Вінницький національний технічний університет

Факультет інформаційних технологій та комп'ютерної інженерії

Кафедра програмного забезпечення

Бакалаврська кваліфікаційна робота

на тему: Розробка інтерактивного бота для музичних рекомендацій на основі SMMC алгоритму для платформи Telegram з використанням бібліотек telebot та matplotlib

Виконав: студент ___4___ курсу
групи ___4ПІ-21Б_____
спеціальності
121 – Інженерія програмного забезпечення
(шифр і назва напряму підготовки, спеціальності)
Тіслін О.Ю.
(прізвище та ініціали)
Керівник: к.т.н., доц. каф. ПЗ Катєльніков Д.І.
(прізвище та ініціали)
Рецензент:. к.т.н., доц. каф ОТ Савицька Л.А.
(прізвище та ініціали)

Допущено до захисту

Зав. кафедри ПЗ _____
«9» ____08____ 2025 р.

ВНТУ – 2025

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

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

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

Тісліну Олексію Юрійовичу

1. Тема роботи – розробка інтерактивного бота для музичних рекомендацій на основі SMMC алгоритму для платформи Telegram з використанням бібліотек telebot та matplotlib.

Керівник роботи: Катєльніков Денис Іванович, к.т.н., доцент кафедри ПЗ, затверджені наказом вищого навчального закладу від 20 березня 2025 р. №97.

2. Строк подання студентом роботи 2 червня 2025

3. Вихідні дані до роботи: функціональні можливості Telegram-бота охоплюють: формування персоналізованих музичних рекомендацій на основі побудованої Марковської моделі; додавання треків до системи; перегляд, оновлення та аналіз статистики переходів між треками; генерацію графа ймовірнісних переходів; обробку запитів користувача через Telegram Bot API; логування дій користувача й етапів побудови моделі; виведення інформаційних повідомлень, зокрема рекомендацій, підказок щодо використання та повідомлень про помилки; візуалізацію структури моделей за допомогою відповідних бібліотек Python; забезпечення інтерактивної взаємодії з користувачем у текстовому середовищі Telegram.

4. Зміст текстової частини: вступ; аналіз та обґрунтування вибору методу розробки та постановка задачі дослідження; аналіз стану проблеми; порівняльний аналіз аналогів; розвиток машинного навчання; постановка задачі розробки; розробка архітектури та алгоритмів програмного продукту; опис інтерфейсу; розробка діаграм; розробка блок-схем алгоритмів роботи програми; розробка модуля музичних рекомендацій; розробка модуля навчання бота; розробка програмного продукту; обґрунтування вибору програмних засобів для

розробки; вибір середовища розробки; інструкція користувача; програмна реалізація; тестування програми; вибір методики тестування; тестування програмного забезпечення.

5. Перелік ілюстративної частини: актуальність теми; мета, об'єкт, предмет та завдання дослідження; постановка задачі; наукова новизна отриманих результатів; порівняльний аналіз аналогів; блок схема загальної роботи; блок схема роботи алгоритмів; обґрунтування вибору мови програмування Python; обґрунтування вибору середовища Visual Studio; блок схеми методів; апробація матеріалів бакалаврської роботи; тестування функціоналу бота.

6. Консультанти розділів роботи

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

7. Дата видачі завдання 24 березня 2025 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів бакалаврської кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Аналіз програмного забезпечення для формування музичних рекомендацій	25.03.25 – 31.03.2025	вик.
2	Розробка алгоритмів та архітектури програмного забезпечення	31.03.2025 – 05.04.2025	вик.
3	Розробка модуля музичних рекомендацій	05.04.2025 – 12.04.2025	вик.
4	Розробка модуля навчання бота	12.04.2025 – 23.04.2025	вик.
5	Розробка програмного забезпечення	23.04.2025 – 01.05.2025	вик.
6	Тестування програмного забезпечення	01.05.2025 – 10.05.2025	вик.
7	Оформлення матеріалів до захисту БКР	10.05.2025 – 30.05.25	вик.

Студент _____ **О.Ю Тіслін**
(підпис) (ініціали та прізвище)

Керівник бакалаврської кваліфікаційної роботи _____ **Д.І. Катєльніков**
(підпис) (ініціали та прізвище)

Анотація

УДК 004.4:78.01

Тіслін О.Ю. Розробка Telegram-бота для рекомендації музичних треків на основі Марковських ланцюгів, бакалаврська кваліфікаційна робота зі спеціальності 121 – інженерія програмного забезпечення, освітня програма – інженерія програмного забезпечення. – Вінниця: ВНТУ, 2025. – 60 с.

Бакалаврська дипломна робота складається з 60 сторінок формату А4, на яких є 23 рисунки, 6 таблиці, список використаних джерел містить 25 найменувань

У бакалаврській кваліфікаційній роботі обґрунтовано доцільність та практичну цінність створення програмного забезпечення у вигляді Telegram-бота, що реалізує систему рекомендацій музичних треків з використанням Марковської моделі. Здійснено аналіз існуючих підходів до побудови рекомендаційних систем, проведено дослідження ефективності використання стохастичних процесів у задачах прогнозування вибору контенту.

Розроблено алгоритм побудови Марковського графа переходів між треками, алгоритм генерації персоналізованих рекомендацій, а також реалізовано інструменти для збору статистики, візуалізації структури переходів та інтерактивної взаємодії з користувачем. Telegram-бот реалізовано за допомогою мови програмування Python та Telegram Bot API у середовищі розробки Visual Studio Code. Проведено модульне тестування функціоналу, розроблено інструкцію користувача та визначено вимоги до середовища експлуатації.

Результати роботи можуть бути використані для створення рекомендаційних сервісів, що працюють у месенджерах, і для розширення можливостей автоматизованої взаємодії з користувачами в системах доставки контенту.

Ключові слова: Telegram-бот, рекомендаційна система, Марковські ланцюги, Python, API, взаємодія з користувачем, автоматизація

ABSTRACT

UDC 004.4

Tislin O.Y. Development of a Telegram Bot for Music Track Recommendation Based on Markov Chains: bachelor's thesis in specialty 121 – Software Engineering, educational program – Software Engineering. – Vinnytsia: VNTU, 2025. – 60 p.

The bachelor's thesis consists of 60 A4 pages, which have 23 figures, 6 tables, the list of sources used contains 26 items

The bachelor's qualification thesis substantiates the relevance and practical value of developing software in the form of a Telegram bot that implements a music track recommendation system using a Markov model. The study includes an analysis of existing approaches to recommendation systems and investigates the applicability of stochastic processes in predicting user content preferences.

Algorithms have been developed for constructing a Markov transition graph between tracks, generating personalized recommendations, collecting statistics, visualizing transition structures, and ensuring interactive user communication. The Telegram bot was implemented in Python using the Telegram Bot API within the Visual Studio Code environment. Software modules were tested, a user manual was prepared, and system requirements were specified.

The results of the work can be applied to the development of recommendation services integrated into messaging platforms, enhancing automated user engagement in content delivery systems.

Keywords: Telegram bot, recommendation system, Markov chains, Python, API, user interaction, automation.

.

ЗМІСТ

ВСТУП

Обґрунтування вибору теми дослідження.

Сучасна епоха характеризується безпрецедентним зростанням цифрової комунікації, де технології створюють нові виміри людської взаємодії. Музика одна з найпотужніших форм самовираження та культурного обміну потребує нових інструментів персоналізації в цифровому просторі. Пандемічні обмеження та геополітичні виклики останніх років значно прискорили дистанціювання людських спільнот, що посилило роль цифрових інновацій у створенні почуття спільності та емоційного зв'язку.

Месенджер Telegram [1], з його відкритою екосистемою та потужним API, став одним із найвпливовіших комунікаційних каналів сучасності. За даними останніх досліджень, понад 700 мільйонів користувачів щодня проводять на платформі більше 2 годин, що робить цей простір ідеальним для впровадження сучасних розробок у сфері особистісно-орієнтованих музичних рекомендацій.

У контексті перенасичення інформаційного простору, де щодня з'являються тисячі нових музичних творів, актуалізується потреба в інтелектуальних системах фільтрації та персоналізації контенту. Традиційні методи рекомендацій часто ігнорують психоемоційний контекст слухача, послідовність прослуховування та динаміку зміни преференцій, що призводить до поверхневих та стереотипних пропозицій.

Запропонований метод SMMC, Sequential Mixed Markov Chain, алгоритм є інноваційним підходом до аналізу музичних уподобань, що враховує не лише характеристики композицій, але й частоту рекомендацій та контекстуальні фактори. Візуалізація цих складних взаємозв'язків за допомогою matplotlib створює новий рівень прозорості та довіри до рекомендаційної системи.

Існуючі музичні рекомендаційні сервіси мають суттєві обмеження: домінування комерційних інтересів над користувацькими потребами; алгоритмічні «бульбашки», що обмежують музичний світогляд; відсутність

інтерактивного зворотного зв'язку; складність інтерфейсів; непрозорість критеріїв рекомендацій. Більшість сервісів фокусуються на популярному контенті, залишаючи поза увагою нішеві музичні напрямки та незалежних виконавців.

Таким чином, розробка інтерактивного бота для музичних рекомендацій на основі SMMC алгоритму для платформи Telegram становить актуальну науково-практичну задачу, що відповідає сучасним викликам персоналізації цифрового контенту.

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

Мета та завдання дослідження. Метою бакалаврської кваліфікаційної роботи є підвищення рівня автономності системи музичних рекомендацій від зовнішніх сервісів та API за рахунок використання локальної логіки аналізу вподобань на основі SMMC-алгоритму. Це також дозволило підвищити рівень відповідності рекомендацій до особистих потреб користувача.

Пропонується розробка екосистеми музичних рекомендацій, що ґрунтується на принципово новому підході до сегментації та кластеризації музичних матриць. Архітектура системи передбачає модульну організацію компонентів: рекомендаційна система на базі SMMC, інтерфейсний шар реалізований через telebot, візуалізаційний компонент за допмогою matplotlib.

Відповідно до поставленої мети необхідно виконати такі завдання:

– проаналізувати засоби та методи для реалізації інтерактивної системи;

– дослідити нейропсихологічні аспекти сприйняття музики та їх вплив на алгоритми рекомендацій;

– розробити математичну модель SMMC алгоритму з адаптацією до контекстуальних факторів;

– створити багаторівневу архітектуру системи з мікросервісним підходом;

– імплементувати інтелектуальний інтерфейс взаємодії через Telegram API з використанням бібліотеки telebot;

– розробити систему інтерактивної візуалізації музичних кластерів за допомогою matplotlib.

Об’єкт дослідження – процес розробки інтерактивного Telegram бота для музичних рекомендацій.

Предмет дослідження – методи та засоби лінійної алгебри для роботи з матрицями та формування музичних рекомендацій на їх основі.

Методи дослідження. Протягом дослідження було використано: теорія лінійної алгебри для розробки алгоритму для формування рекомендацій; методи теорії ймовірностей для моделювання переходів між станами системи; методи об’єктно-орієнтованого програмування для розробки архітектури класів та сервісів програмного забезпечення; комп’ютерне моделювання для аналізу та перевірки отриманих теоретичних положень.

Новизна отриманих результатів.

1. Подальшого розвитку отримав метод обробки користувацьких запитів у SMMC-боті музичних рекомендацій, у якому, на відміну від традиційних ботів, реалізовано локальну логіку аналізу вподобань без підключення до зовнішніх сервісів. Це дозволило зробити бота незалежним від зовнішніх сервісів, швидким у відповіді та придатним для роботи в умовах обмеженого доступу до Інтернету.

2. Подальшого розвитку отримав метод доступу до музичних композицій, який, на відміну від існуючих методів, базується на завчасно підготовлених списках треків та простих текстових структурах без залучення баз даних або зовнішніх API. Такий підхід забезпечив високу швидкість роботи та автономність системи.

3. Подальшого розвитку отримав метод рекомендації музики на основі SMMC- процедур, у якому, на відміну від систем, що використовують машинне навчання, реалізовано просту логіку порівняння параметрів треків і вподобань користувача. Це дозволило створити легкий, прозорий і керований механізм

рекомендацій, адаптований до потреб користувачів.

Практична цінність отриманих результатів полягає в тому, що на основі сформульованих у роботі теоретичних положень було запропоновано алгоритми та розроблено програмні засоби боту для музичних рекомендаці, що ґрунтуються на SMMC-алгоритмах та Марковських моделях.

Особистий внесок здобувача. Усі наукові результати, викладені у бакалаврській кваліфікаційній роботі і в науковій публікації[2], отримані особисто автором.

Апробація матеріалів бакалаврської кваліфікаційної роботи. Описані у дослідженні положення доповідались на конференції «LIII Всеукраїнській науково-технічній конференції факультету інформаційних технологій та комп'ютерної інженерії (2025)» та опубліковані в тезах доповіді [2].

Публікації. За тематикою дослідження опубліковано 1 наукову працю, що доповідалась на конференції «LIII Всеукраїнській науково-технічній конференції факультету інформаційних технологій та комп'ютерної інженерії (2025)» та опубліковано в тезах доповіді [2].

1 АНАЛІЗ ТА ОБҐРУНТУВАННЯ ВИБОРУ МЕТОДУ РОЗРОБКИ ТА ПОСТАНОВКА ЗАДАЧІ ДОСЛІДЖЕННЯ

1.1 Аналіз стану проблеми

З огляду на динамічну еволюцію інформаційних технологій, впровадження інтелектуальних ботів, здатних до контекстно-залежної обробки даних, набуває дедалі більшого значення у царині цифрових сервісів, зокрема в екосистемах галузі медіа. Однією з перспективних парадигм у цьому напрямі є розроблення інтерактивних агентів, інтегрованих у середовище месенджерів, що забезпечують адаптивні рекомендації на основі аналізу користувацьких обраних треків. Зокрема, Telegram, завдяки відкритому API, гнучкості платформи та високій проникності серед цільової аудиторії, створює сприятливі умови для реалізації таких застосунків [3].

Порівняно з традиційними мобільними застосунками, Telegram боти функціонують без необхідності в інсталяції додаткового програмного забезпечення, що істотно знижує поріг входження та полегшує доступ до запитуваного вмісту. Разом із мінімізацією технічних вимог до кінцевого користувача, інтеграція ботів у середовище месенджерів дозволяє досягти високого рівня доступності функціоналу, що традиційно був притаманний лише повноцінним програмним застосункам. Така модель взаємодії значно оптимізує процес споживання контенту, зокрема музичного, забезпечуючи прямий і безперешкодний канал комунікації між сервісом і юзером. У цьому контексті постає необхідність не лише створення ефективної логіки обробки запитів, а й забезпечення інтелектуального аналізу вподобань, що включає залучення засобів математичного моделювання й інструментів машинного навчання [4].

Сучасні рекомендаційні системи дедалі частіше застосовують методи обробки взаємодії між користувачами та об'єктами, зокрема алгоритму SMMC, що здійснює точніші передбачення на основі схожості між користувачами або елементами. Застосування SMMC у Telegram-боті відкриває широкі можливості

для персоналізованих рекомендацій, адаптованих до смаків конкретного користувача, навіть за умови обмеженості початкових даних [5].

Розробка таких систем потребує не лише технічних знань, але й уваги до потреб і очікувань юзерів. Інтерфейс має бути інтуїтивно зрозумілим, а відповіді швидкими та точними. Зручна взаємодія, можливість побачити візуалізований аналіз музичних вподобань, гнучкість у налаштуваннях і швидкість відповіді, усе це формує позитивний досвід користувача, що, у свою чергу, підвищує інтерес та бажання використовувати сервіс.

Отже, створення інтерактивного бота для музичних рекомендацій на базі SMMC-алгоритму є актуальним напрямом у сучасних інформаційних технологіях. Поєднання алгоритмічної точності, зручності систем обміну повідомленнями і можливостям Python бібліотек створює ефективний інструмент, який дозволяє зробити музичний контент ближчим і доступнішим. У світлі постійного розвитку інформаційних платформ, необхідно стежити за прогресом в області персоналізованих рекомендацій і впроваджувати актуальні програмні розробки [6].

1.2 Порівняльний аналіз аналогів

Зі зростанням темпів розвитку цифрових технологій змінюється не лише спосіб споживання контенту, а й очікування користувачів щодо персоналізованих рекомендацій. Особливо це помітно в музичній індустрії, де автоматизація рекомендаційних процесів стала важливою частиною взаємодії з аудиторією. У поданому контексті дедалі більше уваги привертають розумні системи на основі ботів, що працюють у популярних системах обміну повідомленнями, зокрема в Telegram.

Інтерактивні боти забезпечують зручну та інтуїтивну взаємодію з користувачем без потреби в окремих застосунках чи складних параметрах, а також дозволяють автоматизувати процес підбору музики відповідно до вподобань слухача, зберігаючи при цьому швидкість реакції системи. У таких системах ключову роль відіграє точність алгоритмів, зокрема SMMC, який дає

змогу робити персоналізовані музичні рекомендації навіть на основі обмеженого набору початкових даних.

