналі, щоб використовувати цю команду.');  return;  }  const audioUrl = options.getString('url');  try {  const streamData = await stream(audioUrl);  const connection = joinVoiceChannel({  channelId: voiceChannel.id,  guildId: interaction.guildId,  adapterCreator: interaction.guild.voiceAdapterCreator,  });  const player = createAudioPlayer({  behaviors: {  noSubscriber: NoSubscriberBehavior.Play  }  });  const resource = createAudioResource(streamData.stream, {  inputType: streamData.type  });  const subscription = connection.subscribe(player);  player.play(resource);  interaction.reply(`Приєднався та почав відтворення в каналі ${voiceChannel.name}. Посилання на трек:${audioUrl}`);  player.on('idle', () => {  if (player.queue && player.queue.length > 0) {  const nextResource = player.queue.shift();  player.play(nextResource);  }  });  } catch (error) {  interaction.reply('Ви ввели неправильне посилання.');  }  } catch (error) {  console.error(error);  interaction.reply('Виникла помилка при обробці команди play.');  }  }; |

Пауза музики:

|  |
| --- |
| const handlePauseCommand = (interaction) => {  const connection = getVoiceConnection(interaction.guildId);  if (!connection) {  interaction.reply('Бот не знаходиться у голосовому каналі.');  return;  }  const player = connection.state.subscription.player;  if (player) {  player.pause();  interaction.reply('Відтворення музики призупинено.');  } else {  interaction.reply('В даний момент музика не грає.');  }  }; |

Відновити відтворення музики:

|  |
| --- |
| const handleResumeCommand = (interaction) => {  const connection = getVoiceConnection(interaction.guildId);  if (!connection) {  interaction.reply('Бот не знаходиться у голосовому каналі.');  return;  }  const player = connection.state.subscription.player;  if (player) {  player.unpause();  interaction.reply('Music playback resumed.');  } else {  interaction.reply('В даний момент музика не поставлена на паузу.');  }  }; |

Пропустити поточну пісню:

|  |
| --- |
| const handleSkipCommand = async (interaction) => {  try {  const connection = getVoiceConnection(interaction.guildId);  if (!connection) {  interaction.reply('Бот не знаходиьтся у голосовому каналі.');  return;  }  const player = connection.state.subscription.player;  if (!player) {  interaction.reply('В даний момент музика не грає.');  return;  }  if (!player.queue || player.queue.length === 0) {  interaction.reply('В черзі відсутня наступна пісня.');  return;  }  player.stop();  const nextResource = player.queue.shift();  player.play(nextResource);  interaction.reply('Пропустити поточну пісню, що відтворюється.');  } catch (error) {  console.error(error);  interaction.reply('При обробці команди "Пропустити" виникла помилка.');  }  }; |

Зупинити відтворення музики:

|  |
| --- |
| const handleStopCommand = (interaction) => {  const connection = getVoiceConnection(interaction.guildId);  if (!connection) {  interaction.reply('Бот не знаходиться у голосовому каналі.');  return;  }  const player = connection.state.subscription.player;  if (player) {  player.stop();  interaction.reply('Відтворення музики зупинено.');  } else {  interaction.reply('В даний момент музика не грає.');  }  }; |

Додати пісню в чергу:

|  |
| --- |
| const handleQueueCommand = async (interaction, options) => {  try {  const voiceChannel = interaction.member?.voice.channel;  if (!voiceChannel) {  interaction.reply('Для використання цієї команди ви повинні знаходитись у голосовому каналі.');  return;  }  const audioUrl = options.getString('url');  try {  const streamData = await stream(audioUrl);  if (!streamData || !streamData.stream) {  interaction.reply('Не вдалось отримати дані аудіопотоку.');  return;  }  const connection = getVoiceConnection(interaction.guildId);  if (!connection) {  interaction.reply('Бот не знаходиться у голосовому каналі.');  return;  }  const player = connection.state.subscription.player;  if (!player.queue) {  player.queue = [];  }  const resource = createAudioResource(streamData.stream, {  inputType: streamData.type  });  player.queue.push(resource);  player.on('end', () => {  if (player.queue.length > 0) {  const nextResource = player.queue.shift();  player.play(nextResource);  }  });  interaction.reply({  content: `Додано в чергу. Посилання на трек: ${audioUrl}`,  ephemeral: false,  });  } catch (error) {  interaction.reply('Ви ввели неправильне посилання.');  }  } catch (error) {  console.error(error);  interaction.reply('При обробці команди черги виникла помилка.');  }  }; |

Залишити канал:

|  |
| --- |
| const handleLeaveCommand = (interaction) => {  const currentConnection = getVoiceConnection(interaction.guild.id);  if (currentConnection) {  currentConnection.destroy();  interaction.reply('Покинув голосовий канал.');  } else {  interaction.reply('Бот не знаходиться у поточному каналі');  }  }; |

## 

## 3.3 Інструкція з використання музичного бота

Крок 1: Додавання бота на сервер:

1. Перейдіть на сайт Discord Developer Portal.
2. Натисніть на бота.
3. Потім натисніть на OAuth2.
4. Додайте йому привілеї.
5. Скопіюйте посилання.
6. Виберіть, в який сервер додати бота.
7. Натисніть кнопки «продовжити» й «авторизувати».

