МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ЛЬВІВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ІМЕНІ ІВАНА ФРАНКА

Факультет прикладної математики та інформатики
(повне найменування назва факультету)

Кафедра інформаційних систем
(повна назва кафедри)

ДИПЛОМНА РОБОТА

ІНФОРМАЦІЙНА СИСТЕМА ДЛЯ ПРОГРАВАННЯ АУДІО В ГОЛОСОВИХ КАНАЛАХ ЗВ'ЯЗКУ, З МОЖЛИВОСТЯМИ ДЕТАЛЬНОГО НАЛАШТУВАННЯ ТА ІГРОВИМ ФУНКЦІОНАЛОМ

Виконав(ла): студент(ка) групи ПМІ-42
спеціальності 122 – комп'ютерні науки
(шифр і назва спеціальності)

Балабух Д.Я.
(підпис) (прізвище та ініціали)

Керівник Соколовський Я.І
(підпис) (прізвище та ініціали)

Рецензент
(підпис) (прізвище та ініціали)

2023

ЗМІСТ

АНОТАЦІЯ

UA

Дипломна робота Балабуха Дмитра Ярославовича на тему «Інформаційна система для програвання аудіо в голосових каналах зв’язку, з можливостями детального налаштування та ігровим функціоналом» містить 42 сторінки та 7 джерел.

У роботі розглянуто методи побудови сучасної інформаційної системи, основою ціллю якої є програвання та налаштування аудіо композицій всередині голосових каналів зв’язку. Розглянуто елементи машинного навчання для групування великих масивів аудіо даних в кластери.

Ключові слова: діскорд, бот, аудіо, кластеризація, машинне навчання

EN

Dmytro Yaroslavovich Balabukh's diploma thesis on the topic "Information system for playing audio in voice communication channels, with the possibility of detailed adjustment and game functionality" contains 42 pages and 7 sources.

The work considers the methods of building a modern information system, the main purpose of which is to play and adjust audio compositions within voice communication channels. The elements of machine learning for grouping large arrays of audio data into clusters are considered.

Keywords: discord, bot, audio, clustering, machine learning

ТЕХНІЧНЕ ЗАВДАННЯ

- Опис завдання:

Розробити інформаційну систему, що дозволяє користувачам програвати аудіо файли в голосових каналах зв'язку. Система має забезпечувати можливість детального налаштування параметрів аудіо та містити ігровий функціонал. Крім того, система повинна мати можливість кластеризувати масиви даних за схожими характеристиками. Програма має використовувати методи та інші частини прикладного інтерфейсу Discord.

- Функціональні вимоги:
    1) Забезпечити можливість пошуку будь-яких аудіо композицій в сервісі Youtube.
    2) Дозволити користувачам програвати аудіо файли в режимі реального часу під час спілкування в голосових каналах зв'язку.
    3) Забезпечити можливість детального налаштування аудіо.
    4) Реалізувати ігровий функціонал, такий як вікторина пісень.
    5) Забезпечити можливість кластеризації масивів аудіо даних на групи згідно з їх характеристиками, такими як: жанр, артист, рік тощо.
- Нефункціональні вимоги:
    1) Забезпечити підтримку різних форматів аудіо файлів для відтворення.
    2) Забезпечити стабільну роботу системи та мінімальну затримку аудіо під час програвання.
    3) Забезпечити інтуїтивно зрозумілий інтерфейс користувача для відтворення, налаштування та організації аудіо файлів.
    4) Інформаційне забезпечення українською.

ВСТУП

Багато з нас любить проводити час з друзями, проте ситуація в світі на сьогоднішній день обмежує нас в багатьох речах, які ми вподобаємо. На жаль чи на щастя, але живе спілкування замінюється сучасними технологіями, кожна особа має змогу комунікувати з іншими за допомогою платформ обміну миттєвими повідомленнями. Бізнес, який забезпечує онлайн комунікацію настільки виріс, що кожна людина знає назву хоча б однієї корпорації з доставки таких послуг. Зі зростанням попиту, збільшується й запит на різного роду сервіси, що побудовані навколо месендж провайдерів.

Одним із гігантів онлайн комунікації є Discord. Основною відмінністю від вже звичних аналогів, якими людство користується не одне десятиліття, є можливість контактувати через канали зв’язку, які постійно відкриті для з’єднання. Раніше, щоб поговорити один з одним через мережу інтернет, ми ініціювали дзвінок, тим самим відкривали канал, а коли закінчували розмову, то відповідно й з’єднання знищувалось. Така зручність у використанні об’єднала мільйони людей зі спільними інтересами, кожна група має змогу ініціювати свій сервер, створити там декілька каналів і насолоджуватись спілкуванням.

Проблемою для мене стало просте прослуховування пісень. Справа в тому, що іноді виникає потреба прослухати якесь аудіо та, можливо, висловити свої думки, критику чи тому подібне. Наразі, немає надзвичайно мінімалістичного застосунку, який би задовільнив конкретну цю вимогу.

Практично кожна відкрита компанія надає доступ до свого прикладного інтерфейсу (API), за рахунок цього кожна особа володіє можливістю автоматизувати процес у вигляді бота. Discord також дозволяє користувачам використовувати свої відкриті методи. В даному випадку наша програма виступає клієнтом обміну голосових сигналів, яка транслює бажані користувачем аудіо композиції. Інтерфейс надає можливість створення голосового зв’язку, за допомогою VoIP (Voice over

Internet Protocol[1]). Зв'язок на цій платформі базується на відкритій системі, де аналоговий звуковий сигнал, що надходить від абонента, перетворюється в цифрову форму через процес дискретизації та кодування. На стороні отримувача здійснюється зворотна операція, яка включає декомпресію, декодування та відтворення аналогового сигналу. Таким чином, забезпечується ефективний обмін інформацією між абонентами.

АКТУАЛЬНІСТЬ

Розвиток ботів в поєднанні з алгоритмами машинного навчання є актуальними темами, які мають значний вплив на соціальні спільноти та інтерактивний досвід користувачів. Використання новітніх алгоритмів дозволяє застосункам ставати більш ефективними, інтелектуальними та персоналізованими, спрощуючи взаємодію, а також забезпечуючи точнішу аналітику та рекомендації. В контексті аудіо бота, що програє композиції в каналах зв’язку, це допоможе групувати пісні за різними ознаками, розпізнавати музичні уподобання користувачів, пропонувати персоналізовані рекомендації, прогнозувати користувацькі музичні вподобання та забезпечувати відповідні кластери для їхнього комфортного прослуховування. Крім того, важливо використовувати машинне навчання, оскільки це відкриває шлях до постійного вдосконалення функціональності. Загалом, актуальність аплікації полягає в тому, що вона створює інноваційний, соціальний і особистісний підхід до спільного прослуховування музики. Вона об'єднує людей, допомагає відкривати нову музику і розширює можливості музичної спільноти. Усе це підкреслює важливість та актуальність розвитку діскорд ботів та алгоритмів в машинному навчанні для поліпшення соціальних спільнот та забезпечення задоволення користувачів.

ОБ’ЄКТ ДОСЛІДЖЕНЬ

Процес поширення інформації в каналах аудіо зв’язку.

ПРЕДМЕТ ДОСЛІДЖЕНЬ

Програмне забезпечення для програвання аудіо в голосових каналах зв’язку з можливостями налаштування та ігровим функціоналом на підставі машинного навчання.

МЕТА РОБОТИ

Розроблення програмного застосунку Discord для програвання аудіо в голосових каналах зв’язку та управління аудіо композиціями з можливостями налаштування та ігровим функціоналом і кластеризацією.

ДЛЯ ДОСЯГНЕННЯ ПОСТАВЛЕНОЇ МЕТИ У РОБОТІ НЕОБХІДНО ВИКОНАТИ НАСТУПНІ ЗАВДАННЯ

- Здійснити аналіз можливостей доступних провайдерів Discord API з метою оцінки спектру їх можливостей.
- Проаналізувати функціонал обраного провайдера та його взаємодію з голосовими каналами і чатом.
- Зареєструвати клієнт бота за допомогою девелоперського порталу Discord.
- Розробити інформаційне забезпечення для функціонування стану бота.
- Програмно реалізувати комунікацію з ботом в чаті Discord-каналу, зокрема, забезпечити користувача командами управління музикальними цифровими медіа файлами та кластеризації масивів аудіо даних.

- Здійснити програмну реалізацію ігрового функціоналу за допомогою рандомізації активної аудіо композиції.
- Передбачити можливість розширення функціоналу системи, зокрема, враховувати взаємодію різних ефектів на аудіо композиції.
- Розробити програмний застосунок для кластеризації аудіо ресурсів Spotify на Python сервері.

ПРАКТИЧНА ЗНАЧИМІСТЬ РОБОТИ

Практичну значимість цієї роботи можна розділити на декілька категорій: соціальна, технічна та розважальна.

- Інформаційна значимість:
  розроблений програмний застосунок Discord для програвання аудіо в голосових каналах зв’язку та управління аудіо композиціями з можливостями налаштування та ігровим функціоналом і кластеризацією. Це розширює можливості користувачів для налаштування звукового контенту під свої уподобання.
- Соціальна значимість:
  застосунок сприяє організації спільноти та підвищує соціальну взаємодію учасників, а також створює можливість спільного прослуховування музики, обговорення музичних уподобань та спільного досвіду.
- Розважальна значимість:
  програмний застосунок Discord дозволяє соціальній спільноті вдосконалювати soft skills, надає можливість користувачам захоплюватись музикою, відкривати нові треки та ділитися ними з іншими.

1 ОГЛЯД ІСНУЮЧИХ ЗАСТОСУНКІВ ТА ВИБІР ІНСТРУМЕНТІВ ДЛЯ РЕАЛІЗАЦІЇ ЗАСТОСУНКУ

1.1 Огляд існуючих застосунків

Цими днями Discord заполонили тисячі, а то й десятки тисяч ботів різного рівня. Починаючи з локальних застосунків, які використовуються в обмеженому колі, закінчуючи мастодонтами бізнесу, що хостяться чи не на кожному Discord сервері. Проте використання популярних сервісів часто не виправдовує себе, адже з плином часу їх закривають через малу дохідну частину та збільшення попиту, який неможливо контролювати. Крім цього, часто можна спостерігати, що розробники стараються зробити свою програму мультизадачною, в той час, коли кінцевий користувач зацікавлений лише в конкретній частині застосунку.

Розглянемо два найпоширеніших на сьогоднішній день аналоги, а саме Vexera та Pankace.