Серед існуючих застосунків, схожих за функціональністю, варто згадати декілька популярних ботів і сервісів, які становлять конкуренцію під час створення нових застосунків. Наприклад: Spotybot, MusicTubeBot_Bot , TuneMyMusic , FredBoat

На рисунку 1.1 представлене один із таких застосунків – Spotybot, телеграм-бот, що дозволяє здійснювати пошук музичних композицій, переглядати альбоми, а також прослуховувати короткі уривки пісень завдяки інтеграції з сервісом Spotify. Застосунок вирізняється лаконічним інтерфейсом і орієнтований переважно на швидкий пошук музичних композицій за назвою або артистом. Його функціональність обмежується базовим обміном даних через API музичного сервісу, однак бот демонструє потенціал інтеграції сторонніх застосунків для покращення користувацького сприйняття [7].

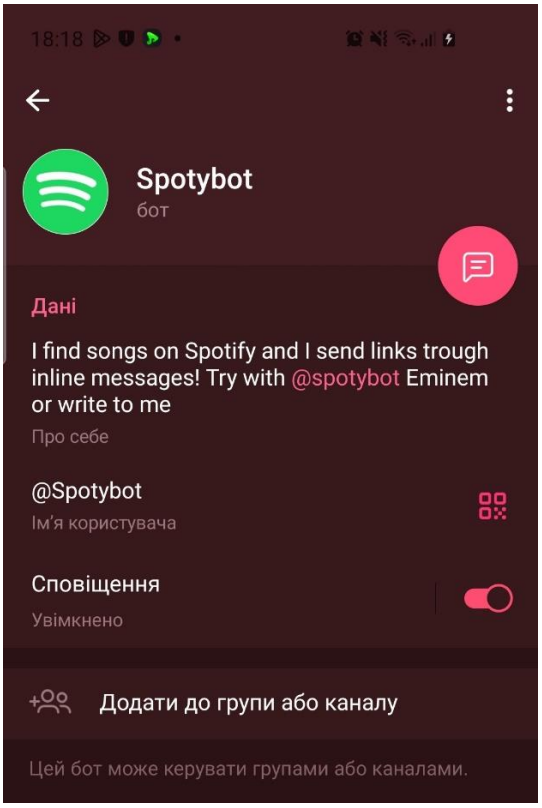

Рисунок 1.1 – Опис можливостей Spotybot

На рисунку 1.2 зображено інший приклад Musictube Bot, який працює з базою YouTube Music. Його основною особливістю є можливість слухати музику безпосередньо в чаті, що значно спрощує доступ до шуканого. Такий підхід добре ілюструє актуальну тенденцію до зменшення кількості кліків і переходів для користувача, забезпечуючи більш «негайну» взаємодію із сервісом. Незважаючи на нескладну реалізацію, цей бот демонструє, як навіть з обмеженим функціоналом можна досягти високого рівня наявності віртуальних користувачів [8].

Рисунок 1.2 – Конвертація посилань MusicTubeBot_Bot

На рисунку 1.3 зображено TuneMyMusic, веб-сервіс, який дозволяє користувачам переносити музичні плейлисти між різними потоковими

платформами, такими як Spotify, Apple Music, YouTube Music, Deezer, Tidal, а також багато у нього інтегровано багато корисних сервісів окрім. Він забезпечує швидке та зручне перенесення плейлистів, підтримує синхронізацію та резервне копіювання музичних колекцій, що дозволяє зберігати улюблені треки навіть при зміні додатку. Однією з ключових переваг TuneMyMusic є можливість синхронізації плейлистів між декількома застосунками. Наприклад, можна налаштувати автоматичне оновлення, щоб нові треки на Spotify автоматично додавалися до плейлиста на YouTube Music. Це дозволяє зберігати актуальність музичної колекції на різних сервісах без непотрібних операцій. TuneMyMusic також надає функцію резервного копіювання, що дозволяє експортувати плейлисти у форматах CSV або TXT для збереження на локальному пристрої. Це корисно за умови, коли користувач хоче створити резервну копію або проаналізувати свою музичний плейлист. Крім того, сервіс підтримує перенесення не тільки особистих плейлистів, але й загальнодоступних, що значно розширює можливості збереження музики [9].

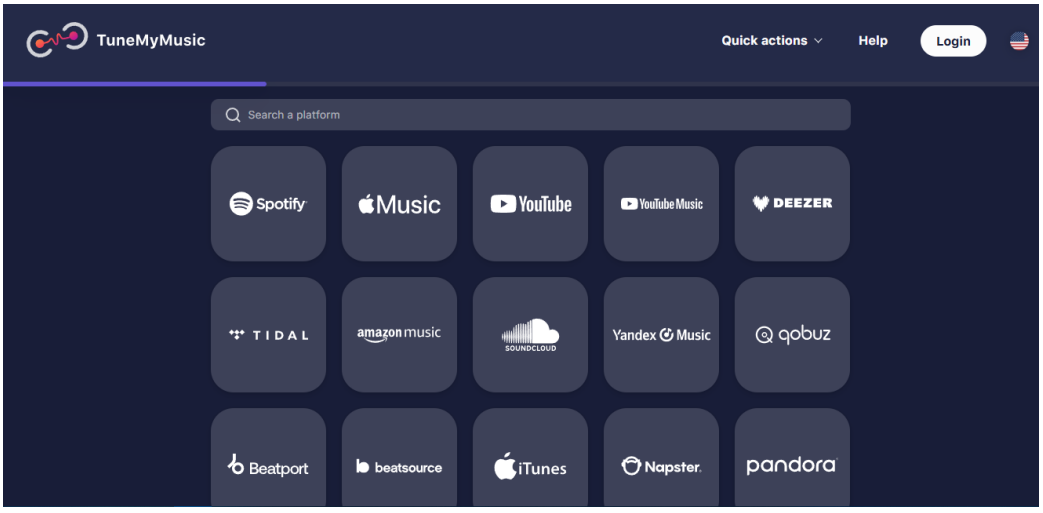

Рисунок 1.3 – Можливості інтегрованих сервісів у TuneMyMusic

На рисунку 1.4 зображено FredBoat, музичний Discord-бот, який робить акцент на аналізі музичних уподобань на основі контексту та настрою користувача. Він використовує комбінацію алгоритмів моделювання поведінки користувача на сайті та аналізу текстових запитів для визначення поточного

настрою. FredBoat підтримує відтворення музики з різних джерел, наприклад, SoundCloud, Bandcamp, Twitch, Deezer, Vimeo, Dailymotion. Користувачі можуть додавати треки до черги, перемішувати плейлисти та налаштовувати повторне відтворення, що забезпечує гнучкість у керуванні музичним плейлистом [10].

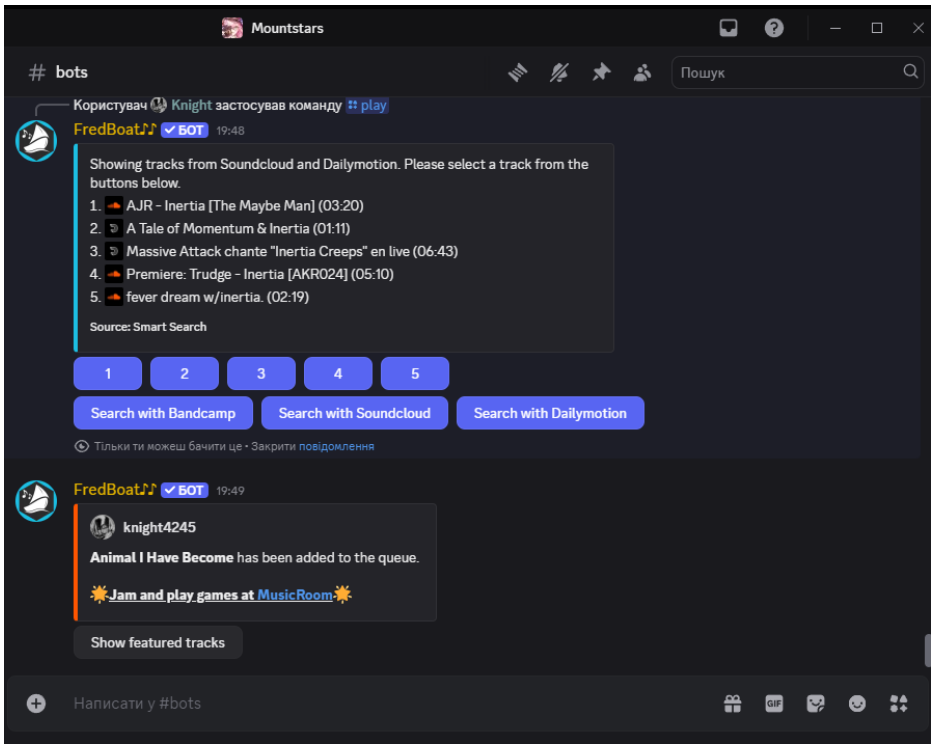

Рисунок 1.4 – Замовлення треків у FredBoat

У процесі розробки інтерактивного Telegram-бота для рекомендацій на основі алгоритму SMMC було проаналізовано наявні рішення у цій сфері, що дало змогу краще зрозуміти наявні інноваційні тренди. Зокрема, такі застосунки, Spotybot, MusicTubeBot_Bot, TuneMyMusic, FredBoat, продемонстрували ключові функції, які очікують сучасні користувачі: інтеграцію з потоковими сервісами, доступ до історії прослуховувань, побудову персоналізованих порад тощо. Проте жоден із них не об’єднує в собі повного функціонального спектру, включно з побудовою візуалізацій для аналізу рекомендацій або збереженням музики для офлайн-доступу, як це реалізовано у власній розробці YoutubeConverter. Це дозволяє не лише запропонувати нові

підходи до рекомендацій, але й створити зручний інтерфейс на базі бібліотеки telebot із використанням matplotlib для графічного представлення результатів. Підсумки порівняння функціональних характеристик систем наведені у таблиці 1.1, що чітко визначає переваги запропонованого рішення.

Таблиця 1.1 – Порівняльна характеристика онлайн платформ

Назва функціоналу	Spotybot	MusicTubeBot_Bot	TuneMyMusic	FredBoat	YoutubeConverter власна розробка
Сумісність з YouTube та Spotify	1	1	1	1	1
Рекомендації на основі смаків	1	1	1	0	0
Перегляд попередніх треків	0	1	0	1	1
Побудова графічних моделей для рекомендацій	0	0	0	0	1
Функція офлайн-збереження музики	0	0	0	0	1
Сумарний коефіцієнт	2	3	2	2	4

Згідно вищенаведеної таблиці проаналізовані інструменти: FredBoat, TuneMyMusic, Spotybot, MusicTubeBot_Bot, які демонструють потенціал інформаційних технологій у цій сфері, однак водночас виявляють і низку обмежень, які створюють підґрунтя для вдосконалення. З урахуванням порівняльного аналізу функціональності, розробка власного Telegram-бота

YouTubeConverter із використанням SMMC-алгоритму та бібліотек telebot і matplotlib є доцільною й конкурентоспроможною.

Порівняно з наявними рішеннями, запропонований бот охоплює увесь спектр базових можливостей: підтримку платформ YouTube та Spotify, генерацію рекомендацій, збереження історії прослуховувань, а також пропонує додатковий функціонал, зокрема, візуалізацію процесу рекомендацій за допомогою графіків та можливість офлайн-прослуховування. Вищенаведені пункти відображають переваги YouTubeConverter між аналогами.

Окрема увага приділяється створенню прозорого й зрозумілого інтерфейсу, де користувач не лише отримує рекомендації, але й бачить, як саме вони формуються, завдяки графікам, згенерованим у matplotlib. Така прозорість створює та покращує довіру до системи й дозволяє користувачу краще зрозуміти власні музичні смаки. Крім того, реалізація збереження історії треків допомагає підтримувати довготривалу персоналізацію, що актуально для користувачів, які систематично користуються рекомендаційними сервісами.

Також вагомим фактором є підтримка офлайн-доступу до музичних композицій. Цей функціонал відсутній у всіх проаналізованих аналогів, але надзвичайно актуальний для користувачів із нестабільним інтернет-з'єднанням або тих, хто часто в дорозі. Такий підхід покращує загальний користувацький досвід та розширює потенційне коло прослуховувачів музики.

Згідно з результатами таблиці 1.1, сумарний коефіцієнт функціональності, який має YouTubeConverter складає 4 балів, що є найвищим показником серед проаналізованих застосунків. Наприклад, FredBoat та TuneMyMusic мають лише по 2 бали, Spotybot та MusicTubeBot_Bot 3. Це є значною перевагою розроблюваного рішення, що має ширший набір функцій та краще адаптоване до сучасних вимог користувачів.

Отже, розробка інтерактивного бота з використанням бібліотеки telebot з метою інтерактивної взаємодії з користувачем та засобами matplotlib для візуалізації музичних переваг користувача є актуальною. Вбудування алгоритму SMMC працює ефективно навіть у випадках зі спарсованими або

фрагментарними даними, що відповідає реаліям нових користувачів системи, які ще не мають сформованого профілю. Створений інтерактивний чат бот не тільки сприяє кращій адаптації до потреб користувачів, але й значно підвищує їхню задоволеність, відповідаючи стандартам новітнього цифрового простору.

1.3 Розвиток машинного навчання

Машинне навчання стрімко змінило та покращило численні галузі, включаючи сферу мультимедійних рекомендацій, де адаптивні алгоритми дозволяють забезпечувати персоналізований користувацький досвід. Упродовж останнього десятиліття спостерігається невпинний інтерес до використання стохастичних моделей та нейромереж у завданнях фільтрації та передбачення вподобань. Це особливо актуально для музичних сервісів, таких як Spotify, YouTube Music та Telegram-боти, які інтегрують інтелектуальні системи на основі історії прослуховувань користувача.

Значну роль у рекомендаційних системах відіграють марковські процеси, математичні моделі [11], які дозволяють описувати ймовірнісні переходи між станами, що у музичному контексті відповідає трекам або жанрам. Використання таких моделей дозволяє не лише фіксувати шаблони прослуховування, а й прогнозувати наступні музичні вподобання з урахуванням контексту та динаміки користувача.

При розробці інтерактивного Telegram-бота для музичних рекомендацій було застосовано вдосконалений стохастичний підхід SMMC, що поєднує класичну Марковську модель із адаптивним зважуванням переходів. Основна ідея полягає у побудові перехідної матриці, яка відображає ймовірність переходу від одного треку до іншого на основі дій користувача. Таким чином, система поступово навчається на індивідуальних уподобаннях, зберігаючи їх у вигляді графа переходів.

Після кожної взаємодії, тобто прослуховування чи запиту, треку бот оновлює свою перехідну матрицю. Якщо користувач слухає певну послідовність треків, бот фіксує цю закономірність, коригуючи ймовірності

переходів відповідно. При першій взаємодії нова комірка ініціалізується рівномірно для всіх можливих наступних треків, що відповідає принципу максимальної ентропії. У разі повторного прослуховування конкретної послідовності, відповідна ймовірність посилюється, що забезпечує самонавчання системи та зменшує частоту хибних рекомендацій.

Унікальність цього підходу в адаптації моделі під конкретного користувача без залучення складних нейронних мереж чи зовнішніх баз даних. Таким чином, Telegram-бот є легким, автономним і прозорим у функціонуванні. Додатково, можливість візуалізації ймовірнісних зв'язків між треками через бібліотеку matplotlib дозволяє користувачеві бачити логіку рекомендацій, що є важливим чинником довіри до системи.

На відміну від традиційних алгоритмів колаборативної фільтрації, які вимагають значних обчислювальних ресурсів і даних про багатьох користувачів, реалізація через SMMC [12] дозволяє досягти високої точності при мінімальних витратах – це ідеальне рішення для інтерактивних ботів у месенджерах, таких як Telegram.

Таким чином, використання стохастичних методів у побудові рекомендаційної системи дозволяє досягти балансу між адаптивністю, швидкодією та інтерпретованістю, що й стало основою розробки Telegram-бота для персоналізованих музичних рекомендацій.

1.4 Постановка задачі розробки

Після ретельного аналізу існуючих музичних ботів та оцінки їх сильних і слабких сторін, було визначено низку ключових завдань для розробки нашого інтерактивного музичного бота:

1. Оптимізувати метод рекомендацій таким чином, щоб підбір музики був ефективний, що відповідає вподобанням користувачів, який забезпечить точні та персоналізовані рекомендації, адаптуючись до існуючої популярної музики.

2. Розробка інтуїтивно зрозумілого інтерфейсу, створити зручний графічний інтерфейс користувача, який дозволить користувачам швидко

отримувати музичні рекомендації, переглядати історію прослуховувань та взаємодіяти з контентом без складнощів.

3. Інтегрувати з популярними платформами, такими як Spotify та YouTube, що дозволить користувачам отримувати рекомендації з різних джерел. Крім того, необхідно впровадити функції візуалізації рекомендацій за допомогою діаграм та підтримку офлайн-прослуховування.

4. Провести всебічне тестування, щоб перевірити надійність та коректність його роботи в реальних умовах.

Ці завдання є основою для створення музичного бота, який не лише задовольнятиме потреби користувачів у швидкому та зручному доступі до музики, але й надасть інноваційний підхід до рекомендацій через використання сучасних алгоритмів та інтеграцію з популярними платформами.