Крок 2: Прослуховування музики через бота:

1. Зайдіть в голосовий канал.
2. Напишіть або оберіть у списку команду /play <URL-посилання>.

Основні команди для використання бота:

/leave Відключення бота від голосового каналу (див. рис. 3.1)

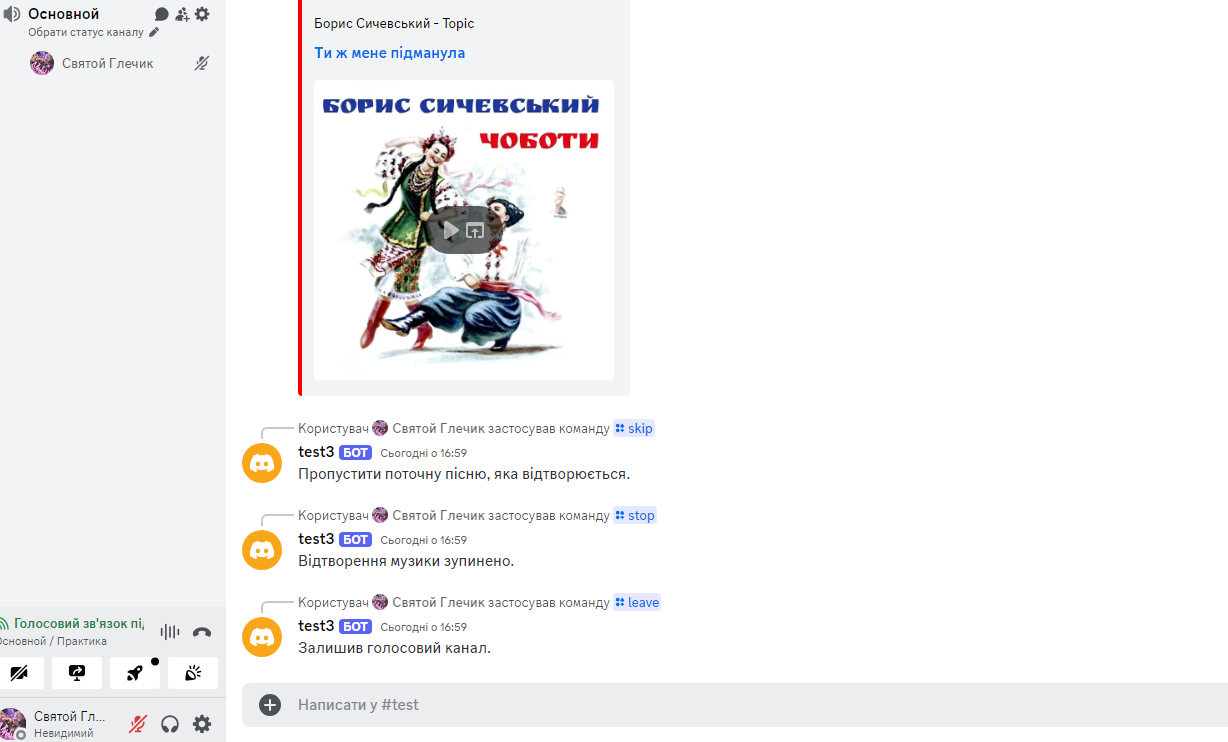


Рисунок 3.1 – Команда /leave

/pause Пауза відтворення (див. рис. 3.2)