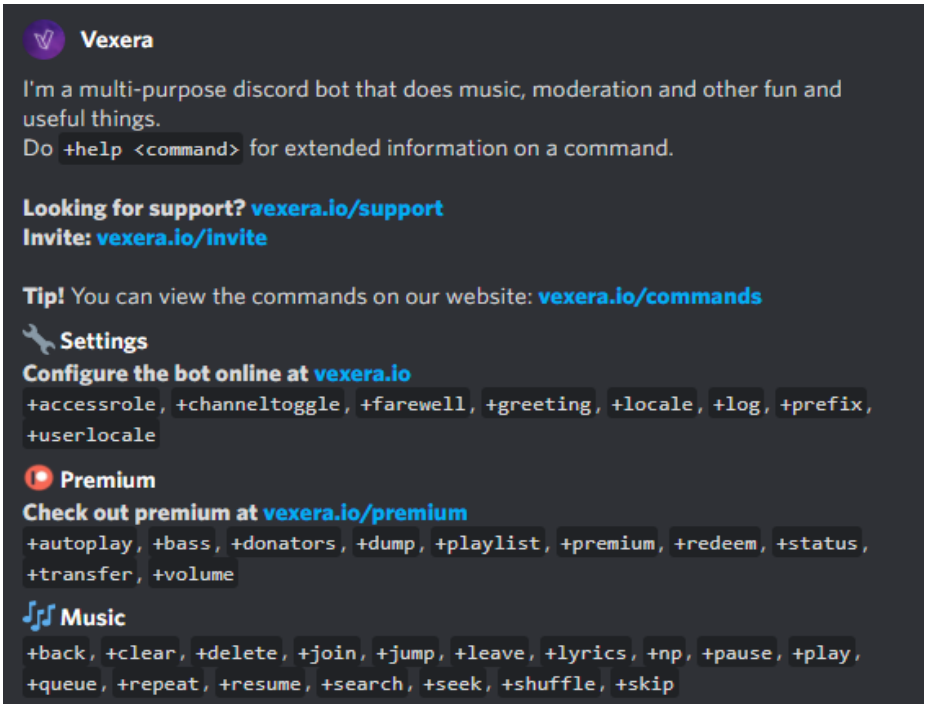

Рисунок 1.1 – Опис та основний стек команд боту Vexera

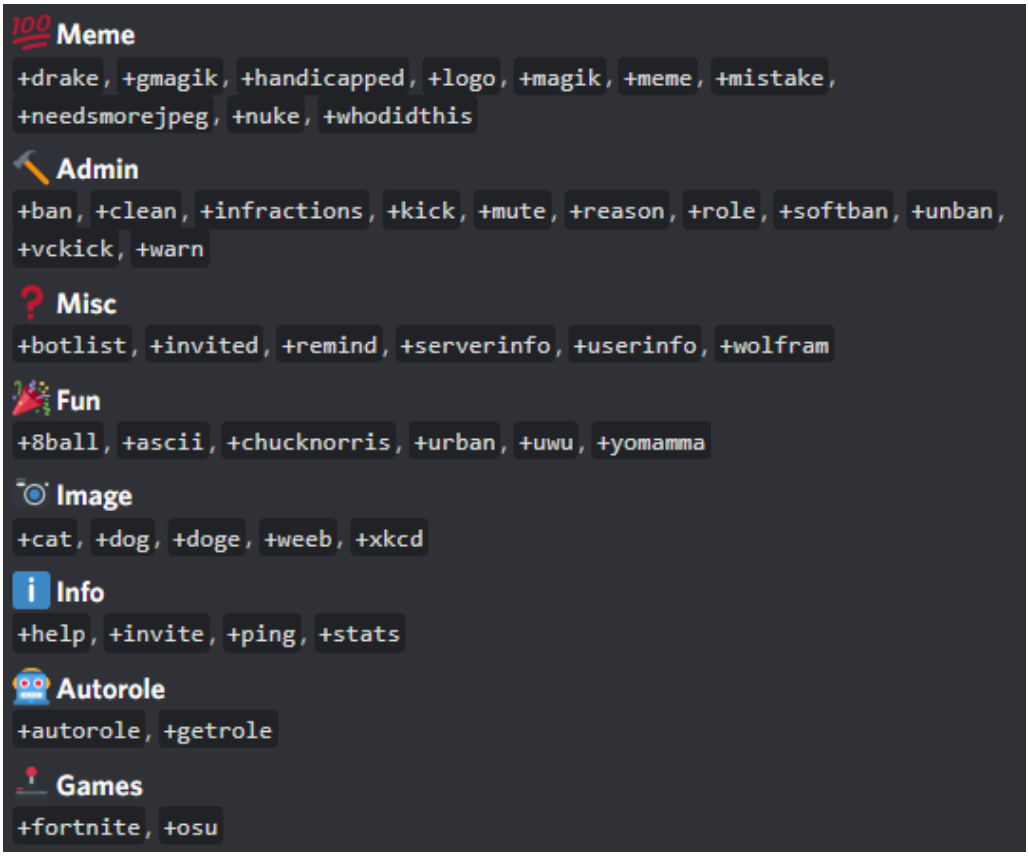

Рисунок 1.2 – Інші команди Vexera

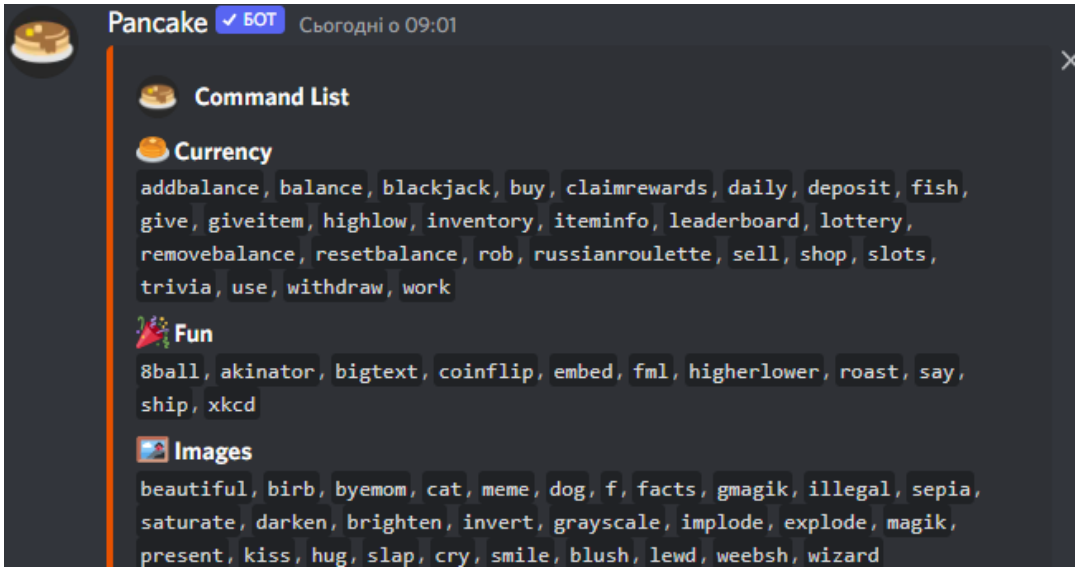

Рисунок 1.3 – Перша частина команд Pancake

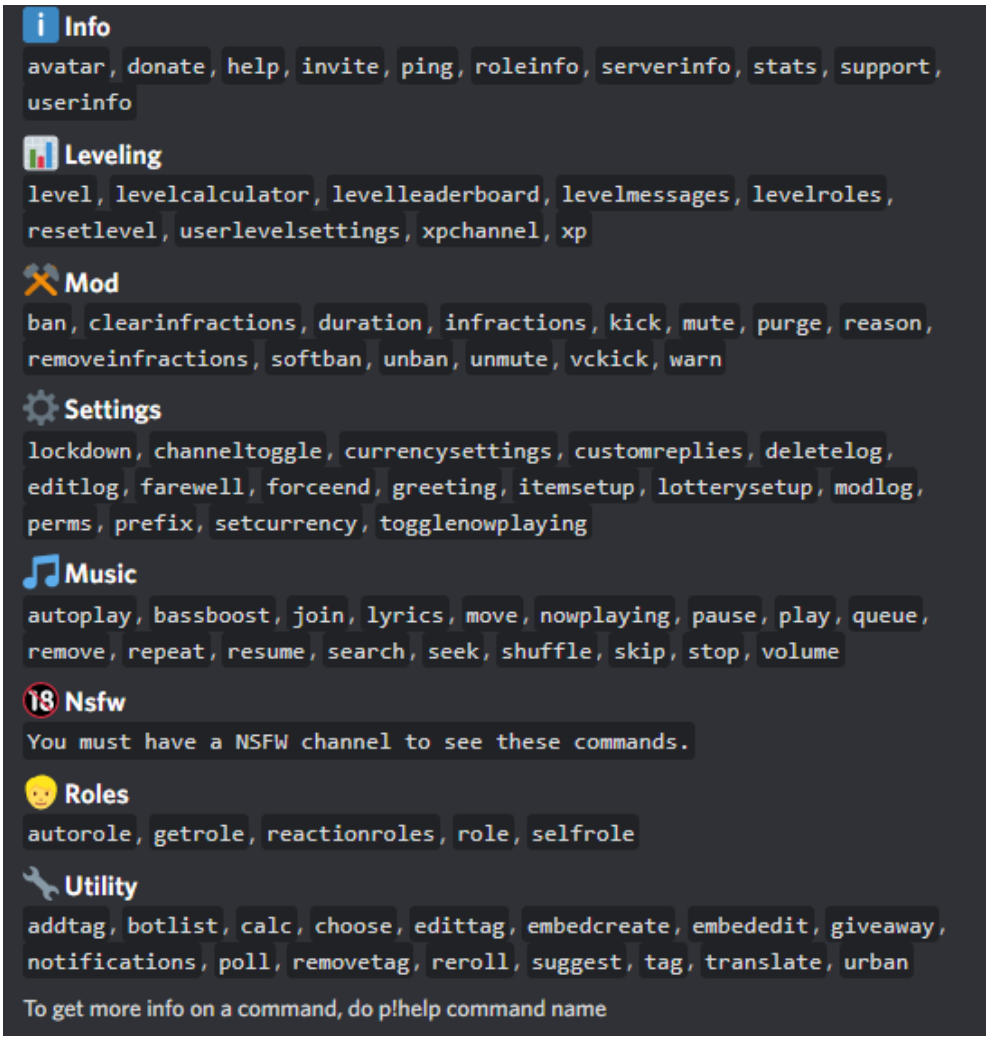

Рисунок 1.4 – Решта команд Pancake

Спостерігаємо, що клієнту надається надлишковий функціонал, який, між іншим, потребує постійної підтримки. Коли виникає проблема з одним сервісом, тоді користувачі всіх інших повинні очікувати на виправлення помилок. Все це відбувається тому, що бот має централізоване керування.

Переглядаючи багато інших застосунків, я помітив, що описані вище проблеми є спільними для них усіх. Крім цього, велике навантаження через масове використання цих ботів, спричиняє проблеми у програванні аудіо і відбувається це регулярно, адже є час пік, коли багато серверів зацікавлені в функціоналі цих застосунків.

Локальним каналам з конкретно визначеними цілями, не потрібен весь спектр можливостей, тому з побаченого можемо зробити для себе висновки, що кінцевого користувача цікавлять дві речі, а саме:

- Необхідний та достатній функціонал, який буде максимально простий у використанні.
- Безперебійна робота застосунку та стабільне програвання аудіо.

1.2 Вибір алгоритмів та інструментів для реалізації застосунку

1.2.1 Вибір алгоритму кластеризації та опис його моделі

1.2.1.1 Загальний опис та обґрунтування

Для групування великих масивів аудіо даних було обрано алгоритм машинного навчання з метою досягнення більшої ефективності використання ресурсів, зменшення накладних витрат. Звична нам послідовна обробка даних є повільною та ресурсоємною, адже кожна окрема одиниця рахується та аналізується окремо. Це ніяк не годиться навіть для помірного використання користувачами, адже навіть декілька запитів на послідовний аналіз великої кількості даних перенавантажать систему.

Кластеризація – це процес розбиття масивів даних на групи подібних, що називаються кластерами. «Подібність» визначається за спільними властивостями об’єктів над якими проводиться кластерний аналіз. Вона належить до типу «навчання без вчителя». Відмінність від навчання з учителем полягає в тому, що тут немає наперед заданої мети[2].

Існують різні моделі кластеризації, ось декілька з них:

- Зв’язності, така як ієрархічна кластеризація.
- Базовані на центроїді – k-середніх.
- Щільності.

Для виконання цього завдання було обрано модель k-середніх, що ґрунтується на центроїді. Розглянемо переваги та недоліки.

Переваги:

- Беззаперечною перевагою можемо вважати інтуїтивну простоту цього алгоритму. Тут використовуються добре зрозумілі математичні основи.
- Ефективний для наборів з малою та середньою кількістю даних.

Недоліки:

- Вибір числа k - кількості кластерів повинен бути зроблений перед виконанням самого алгоритму, що в деяких випадках може бути складно.
- Як наслідок попереднього пункту, в залежності від кількості кластерів, можемо отримати й різні результати.
- Може бути неефективним для даних з великою кількістю характеристик.

Незважаючи на присутні недоліки, алгоритм все ще задовольняє вимоги та підходить для виконання поставленої задачі.

Перед тим як описати алгоритм та пояснити що таке центроїд, визначу властивості моделі:

- Перша властивість алгоритму кластеризації k-середніх полягає у тому, що всі точки даних в кластері повинні бути подібні одна до одної.

Якщо розглядати це в контексті аудіо композицій, то ми говоримо про спільні якості, такі як: гучність, мелодійність, інструментальність тощо. Ця властивість інтуїтивно виходить з поняття кластера, адже в кластері повинні зберігатись якомога більш схожі об`єкти.