1.5 Висновки

У першому розділі було проведено аналіз існуючих аналогів для автоматизованих музичних рекомендацій, що дозволило виявити основні сильні та слабкі сторони таких систем. Хоча існуючі сервіси мають певні переваги, зокрема у зручності користування, вони не завжди забезпечують високий рівень персоналізації рекомендацій, а також обмежені можливості інтерактивного взаємодії з користувачем. Ці недоліки стали основою для розробки нового інтерактивного бота, який буде використовувати алгоритм SMMC для точних та індивідуальних рекомендацій музики. Унікальність цього рішення полягає в тому, що бот не тільки пропонуватиме рекомендації, але й дозволить користувачам активно взаємодіяти з контентом через Telegram, використовуючи інтуїтивно зрозумілий інтерфейс. Вдосконалення алгоритму і інтеграція з популярними музичними платформами, а також використання бібліотек telebot та matplotlib для візуалізації даних дозволить створити продукт, який значно покращить користувацький досвід у порівнянні з існуючими аналогами.

2 РОЗРОБКА АРХІТЕКТУРИ ТА АЛГОРИТМІВ ПРОГРАМНОГО ПРОДУКТУ

2.1 Розробка методу обробки користувацьких запитів у SMMC-боті музичних рекомендацій

У процесі розробки діаграм особливу увагу було приділено зручності сприйняття та наочності поданих даних. Для цього використовувалися інструменти бібліотеки Matplotlib [13], що дозволяють гнучко налаштовувати параметри візуалізації, зокрема, колір, підписи осей, масштабування та обертання підписів, що особливо актуально при виведенні довгих URL-адрес. Важливо також, що гістограма генерується динамічно на основі реальних користувацьких даних, що забезпечує актуальність статистики при кожному зверненні до команди. Такий підхід у розробці не лише автоматизує формування аналітичних звітів, а й робить їх адаптивними до змін у поведінці користувача. В результаті діаграми стають не просто візуальним елементом, а повноцінним інструментом зворотного зв’язку, який перетворює числову інформацію на зрозумілу, структуровану форму для аналізу.

На рисунку 2.1 представлена візуалізація цієї ж статистики у вигляді гістограми за допомогою команди меню /stats. Такий графічний підхід дозволяє значно легше сприймати й аналізувати дані, порівнюючи частотність рекомендацій для кожного треку. Осі графіка підписані належним чином: по горизонталі вказані URL-адреси треків, а по вертикалі кількість рекомендацій. Візуальна репрезентація суттєво підсилює сприйняття інформації, особливо коли йдеться про динаміку прослуховувань або вподобання за певний період часу.

Гістограма виконує роль інструмента, який перетворює сухі цифри у зрозумілу й інтуїтивно доступну форму, полегшуючи порівняння між треками.

Горизонтальна вісь містить підписи у вигляді URL-адрес відповідних треків, що дає змогу безпосередньо ідентифікувати кожну музичну одиницю, а вертикальна демонструє кількість рекомендацій, отриманих кожним треком у

межах заданого періоду. Така структура дозволяє не лише здійснювати загальний огляд популярності контенту, а й робити висновки щодо вподобань користувачів, динаміки змін інтересу чи ефективності окремих рекомендацій.

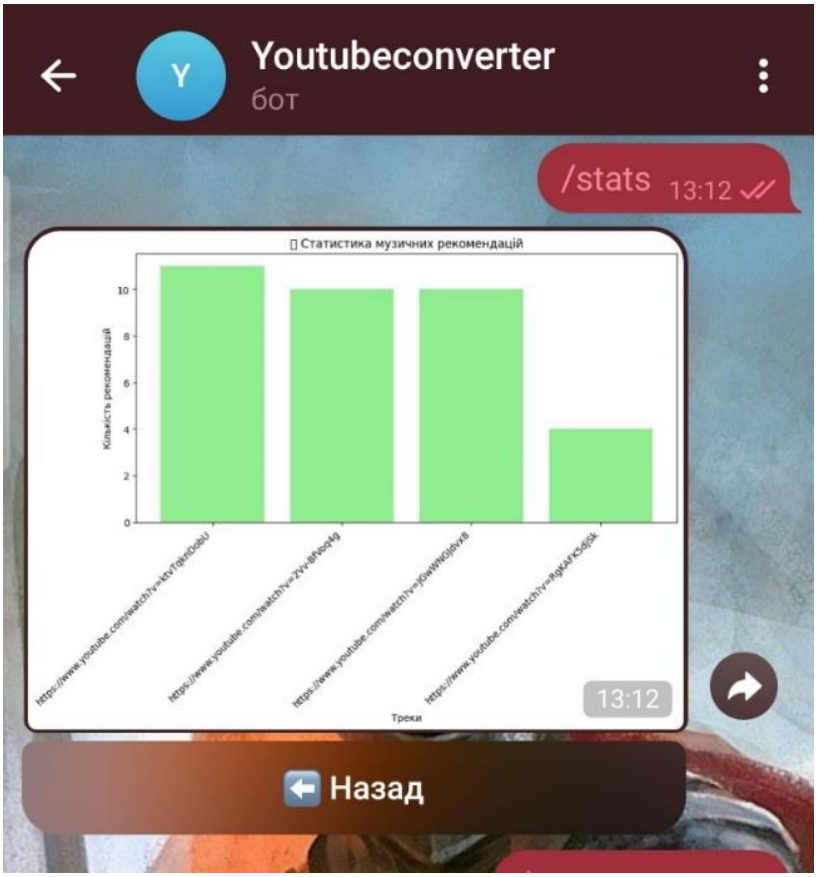

Рисунок 2.1 – Візуалізація статистики у вигляді графіку

Крім того, важливою особливістю є наявність окремих блоків статистики, що підвищує залученість користувачів. Наявність як загальної, так і персональної статистики дозволяє користувачеві оцінити свій внесок у формування моделі, відчути взаємозв’язок з системою й бачити результати власної активності. Аналогічно, візуалізація графа переходів виконує роль зворотного зв’язку, перетворюючи абстрактну модель рекомендацій на зрозумілу структуру.

На рисунку 2.2 представлено граф переходів за допомогою команди меню /graph між треками, який виконує ключову роль у візуалізації внутрішньої логіки роботи рекомендаційної системи. Цей граф побудований на основі

Марковської моделі, де кожна вершина відповідає конкретному треку, а ребра між ними символізують переходи тобто, як часто після прослуховування одного треку система рекомендувала інший. Таким чином, уся історія взаємодії користувача з ботом перетворюється на наочну структуру, яка демонструє не лише вибір, а й послідовність музичних переваг.

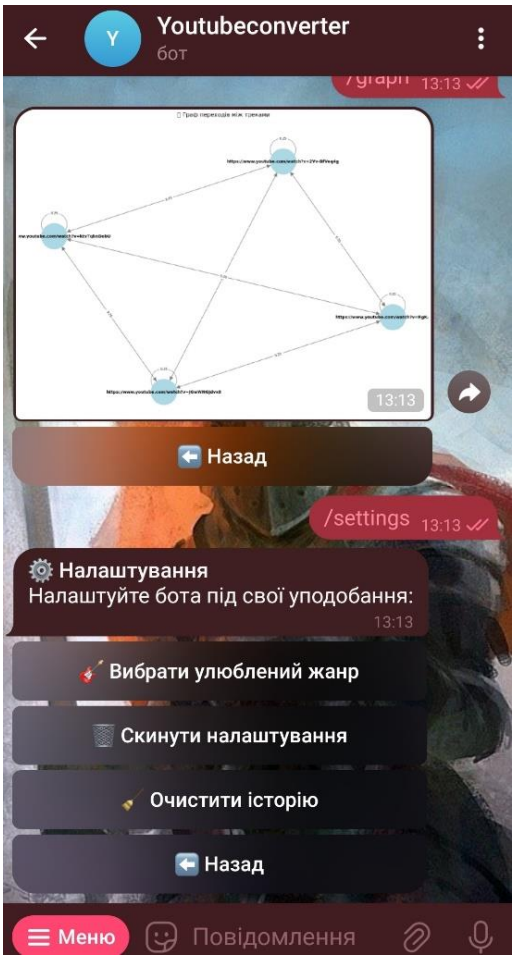

Рисунок 2.2 – Граф переходів та налаштування

Особливістю цього графа є те, що він відображає не просто загальні рекомендації, а динаміку переходів, які формують основу подальших пропозицій. Якщо певна композиція з’являється як вихідна точка для багатьох інших треків, це свідчить про її високу роль у музичному профілі користувача, вона є своєрідним вузлом, що спрямовує хід рекомендацій. У той час як треки, до яких веде велика кількість стрілок, можуть сприйматися як найочікуваніші фінали або найпопулярніші цілі середньої послідовності.

Наявність цього графа значно підвищує прозорість системи. Користувач бачить не просто набір пісень, а логічну схему, в якій він сам є джерелом даних. Такий підхід підсилює ефект взаємодії, коли інтерактивний бот не видається «чорним ящиком», а перетворюється на партнера, чий хід думки можна прослідкувати. Відповідно, це не лише зміцнює довіру до бота, а й пробуджує зацікавлення, користувач прагне змінити або доповнити модель, граючись із власними вподобаннями. Як видно з рисунка, такі візуалізації роблять невидимі процеси видимими, в цьому полягає їхня цінність.

Отже, було здійснено візуальне представлення функціонування Telegram-бота шляхом демонстрації ключових етапів його роботи за допомогою рисунків роботи. Такий підхід дозволив наочно відобразити інтерактивність інтерфейсу, реакцію системи на команди користувача, а також візуалізацію статистичних даних і графа переходів. При розробці особливу увагу було приділено графічним елементам, які не лише покращують сприйняття інформації, але й підвищують загальну зручність користування ботом. Гістограми та графи, що відображають динаміку прослуховувань і зв’язки між треками, слугують інструментом зворотного зв’язку, формуючи у користувача уявлення про принципи роботи рекомендаційної системи.

2.2 Розробка методу доступу до музичних композицій

На рисунку 2.3 зображено блок-схему, яка описує загальну логіку функціонування Telegram-бота, який надає користувачу різноманітні функції, зокрема рекомендації музики, перегляд статистики та керування налаштуваннями.

Робота застосунку починається з ініціалізації, під час якої викликається функція setup_bot_commands. Вона відповідає за налаштування команд, які будуть доступні користувачу у взаємодії з ботом. Після цього бот переходить у режим постійного очікування повідомлень через виклик bot.infinity_polling, що забезпечує обробку команд у реальному часі.

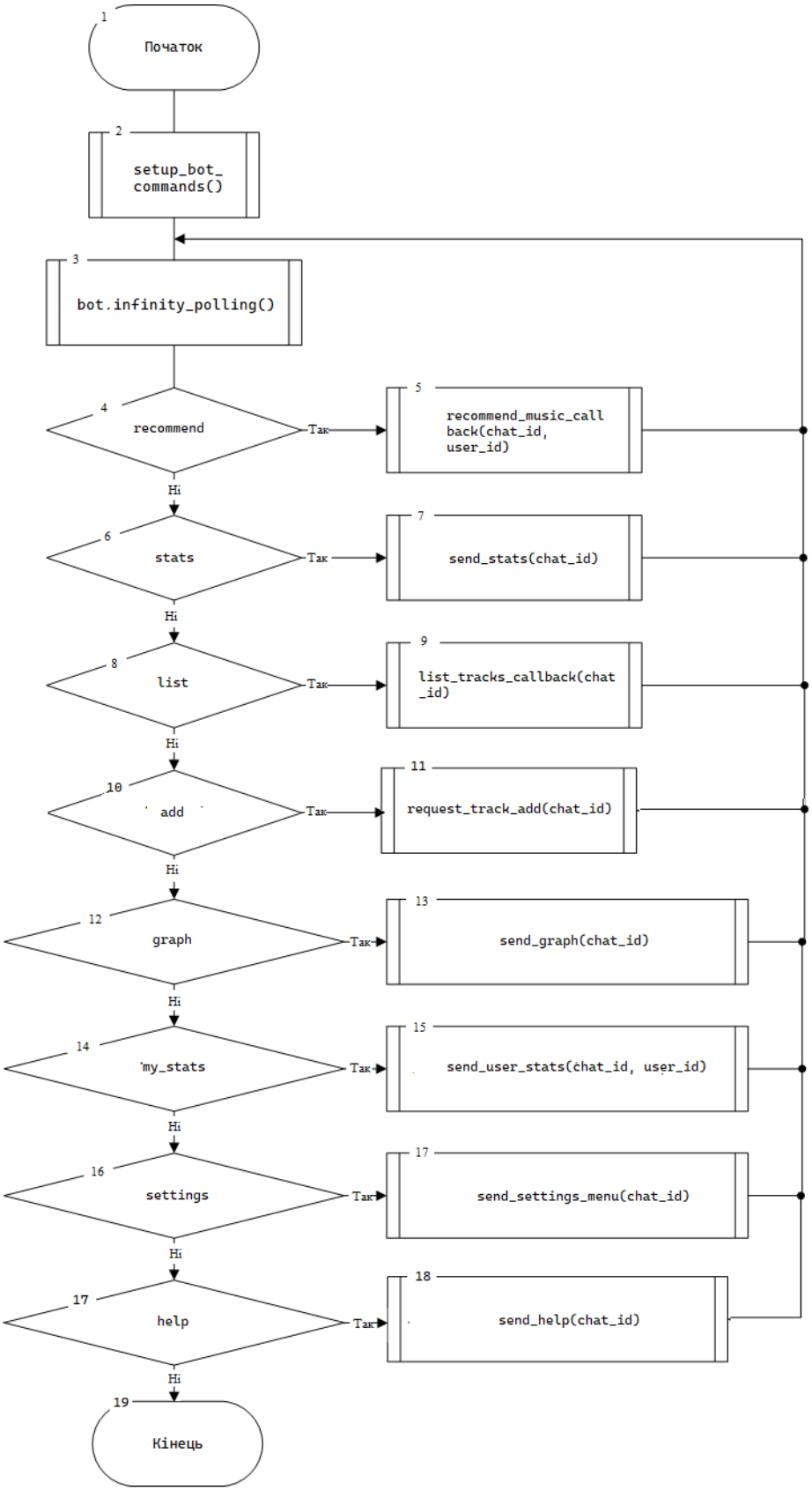

Рисунок 2.3 – Блок-схема роботи застосунку

Далі, у відповідь на введену користувачем команду, бот виконує відповідну перевірку. Якщо команда відповідає ключовому слову recommend, викликається функція recommend_music_callback, яка реалізує логіку рекомендації музичних треків. У разі команди stats користувач отримує загальну статистику через функцію send_stats. Команда list ініціює виведення списку доступних треків. Якщо користувач надсилає команду add, бот запитує додавання нового треку.

Команда graph дозволяє побачити графічне відображення даних, тоді як my_stats відображає персоналізовану статистику користувача. Команда settings відкриває меню налаштувань, а help викликає функцію довідки для отримання інформації про доступні можливості.

Після обробки команди бот повертається у початковий стан очікування наступного запиту, забезпечуючи безперервний цикл взаємодії з користувачем до завершення роботи застосунку.

На рисунку 2.4 зображена блок-схема демонструє логіку роботи функції рекомендації музичних треків на основі користувацьких даних і жанрових вподобань у Telegram-боті.

Процес починається з отримання ідентифікатора користувача user_id. Після цього розраховується загальна кількість збережених взаємодій користувача через функцію sum. Якщо значення цієї суми дорівнює нулю, тобто користувач не має жодної активності або даних, алгоритм одразу завершується й повертає порожній список треків, оскільки рекомендації сформувати неможливо.

У разі наявності активності, система завантажує дані користувача та його жанрові вподобання. Далі перевіряється, чи задано жанр. Якщо так, відбувається фільтрація доступних треків за обраним жанром. Після цього перевіряється, чи залишилися треки після фільтрації. Якщо жодного не знайдено, повернення знову завершується порожнім списком.

Алгоритм завершується після того, як система надає відповідні рекомендації, що є логічним завершенням процесу.