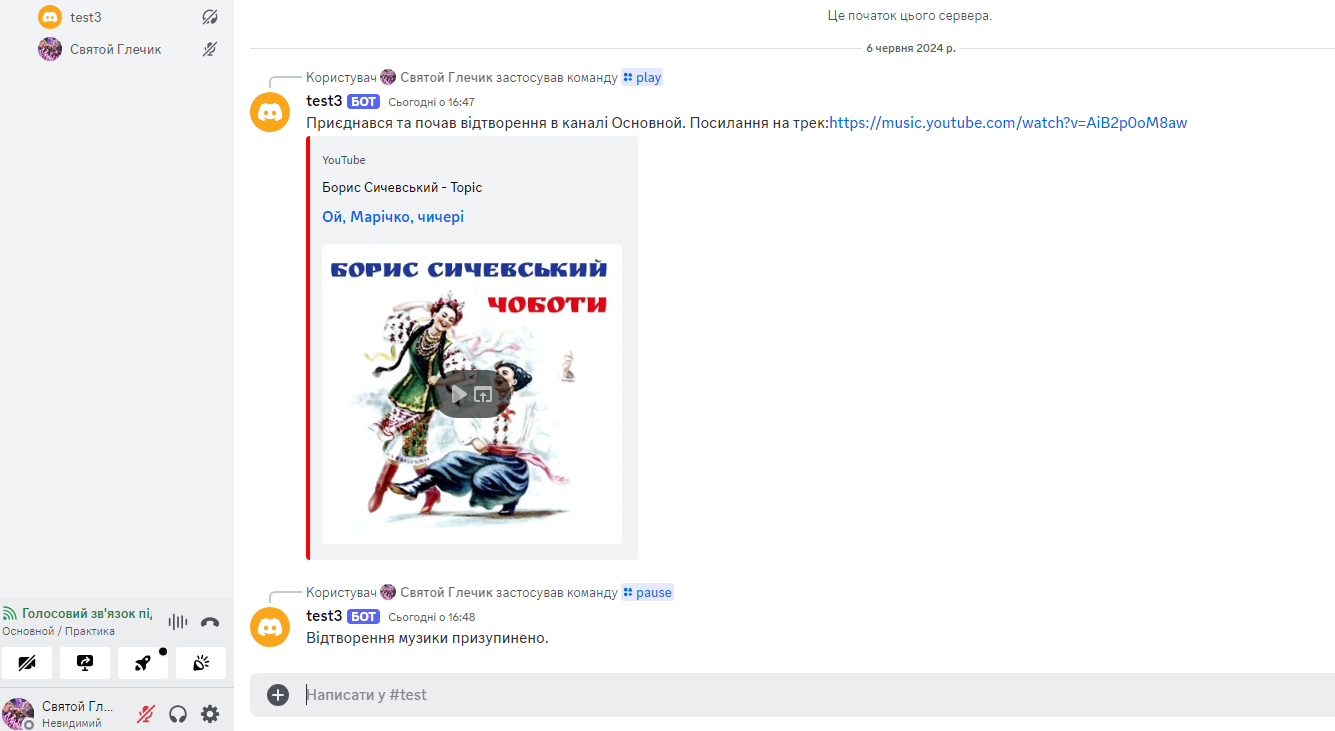


Рисунок 3.2 – Команда /pause

/play <Url-посилання> Підключення та відтворення музики (див. рис. 3.3)

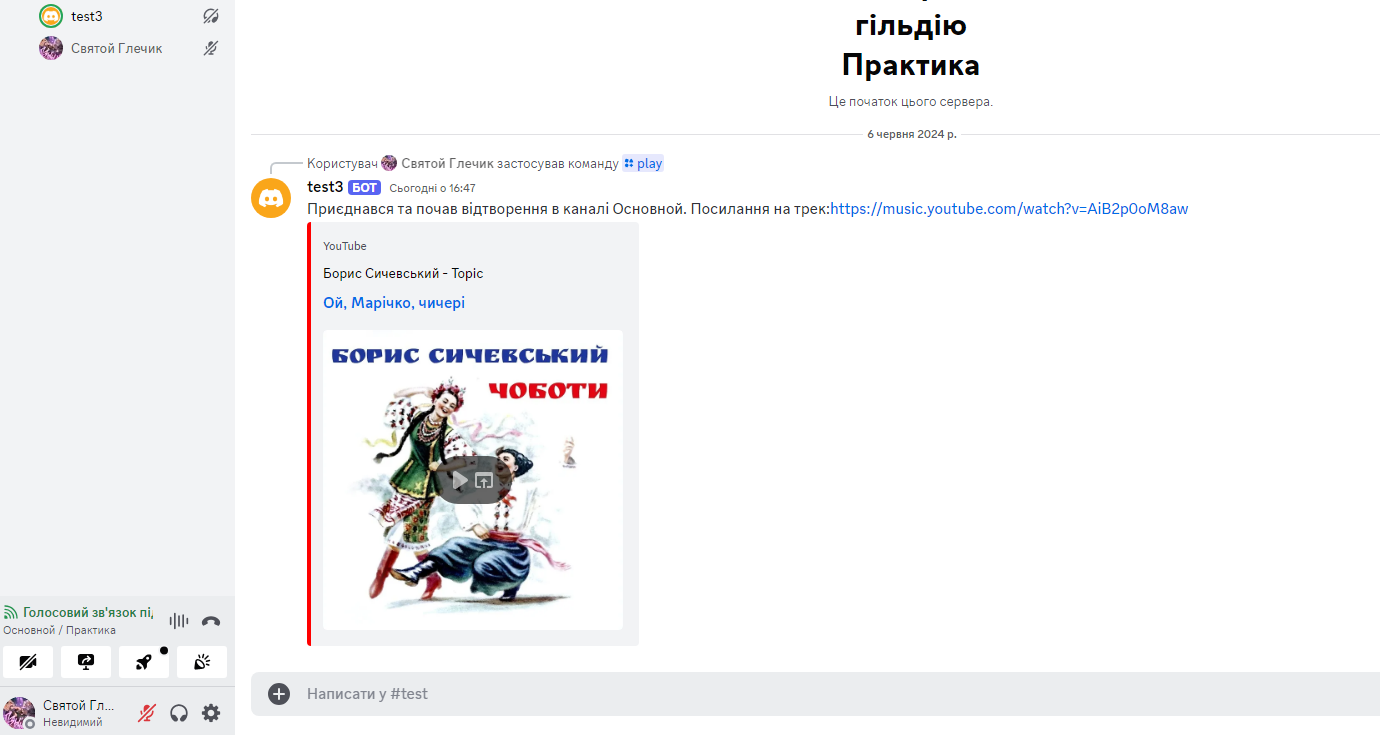


Рисунок 3.3 – Команда /play

/queue <Url-посилання> Сторення черги (див. рис. 3.4)