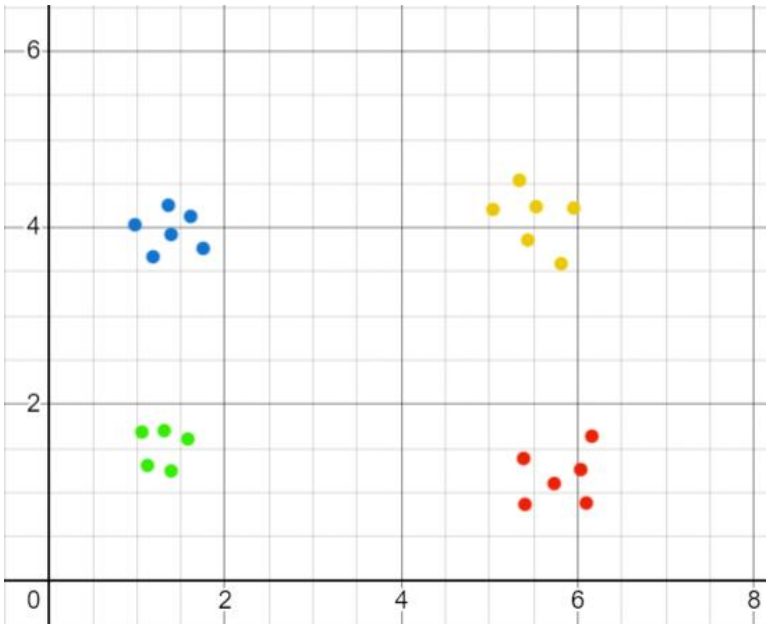

Рисунок 1.5 – Випадково розміщені дані для кластеризації

Очевидно, що об`єкти на графіку найлегше об`єднати в декілька округлих кластерів.

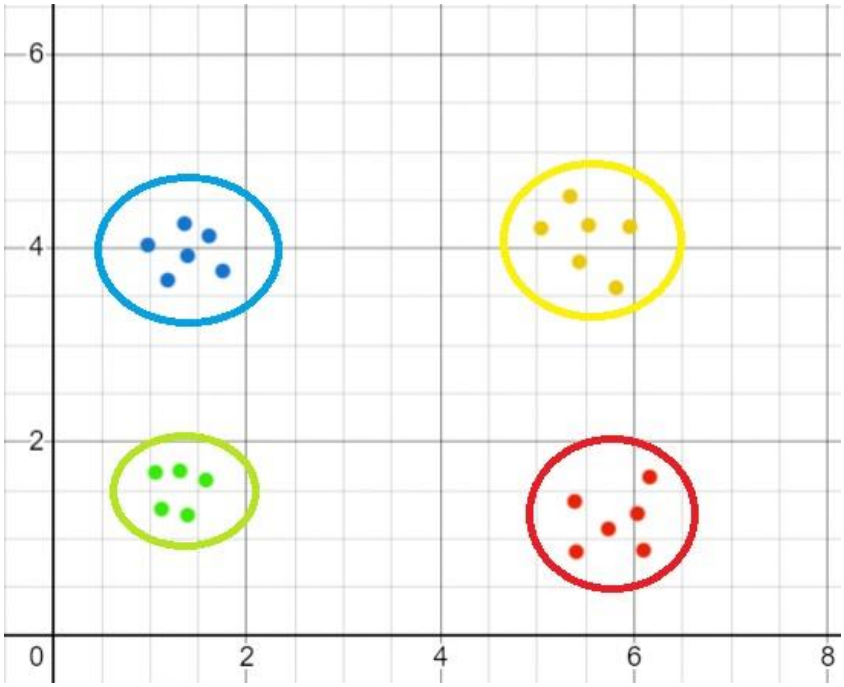

Рисунок 1.6 – Дані розподілені в чотири кластери

- Точки з різних кластерів мають бути якомога різноманітнішими в порівнянні один з одним.

З першої властивості розуміється й друга, хоча вони не завжди одночасно задовольняються. Таким чином, ми забезпечуємо різницю, яку може побачити не лише комп`ютерна програма, а й людина.

Якщо б ми спробували об’єднати наші аудіо дані інакше, як на рисунку нижче, то спостерігаємо точки всередині одного кластера, що знаходяться близько один з одним. Але чітко видно, що значно гірше виконується друга властивість в порівнянні з першим рисунком кластеризації.[3]

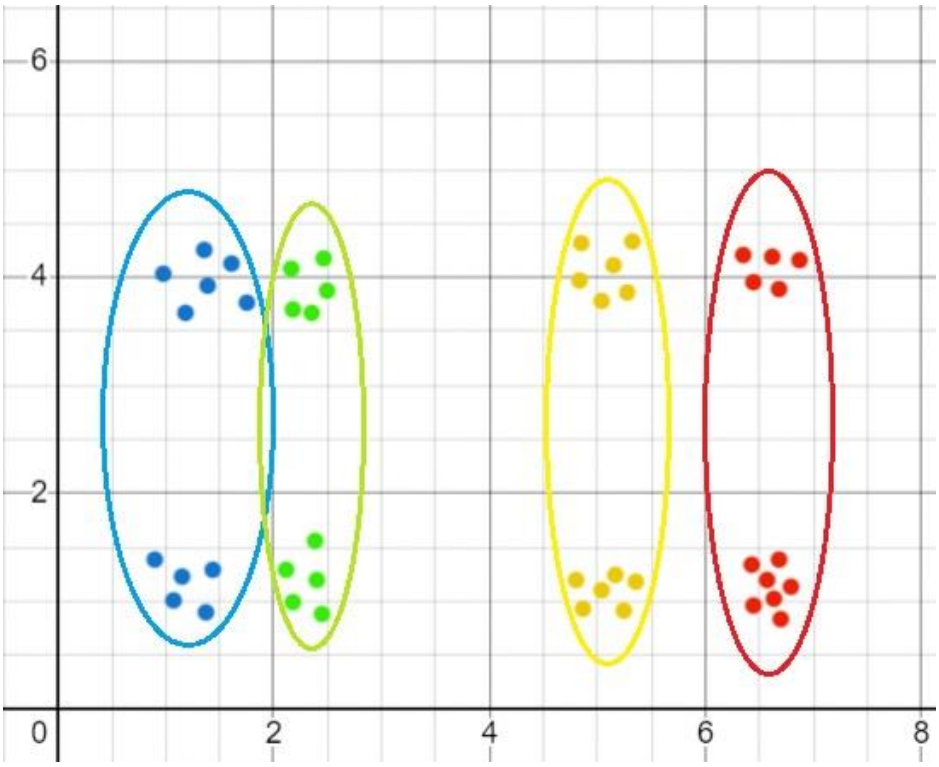

Рисунок 1.7 – Приклад кластеризації, що порушує другу умову

1.2.1.2 Знаходження оптимальної кількості кластерів

В методі k-середніх знаходження початкової кількості кластерів є надзвичайно важливим кроком, адже він заздалегідь визначає точність розбиття даних та й взагалі нашої моделі. Одним із поширених методів визначення оптимальної кількості кластерів є метод ліктя, який полягає в побудові графіка суми квадратів евклідових відстаней між точками даних і центром їх кластерів. Після цього обирається така кількість кластерів, де зміна суми квадратів відстаней починає вирівнюватися, тобто значення близькі одне до одного. Фактично, ми обраховуємо різні сценарії розвитку нашої системи для різної кількості кластерів, а після цього будуємо з цього графік.

Алгоритм може обробляти мільйони точок даних і видавати результати за лічені секунди або хвилини, що робить його актуальним для використання в нашій задачі.

1.2.1.3 Початкове визначення центроїдів

У звичайному алгоритмі k-середніх початкові центроїди обираються випадково, тому існує ризик неоптимальної роботи алгоритму в подальшому. Починаючи з того,

що це може призвести до більшої кількості ітерацій для визначення якісних кластерів й закінчуючи тим, що модель попаде в пастку локального мінімуму. Пастка локального мінімуму - це ситуація, коли алгоритм застрягає в недосконалому розв'язку, який є локальним мінімумом цільової функції. Це означає, що хоча поточний розв'язок може бути найкращим серед усіх сусідніх точок, він все ще є гіршим за глобальний, який може знаходиться в іншій частині простору розв’язків.

На рисунках 1.8 та 1.9 спостерігаємо приклад, де ми намагаємося використовувати k-середні з урахуванням 4-х центроїдів, і, як ми бачимо, дані можна легко розділити на 4 групи, розташовані більш-менш у чотирьох кутах квадратного зображення. Однак, через випадкову ініціалізацію центроїдів у верхньому лівому куті, маємо два центроїди, розташовані близько один до одного. Це впливає на продуктивність, оскільки на останньому зображенні, коли алгоритм нібито зійшовся, ми бачимо, що червоний колір покриває два кластери даних, які розглядаються як один. Один кластер був розділений на два (рожевий і коричневий), тому аналіз проведено хибно.

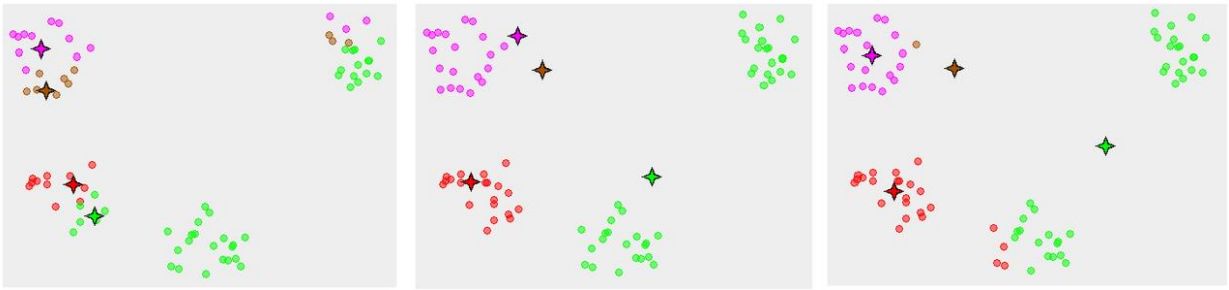

Рисунок 1.8 – Приклад пастки локального мінімуму

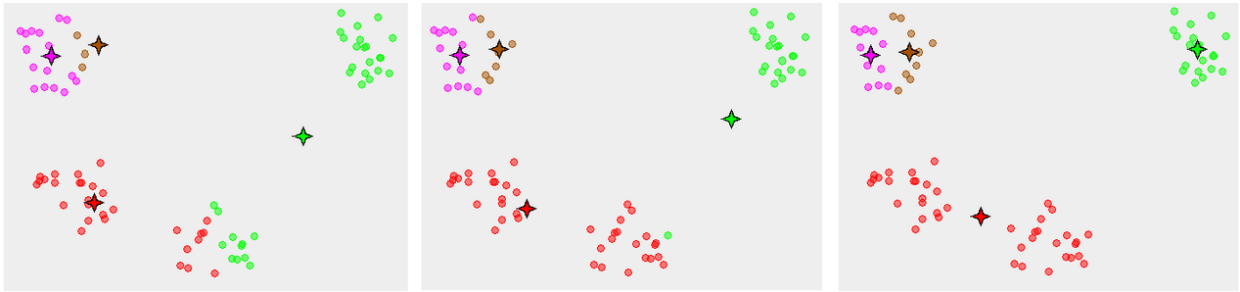

Рисунок 1.9 – Приклад пастки локального мінімуму

В такому випадку допомагає покращений метод k-середніх або k-середніх++. Він визначає процедуру ініціалізації центрів кластерів перед тим, як рухатися вперед із стандартним алгоритмом. Ідея полягає в тому, щоб розподілити початковий центроїд шляхом випадкового призначення першого, а потім шляхом вибору решти на основі максимального квадрата відстані. Тобто, потрібно відсунути центроїди якомога далі один від одного.

Алгоритм покращеного методу k-середніх[4]:

- Обираємо довільну точку як центроїд.
- Для кожної точки x обраховуємо значення P(x) за допомогою формули:

$$P(x) = \frac{D(x)^2}{\sum_{x' \in X} D(x')^2}, \quad (1.1)$$

де *P(x)* - ймовірність обрати точку як центроїд; *D(x)* - найближча відстань від точки *x до* найближчого центроїду.

- Далі точка для центроїда обирається з ймовірністю, пропорційною значенню *P(x).* Тобто, точка з більшим значення обирається як центроїд.
- Повторювати кроки 2 – 3 до тих пір, поки не буде обрано всі початкові центроїди.

1.2.2 Технологічний стек

Для розробки програмного забезпечення було обрано наступний технологічний стек:

- Typescript – мова програмування, що розширює можливості Javascript. Основний інструмент для написання бота та більшості функціоналу.
- Discord.js – модуль Node.js, який надає доступ до зручної інтеракції з прикладним програмним інтерфейсом Discord. Також ця бібліотека надає змогу кодувати цифрові аудіо файли.
- Python та Pandas – мова програмування Python в зв’язці з бібліотекою Pandas чудово підходять для вирішення задач машинного навчання, обробки великих масивів даних.

Typescript грає більш синтаксичну роль. Справа в тому, що на момент виконання програми весь код трансформується у Javascript. Основними перевагами є можливість явного визначення типів, підтримка використання класів і тому подібне. Ідея розробників була в тому, щоб покращити читабельність, повторне використання та рефакторинг коду. Ця мова є основним інструментом розробки застосунку, оскільки найкращі бібліотеки для взаємодії з Discord написані спеціально під неї.