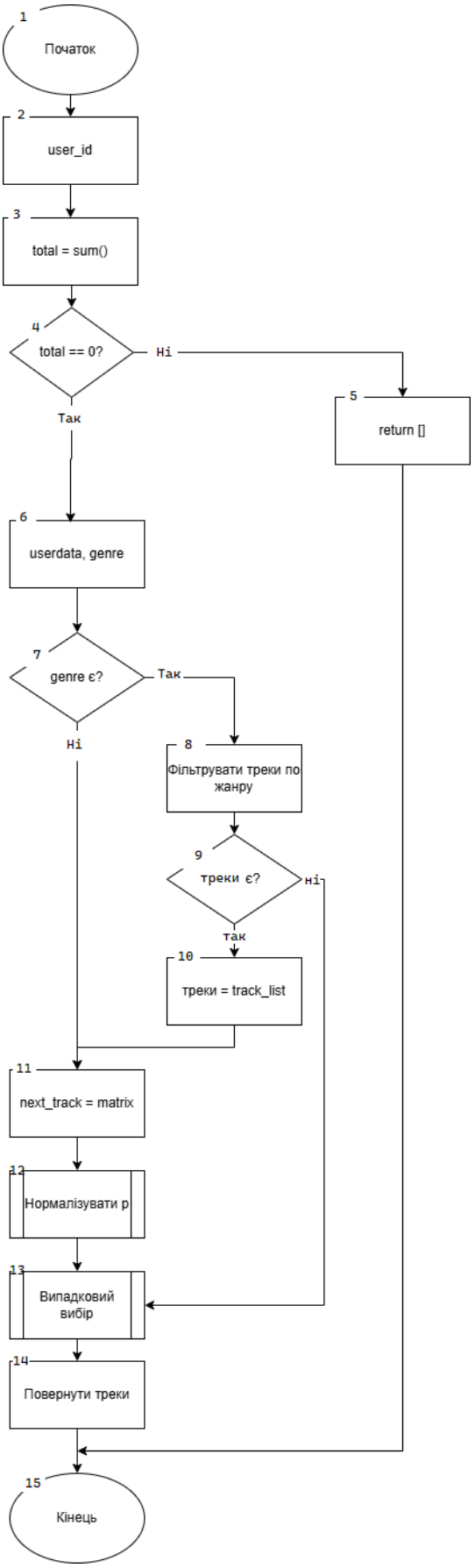

Рисунок 2.4 – Блок схема робота алгоритму Markov_recommendation

Якщо жанр не був заданий або фільтрація успішна, система формує список треків для подальшої обробки. На основі матриці ймовірностей вибору matrix визначається наступний трек. Ймовірності нормалізуються для забезпечення коректного розподілу, після чого відбувається випадковий вибір треку з урахуванням цих ймовірностей. В результаті, користувачу повертається список рекомендованих треків.

Отже, розроблені блок-схеми відображають структуровану та логічно послідовну архітектуру функціонування Telegram-бота для музичних рекомендацій. Перша схема демонструє загальний алгоритм взаємодії користувача з ботом через набір команд, кожна з яких викликає відповідну функцію: рекомендація музики, перегляд статистики, графічне представлення, доступ до налаштувань тощо. Це є підтвердженням модульної побудови застосунку, що полегшує масштабування та підтримку коду.

2.3 Розробка методу рекомендації музики на основі SMMC-процедур

Модуль музичних рекомендацій у Telegram-боті базується на стохастичному підході, який використовує ланцюги Маркова [14] першого порядку для моделювання переходів між музичними треками. Ланцюги Маркова є ефективним інструментом для прогнозування наступного стану системи, у цьому випадку наступного треку, на основі поточного стану, що ідеально відповідає задачі створення персоналізованих рекомендацій.

$$\{X_t\}, \quad t \in T = \{0,1,2,\dots\}, \tag{2.1}$$

де X(t) – стан системи (тобто трек) у момент часу t, T – множина дискретних моментів часу.

Даний стохастичний процес задовольняє марковську властивість.

Такий підхід спрощує задачу прогнозування наступного треку, оскільки система не потребує аналізу всієї історії прослуховувань, а лише враховує

поточний трек. Це знижує обчислювальну складність і робить алгоритм придатним для реального часу.

Марковська властивість забезпечує простоту моделі, дозволяючи системі фокусуватися на локальних взаємозв'язках між треками. Наприклад, якщо користувач прослухав трек s(i), бот використовує ймовірності p(ij) для вибору наступного треку s(j), що робить рекомендації адаптивними до останніх уподобань користувача.

$$P(X_{t+1} = j \mid X_t = i, X_{t-1} = i_{t-1}, \dots, X_0 = i_0) = P(X_{t+1} = j \mid X_t = i) = p_{ij} \quad (2.2)$$

де X(t) – стан системи (тобто трек) у момент часу t, i,j – індекси треків (відповідно поточного та наступного), p(ij) – ймовірність переходу від треку s(i) до s(j), T – множина дискретних моментів часу.

Тобто ймовірність переходу в наступний стан залежить лише від поточного стану, а не від попередньої історії.

Нехай зображена скінченна множина всіх треків у системі, де n кількість треків. Ця множина є основою для побудови рекомендаційної системи, оскільки алгоритм оперує скінченною кількістю станів. Обмеження кількості треків n дозволяє ефективно організувати обчислення ймовірностей переходів, що є ключовим для масштабованості системи.

$$S = \{s_1, s_2, \dots \ , s_n\}, \quad (2.3)$$

де S – множина треків

Множина S представляє повний набір треків, доступних у системі, де n загальна кількість треків. Кожен елемент s(i) відповідає унікальному треку, який ідентифікується за назвою та, за наявності, URL-посиланням.

Будується матриця переходів P розмірністю n×n, де елемент p(ij) представляє ймовірність переходу від треку s(i) до треку s(j).

$$P = \left[p_{ij}\right]_{n \text{ x } n}, \tag{2.4}$$

де n – кількість треків;

p(ij) – ймовірність переходу з треку s(i) на s(j).

При ініціалізації системи всі переходи мають однакову ймовірність.

Матриця P є основною структурою даних для зберігання і використання знань про вподобання користувача. Вона дозволяє боту динамічно обирати наступний трек шляхом випадкового вибору, зваженого ймовірностями p(ij), що відображають частоту переходів між треками на основі історії взаємодій.

$$p_{ij} = \frac{1}{n}, \quad \forall i, j \in \{1, \dots \ , n\}, \tag{2.5}$$

де ймовірність p(ij) обчислюється як результат ділення 1 на n кількість треків

Ця ініціалізація забезпечує нейтральний стартовий стан, коли бот ще не має інформації про вподобання користувача. У міру накопичення даних, тобто переходів між треками, ймовірності оновлюються, що дозволяє системі поступово адаптуватися до індивідуальних смаків.

Припустимо, що c(ij) лічильник переходів від треку s(i) до треку s(j). При новому переході.

$$c_{ij} \leftarrow \ c_{ij} + 1, \tag{2.6}$$

де c(ij) – кількість переходів між треками

Лічильник c(ij) відображає кількість зафіксованих переходів від треку s(i) до треку s(j). При кожній новій взаємодії, коли користувач переходить від одного треку до іншого, відповідний лічильник інкрементується.

Після оновлення лічильника, нові ймовірності обчислюються.

$$p_{ij} = \frac{c_{ij}}{\sum_{k=1}^{n} c_{ik}}, \tag{2.7}$$

де c(ij) – кількість переходів від треку s(i) до треку s(j), $\sum_{k=1}^{n} c_{ik}$ – загальна кількість усіх переходів із треку s(i), p(ij) – оновлена ймовірність переходу.

Це забезпечує дотримання аксіоми ймовірності. Ця формула забезпечує нормалізацію ймовірностей, гарантуючи, що сума ймовірностей для всіх можливих наступних треків із поточного стану дорівнює 1, аксіома ймовірності. Це дозволяє системі коректно генерувати рекомендації, використовуючи зважену випадкову вибірку на основі накопичених даних.

$$\sum_{j=1}^{n} p_{ij} = 1\,, \tag{2.8}$$

де сума ймовірностей p(ij) дорівнює 1.

Це гарантія того, що ймовірності нормалізовані, забезпечує стабільність алгоритму. Без цього система могла б генерувати некоректні рекомендації, якщо сума ймовірностей відхилялася б від 1.

Задля врахування жанру впроваджується функція належності.

$$g(s_j) = \begin{cases} 1, & \text{якщо трек } s_j \text{ належить до жанру } g \\ 0, & \text{інакше} \end{cases}, \tag{2.9}$$

де g(s(j)) перевіряє належність до жанру.

Функція g(s(j)) дозволяє враховувати жанрові вподобання користувача, роблячи рекомендації більш орієнтованими на юзера. Наприклад, якщо користувач віддає перевагу року, треки цього жанру отримують більшу вагу в процесі вибору.

Тоді модифікована ймовірність переходу обчислюється як добуток ймовірності і жанрового коєфіцієнту.

$$p'_{ij} = p_{ij} \cdot \left(1 + \beta \cdot g(s_j)\right), \tag{2.10}$$

де β – коефіцієнт підсилення жанрових вподобань (наприклад, β=0,5), p(ij) – ймовірність, модифікована відповідно до жанру треку.

Ця модифікація дозволяє системі адаптувати рекомендації до індивідуальних уподобань користувача, підвищуючи їх релевантність. Наприклад, якщо користувач обирає жанр «рок», треки цього жанру отримують вищий пріоритет, але система все ще може рекомендувати треки інших жанрів, якщо їхня базова ймовірність p(ij) достатньо висока.

Щоб зберегти стохастичність матриці переходів, необхідна нормалізація.

$$p''_{ij} = \frac{p'_{ij}}{\sum_{k=1}^{n} p'_{ik}}, \tag{2.11}$$

де p(ij)′ – фінальна нормалізована ймовірність переходу, $\sum_{k=1}^{n} p'_{ik}$ – сума модифікованих ймовірностей для поточного треку s(i).

Нормалізація гарантує, що модифіковані ймовірності залишаються коректними з точки зору ймовірнісної моделі. Це дозволяє системі генерувати рекомендації, які враховують жанрові вподобання, але при цьому залишаються стохастично правильними.

2.4 Розробка модуля навчання бота

Процес навчання бота у запропонованій реалізації базується на простому, але ефективному SMMC алгоритму, який оновлюється динамічно в ході взаємодії користувача з системою. На рисунку 2.5 зображено алгоритми як саме відбувається навчання на основі двох ключових функцій initialize_matrix та update_matrix.

На початковому етапі, коли запускається бот або коли відбувається ініціалізація системи, викликається функція initialize_matrix. Вона створює так

звану матрицю переходів, структуру, яка для кожного треку з наявного списку зберігає ймовірності переходу до будь-якого іншого треку. У початковому стані ці ймовірності рівномірні, тобто бот не має упередженості до жодного конкретного треку усі мають однакову вагу. Це є базовий стан системи до того, як вона отримає достатню кількість даних від користувачів.

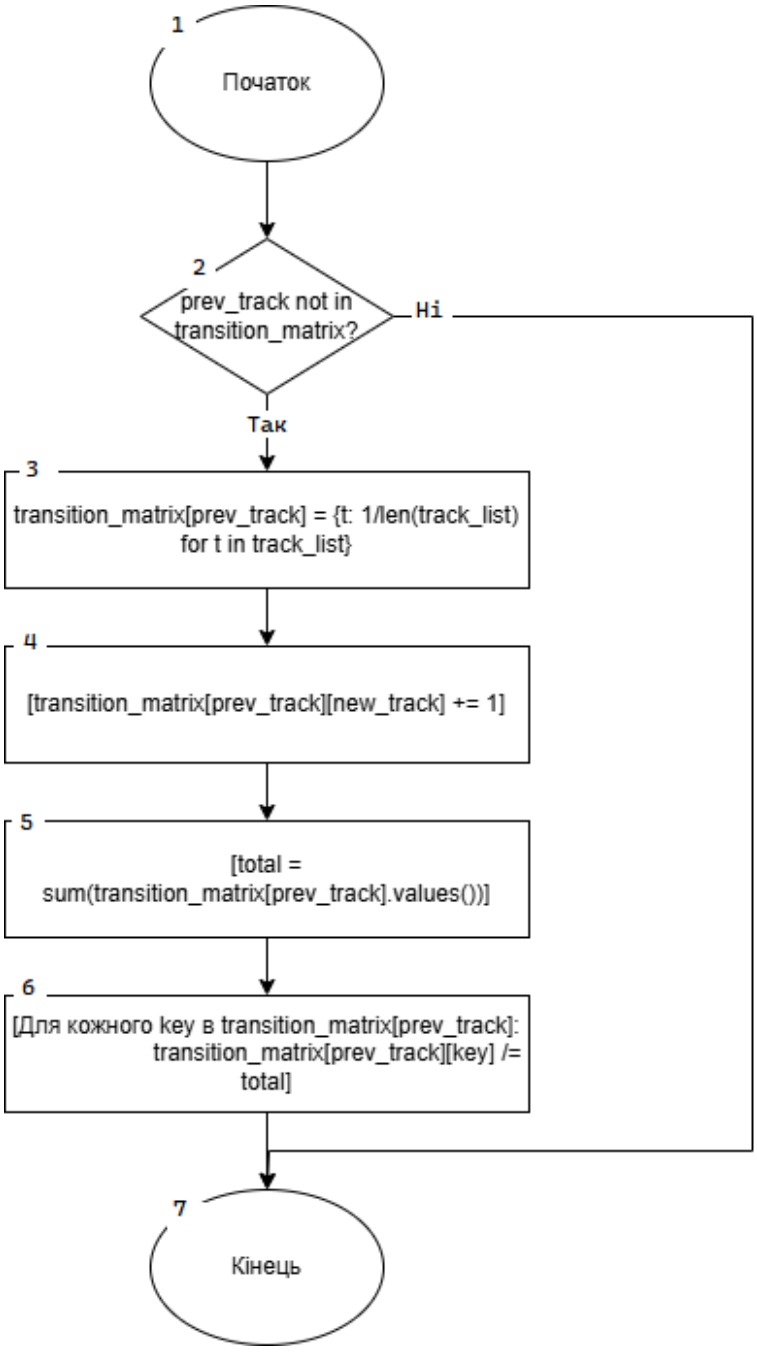

Рисунок 2.5 – Блок схема роботи методу update_matrix

У момент, коли користувач прослуховує новий трек після попереднього, викликається функція update_matrix, яка і реалізує механізм навчання. Алгоритм фіксує факт переходу від одного треку до іншого та інкрементує відповідне значення у матриці. Це означає, що система починає запам’ятовувати, які переходи відбуваються частіше.

Після оновлення ваг проводиться нормалізація: усі значення в рядку для попереднього треку приводяться до ймовірностей, тобто сума всіх переходів з цього треку дорівнює 1. Такий підхід гарантує, що модель зберігає коректну ймовірнісну інтерпретацію даних.

Отже, бот вчиться на основі реальної поведінки користувачів, поступово зменшуючи ймовірності для менш актуальних переходів і підвищуючи вагу тих, що трапляються частіше. У підсумку це дозволяє системі адаптуватися до змін у смаках користувачів, а також формувати більш точні й персоналізовані рекомендації з урахуванням накопиченої історії переходів між треками.

2.5 Розробка інтерфейсу

У сучасній розробці програмного забезпечення, зокрема в галузі чат-ботів, інтерфейс користувача UI [15] виступає не лише візуальною оболонкою, а й ключовим елементом, що формує загальний досвід взаємодії. У випадку музичного Telegram-бота, який реалізує функцію рекомендацій треків, збирання статистики, побудову графів переходів тощо, інтерфейс відіграє роль комунікаційного мосту між системою і користувачем.

На відміну від традиційних мобільних додатків із повноцінними графічними інтерфейсами, Telegram-боти працюють в умовах текстово-контекстного середовища. Користувацька взаємодія відбувається через текстові команди, кнопки меню та швидкі відповіді. Такий інтерфейс базується на принципах простоти, зрозумілості та швидкого доступу до ключового функціоналу.

На рисунку 2.6 бот пропонує чітко структуроване меню з командами: запуск бота, виклик головного меню, отримання рекомендації, перегляд списку

треків, додавання нового треку, загальна та індивідуальна статистика, налаштування, побудова графа переходів, а також допомога. Таким чином, організація інтерфейсу відображає логіку роботи самого алгоритму, користувач вносить вхідні дані, наприклад, додає трек, а система на основі Марковських моделей генерує рекомендації, фіксує переходи й формує статистику. Концептуально вибір текстового меню замість традиційного візуального GUI цілком виправданий. По-перше, Telegram забезпечує зручний механізм реалізації команд через інлайн-кнопки та меню, що дозволяє зменшити когнітивне навантаження на користувача. По-друге, у текстовому середовищі основне навантаження лягає на семантичну точність команд. Користувач має розуміти, що саме виконає бот після натискання на кнопку або введення команди. Тому особлива увага приділяється формулюванню назв команд: вони мають бути короткими, описовими, однозначними та відповідати очікуванням цільової аудиторії.

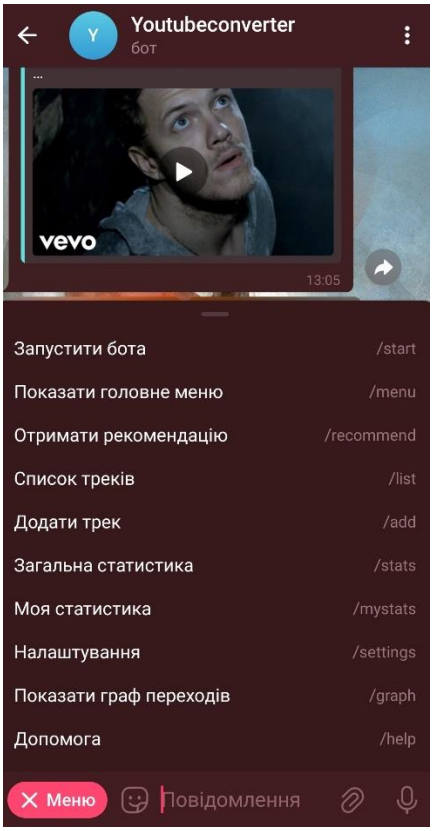

Рисунок 2.6 – Інтерфейс інтерактивного бота Youtubeconverter

Інтерфейс телеграм-бота Youtubeconverter демонструє високий рівень зручності, доступності та інформативності для кінцевого користувача. У межах функціонального меню користувач має змогу швидко викликати команди, які відповідають за запуск бота, генерацію рекомендацій, перегляд списку треків, додавання нових композицій, перегляд статистики, графу переходів та отримання довідки. Такий інтерфейс є не лише інтуїтивно зрозумілим, але й відповідає принципам ефективного UX-дизайну, де кожна команда має чітке призначення, що мінімізує когнітивне навантаження на користувача.