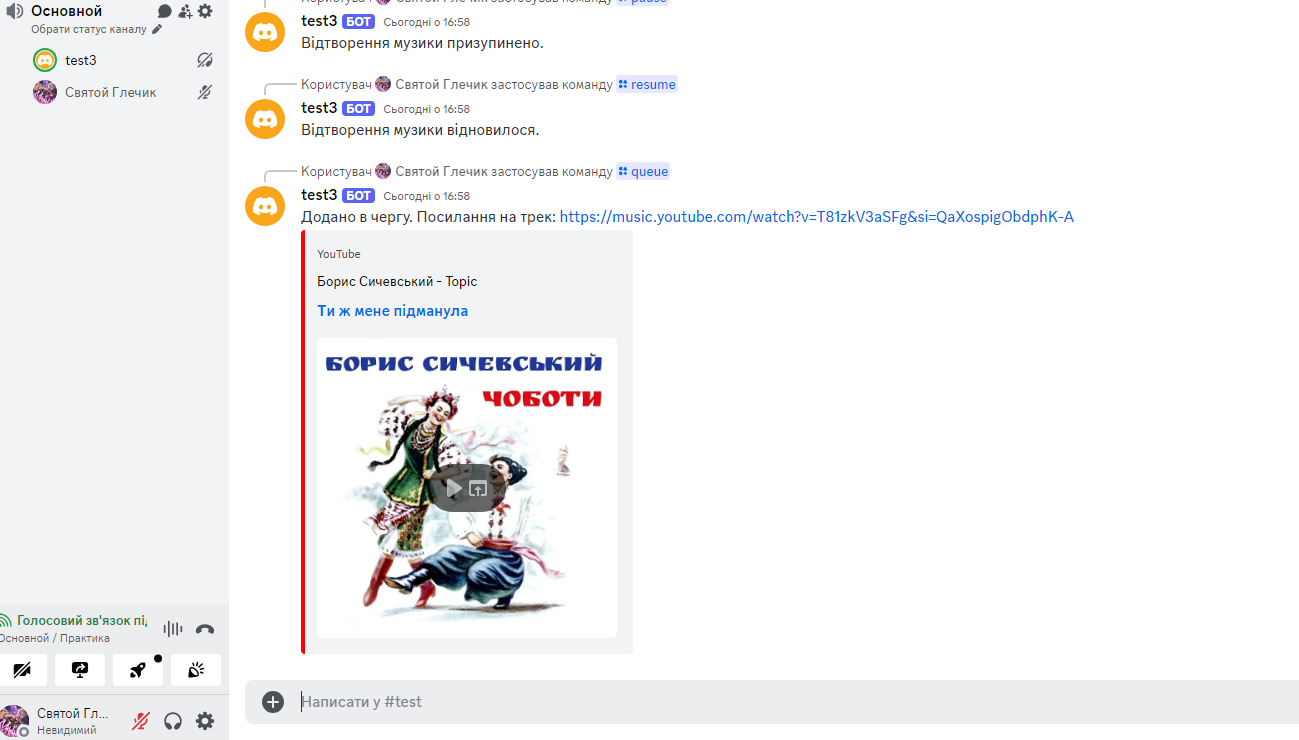


Рисунок 3.4 – Команда /queue

/resume Продовження відтворення (див. рис. 3.5)