Discord.js є обгорткою над API основної платформи. Фреймворк надає зрозумілу документацію та гнучкий інтерфейс користування, що приховує складну логіку. Він потребує набагато більш об’єктно-орієнтованого підходу, ніж більшість інших Javascript Discord бібліотек. Це робить код значно акуратнішим і легшим для розуміння. Послідовність і продуктивність є ключовими аспектами Discord.js, також фреймворк охоплює майже 100% Discord API. Для нашої конкретної задачі це не настільки важливо, але це свідчить, що йде постійна технічна підтримка цього застосунку.

Discord.js/opus модуль призначений для стиснення та розпакування аудіо даних відповідно до заданого формату. Ця підбібліотека відрізняється високою якістю кодування і мінімальною затримкою як при стисненні потокового звуку з високим бітрейтом, так і при стисненні голосу в обмежених за пропускною здатністю застосунках.

Python в зв’язці з бібліотекою Pandas стали платформою для реалізації функціоналу кластеризації аудіо даних. Простота та зручність у використанні – основні причини вибору цих інструментів. Крім цього під Python написана велика кількість мікро бібліотек, що дають змогу ефективно та зрозуміло візуалізувати модель на стадії розробки.

Висновок до розділу

В цьому розділі було розглянуто вже існуючі застосунки, а конкретно їх можливості, переваги та недоліки. Окрім цього, було описано обрані алгоритми та інструменти, що призначені для програмної реалізації системи.

2 ІНФОРМАЦІЙНЕ ЗАБЕЗПЕЧЕННЯ

2.1 Збір та обробка даних для відбору характеристик кластеризації

2.1.1 Опис характеристик аудіо композиції

Перш ніж почати імплементацію, потрібно проаналізувати дані, які буде отримувати наша модель. Для цього із Spotify за допомогою прикладного інтерфейсу на мові Python, отримуємо сотні тисяч пісень у вигляді таблиці, в форматі csv. Кожен із рядків описує одну аудіо композицію, починаючи з унікального ідентифікатора, назви пісні, авторів й закінчуючи великою кількістю аудіо характеристик, що задають кожній пісні свій унікальний стиль.

	id	instrumentalness	key	liveness	loudness	mode	name
1	4BJqT0PrAfrxzMOxyt…	0.878	10	0.665	-20.096	1	Piano Concerto No. 3…
2	7xPhfUan2yNtyFG0c…	0.0	7	0.16	-12.441	1	Clancy Lowered the B…
3	1o6I8BglA6ylDMrIELy…	0.913	3	0.101	-14.85	1	Gati Bali
4	3ftBPsC5vPBKxYSee…	2.77e-05	5	0.381	-9.316	1	Danny Boy
5	4d6HGyGT8e121Bsd…	1.68e-06	3	0.229	-10.096	1	When Irish Eyes Are …

Рисунок 2.1 – Приклад даних отриманих із Spotify

Speechines – демонструє наявність вокалу в пісні. Що значення ближче до одиниці, тим більше слів там є. Якщо значення більше за 0.5, то це скоріш за все подкаст чи аудіокнига.

Liveness – сигналізує про те, чи аудіо було записано наживо. Чим нижче значення, тим більша вірогідність того, що це студійний запис.

Valence – відносна міра позитиву всередині треку. Чим більше значення, тим більш він захопливий, веселий. Значення близькі до нуля означають, що композиція є сумною.

Tempo – оцінка швидкості програвання аудіо. Значення вимірюється кількістю ударів метронома за одну хвилину. Логічно, що чим більше значення, тим швидший темп композиції.

Energy – показник, що демонструє наскільки активна, швидка, голосна, енергійна та шумна є композиція.

Acousticness – міра, що визначає наскільки активно використовуються акустичні інструменти.

Duration_ms – час програвання в мілісекундах.

Track_Name - назва композиції.

Track_Id – унікальний ідентифікатор.

Artist_Name – назва виконавця.

Та декілька інших, що схоже описують властивості аудіо композиції.

2.1.2 Кореляційний аналіз характеристик аудіо даних

Для розуміння роботи моделі на основі кластеризації, потрібно провести кореляційний аналіз. Це дозволить визначити які пісні слід очікувати всередині одного кластеру, а які точно в різних. Окрім цього, ми побачимо які дані наша модель буде опрацьовувати в майбутньому.

Коефіцієнти кореляції між групами змінних, що відображаються таблицею, називаються матрицею залежностей. Коефіцієнтом кореляції є міра, що використовується в статистиці, для того, щоб визначити наскільки зв`язані змінні. Матриця залежностей може мати значення в межах -1 до 1, де -1 відображає строгу негативну кореляцію між змінними, 0 - відсутність, та 1 - строгу позитивну.

Для визначення залежностей між різними змінними, будемо використовувати метод Пірсона[5]. Причиною вибору саме цього методу є його простота та зрозумілість. Оскільки ми вже використовуємо доволі базовий метод для процесу кластеризації, то ускладнювати проміжні кроки також немає сенсу.

Спочатку обчислимо середні значення для кожної змінної (властивості) аудіо.

$$\bar{X} = \frac{1}{n}\sum_{i=1}^{n} X_i, \qquad \bar{Y} = \frac{1}{n}\sum_{i=1}^{n} Y_i, \tag{2.1}$$

де $\bar{X}$ – середнє арифметичне для змінної X; $\bar{Y}$ – середнє арифметичне для змінної Y; X_i – кожне значення для змінної X; Y_i – кожне значення для змінної Y; n – загальна кількість спостережень.

Після цього обчислимо відхилення від середнього значення для всіх композицій.

$$d_i(X) = X_i - \bar{X}, \qquad d_i(Y) = Y_i - \bar{Y}, \tag{2.2}$$

де $d_i(X)$ – різниця від кожного значення до середнього по X; $d_i(Y)$ – різниця від кожного значення до середнього по Y; $\bar{X}$ – середнє арифметичне для змінної X; $\bar{Y}$ – середнє арифметичне для змінної Y; X_i – кожне значення для змінної X; Y_i – кожне значення для змінної Y;

Далі знаходимо стандартне відхилення значень, що обчислюється по формулах

$$s_X = \sqrt{\frac{1}{n-1}\sum_{i=1}^{n}(X_i - \bar{X})^2}, \qquad s_Y = \sqrt{\frac{1}{n-1}\sum_{i=1}^{n}(Y_i - \bar{Y})^2} \tag{2.3}$$

де s_X – середньоквадратичне відхилення по X; s_y - середньоквадратичне відхилення по Y; $\bar{X}$ – середнє арифметичне для змінної X; $\bar{Y}$ – середнє арифметичне для змінної Y; X_i – кожне значення для змінної X; Y_i – кожне значення для змінної Y; n – загальна кількість спостережень.

Й нарешті отримуємо коефіцієнти кореляції

$$r = \frac{\sum_{i=1}^{n} d_i}{(n-1)s_X s_Y} \tag{2.4}$$

де r – коефіцієнт кореляції Пірсона, d_i – добуток відхилень для кожної пари значень $(X_i Y_i)$; s_X – середньоквадратичне відхилення по X; s_y – середньоквадратичне відхилення по Y; n – загальна кількість спостережень.

Після проведених обчислень, маємо змогу розглянути кореляційні коефіцієнти. Для цього скористаємось теплокартою, що є графічним способом представлення даних, в якій значення відображаються за допомогою кольорів. Цифрові значення можемо побачити на відповідних позначках шкали.

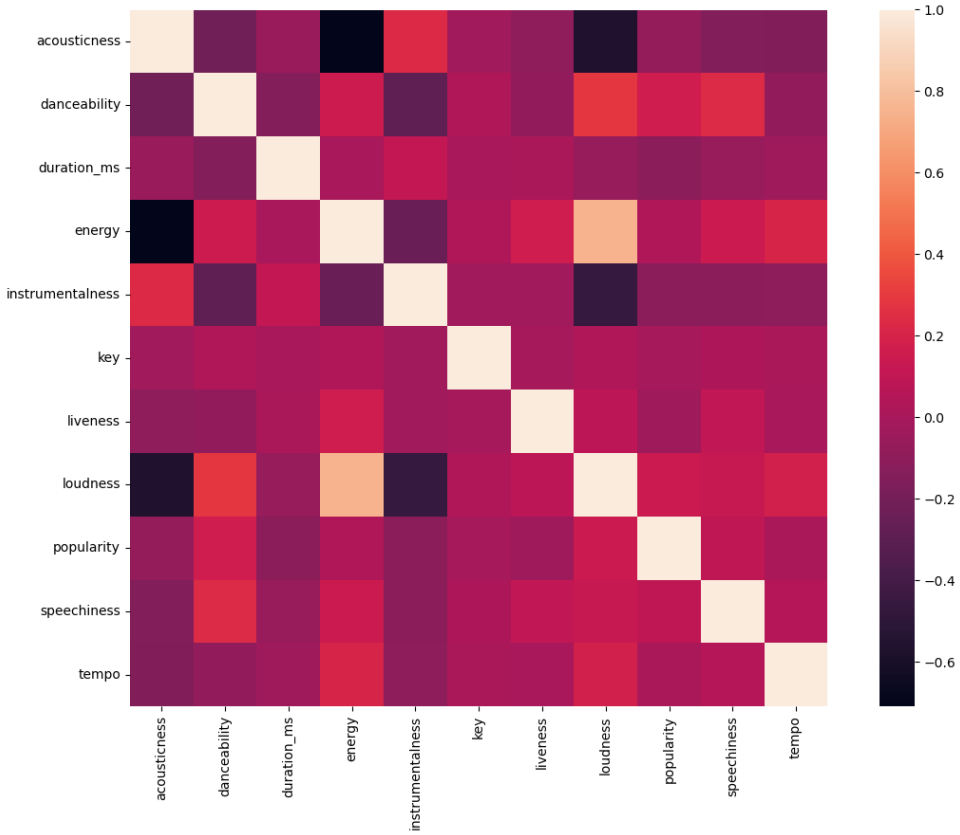

Рисунок 2.2 – Кореляційна матриця у вигляді теплокарти

Спостерігаємо строгу негативну кореляцію між energy та acousticness, що свідчить про те, що пісні, де використовуються в основному акустичні інструменти (напр., народна музика), енергійність буде низькою. Аналогічно надзвичайно енергійна музика, наприклад, в жанрі металу, навряд чи буде багата на акустичні інструменти. Оскільки loudness добре корелює з energy, то й відповідно бачимо негативну кореляцію з акустичністю. Також варто відмітити, що liveness (характеристика того, чи композиція виконувалась вживу) не корелює практично ні з чим, що є логічно.

Оскільки розгляд за всіма характеристиками буде занадто витратним та, насправді, непотрібним для нашої моделі, будемо розглядати лише частину, а саме ті, що найбільше корелюють по відношенню до популярності. Таким чином, ми

забезпечимо більшу пізнаваність кластеризованих пісень й сконцентруємось на конкретно визначених якостях.