Особливу увагу заслуговує модуль статистики, який забезпечує зворотний зв’язок щодо музичних вподобань користувача. На рисунку 2.7 видно, що при виклику команди /mystats бот повертає текстову зведену інформацію: загальну кількість згенерованих рекомендацій, 35.

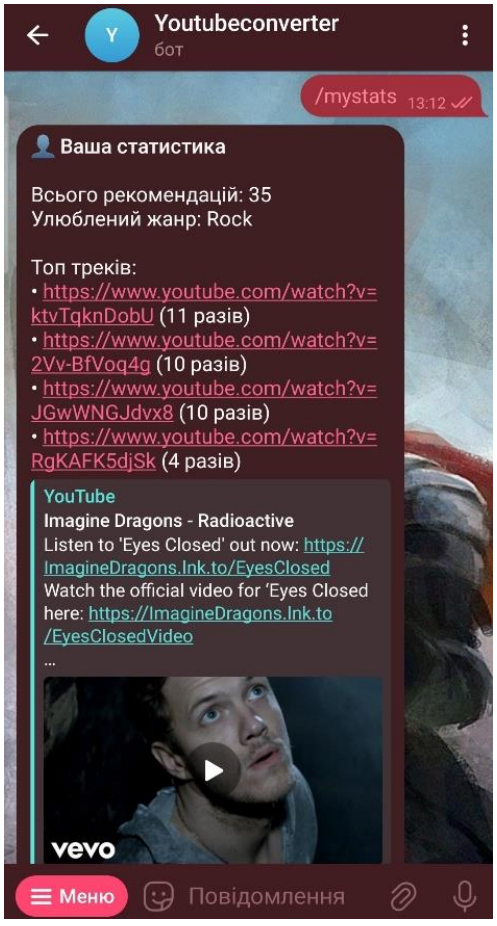

Рисунок 2.7 – Відображення статистики рекомендацій

Також визначає улюблений музичний жанр користувача Rock. Додатково бот надає список найбільш прослуховуваних треків, що базується на кількості звернень до певних YouTube-посилань. Найпопулярнішим є трек із посиланням https://www.youtube.com/watch?v=ktvTqknDobU, який прослуховували 11 разів. Такий підхід дозволяє користувачу не лише відстежувати свою активність, а й мимоволі формувати свій музичний профіль, що може бути надалі використано для персоналізованих рекомендацій.

Отже, інтерфейс Telegram-бота вирізняється високим рівнем інформативності та взаємодійності, що значно підвищує користувацький досвід. Поєднання текстових повідомлень зі зведеною статистикою, графічних візуалізацій у вигляді діаграм та графа переходів забезпечує багаторівневе представлення даних. Такий підхід дозволяє не лише швидко отримувати відповіді на команди, а й бачити динаміку, закономірності та контекст власної музичної активності. Ключовим чинником ефективності є інтуїтивність інтерфейсу: користувач без зайвих зусиль розуміє, як взаємодіяти з ботом і що означає кожен елемент. Команди типу /mystats і /stats відкривають доступ до особистих аналітичних звітів, у той час як графічні блоки виконують функцію зворотного зв’язку, візуалізуючи результати. Водночас, збереження мінімалістичного стилю відповідає формату месенджера, не перевантажуючи екран зайвими деталями. Таким чином, інтерфейс бота поєднує простоту й функціональність, перетворюючи процес отримання рекомендацій на залучаючий і прозорий досвід для кожного користувача.

2.6 Висновки

У цьому розділі було сформовано цілісне уявлення про архітектуру та ключові компоненти програмного продукту інтерактивного Telegram-бота для музичних рекомендацій. Опрацьовані рішення охоплюють як зовнішній вигляд і зручність взаємодії, так і внутрішню логіку та алгоритми функціонування системи. Інтерфейс реалізований у форматі командного меню, яке забезпечує просту й інтуїтивну навігацію для користувача. Гістограма дозволяє оцінити

активність і вподобання, тоді як граф переходів відображає логіку рекомендаційної моделі, заснованої на алгоритмі SMMC. Блок-схеми алгоритмів стали основою структурного підходу до реалізації логіки програми, а модулі рекомендацій і навчання забезпечують адаптивність і персоналізований підхід до кожного користувача. Бот не лише реагує на команди, а й поступово навчається, удосконалюючи свої рекомендації відповідно до динаміки вподобань.

3 РОЗРОБКА ПРОГРАМНОГО ПРОДУКТУ

3.1 Обґрунтування вибору програмних засобів для розробки

У таблиці 3.1 зроблено вибір мови програмування, який є фундаментальним рішенням, що визначає актуальність розробки, супроводу та масштабування програмних систем. У контексті створення інтерактивного бота для системи музичних рекомендацій з використанням алгоритму SMMC, Python постає як оптимальний інструмент завдяки низці суттєвих переваг.

Python [16] інтерпретована мова високого рівня з динамічною типізацією, що характеризується надзвичайною гнучкістю та виразністю коду. Філософія Python, яка робить наголос на читабельності та зрозумілості, дозволяє розробникам створювати елегантні рішення при мінімальних затратах часу. Важливо підкреслити, що Python надає суттєві переваги саме в контексті розробки ботів та систем рекомендацій завдяки доступу до спеціалізованих бібліотек.

Екосистема Python пропонує зручний інструментарій для обробки даних та реалізації адаптивних алгоритмів. Бібліотеки для роботи з даними, такі як collections, json та простий вбудований функціонал для роботи з текстовими файлами, дозволяють ефективно зберігати історію рекомендацій та аналізувати патерни вподобань користувачів. Такий підхід ідеально підходить для створення адаптивного бота музичних рекомендацій, який враховує попередні вибори користувача без застосування складних методів машинного навчання.

Для інтеграції з месенджерами наявні бібліотеки на кшталт pyTelegramBotAPI, aiogram та python-telegram-bot надають високорівневі абстракції для взаємодії з API Telegram, що суттєво спрощує розробку інтерактивних ботів без необхідності занурюватись у деталі HTTP-комунікації.

Синтаксис Python дозволяє досягти високої швидкості розробки при збереженні якості, що особливо важливо при створенні прототипів та ітеративній розробці. Скорочення циклу написання коду, тестування, рефакторинг призводить до швидшого виходу на ринок. Починаючи з версії 3.5,

Python надає вбудовану підтримку асинхронного програмування через ключові слова async/await, що дозволяє ефективно обробляти велику кількість одночасних запитів користувачів бота.

Таблиця 3.1 - Обґрунтування вибору мови програмування Python

Параметр	Python	JavaScript	Java	C#
Швидка розробка прототипу	1	0	0	0
Високодоступні бібліотеки для обробки даних	1	0	0	0
Проста інтеграція з API месенджерів	1	1	0	0
Висока читабельність коду	1	0	0	1
Висока продуктивність виконання	0	1	1	1
Розширена підтримка спільноти	1	1	1	1
Вбудоване асинхронне програмування	1	1	0	1
Сумарний коєфіцієнт	6	4	2	4

Отже, у контексті реалізації алгоритму SMMC для музичних рекомендацій, Python надає додаткові переваги завдяки простоті реалізації логіки адаптивного підбору треків. Механізм рекомендацій, що базується на

аналізі попередніх виборів користувача та зберіганні цієї інформації у текстових файлах, може бути реалізований із використанням вбудованих функцій Python для роботи з рядками та файлами. Це дозволяє створити ефективну систему рекомендацій без надмірного ускладнення кодової бази та залучення складних фреймворків машинного навчання.

3.2 Вибір середовища розробки

У таблиці 3.2 зроблено вибір інтегрованого середовища розробки IDE, який має суттєвий вплив на продуктивність програмування, якість кінцевого продукту та можливість командної роботи. Visual Studio як комплексна платформа розробки пропонує потужний інструментарій, що робить її оптимальним вибором для реалізації проєкту інтерактивного бота системи музичних рекомендацій.

Visual Studio [17] є не лише редактором коду, але й цілісною екосистемою розробки, що включає інструменти для дизайну, тестування, налагодження та розгортання програмного забезпечення. У контексті розробки Python-застосунків, Visual Studio з розширенням Python Tools for Visual Studio надає широкий спектр функціоналу, що значно оптимізує процес програмування.

IntelliSense для Python забезпечує контекстно-залежні підказки, автодоповнення та рефакторинг, що прискорює написання коду та знижує ймовірність помилок. Потужні інструменти налагодження дозволяють виконувати покроковий аналіз коду, встановлювати контрольні точки, відстежувати змінні та виявляти помилки в логіці програми на ранніх етапах розробки.

Visual Studio спрощує створення, активацію та керування віртуальними середовищами Python, що забезпечує ізоляцію залежностей та запобігає конфліктам між пакетами. Вбудована підтримка Git та інших систем контролю версій сприяє ефективній організації командної роботи, відстеженню змін та управлінню релізами. Інструменти статичного аналізу та профілювання

допомагають виявляти потенційні вади продуктивності та якістю коду ще до етапу виконання.

Таблиця 3.2 – Обґрунтування вибору середовища Visual Studio

Функціональність	Visual Studio	PyCharm	VS Code	Jupyter Notebook
Високоякісне автодоповнення Python	1	1	0	0
Комплексне інтегроване налагодження	1	1	0	0
Повна підтримка віртуальних середовищ	1	1	0	0
Вбудоване профілювання коду	1	0	0	0
Повна інтеграція з Git	1	1	0	0
Висока підтримка мультимовних проєктів	1	0	0	0
Вбудовані інструменти аналізу даних	0	0	0	1
Висока продуктивність при великих проєктах	1	0	0	0
Повна крос-платформна сумісність	0	1	1	1
Сумарний бал	7	5	1	2

Отже, при розробці проєкту інтерактивного бота для системи музичних рекомендацій, Visual Studio забезпечує додаткові переваги через можливість структурування проєкту з виділенням окремих модулів для обробки даних музичних треків, імплементації алгоритму SMMC, інтеграції з API Telegram, тестування та валідації рекомендацій. Така модульна архітектура, підтримувана інструментами Visual Studio, сприяє кращій організації коду, спрощує супровід та забезпечує можливість ефективного масштабування системи в майбутньому.

3.3 Інструкція користувача

Цей документ призначений для користувачів програмного застосунку YouTubeConverter. Інструкція містить опис основного функціоналу, а також покрокові вказівки щодо встановлення та використання застосунку для музичних рекомендацій та завантаження, конвертації відео з платформи YouTube у популярні медіаформати

1. Завантажте програмний застосунок з (див. рисунок 3.1) https://github.com/Knightuk12/Youtubconverter

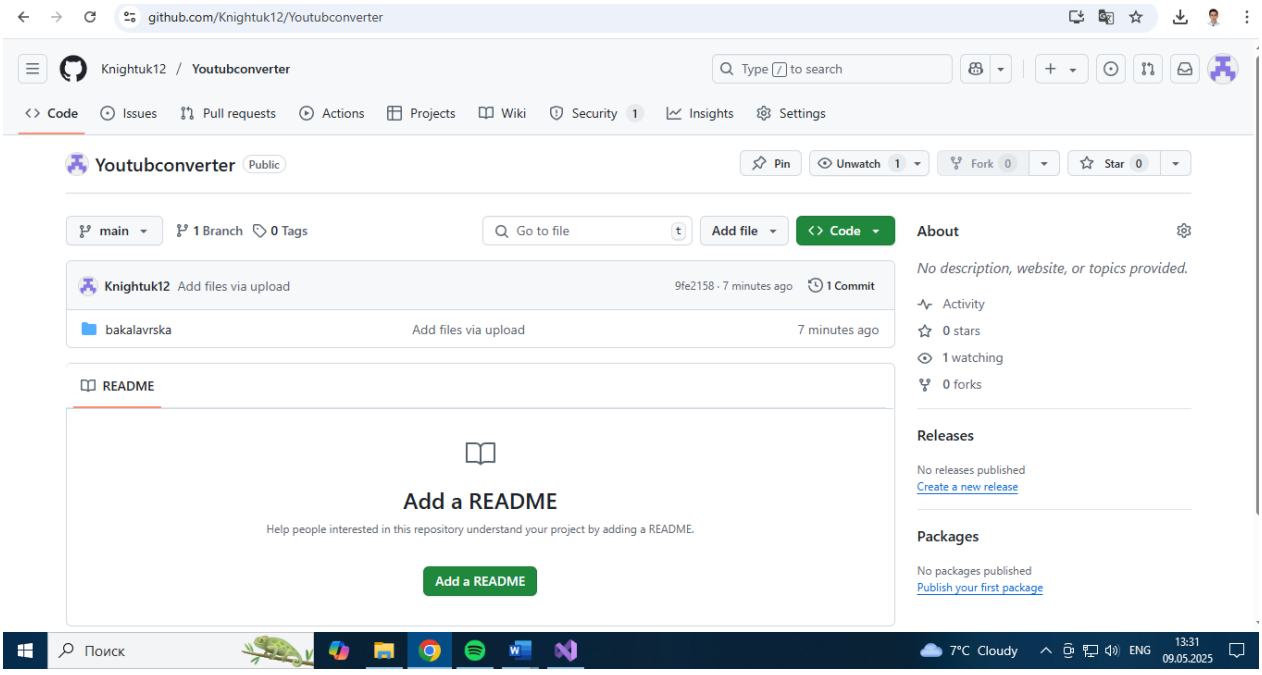

Рисунок 3.1 – Завантаження програмного застосунку з Google Диск

2. Розпакуйте папку з програмним застосунком (див. рисунок 3.2) у директорію C:\Users\user\source\repos, при потребі змінити ім’я користувача.

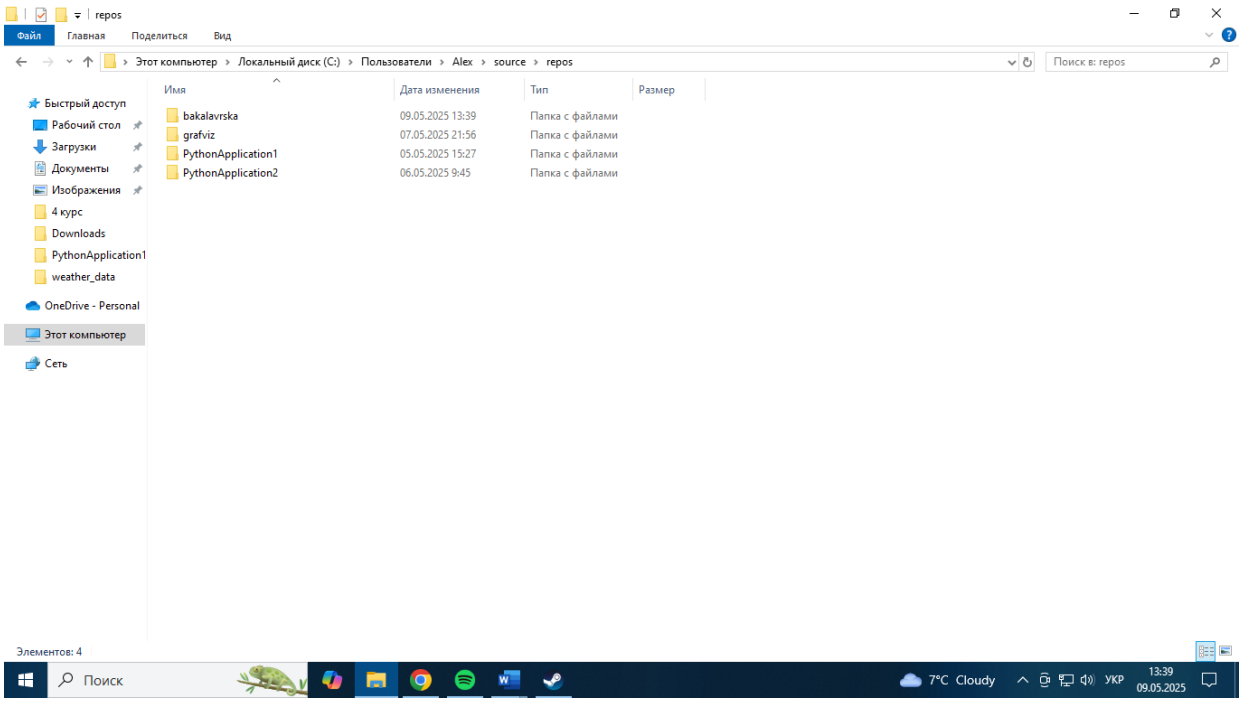

Рисунок 3.2 – Місце розташування програмного застосунку

3. Відкрийте папку з програмним застосунком (див. рисунок 3.3) за допомогою Visual Studio .