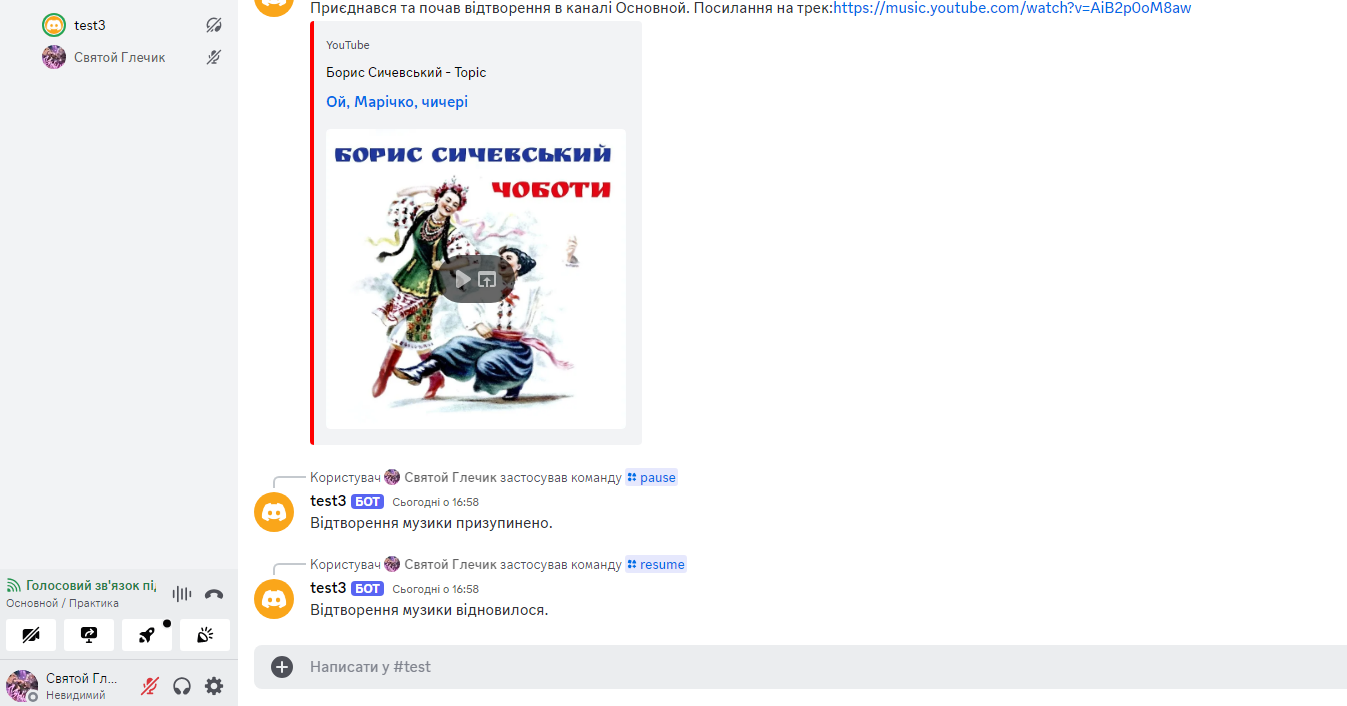


Рисунок 3.5 – Команда /resume

/skip Пропуск пісні (див. рис. 3.6)

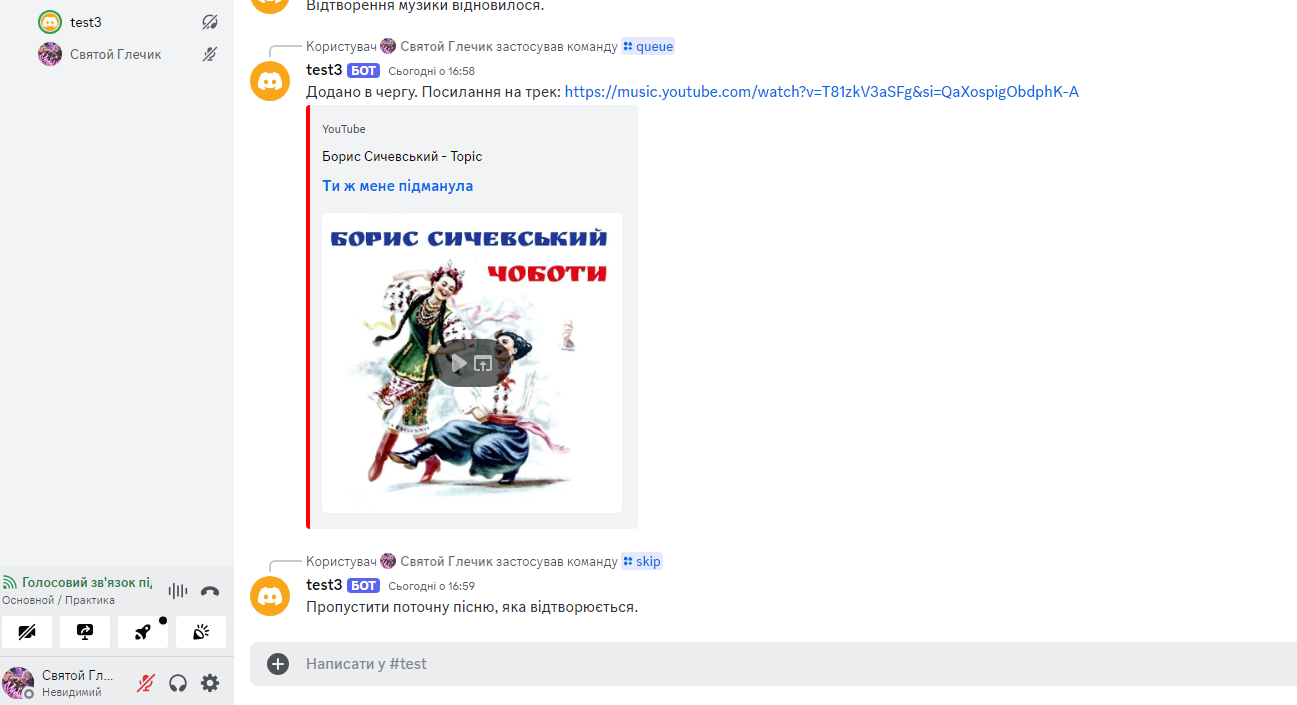


Рисунок 3.6 – Команда /skip

/stop Зупинка відтворення (див. рис. 3.7)