2.2 Use-Case діаграма

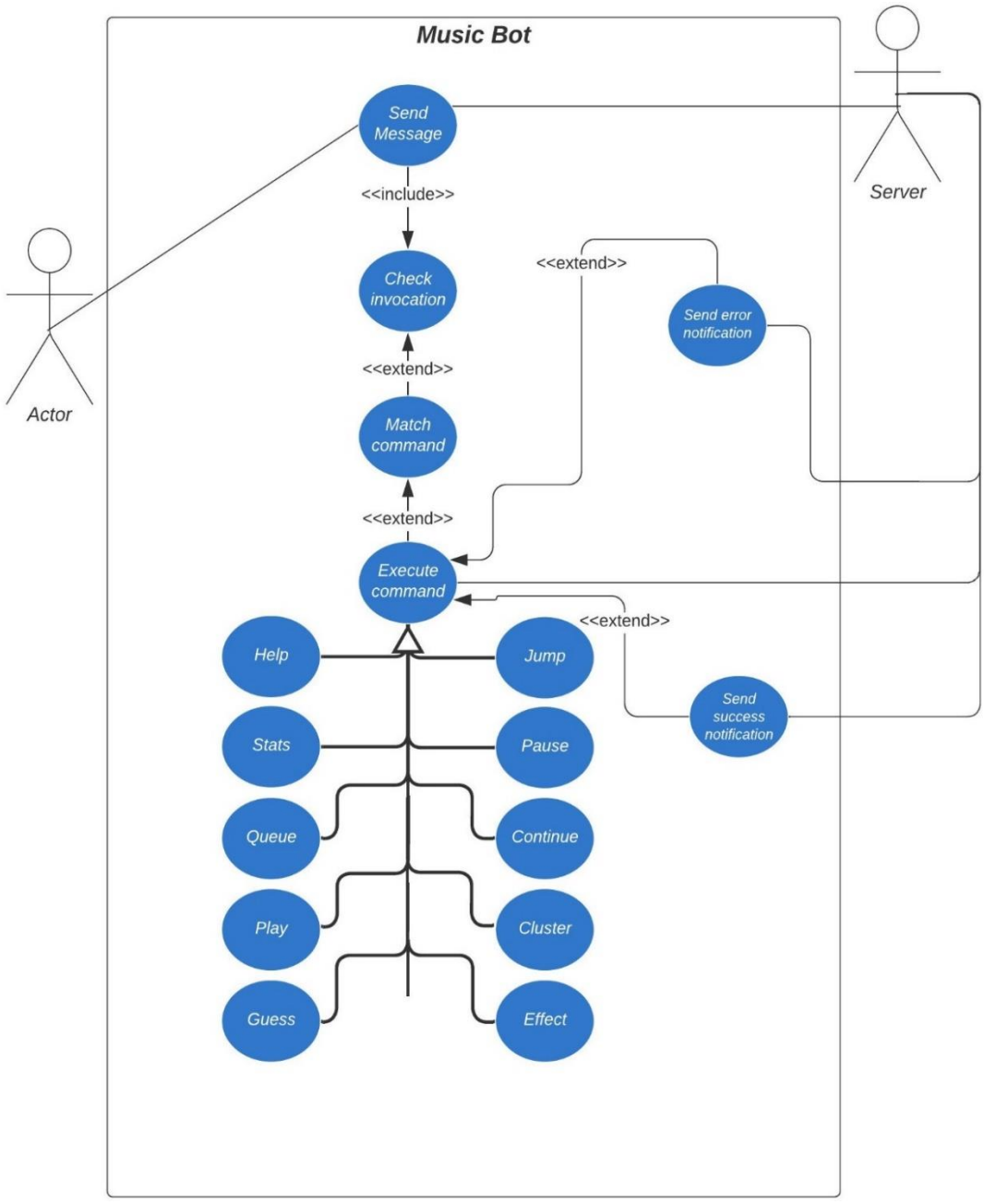

Рисунок 2.3 – Use Case діаграма

Бачимо класично визначеного актора та другорядну систему – Discord server. Якщо не заглиблюватись в деталі, адже це ми зробимо в діаграмі послідовностей, то точкою входу є просте повідомлення. Ніяка інша дія не може тригерити наш

застосунок, саме тому актор з’єднаний лише з “Send Message Use-Case”. Інша справа, що при виклику дій бота за допомогою команд, в нас автоматично підключаються похідні use-cases, які зводяться до двох вихідних дій – або помилка, або успішне виконання команди.

2.3 Sequence діаграма

На діаграмі (рисунок 2.4 та 2.5) продемонстровано чотири сутності. Як вже було сказано, Discord сервер є ключовою сутністю, яка й виконує взаємодію між ботом та актором і служить таким собі посередником. З рисунків видно, що вхідне повідомлення пропускається через код серверу, як і відповідь бота на звернення користувача.

Розглянемо детальніше потік виконання програми. Користувач відсилає повідомлення в чат серверу, це повідомлення зчитується та перевіряється ботом. Якщо надсилання виявилось зверненням до бота, то відбувається перевірка на наявність команди. При успішному результаті, бот почне діставати додаткову інформацію про користувача і канал в якому він знаходиться, а дальше розподіляти виклик по конкретним модулям, що мають свою реалізацію.

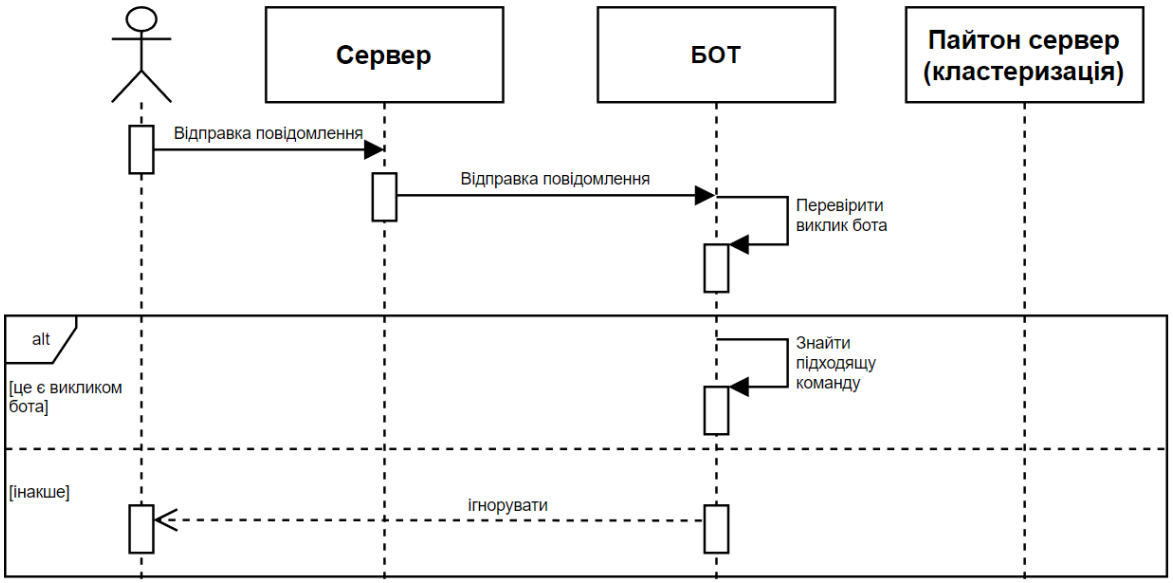

Рисунок 2.4 – Діаграма послідовностей ч.1

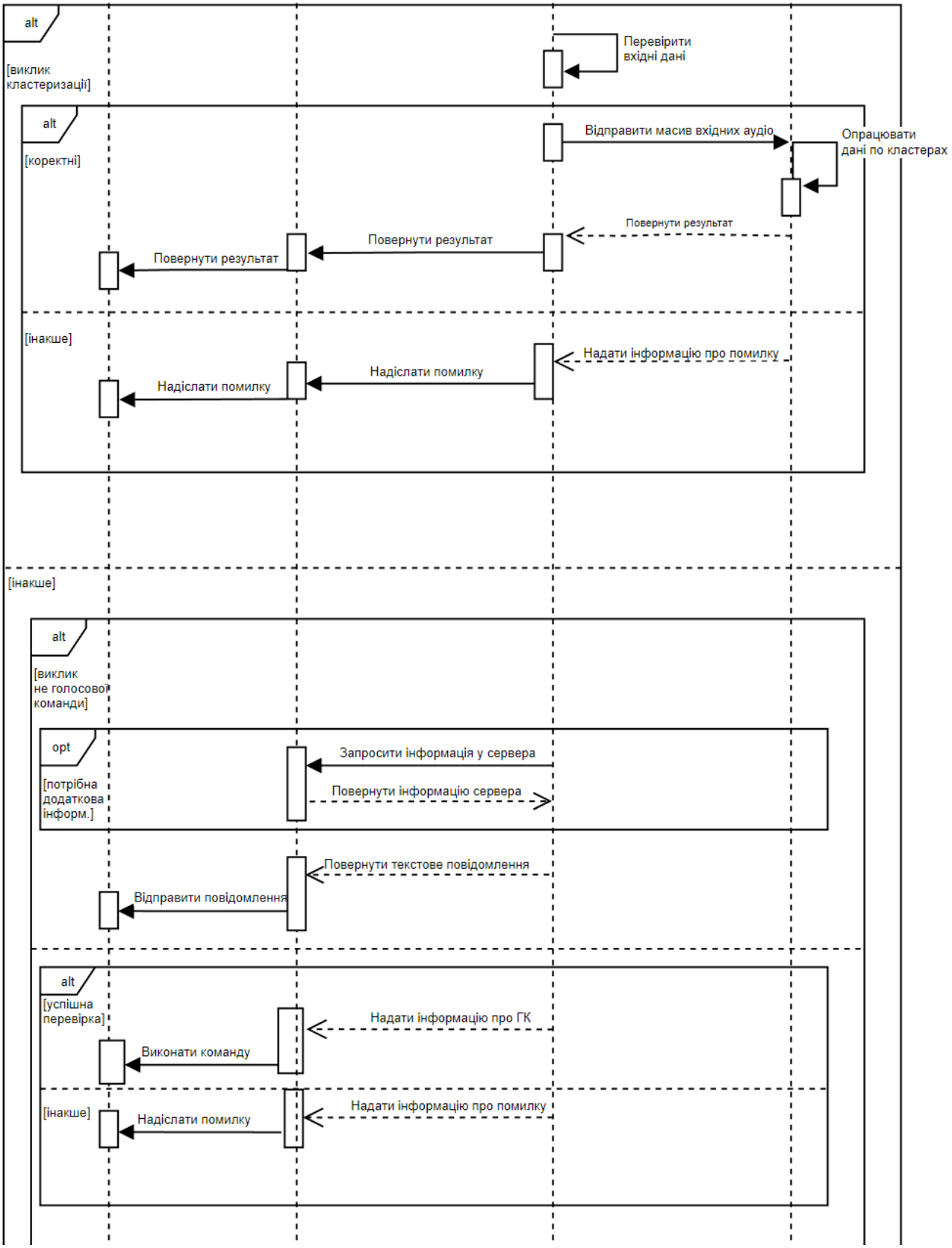

Рисунок 2.5 – Діаграма послідовностей ч.2

Роль пайтонівського сервера відіграється лише тоді, коли наша мета – це кластеризація масивів аудіо даних. Решта дій діляться на ті, що націлені на взаємодію з голосовим каналом і ті, що з текстовим.

Висновок до розділу

В цьому розділі було описано спосіб попереднього огляду даних, що в майбутньому розподіляються на кластери, а також детально описано потік роботи програми у вигляді різних діаграм.

3 РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ТА ДЕМОНСТРАЦІЯ ВИКОНАННЯ

3.1 Розробка програмного забезпечення

3.1.1 Створення боту на девелоперському порталі Discord

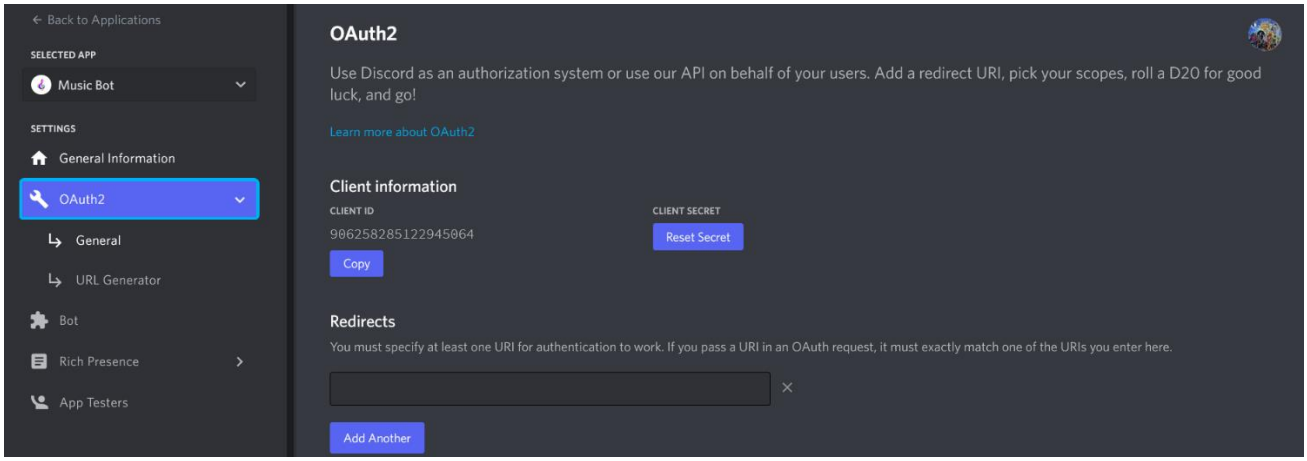

Рисунок 3.1 – Фото налаштування бота з Discord Portal

Для того, щоб бот почав існувати в системі, потрібно його зареєструвати на акаунті конкретного користувача, визначити його права, якими він має володіти при заході на новий сервер. Доступ до бота з програми відбувається за допомогою секретного ключа, який при викраденні надає зловмисникам повний доступ.[6]

3.1.2 Короткий опис event-driven архітектури

Після першого запуску бот готовий слухати різні події. Це може бути, наприклад, повідомлення в чат[7].