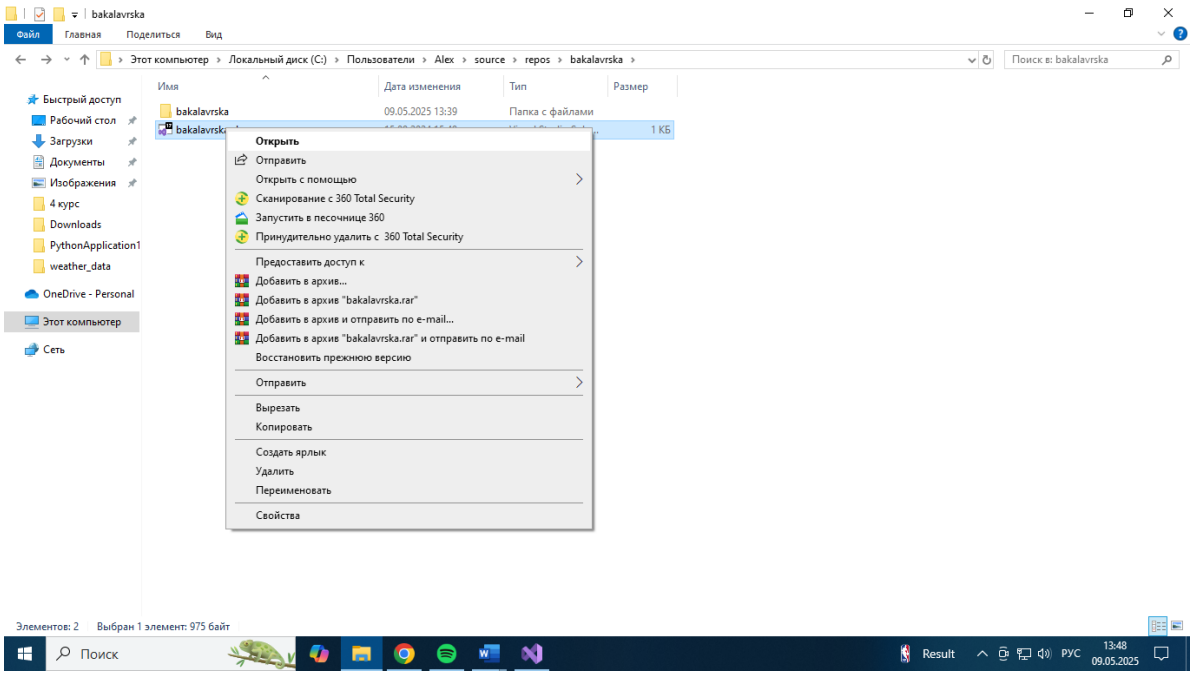

Рисунок 3.3 – Відкриття програми за допомогою Visual Studio

4. Після завершення завантаження програмного застосунку, запустіть програму за допомогою кнопки запуску (див. рисунок 3.4).

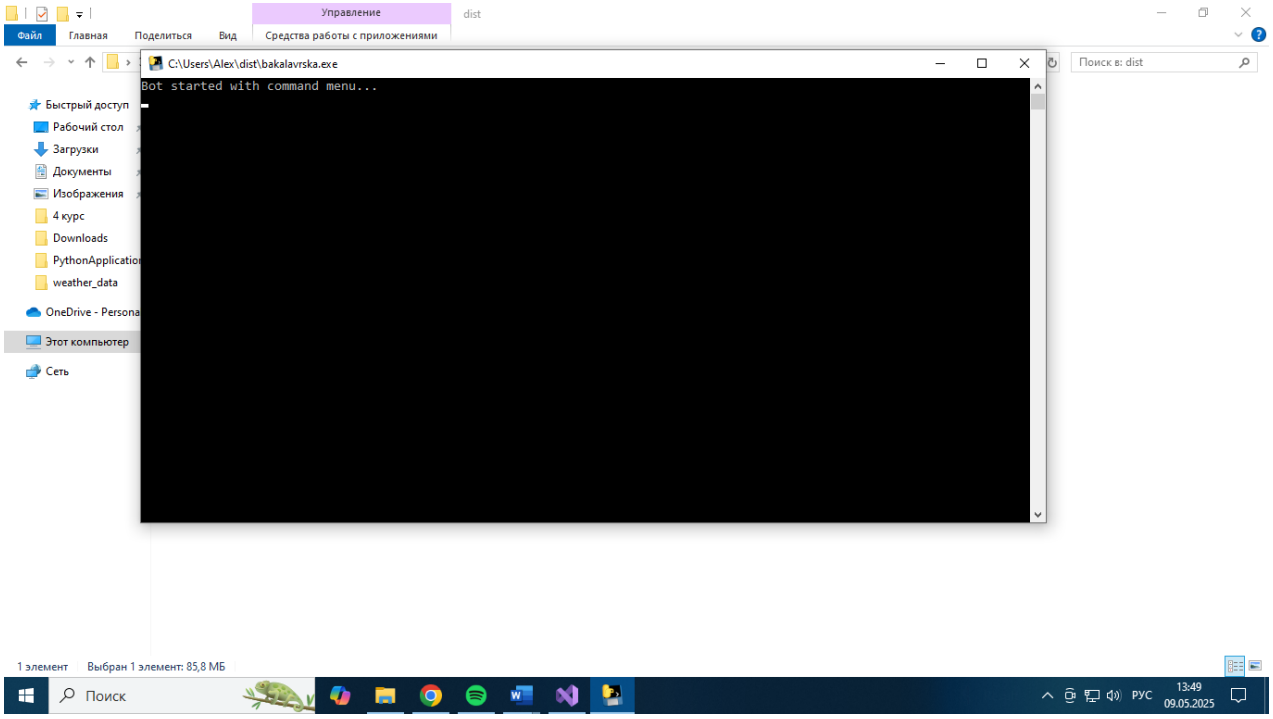

Рисунок 3.4 – Запуск програмного застосунку

Керівництво користувача включає опис технічних умов, необхідних для стабільної роботи програмного продукту. У ньому зазначено параметри пристроїв, на яких застосунок може бути встановлений та ефективно функціонувати. Детальну інформацію щодо мінімальних та рекомендованих характеристик мобільного пристрою для запуску застосунку наведено у таблицях 3.3 та 3.4.

Таблиця 3.3 – Мінімальна конфігурація:

Тип процесора	Qualcomm Snapdragon 450
Об’єм оперативної пам’яті	1,5 ГБ для 64-разрядної системи
Місце на жорсткому диску	500 МБ
Операційна система	Android 8.0 або новіше

Таблиця 3.4 – Рекомендована конфігурація:

Тип процесора	2.4GHz Octa-core Qualcomm Snapdragon 765G
Об’єм оперативної пам’яті	4 ГБ для 64-разрядної системи
Місце на жорсткому диску	2-4 ГБ
Операційна система	Android 12 або новіше

Після запуску інтерактивного бота YoutubeConverter, користувач проходить початкову авторизацію за допомогою команди /start, що забезпечує створення індивідуального профілю для подальшої взаємодії з ботом. Після успішного входу відкривається головне меню, де користувачу доступні основні функції: отримання музичних рекомендацій, додавання треків до персональної колекції, перегляд статистики активності та побудова музичної моделі на основі власних вподобань.

Перед виконанням будь-якої команди, яка потребує взаємодії з персональними даними, наприклад, додавання треку чи побудова графіка, система перевіряє наявність активного профілю. У разі його відсутності бот пропонує пройти авторизацію. Без цього кроку користувач не зможе скористатися повним функціоналом застосунку. Після успішного запуску профілю бот починає накопичувати дані про дії користувача: які треки були додані, як часто запитуються рекомендації, які жанри переважають у виборі.

Зібрана інформація використовується для створення статистики, доступної через команду /statistics, а також для побудови персоналізованої графічної моделі вподобань за допомогою команди /model. Уся історія взаємодії з ботом зберігається, що дозволяє користувачам відстежувати зміну своїх музичних інтересів та оцінювати динаміку розвитку персональної колекції.

Отже, створено інструкцію користувача та детально описано дії, які треба виконати, щоб встановити програмний застосунок на власний комп’ютер, а також його можливі налаштування. Також було вказано доступний функціонал у Youtubeconverter і його переваги.

3.4 Програмна реалізація

На рисунку 3.5 зображено блок-схема загального методу до реалізації системи логування та збору статистичних даних у межах механізму музичних рекомендацій.

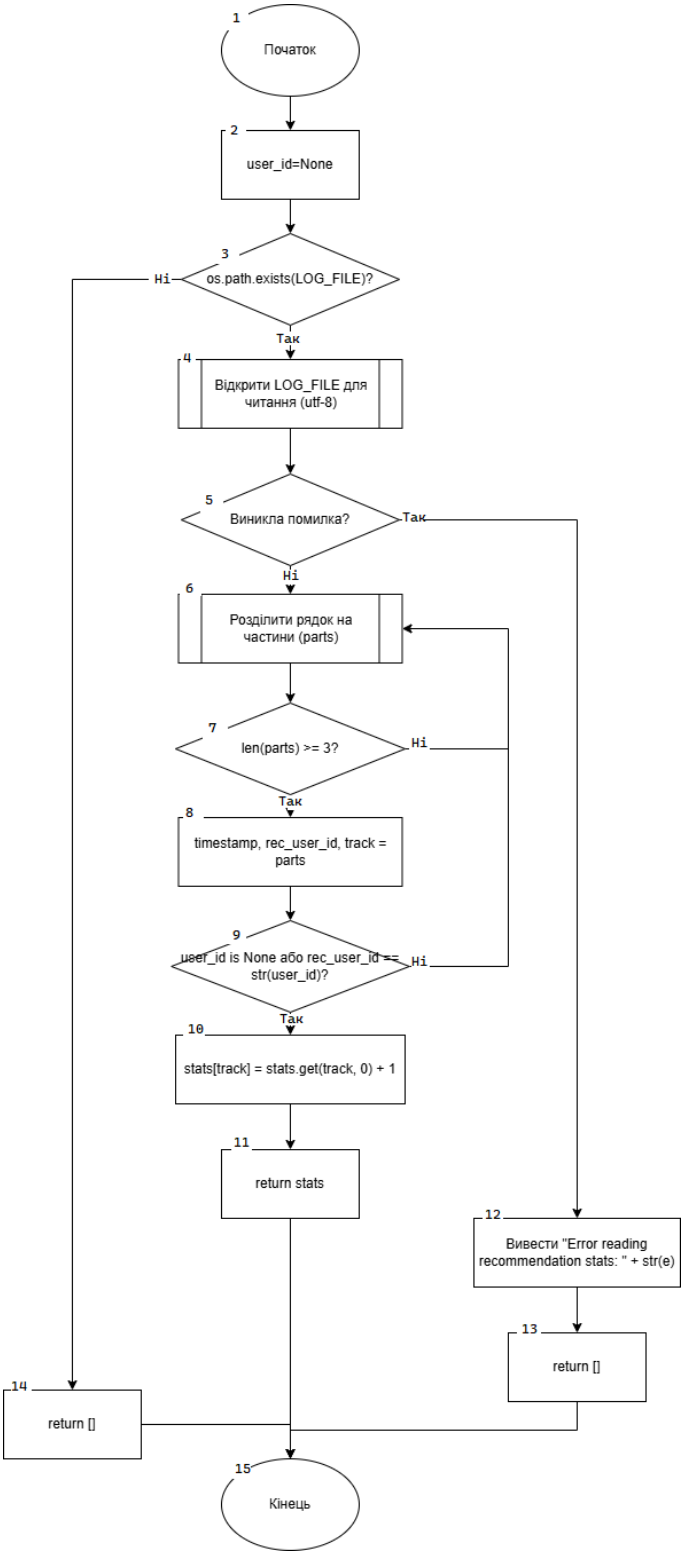

Рисунок 3.5 – Блок-схема методу get_recommendation_stats

Основна мета таких компонентів забезпечити збереження інформації про взаємодію користувача із системою, а також надати можливість подальшого аналізу частоти використання окремих елементів.

Логування [18] дозволяє фіксувати історію дій, пов’язаних із наданням рекомендацій, що створює основу для зворотного зв’язку, адаптації системи під уподобання користувача та удосконалення алгоритмів.

Метод log_recommendation відповідає за запис кожної музичної рекомендації до лог-файлу. У момент виклику функції створюється мітка часу у форматі YYYY-MM-DD HH:MM:SS, після чого у файл додається новий рядок, що містить час, ідентифікатор користувача та назву або URL треку. Таким чином формується історія взаємодії користувачів з ботом, що може бути використана для подальшого аналізу або навчання системи.

Метод get_recommendation_stats призначений для зчитування лог-файлу та побудови статистики рекомендацій. Якщо файл існує, метод обробляє кожен рядок, витягуючи з нього користувача та рекомендований трек. Якщо передано конкретний user_id, то враховуються лише записи цього користувача; якщо ні, враховується загальна статистика по всіх користувачах. Результатом є словник, де ключами є назви треків, а значеннями є кількість рекомендацій. У разі виникнення помилок, наприклад, відсутність доступу до файлу, функція обробляє виняток і повертає порожній словник. Загалом ці методи забезпечують базову, але ефективну систему збору даних про активність, що лежить в основі подальшої побудови аналітики та персоналізації рекомендацій.

На рисунку 3.6 зображено блок-схема методу markov_recommendation реалізує рекомендаційний алгоритм, заснований на Марковській моделі, з урахуванням індивідуальних музичних уподобань користувача.

Його основна мета запропонувати користувачеві наступний трек для прослуховування, використовуючи історію попередніх виборів і, за можливості, улюблений жанр. На початку функція перевіряє, чи існує загальний список треків track_list. Якщо він порожній, метод повертає None, оскільки рекомендація неможлива.

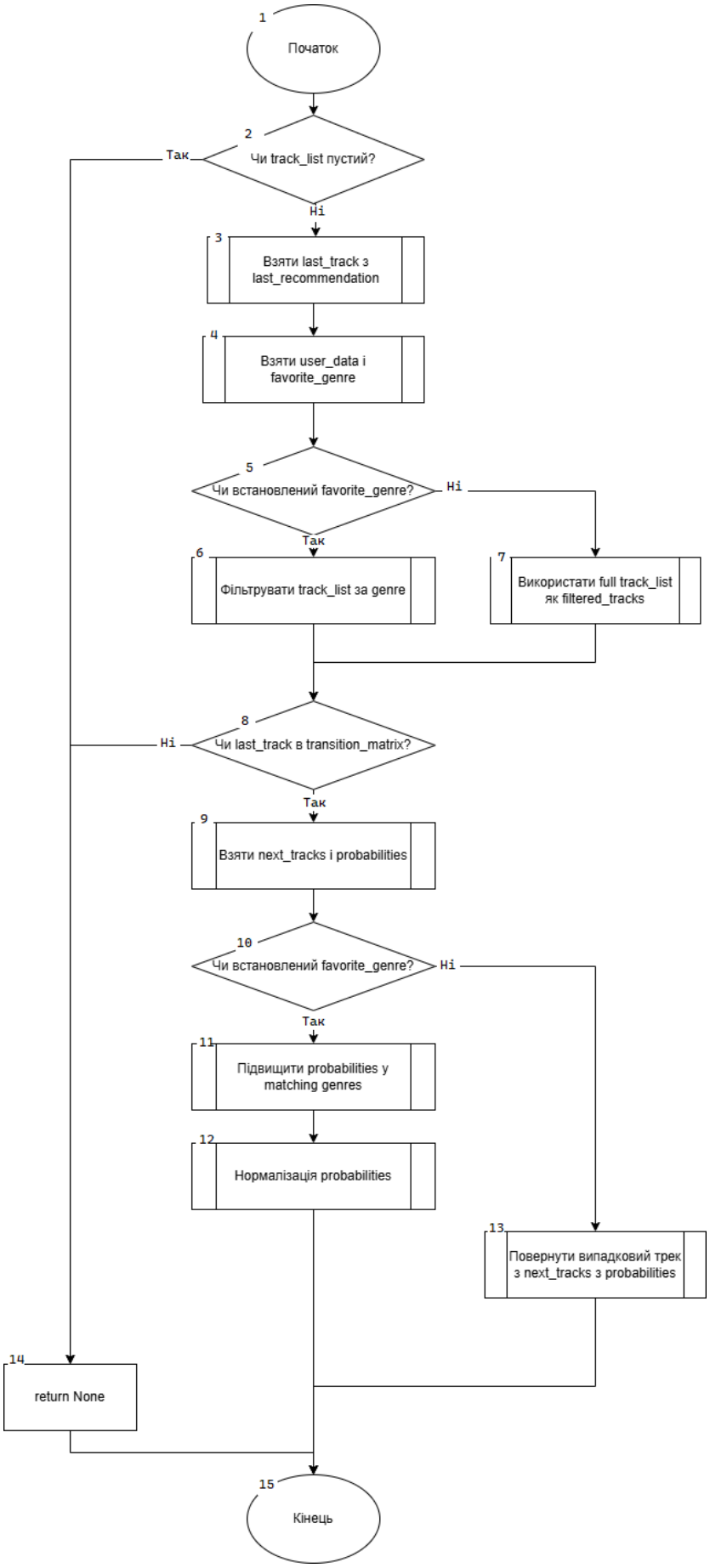

Рисунок 3.6 – Блок-схема методу Markov_recommendation

Далі зберігається останній рекомендований трек для поточного користувача з допомогою словника last_recommendation. Одночасно завантажуються збережені дані про користувача, зокрема його улюблений музичний жанр favorite_genre, якщо такий вказаний.