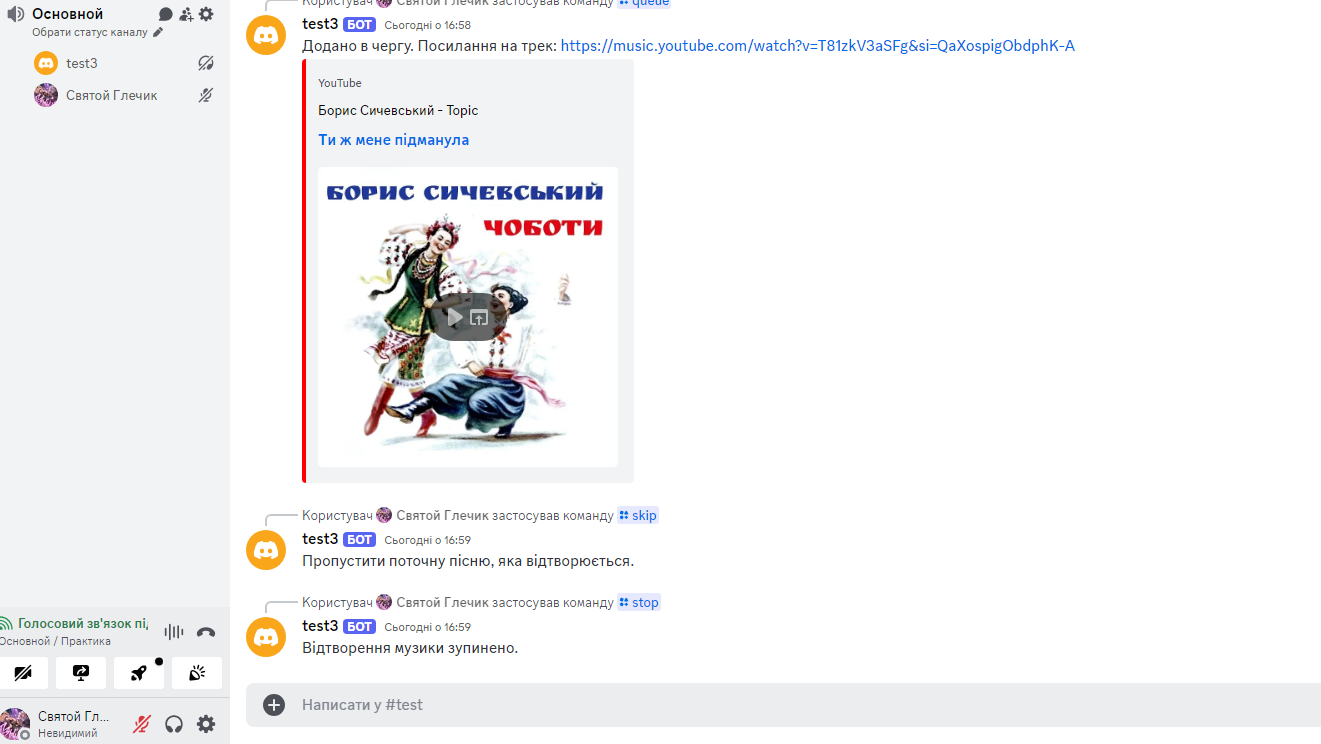


Рисунок 3.7 – Команда /stop

## 3.4 Висновки за розділом 3

У цьому розділі було описано процес розробки бота на Node.js для інтеграції з музичним сервісом. Було створено REST-full сервер, який може взаємодіяти з користувачами через платформу Discord, забезпечуючи можливість відтворення музики з різних джерел.

В результаті проєкт успішно виконано відповідно до поставлених вимог до розробки. Бот відповідає поставленим вимогам і може бути використаний для відтворення музики на платформі Discord.

**ВИСНОВКИ**

У кваліфікаційній роботі було реалізовано модуль бота для інтеграції з сервісом музичного контенту на основі Node.js. Для досягнення поставленої мети вирішено наступні завдання:

1. проведено огляд існуючого інструментарію для створення ботів;
2. вивчено основні поняття та термінологію сервісів музичного контенту;
3. розглянуто методи інтеграції з сервісами музичного контенту за допомогою бібліотек play-dl та discord.js;
4. розроблено алгоритм взаємодії з сервісом музичного контенту;
5. реалізовано програмний модуль бота для музичного сервісу з використанням Node.js, бібліотек play-dl та discord.js.

Досягнення:

1. розроблено бота на node.js з використанням бібліотек discord.js та play-dl;
2. забезпечено можливість відтворення музики з різних джерел;
3. реалізовано команди для керування ботом: відтворення, пауза, пропуск, зупинка, додавання до черги, вихід з каналу;
4. бот успішно протестовано на безкоштовному хостингу.

Подальший розвиток:

1. додати більше команд, наприклад, пошук музики, створення плейлистів, регулювання гучності;
2. розширити можливості бота, наприклад, інтеграція з іншими сервісами, голосове керування;
3. оптимізувати код бота для покращення його продуктивності.

Розроблений бот може бути використаний у музичних сервісах як потужний засіб для здійснення швидкого, гнучкого та релевантного пошуку та відтворення музичного контенту.

# ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Що таке бот. URL: <https://gerabot.com/article/sho_take_bot> (дата звернення: 17.04.2024).

2. Все про чат-боти: типи і приклади, якому бізнесу підійде, список конструкторів для створення. URL: <https://web-promo.ua/ua/blog/vse-o-chat-botah-tipy-i-primery-kakomu-biznesu-podojdet-spisok-konstruktorov-dlya-sozdaniya/> (дата звернення: 24.04.2024).