В першому випадку, така команда як leave викликає подію, що впливає на стан бота, адже після її виконання бот покидає канал голосового зв’язку.

```
private configureAudioPlayerIdleState() {
    this.audioPlayer.on(AudioPlayerStatus.Idle, () => {
        if (this.queueHandler.getQueueSize() === 0) {
            this.messageHandler.setStrategy(new CasualMessageStrategy);
            this.messageHandler.send(new CasualMessage(`> **Bot has finished playing**`));
            return;
        }
        this.executeNextResource();
    })
}

private configureAudioPlayerPlayingState(): void {
    this.audioPlayer.on(AudioPlayerStatus.Playing, async (_, newState) => {
        this.messageHandler.setStrategy(new CasualMessageStrategy());
        this.messageHandler.send(new CasualMessage(`> \`${newState.resource.metadata}\` **is now playing**`));
    })
}
```

Рисунок 3.2 – Налаштування подій

В свою чергу, такі команди, як stop, skip чи тому подібні, не торкаються корінної системи самого бота, а лише його залежності.

3.1.3 Розподілювач команд

Важливим вважаю згадати основний метод, що розподіляє повідомлення, які зчитує бот. В методі визначено префікс, який передує всім командам. Таким чином, він відфільтровує звичайні повідомленням від команд.

```
execute(message: Message): void {
  try {
    if (!this.validateMessage(message)) {
      return;
    }
    let messageArgs: string[] = message.content.substring(1)
      .split(' ')
    let commandName: string = messageArgs.shift() as string;
    let command = this.commandHandler.commands.get(commandName);
    if (!command) {
      return;
    }
    command.execute(message, messageArgs.join(' '));
  } catch (error) {
    console.log('Error occured while executing COMMUNICATE EVENT.', error);
  }
}
```

Рисунок 3.3 – Метод розподілювача команд

3.1.4 Кластеризація на Python сервері

Оскільки більшість основних бібліотек для зручного управління великими масивами даних написані під мову Python, я створив окремий проект, до якого звертається бот при виклику команди кластеризації. Коли користувачі надсилають

список пісень у csv форматі до застосунку, він отримує його і починає парсити. Проводиться аналіз метаданих, щоб впевнитися, що вони відповідають формату аудіо пісень зі Spotify. Після цього, бот відправляє його на пайтон сервер. Коли аналіз проведено, створюються відповідні публічні плейлисти, які доступні користувачеві.

```python
class ServerManager(BaseHTTPRequestHandler):
    def do_POST(self):
        content_length = int(self.headers['Content-Length'])
        form_data = self.rfile.read(content_length)

        # Parse form data to get the CSV file content
        fields = form_data.decode('utf-8')

        sp_client = SpotifyClient('02aca3b049a74aeba6a820a0ba7e0c50',
                                  '513bcf2584eb43518a808ad653dfa9b8',
                                  'Oracle Kintana')
        df = pd.read_csv(io.StringIO(fields), index_col=0)#.dropna(axis=0, s
           # .sort_values(by='popularity', ascending=False)
        cl_manager = ClusteringManager(df)
        cl_manager.process_k_means(sp_client)

        # Send response to client
        self.send_response(200)
        self.send_header('Content-type', 'text/plain')
        self.end_headers()
        self.wfile.write(b'CSV file uploaded successfully')
```

Рисунок 3.4 – API Python сервера, що приймає запити від бота

3.2 Демонстрація виконання

3.2.1 Help

Одна з декількох команд, які не залежать від стану бота та користувача. Виконує роль гіда по решті команд, що призначені для керування аудіо чергою або станом застосунку.

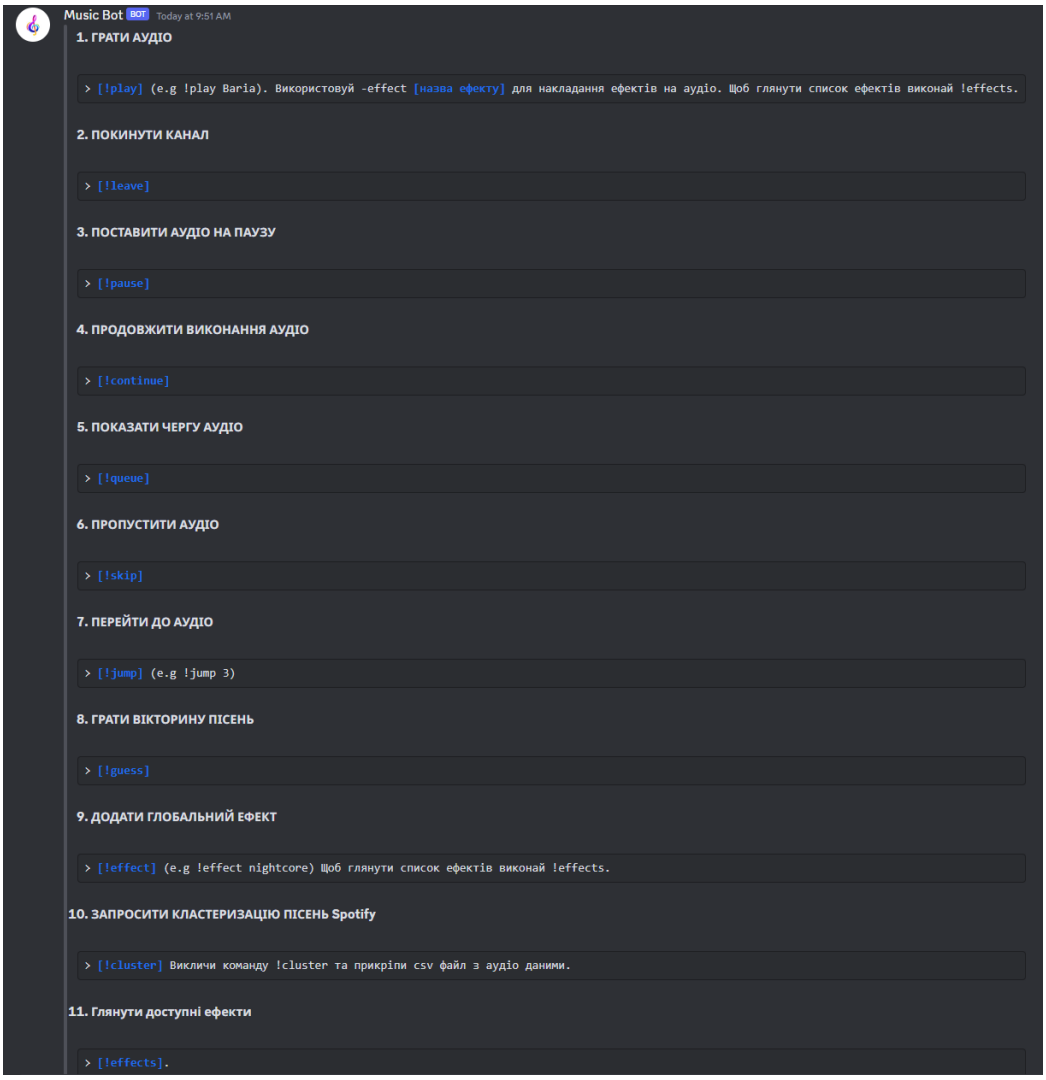

Рисунок 3.5 – Відповідь бота на команду Help

3.2.2 Stats

Друга команда такого ж типу, як і попередня. Надає коротку інформацію про стан сервера.

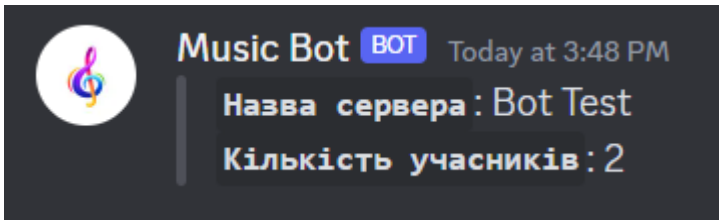

Рисунок 3.6 – Відповідь бота на команду Stats

3.2.3 Play <args>

Одна із головних команд програми. Ця команда є мультизадачною, а тому має декілька застосунків. Якщо в черзі немає пісень, а бот не в каналі, то при виклику цієї команди він заходить в один голосовий канал з користувачем і починає програвати аудіо композицію. В іншому випадку, аудіо композиція додається в чергу і чекає свого виконання. Варто зазначити, що деякі аргументи для цієї команди є обов'язковими, в них передають назву пісні або посилання на конкретний ресурс.

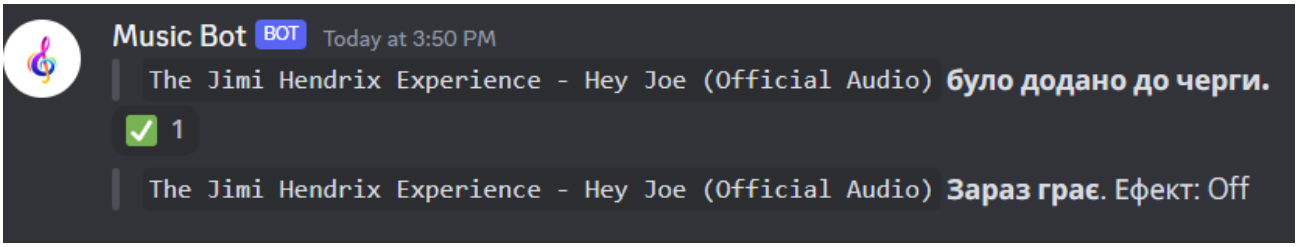

Рисунок 3.7 – Відповідь бота на команду Play

Доповненням до цієї команди є можливість накладати аудіо ефекти на пісні. Через команду допомоги користувач має змогу дізнатись про доступні ефекти та використати один із них при додаванні пісні в чергу.

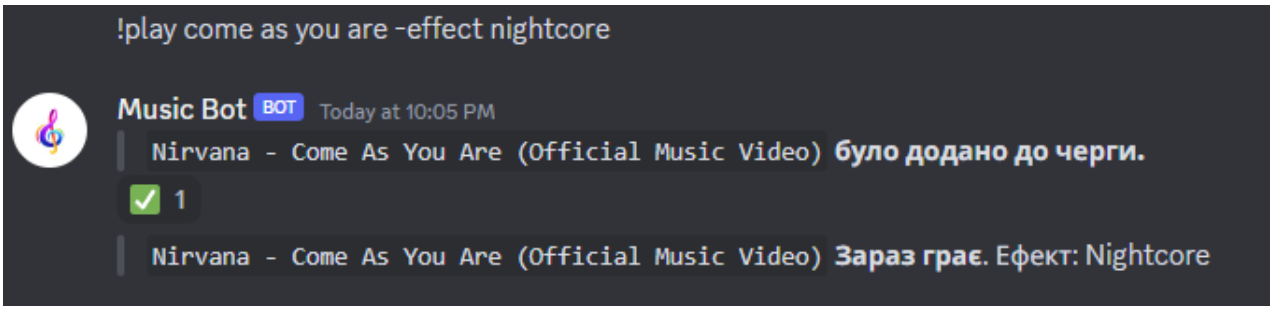

Рисунок 3.8 – Застосування play разом з параметром ефектів.