Якщо жанрові вподобання відомі, список треків фільтрується відповідно до ключового слова жанру, що дає змогу зробити рекомендацію більш персоналізованою. У випадку, якщо після фільтрації не залишилось жодного треку, система повертається до повного списку.

Якщо в transition_matrix є записи для останнього прослуханого треку, алгоритм розглядає можливі переходи, ймовірності переходу від одного треку до іншого згідно з SMMC алгоритмом на основі марковських ланцюгів. Якщо користувач має жанрові уподобання, метод підсилює ймовірності для треків, які відповідають жанру, після чого виконується нормалізація ймовірностей для збереження коректної суми. Рекомендація обирається за допомогою функції np.random.choice з урахуванням цих ймовірностей.

Якщо ж інформація про попередній трек відсутня або перехідна модель не містить відповідного запису, бот здійснює випадковий вибір треку зі списку, пріоритезуючи, за можливості, треки відповідного жанру.

На рисунку 3.7 зображена блок-схема функції send_graph, яка реалізує побудову орієнтованого зваженого графа на основі матриці переходів між музичними треками, що репрезентує поведінку користувача у процесі прослуховування. Основна мета створення наочної візуалізації, яка відображає ймовірнісні переходи між об'єктами, треками, у вигляді графа, вузлами якого є назви треків, а дугами емпірично зафіксовані переходи між ними.

На початку виконується перевірка наявності вхідних даних. У разі їх відсутності або недостатності функція припиняє виконання, повідомивши користувача про неможливість побудови графа. За наявності достатньої кількості даних ініціалізується порожній орієнтований граф з бібліотеки NetworkX [19].

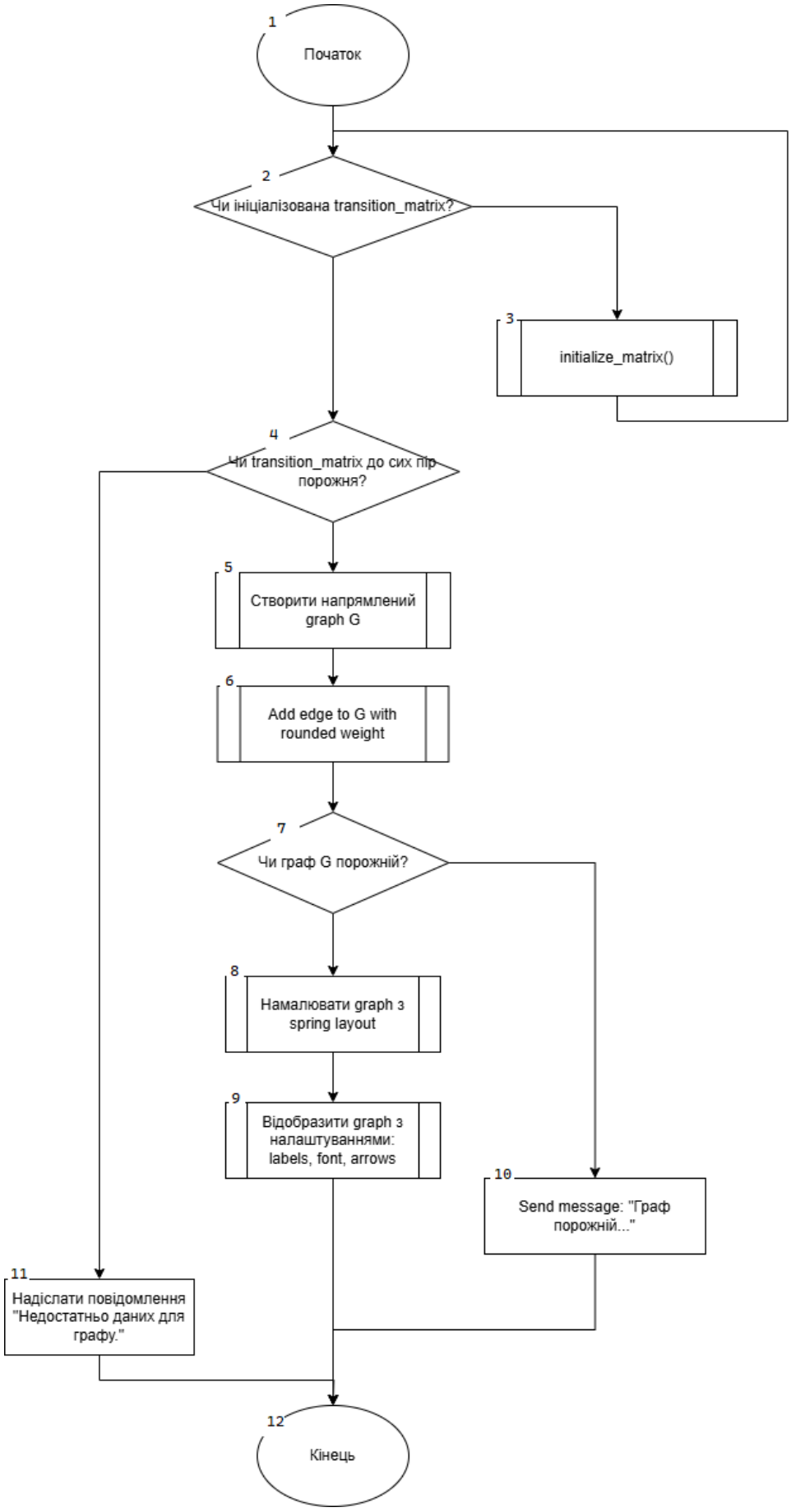

Рисунок 3.7 – Блок-схема методу send_graph

Далі здійснюється ітерація по матриці переходів: для кожної пари треків додається відповідне ребро з вагою, що характеризує частоту переходу. Імена треків при цьому скорочуються до 20 символів для покращення читабельності графічного зображення. Щоб уникнути надлишкової деталізації, встановлюється мінімальний поріг для додавання ребер, наприклад, лише якщо частота переходу перевищує 0.05.

Після побудови всієї структури графа проводиться додаткова перевірка на наявність вузлів. Якщо граф порожній, виводиться повідомлення про відсутність значущих даних. У разі великої кількості вузлів, понад 15, відбираються лише ті, що мають найвищий ступінь, кількість входів і виходів, таким чином, акцент візуалізації робиться на найбільш релевантних елементах.

Фінальний етап полягає у побудові графічного зображення за допомогою matplotlib. Застосовується силовий алгоритм розміщення вузлів для досягнення естетично привабливого та інформативного вигляду. Кожен вузол підписується, стрілки мають відповідне форматування, а товщина ліній співвідноситься з вагою переходу.

Таким чином, функція реалізує трансформацію статистичних даних про послідовність дій користувача у формалізовану візуальну модель, що дозволяє інтерпретувати структуру та інтенсивність переходів між об'єктами у системі.

На рисунку 3.8 зображена блок-схема функції send_stats, яка отримує словник зі статистикою прослуховувань або рекомендацій, де ключами є назви треків, а значеннями кількість звернень до кожного з них. Якщо словник порожній, користувач отримує повідомлення про відсутність даних.

У разі наявності статистики, вона сортується за спаданням популярності від найбільш до найменш рекомендованих треків. Далі бібліотека matplotlib використовується для побудови горизонтальної діаграми [20] на осі X відображається кількість рекомендацій, а на осі Y назви треків, обмежено топ-10.

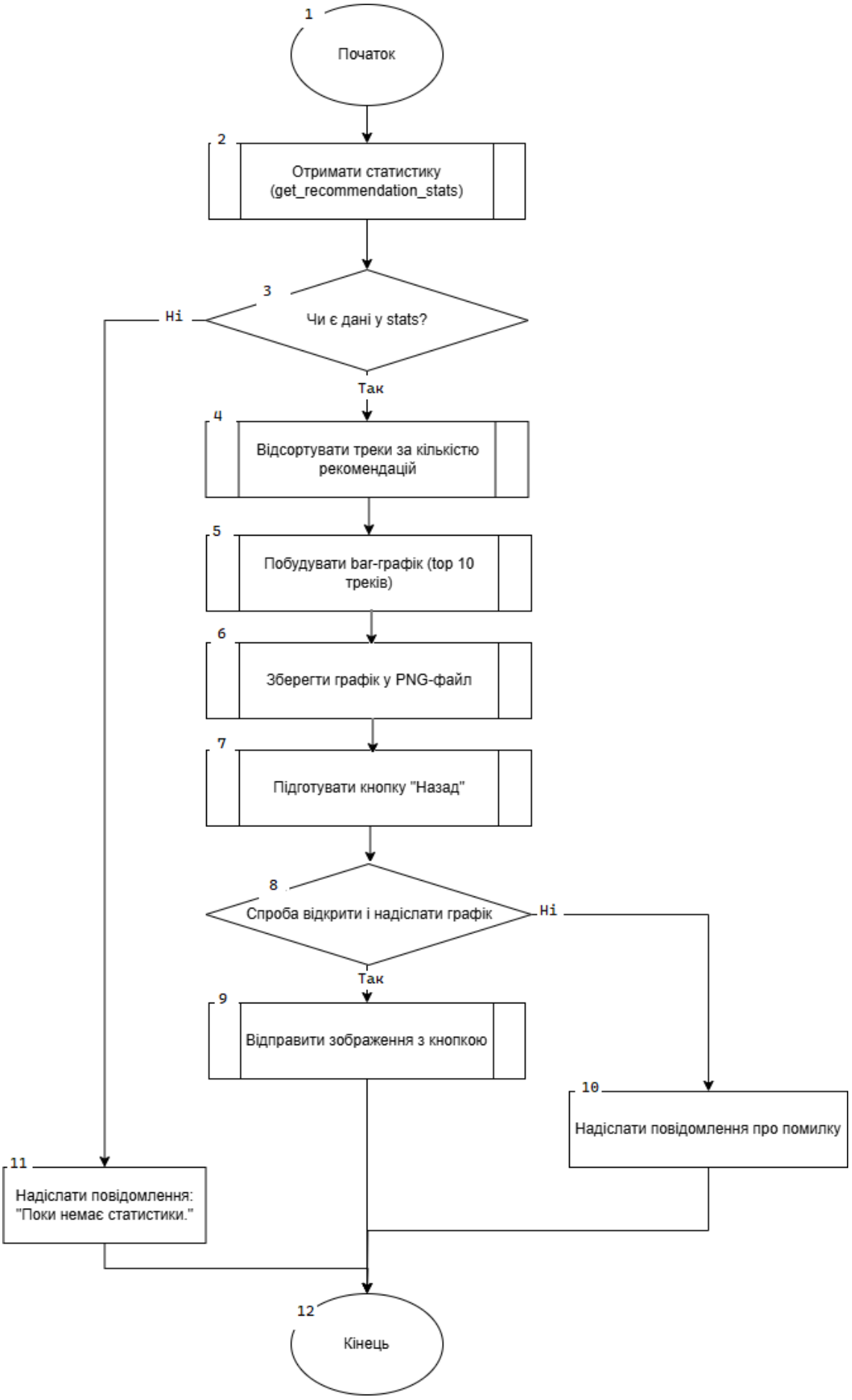

Рисунок 3.8 – Блок-схема методу send_stats

Щоб забезпечити зручність сприйняття, назви треків автоматично обрізаються до перших 20 символів. Побудована діаграма зберігається як PNG-файл із унікальною назвою, що включає поточну дату та час, формат «%Y%m%d%H%M%S».

Далі бот надсилає користувачу отримане зображення та додає кнопку «Назад», яка повертає до головного меню. Після цього тимчасовий файл з графіком видаляється. Якщо ж під час зчитування файлу або його надсилання трапиться помилка, бот повідомить користувача із зазначенням причинної інформації.

На рисунку 3.9 зображена блок-схема функції send_user_stats, яка є частиною програмного забезпечення Telegram-бота і відповідає за формування та надсилання індивідуального звіту користувачу про його музичну активність. Її мета персоналізовано інформувати користувача про те, як часто він користувався системою рекомендацій, які саме треки йому рекомендували найчастіше, а також який музичний жанр він позначив як улюблений.

На початку функція звертається до модуля збору статистики та намагається отримати збережені рекомендаційні дані для всіх користувачів. Після цього вона завантажує файл, що містить інформацію про зареєстрованих користувачів, і знаходить дані для поточного користувача за його ідентифікатором. Якщо виявляється, що для цього користувача ще не накопичено рекомендаційної історії, бот надсилає йому коротке повідомлення про відсутність статистики. На цьому виконання функції припиняється.

Якщо ж рекомендації для користувача наявні, то далі система виконує сортування отриманих даних. Це сортування здійснюється за спаданням кількості рекомендацій, тобто найпопулярніші треки виводяться першими. Далі формується текстовий звіт. У нього включається заголовок, в якому вказується, що це персональна статистика користувача, а також повідомлення про кількість рекомендацій, здійснених ботом для цього користувача. Крім цього, у звіті вказується улюблений музичний жанр користувача.

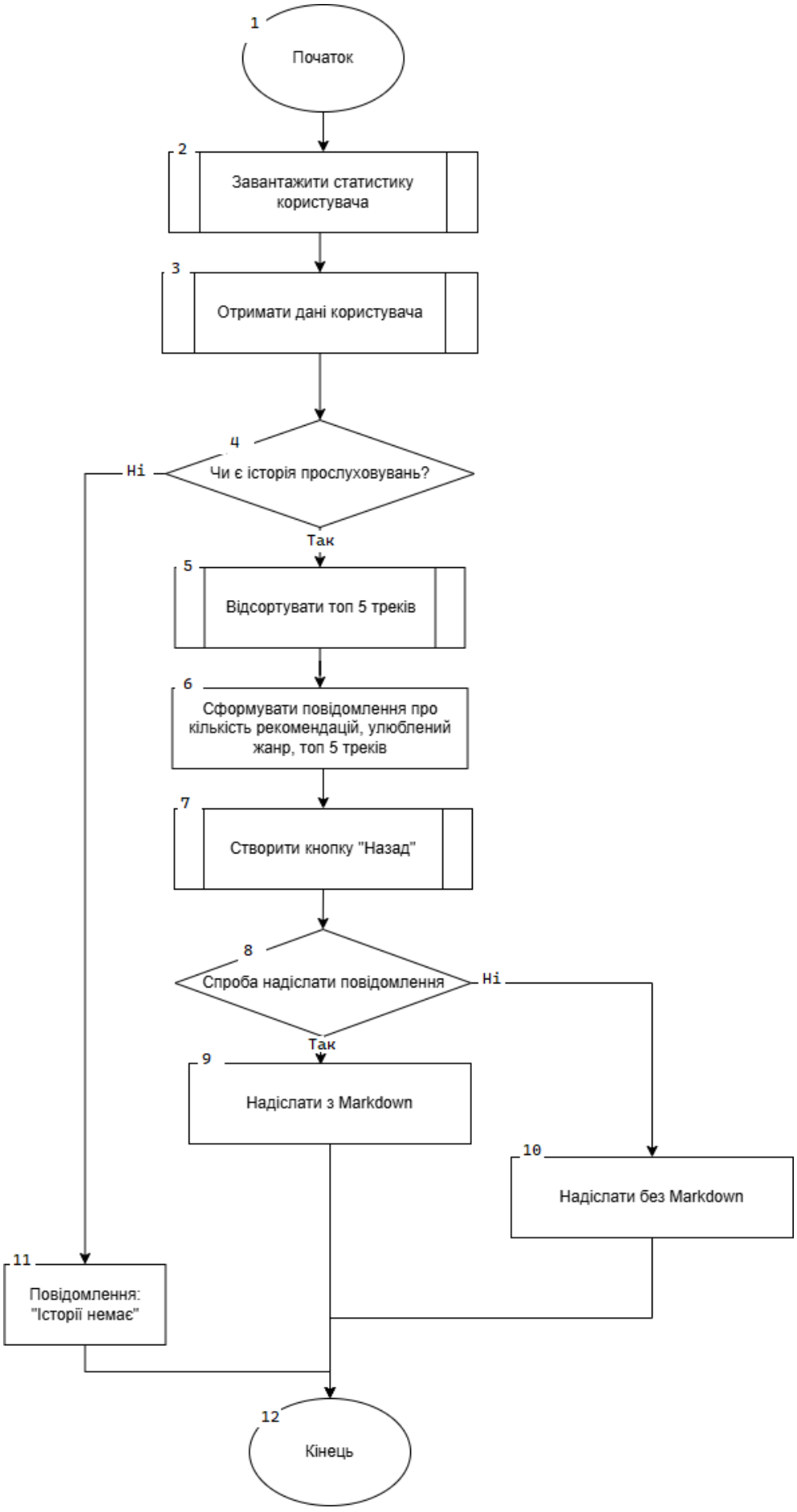

Рисунок 3.9 – Блок-схема методу send_user_stats

Якщо жанр ще не встановлений, система підставляє стандартне повідомлення, яке повідомляє, що жанр не вказано. В кінці додається перелік найпопулярніших треків, які були рекомендовані користувачеві найчастіше. Після формування текстового звіту бот також створює інтерфейсну кнопку для повернення в головне меню. Ця кнопка забезпечує зручність навігації для користувача.