3. MEE6. URL: <https://mee6.xyz/en> (дата звернення: 19.04.2024).

4. ProBot. URL: <https://probot.io/> (дата звернення: 19.04.2024).

5. Hydra. URL: <https://hydra.bot/> (дата звернення: 19.04.2024).

6. DeezerMusicBot. URL: <https://botostore.com/c/deezermusicbot/> (дата звернення: 19.04.2024).

7. mephbot. URL: <https://botostore.com/c/mephbot/> (дата звернення: 19.04.2024).

8. LyBot: URL: <https://botostore.com/c/lybot/> (дата звернення: 19.04.2024).

9. Діаграма прецедентів. URL: <https://uk.wikipedia.org/wiki/%D0%94%D1%96%D0%B0%D0%B3%D1%80%D0%B0%D0%BC%D0%B0_%D0%BF%D1%80%D0%B5%D1%86%D0%B5%D0%B4%D0%B5%D0%BD%D1%82%D1%96%D0%B2> (дата звернення: 21.04.2024).

# ДОДАТОК А

**Створення основного файлу (index.js)**

//index.js

import { Client, GatewayIntentBits, REST } from 'discord.js';

import { commands, handleCommandInteraction } from './commands.js';

import { DISCORD\_TOKEN } from './config.js';

const client = new Client({

intents: [

GatewayIntentBits.Guilds,

GatewayIntentBits.GuildMessages,

GatewayIntentBits.GuildVoiceStates,

],

});

const rest = new REST({ version: '10' }).setToken(DISCORD\_TOKEN);

client.once('ready', async () => {

try {

console.log('Початок оновлення глобальних команд додатку (/).');

await rest.put(

`/applications/${client.user.id}/commands`,

{ body: commands }

);

console.log('Глобальні команди додатку (/) успішно оновлено.');

} catch (error) {

console.error('Помилка при оновленні глобальних команд:', error);

}

});

client.on('interactionCreate', async (interaction) => {

try {

if (!interaction.isCommand()) return;

const voiceChannel = interaction.member.voice.channel;

if (!voiceChannel) {

await interaction.reply('Ви повинні знаходитись у голосовому каналі, щоб використовувати цю команду.');

return;

}

const permissions = voiceChannel.permissionsFor(interaction.client.user);

if (!permissions || !permissions.has('CONNECT') || !permissions.has('SPEAK')) {

await interaction.reply('Мені потрібні дозволи для підключення та спілкування у вашому голосовому каналі!');

return;

}

// Викликаємо обробник команди

await handleCommandInteraction(interaction);

} catch (error) {

console.error('Помилка при обробці взаємодії:', error);

await interaction.reply('Виникла помилка при обробці вашого запиту.');

}

});

// Авторизація бота

client.login(DISCORD\_TOKEN);

# ДОДАТОК Б

**Створення файлу з командами (commands.js)**

// commands.js

import { getVoiceConnection, joinVoiceChannel, createAudioPlayer, createAudioResource, NoSubscriberBehavior } from '@discordjs/voice';

import { stream } from 'play-dl';

const commands = [

{

name: 'skip',

description: 'Skip the currently playing song',

},

{

name: 'leave',

description: 'Leave your channel!',

},

{

name: 'play',

description: 'Play music from a URL',

options: [

{

name: 'url',

type: 3,

description: 'URL of the audio to play',

required: true,

},

],

},

{

name: 'pause',

description: 'Pause the music',

},

{

name: 'resume',

description: 'Resume music playback',

},

{

name: 'stop',

description: 'Stop the music playback',

},

{

name: 'queue',

description: 'Add a song to the queue',

options: [

{

name: 'url',

type: 3,

description: 'URL of the audio to add to the queue',

required: true,

},

],

},

];

const handleLeaveCommand = (interaction) => {

const currentConnection = getVoiceConnection(interaction.guild.id);

if (currentConnection) {

currentConnection.destroy();

interaction.reply('Залишив голосовий канал.');

} else {

interaction.reply('Бот не знаходиться у поточному каналі');

}

};

const handlePlayCommand = async (interaction, options) => {

try {

const voiceChannel = interaction.member?.voice.channel;

if (!voiceChannel) {

interaction.reply('Ви повинні знаходитись у голосовому каналі, щоб використовувати цю команду.');

return;

}

const audioUrl = options.getString('url');

try {

const streamData = await stream(audioUrl);

const connection = joinVoiceChannel({

channelId: voiceChannel.id,

guildId: interaction.guildId,

adapterCreator: interaction.guild.voiceAdapterCreator,

});

const player = createAudioPlayer({

behaviors: {

noSubscriber: NoSubscriberBehavior.Play

}

});

const resource = createAudioResource(streamData.stream, {

inputType: streamData.type

});

const subscription = connection.subscribe(player);

player.play(resource);

interaction.reply(`Приєднався та почав відтворення в каналі ${voiceChannel.name}. Посилання на трек:${audioUrl}`);

player.on('idle', () => {

if (player.queue && player.queue.length > 0) {

const nextResource = player.queue.shift();

player.play(nextResource);