3.2.4 Pause

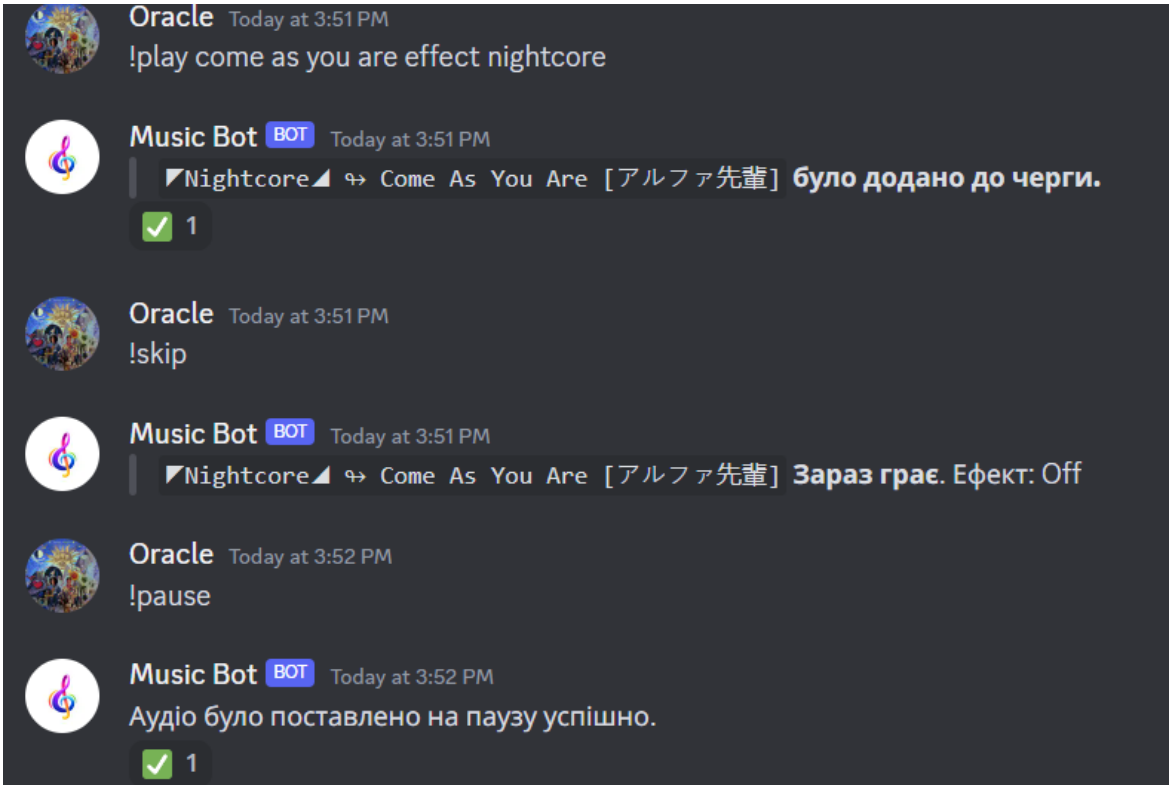

Рисунок 3.9 – Відповідь бота на команду Pause

Ця команда зупиняє програвання поточного аудіо за умови, що бот виконує якусь композицію із черги. В іншому ж випадку, ми отримаємо помилку валідації, як і у всіх інших командах.

3.2.5 Continue

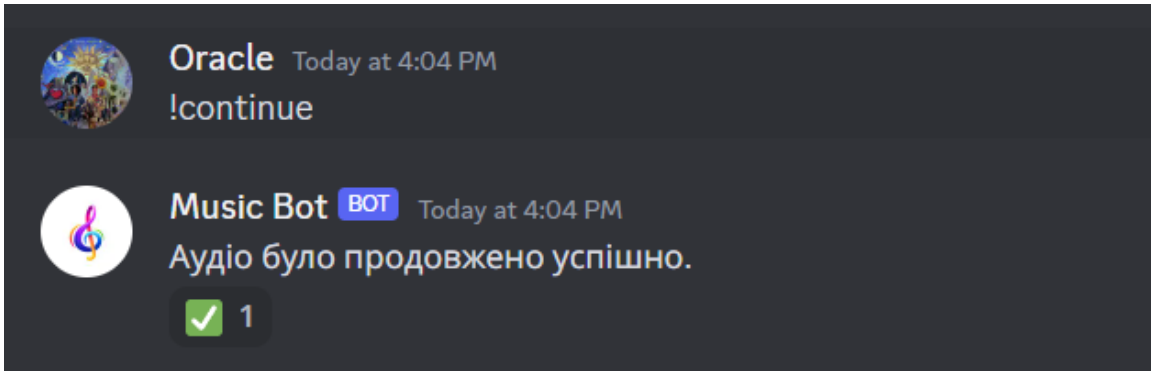

Рисунок 3.10 – Відповідь бота на команду Continue

Фактично є протилежністю до pause, продовжує відтворення ресурсу. Також виводить повідомлення в чат.

3.2.6 Skip

Якщо ми хочемо пропустити виконання композиції, то ця команда слугує для такої мети. На противагу jump, вона обмежена лише однією композицією – поточною. На фото видно, що ми викликали бота цією командою, цим самим пропустивши виконання аудіо, а оскільки черга була пуста, то бот завершив програвання.

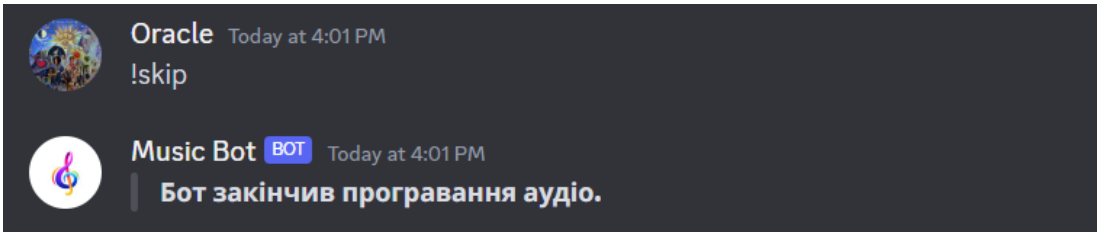

Рисунок 3.11 – Відповідь бота на команду Skip

3.2.7 Queue

Відображає аудіо композиції, які користувач додав в чергу. Ця команда призначена для того, щоб надавати актуальну інформацію по стану програваних ресурсів. Черга автоматично оновлюється з тим, як користувач додає нові композиції, видаляє їх, чи вони самі закінчують своє програвання.

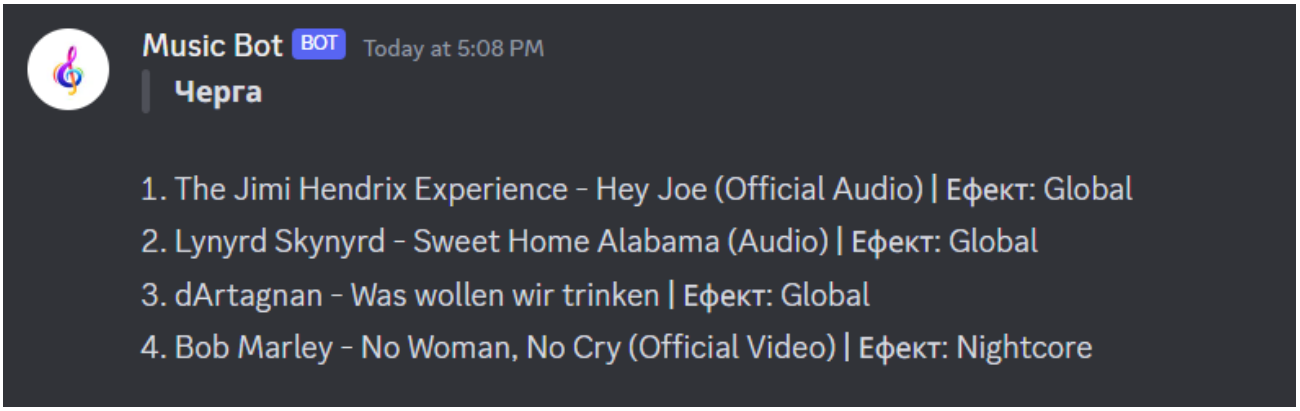

Рисунок 3.12 – Відповідь бота на команду Queue

3.2.8 Jump <index>

Іноді колекція пісень може набувати великих розмірів, а ми хочемо дістатись до пісні, яка далеко в черзі. Використання однієї команди skip було б занадто затратним для такої цілі, тому у нас є jump, яка переносить вказівник на конкретний ресурс.

Рисунок 3.13 – Відповідь бота на команду Jump

3.2.9 Leave

Команда контролю станом самого бота. Незважаючи на стан черги чи інших залежних речей, бот покидає канал, знищує всі поточні ресурси та переходить в стан очікування.

3.2.10 Effect

Як вже було описано в команді play, користувач має змогу накладати ефекти на аудіо композиції. Також є окрема команда, що призначена для встановлення глобального ефекту. Відтак всі пісні, що не мають спеціально заданого ефекту, будуть використовувати глобальний.

3.2.11 Guess

Реалізацією ігрового функціоналу стала команда guess. Користувач в текстовому каналі викликає цю команду й запускає пошук пісень в сервісі Youtube. Таким чином обирається п’ять варіантів, один з яких правильний. Вони надсилаються у вигляді опцій в чат. Після того як час обирання варіантів минає, бот виводить список учасників, час вгадування та результат.

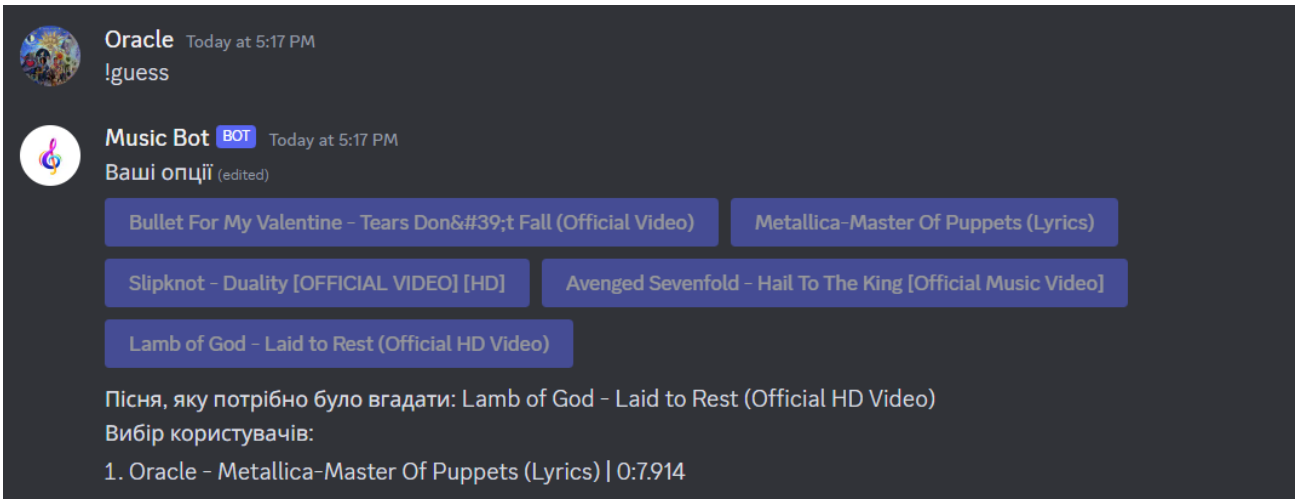

Рисунок 3.14 – Приклад ігрового функціоналу вікторини

3.2.12 Cluster

Для виконання кластеризації спочатку потрібно отримати аудіо дані із Spotify. Це можуть бути як окремо зібрані списки відтворення, які передані в API стрімінгової платформи так і згенерована історія прослуховувань користувача. Остання опція є найбільш легкою для середньостатичного користувача, адже не потребує додаткових знань по виклику API.

Заходимо в особистий кабінет користувача Spotify та обираємо розділ «Налаштування конфіденційності». Після цього у нас є декілька опцій по завантаженню історії прослуховування.

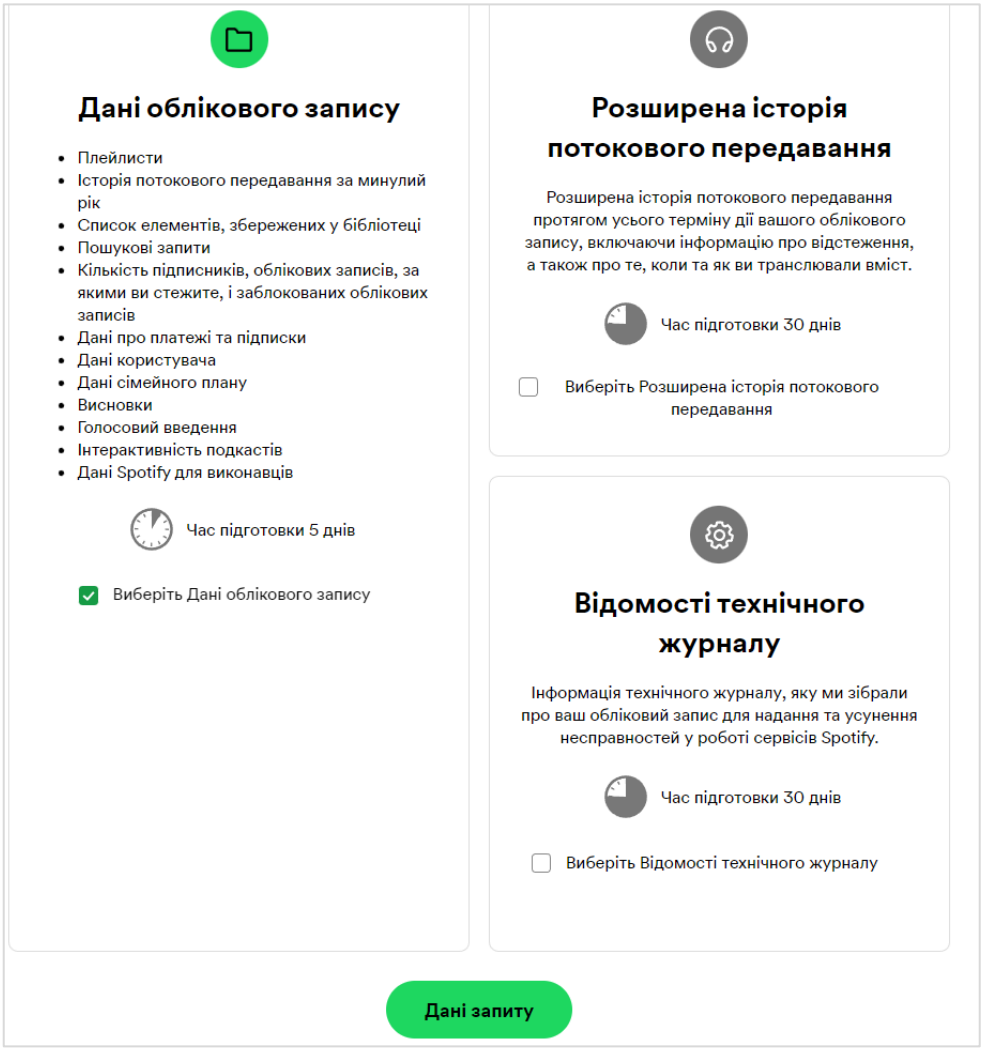

Рисунок 3.15 – Приклад отримання аудіо

Після того як дані будуть згенеровані, користувач отримає повідомлення на свою електронну адресу, щоб їх завантажити.

Тепер потрібно зайти в канал текстового зв’язку й виконати команду cluster, попередньо прикріпивши сам масив аудіо даних в форматі csv.

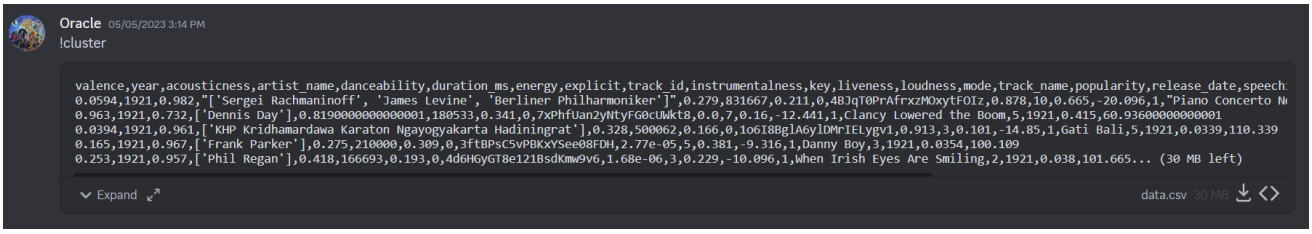

Рисунок 3.16 – Приклад виклику кластеризації

Після цього буде проведена певна валідація файлу й відправка на Python сервер.

При успішній обробці, всі кластеризовані списки відтворення будуть відправлені нам в особистий кабінет.

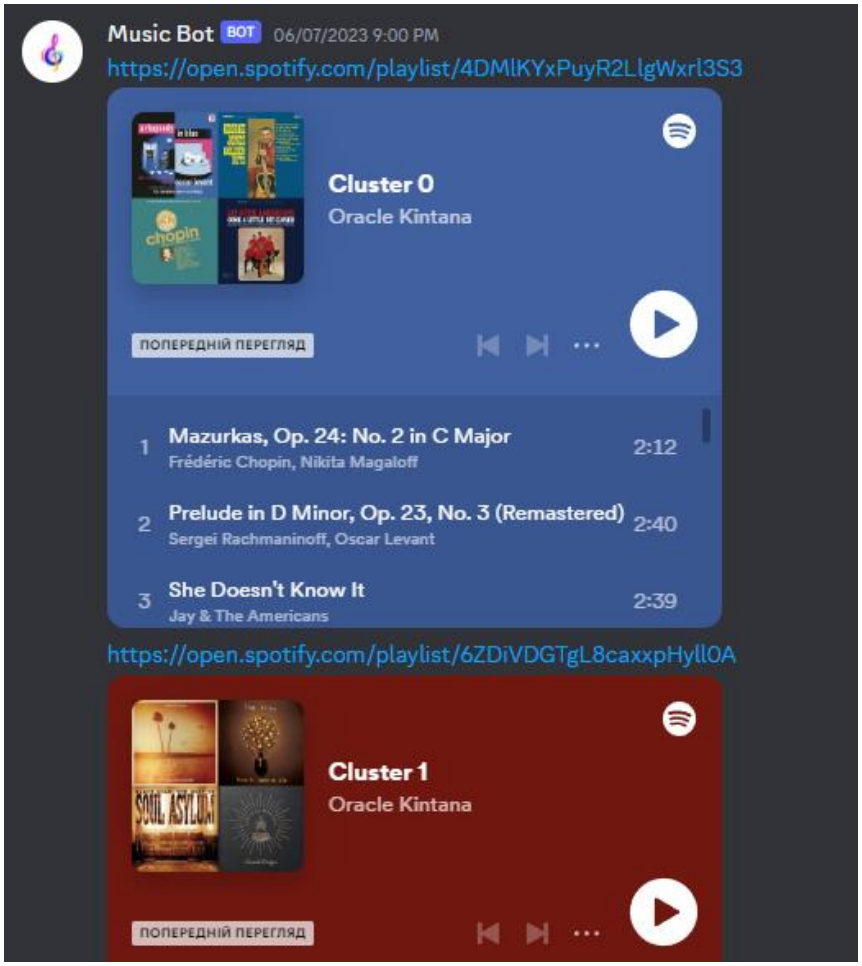

Рисунок 3.17 – Згенеровані списки відтворення

Відтак можемо відкривати клієнт Spotify та кластери. Прослухавши пісні з різних груп, можна експериментально дослідити якість розбиття композицій. Наприклад, в дев'ятому кластері можна було почути багато пісень в жанрі funk.

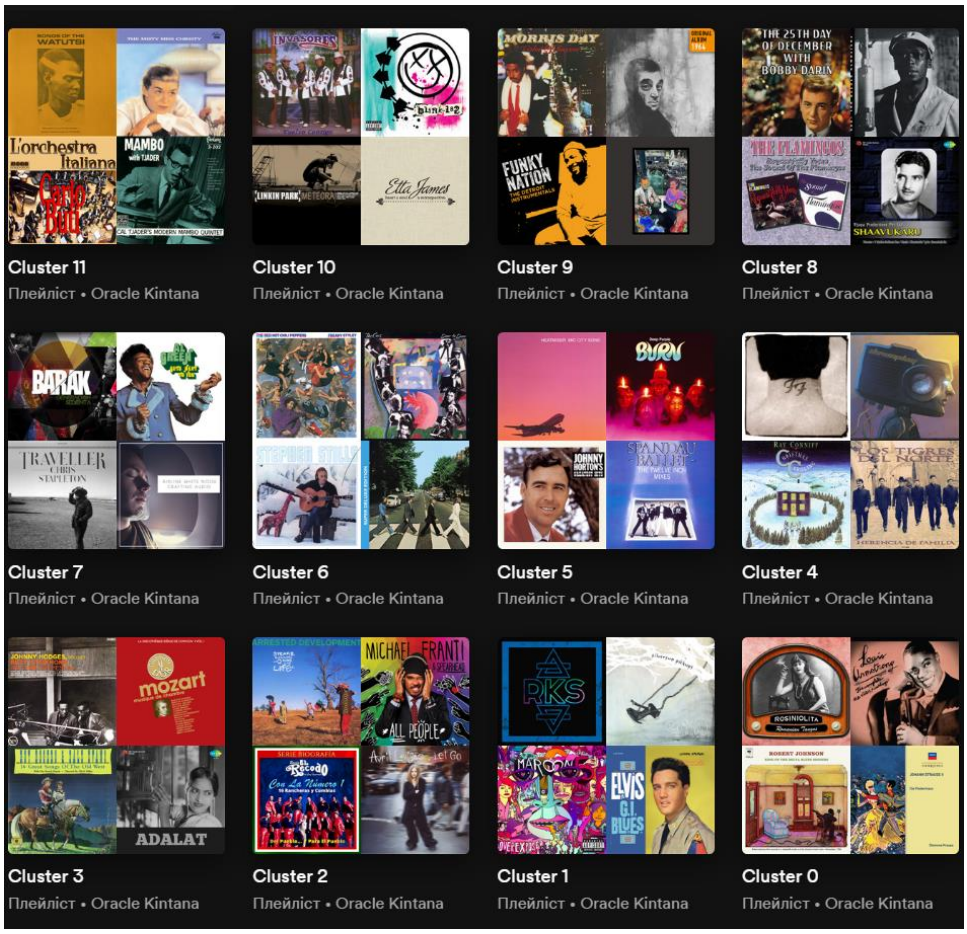

Рисунок 3.18 – Кластеризовані аудіо композиції.

Висновок до розділу

У даному розділі було викладено основні моменти розробки програмного застосунку, а також продемонстровано приклади роботи із ним. Ми детально описали всі можливі способи взаємодії з програмою, починаючи з отримання інформації про неї й закінчуючи кластерним аналізом аудіо даних.

ВИСНОВОК

В цьому документі було описано основні аспекти побудови інформаційної системи для програвання аудіо в голосових каналах зв`язку з ігровим функціоналом та елементами машинного навчання. Всі поставлені цілі були виконані, в результаті користувач отримав багатофункціональний застосунок, що покращує досвід Discord.

Я експериментально дослідив декілька звукових кодеків, зрозумів як працює цифровізація голосових повідомлень та відповідне декодування. Було проаналізовано масиви аудіо даних на основі яких побудована модель кластеризації. Незважаючи на те, що вона базувалась на простих математичних основах, результати, які вона продукувала виявились цілком задовільними. Користувач може налаштовувати параметри програвання аудіо, такі як гучність, баланс звуку, реверберація тощо. Крім того, система має ігровий функціонал, що дозволяє рандомізувати поточну аудіо композицію.

Застосунок доступний для розширення й має потенціал для розвитку в майтубтньому, особливо можу підкреслити напрям машинного навчання, а саме рекомендаційні алгоритми. Оскільки історія прослуховувань користувача є публічною, то збір даних не є проблемою. На базі списків відтворювань та аналізу аудіо характеристик композицій, можемо знаходити аудіо, що мають схожі властивості й пропонувати користувачу до прослуховування нові. Ось декілька алгоритмів, на які можна було б звернути увагу:

- Матриця факторизації
- Колаборативне фільтрування на основі згорткових нейронних мереж
- Глибокі генеративні моделі

Очевидним стало те, що великі аплікації орієнтуються на потреби широкої аудиторії, щоб отримати максимальний прибуток, отож деколи варто створити щось своє, що орієнтується на конкретну проблему.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Alexander S. Gillis. VoIP (voice over Internet Protocol) [Electronic resource] / Alexander S. Gillis, 2023. Available from: https://www.techtarget.com/searchunifiedcommunications/definition/VoIP
2. Richard O. Duda. Pattern Classification / Richard O. Duda, Peter E. Hart, David G. Stork. - John Wiley & Sons, 2012. – C. 1-17.
3. Junjie Wu. Advances in K-means Clustering / Junjie Wu - Springer Berlin Heidelberg, 2012, C. 1- 34.
4. Pulkit Sharma. The Ultimate Guide to K-Means Clustering: Definition, Methods and Applications [Electronic Resource] / Pulkit Sharma, 2019. Available from: https://www.analyticsvidhya.com/blog/2019/08/comprehensive-guide-k-means-clustering/
5. Vijay Kotu. Data Science: Concepts and Practice / Vijay Kotu, Bala Deshpande – Jonathan Simpson, Chennai, India – P.3 – Data Exploration
6. Setting up a bot application [Electronic Resource], 2022. Available from: https://discordjs.guide/preparations/setting-up-a-bot-application
7. Samer Buna. Understanding Node.js Event-Driven Architecture [Electronic Resource] / Samer Buna, 2017. Available from: https://www.freecodecamp.org/news/understanding-node-js-event-driven-architecture-223292fcbc2d/