}

});

} catch (error) {

interaction.reply('Ви ввели неправильне посилання.');

}

} catch (error) {

console.error(error);

interaction.reply('Виникла помилка при обробці команди play.');

}

};

const handleResumeCommand = (interaction) => {

const connection = getVoiceConnection(interaction.guildId);

if (!connection) {

interaction.reply('Бот не знаходиться у голосовому каналі.');

return;

}

const player = connection.state.subscription.player;

if (player) {

player.unpause();

interaction.reply('Відтворення музики відновилося.');

} else {

interaction.reply('В даний момент музика не поставлена на паузу.');

}

};

const handleStopCommand = (interaction) => {

const connection = getVoiceConnection(interaction.guildId);

if (!connection) {

interaction.reply('Бот не знаходиться у голосовому каналі.');

return;

}

const player = connection.state.subscription.player;

if (player) {

player.stop();

interaction.reply('Відтворення музики зупинено.');

} else {

interaction.reply('В даний момент музика не грає.');

}

};

const handleQueueCommand = async (interaction, options) => {

try {

const voiceChannel = interaction.member?.voice.channel;

if (!voiceChannel) {

interaction.reply('Для використання цієї команди ви повинні знаходитись у голосовому каналі.');

return;

}

const audioUrl = options.getString('url');

try {

const streamData = await stream(audioUrl);

if (!streamData || !streamData.stream) {

interaction.reply('Не вдалося отримати дані аудіопотоку.');

return;

}

const connection = getVoiceConnection(interaction.guildId);

if (!connection) {

interaction.reply('Бот не знаходиться у голосовому каналі.');

return;

}

const player = connection.state.subscription.player;

if (!player.queue) {

player.queue = [];

}

const resource = createAudioResource(streamData.stream, {

inputType: streamData.type

});

player.queue.push(resource);

player.on('end', () => {

if (player.queue.length > 0) {

const nextResource = player.queue.shift();

player.play(nextResource);

}

});

interaction.reply({

content: `Додано в чергу. Посилання на трек: ${audioUrl}`,

ephemeral: false,

});

} catch (error) {

interaction.reply('Ви ввели неправильне посилання.');

}

} catch (error) {

console.error(error);

interaction.reply('При обробці команди черги виникла помилка.');

}

};

const handleSkipCommand = async (interaction) => {

try {

const connection = getVoiceConnection(interaction.guildId);

if (!connection) {

interaction.reply('Бот не знаходиться у голосовому каналі.');

return;

}

const player = connection.state.subscription.player;

if (!player) {

interaction.reply('В данный момент музика не грає.');

return;

}

if (!player.queue || player.queue.length === 0) {

interaction.reply('В черзі відсутня наступна пісня.');

return;

}

player.stop();

const nextResource = player.queue.shift();

player.play(nextResource);

interaction.reply('Пропустити поточну пісню, яка відтворюється.');

} catch (error) {

console.error(error);

interaction.reply('При обробці команди "Пропустити" виникла помилка.');

}

};

const handlePauseCommand = (interaction) => {

const connection = getVoiceConnection(interaction.guildId);

if (!connection) {

interaction.reply('Бот не знаходиться в голосовому каналі.');

return;

}

const player = connection.state.subscription.player;

if (player) {

player.pause();

interaction.reply('Відтворення музики призупинено.');

} else {

interaction.reply('В даний момент музика не грає.');

}

};

const handleCommandInteraction = async (interaction) => {

try {

const { commandName, options } = interaction;

switch (commandName) {

case 'leave':

handleLeaveCommand(interaction);

break;

case 'play':

await handlePlayCommand(interaction, options);

break;

case 'resume':

handleResumeCommand(interaction);

break;

case 'stop':

handleStopCommand(interaction);

break;

case 'queue':

await handleQueueCommand(interaction, options);

break;

case 'skip':

await handleSkipCommand(interaction);

break;

case 'pause':

handlePauseCommand(interaction);

break;

default:

interaction.reply('Unknown command');

}

} catch (error) {

console.error(error);

interaction.reply('An error occurred while processing the command.');

}

};

export {

commands,

handleCommandInteraction,

};

**Декларація**

**академічної доброчесності**

**здобувача освіти ВСП «Економіко-правничого фахового коледжу ЗНУ»**

Я, Селявін Дмитро Сергійович, здобувач(-ка) освіти 4 курсу, спеціальності/освітньо-професійної програми Інженерія програмного забезпечення, групи К121-20, адреса електронної пошти epfk.121.20selyavin.d@gmail.com

- підтверджую, що написана мною кваліфікаційна робота на тему «Розробка боту на основі Node.js для інтеграції з сервісом музичного контенту» відповідає вимогам академічної доброчесності та не містить порушень, що визначені у ст. 42 Закону України «Про освіту», зі змістом яких ознайомлений/ознайомлена;

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

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

Дата \_\_\_\_\_\_\_\_\_ Підпис \_\_\_\_\_\_\_\_\_ Дмитро СЕЛЯВІН

Дата \_\_\_\_\_\_\_\_\_ Підпис \_\_\_\_\_\_\_\_\_ Катерина РЕШЕВСЬКА