Після цього бот намагається надіслати сформоване повідомлення з використанням форматування Markdown, яке дозволяє красиво оформити текстові елементи, наприклад, виділяти заголовки або вставляти списки. Якщо при надсиланні повідомлення виникає помилка, яка пов’язана із невідповідністю Markdown-розмітки, бот повторно надсилає повідомлення, попередньо прибравши потенційно небажані символи, які могли викликати цю помилку.

Таким чином, дана функція забезпечує користувача короткою, але інформативною відповіддю про його історію взаємодії з ботом, створюючи при цьому відчуття персоналізованого сервісу і заохочуючи до подальшого користування системою.

Отже, у процесі реалізації системи було створено низку алгоритмів, які забезпечують інтелектуальну поведінку музичного Telegram-бота. Основна увага була приділена формуванню персоналізованих рекомендацій, адаптованих до інтересів користувача, а також збиранню й аналізу даних про його взаємодію з системою. Реалізовано механізми фіксації кожної рекомендації та побудови статистики прослуховувань, що дало змогу відстежувати індивідуальні та загальні вподобання. Це створює основу для зворотного зв’язку та вдосконалення моделі. Зібрана інформація використовується для адаптації поведінки бота до конкретного користувача.

Алгоритм формування рекомендацій базується на імовірнісному підході, зокрема на використанні переходів між треками відповідно до алгоритму SMMC на базі Марковських ланцюгів. Такий підхід дозволяє відтворити логіку вибору, що залежить від попередніх дій, та забезпечує плавну зміну музичного

контексту. Крім того, було враховано можливість жанрових уподобань, що дозволяє зробити рекомендації більш цільовими й осмисленими.

3.5 Висновки

У третьому розділі було обґрунтовано вибір програмних засобів для реалізації інтерактивного бота, призначеного для надання музичних рекомендацій. З огляду на специфіку задачі та потребу в гнучкій обробці даних, було обрано мову програмування Python, яка відзначається простотою, широким набором бібліотек і зручністю інтеграції з Telegram API. Розробку було здійснено у середовищі Visual Studio, що забезпечує комфортну роботу з кодом, зручне налагодження та розширення функціональності.

Бот реалізує функціонал генерації рекомендацій алгоритму SMMC за допомогою Марковських ланцюгів, де ймовірності переходів між треками будуються на основі попередніх виборів користувача. Замість використання бази даних було обрано просту обробку текстового файлу, що містить список треків, а також окремий файл логів для збереження історії рекомендацій. Для побудови статистики використовується бібліотека matplotlib, яка дозволяє наочно відобразити дані у вигляді графіків. Додатково, бот вміє створювати граф переходів між треками з використанням networkx, що дозволяє користувачу зрозуміти логіку роботи рекомендаційної системи. Такий підхід забезпечує ефективну та наочну взаємодію з користувачем без необхідності складної серверної інфраструктури або баз даних.

4 ТЕСТУВАННЯ ПРОГРАМИ

4.1 Вибір методики тестування

Розроблена методика тестування мобільного застосунку YoutubeConverter спирається на фундаментальні концепції інженерії програмного забезпечення, адаптовані до особливостей сучасних мобільних технологій. В основі методики лежить принцип верифікації та валідації, сформульований у працях Бертрана Мейера[21] у контексті забезпечення якості програмних продуктів. Цей принцип включає не лише перевірку коректності технічної реалізації, але й відповідності продукту реальним потребам користувачів. У випадку Telegram-бота це набуває особливого значення, оскільки інтерфейс взаємодії суттєво відрізняється від традиційних мобільних застосунків.

Тестування [22] розпочинається з підготовчого етапу, під час якого відбувається встановлення тестової версії бота на мобільний пристрій з Android та налаштування тестового середовища в Telegram. На цьому етапі також формується набір тестових даних та сценаріїв взаємодії, що імітують реальні ситуації використання бота користувачами.

Функціональне тестування [23] фокусується на перевірці коректності виконання основних команд бота, таких як /start, /recommend, /add та /stats. Особлива увага приділяється правильності обробки введених даних та логіці роботи рекомендаційної системи. У межах цього етапу також перевіряються аналітичні можливості бота, зокрема генерація статистичних звітів та візуалізацій на основі накопичених даних.

Надзвичайно важливим компонентом методики є тестування поведінки системи в нестандартних ситуаціях. Цей етап включає введення некоректних даних, роботу з порожніми запитами та перевірку реакції бота на відсутність необхідної інформації для виконання аналітичних функцій. Аналіз повідомлень про помилки, які надає бот у таких випадках, дозволяє оцінити інформативність зворотного зв'язку та його корисність для користувача.

Безпека застосунку перевіряється шляхом тестування стійкості до потенційно небезпечних команд та оцінки захисту користувацьких даних. Цей аспект особливо актуальний для бота, який взаємодіє з зовнішніми сервісами та обробляє персональні вподобання користувачів.

Інтеграційне тестування зосереджується на перевірці взаємодії бота з YouTube API та коректності збереження й обробки даних між сеансами взаємодії. Цей етап забезпечує цілісність функціонування системи як єдиного комплексу взаємопов'язаних компонентів.

У рамках методики застосовуються різні підходи до тестування: метод «чорної скриньки» дозволяє оцінити систему з точки зору користувача; тестування на основі сценаріїв забезпечує повноту охоплення функціональних аспектів; негативне тестування перевіряє стійкість системи до некоректних вхідних даних; дослідницьке тестування виявляє неочевидні недоліки через спонтанну взаємодію з ботом.

Результати тестування оцінюються за чіткими критеріями, що включають функціональну відповідність, стійкість системи та зручність використання. Виявлені недоліки документуються за допомогою скріншотів із анотаціями та протоколів з описом невідповідностей, що дозволяє розробникам оперативно реагувати на виявлені недоліки.

4.2 Тестування програмного забезпечення

Тестування мобільного застосунку YoutubeConverter на платформі Android це процес перевірки його функціональності, стабільності та безпеки під час взаємодії з користувачем через інтерфейс Telegram-бота. Основна мета тестування впевнитися, що бот правильно реагує на введені команди, коректно обробляє дані треків і надає очікувану відповідь без збоїв чи втрати даних.

Першим кроком є встановлення та налаштування тестової версії бота на мобільному пристрої з Android та запуск його в середовищі Telegram. Далі проводиться серія функціональних тестів, які охоплюють обробку команд /start, /recommend, /add, /stats тощо. Зокрема, перевіряється, чи бот коректно додає

треки, виводить статистику та надає рекомендації відповідно до заданої логіки. Після цього виконуються тести на безпеку, які включають перевірку стійкості до некоректного вводу, обробку потенційно небезпечних команд та дотримання принципів захисту користувацьких даних.

Наприклад, при введенні некоректної або порожньої назви треку команда /add має спровокувати повідомлення про помилку та вимогу ввести коректну інформацію. Це дозволяє забезпечити надійну роботу системи та покращити досвід взаємодії користувача із ботом. Відповідну поведінку системи наведено на рисунку 4.1.

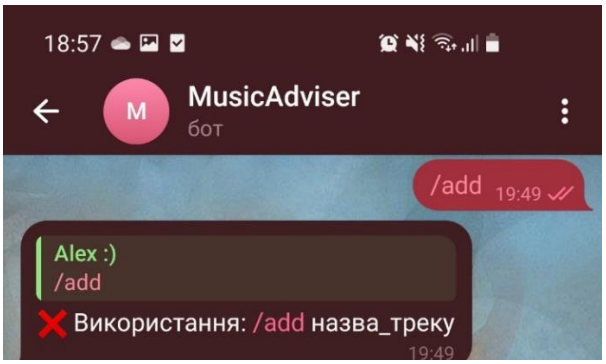

Рисунок 4.1 – Результат введення неправильних даних в команду /add

Наступним етапом тестування мобільного застосунку YoutubeConverter на платформі Android є перевірка коректності виконання команди /statistics. Ця команда призначена для генерації візуальної статистики на основі рекомендацій, які були надані ботом користувачеві протягом певного часу.

Після успішного тестування базової функціональності, як-от обробка команд та валідація введених даних, розпочинається перевірка аналітичних можливостей системи. Під час виконання команди /statistics бот повинен ініціювати процес збору інформації з попередніх взаємодій, сформувати статистичну вибірку та побудувати графік, що наочно відображає популярність треків або частоту запитів. У випадку відсутності даних бот повинен коректно обробити ситуацію та повідомити користувача про неможливість побудови статистики.

Результати тестування підтверджують, що при наявності достатньої кількості інформації бот успішно формує графік у вигляді зображення та надсилає його користувачу. Це дозволяє не лише побачити загальну активність, а й оцінити популярність конкретних музичних рекомендацій. Некоректну поведінку системи у відповідь на команду /statistics наведено на рисунку 4.2.

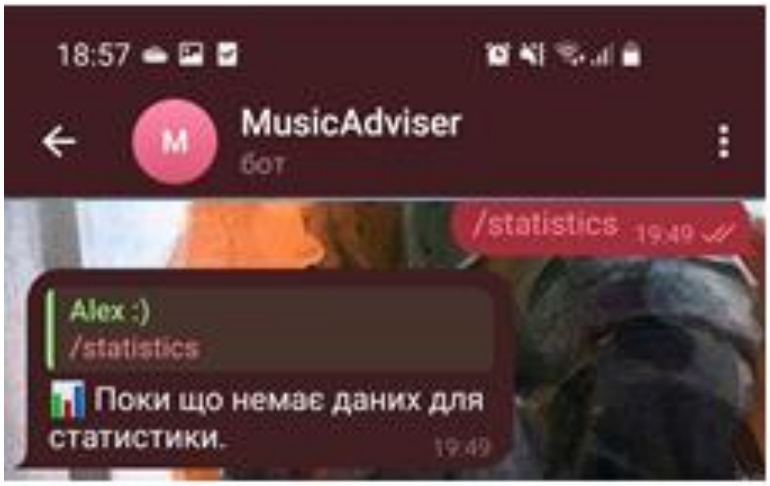

Рисунок 4.2 – Результат без введення даних у команду /statistics

У межах наступного етапу тестування мобільного застосунку YoutubeConverter на платформі Android було перевірено роботу команди /model, яка відповідає за побудову графічної моделі на основі зібраних рекомендацій. Ця команда покликана аналізувати жанрові вподобання користувача або інші ключові показники й будувати відповідну візуалізацію у вигляді графа або діаграми.

Під час тестування було змодельовано ситуацію, коли користувач ще не взаємодіяв із ботом достатньо довго, а база рекомендацій залишалася порожньою або недостатньо заповненою для аналітики. Після виклику команди /model бот намагався зібрати необхідну інформацію для побудови графу, проте, виявивши нестачу даних, коректно повідомив користувача про неможливість виконати побудову на цьому етапі.

Такий результат підтверджує стабільність роботи системи [24] навіть у виняткових випадках та її здатність адекватно реагувати на обмежену кількість

вхідних даних. Повідомлення про помилку є інформативним і дозволяє користувачу зрозуміти, що для повноцінного використання функції побудови моделі необхідно активніше взаємодіяти з ботом. Відповідний результат показано на рисунку 4.3.

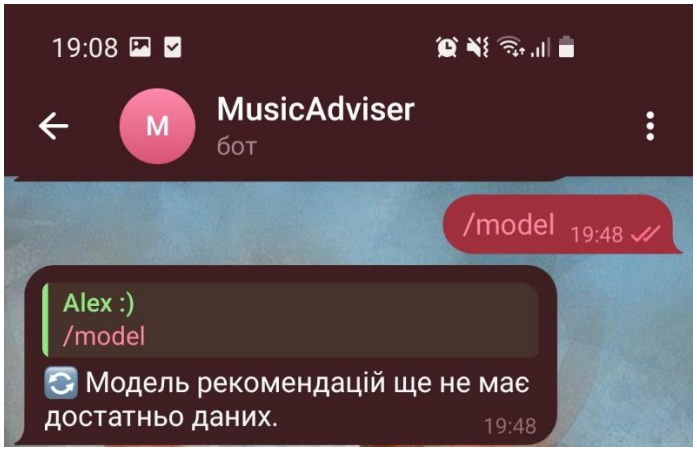

Рисунок 4.3 – Результат тестування графу без даних /model

Ці етапи тестування відіграють ключову роль у забезпеченні стабільної та зручної взаємодії користувача з мобільною версією інтерактивного бота YoutubeConverter на платформі Android. Вони дозволяють своєчасно виявити можливі помилки у функціональності, оцінити поведінку системи в нестандартних ситуаціях, а також перевірити відповідність логіки бота очікуванням користувача. Завдяки послідовному тестуванню команд на різних рівнях: від базових дій до генерації складної статистики й графічних моделей. Система демонструє високу адаптивність і стабільність у роботі.

Загалом, ці перевірки спрямовані на досягнення високої якості продукту, який не лише виконує свої функції, але й забезпечує позитивний користувацький досвід. Надійна робота застосунку на Android-пристроях гарантує, що користувачі зможуть без труднощів отримувати музичні рекомендації, взаємодіяти з ботом через зручний інтерфейс та довіряти системі при збереженні й обробці персональних уподобань. Таким чином, тестування є невід'ємною частиною створення ефективного, безпечного та привабливого цифрового інструменту.

4.3 Висновки

У четвертому розділі було здійснено комплексне тестування Telegram-бота YoutubeConverter, що працює на мобільних пристроях під управлінням Android. Основну увагу було зосереджено на перевірці коректності обробки команд користувача, зокрема валідації початкової авторизації через команду /start, а також правильності взаємодії з основними функціями системи, такими як /add, /recommend, /statistics і /model.

Окрім цього, було створено детальну інструкцію для користувачів, яка описує процес початку роботи з ботом у середовищі Telegram. Інструкція охоплює етапи першого запуску, доступ до функціоналу музичних рекомендацій, додавання треків до колекції та перегляду статистичних і графічних даних.

ВИСНОВКИ

У рамках бакалаврської кваліфікаційної роботи було розроблено інтерактивний бот YoutubeConverter, що забезпечує персоналізовані музичні рекомендації з використанням сучасних інтернет-технологій та засобів штучного інтелекту. Уся розробка здійснювалася у середовищі Visual Studio з використанням мови програмування Python.

При виборі технологічного стеку основним завданням було створення зручного, легкого у розгортанні та зрозумілого для користувача інструменту. Python забезпечив просту інтеграцію з Telegram API, а можливість обробки структурованих даних дала змогу реалізувати систему рекомендацій та статистики без підключення до повноцінної бази даних. Для візуалізації музичних вподобань було використано бібліотеки matplotlib та seaborn, що дозволило створювати графіки за запитом користувача.

У процесі реалізації досягнуто таких результатів:

– проаналізовано засоби та методи для реалізації інтерактивної системи;

– досліджено нейропсихологічні аспекти сприйняття музики та їх вплив на алгоритми рекомендацій;

– розроблено математичну модель SMMC алгоритму з адаптацією до контекстуальних факторів;

– створено багаторівневу архітектуру системи з мікросервісним підходом;

– імплементовано інтелектуальний інтерфейс взаємодії через Telegram API з використанням бібліотеки telebot;

– розроблено систему інтерактивної візуалізації музичних кластерів за допомогою matplotlib.

У першому розділі було проведено аналіз стану проблеми, порівняльний аналіз аналогів, аналіз розвитку машиного навчання, а також поставлено задачі для виконання.

У другому розділі було розроблено та описано інтерфейс розробленого інтерактивного бота, математично обґрунтовано використовувані алгоритми, а також побудовано блок схеми для візуалізації внутрішніх процесів.

У третьому розділі було обґрунтовано вибір мови програмування та середовища розробки, представлено програмну реалізацію та створено інструкцію користувача.

У четвертому розділі було створено методику тестування для створеного бота, а також протестовано його можливості.

Для покращення розуміння логіки роботи системи було створено блок-схеми взаємодії з користувачем. Завершальним етапом стала перевірка працездатності Telegram-бота на мобільному пристрої з операційною системою Android, що підтвердило його функціональну готовність до використання кінцевими користувачами. Результати оформлено згідно методичних вказівок до виконання бакалаврських кваліфікаційних робіт [25].

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Сервіс Telegram. URL: https://web.telegram.org/k/ (дата звернення: 15.04.2025)

2. Тіслін О.Ю. Переваги та недоліки SMMC алгоритму при створенні рекомендацій. Матеріали LIV Всеукраїнської науково-технічної конференції професорсько-викладацького складу, науковців, аспірантів та студентів підрозділів університету з участю працівників підприємств (НТКП ВНТУ-2025). Вінниця, 2025. URL: https://conferences.vntu.edu.ua/index.php/all-fitki/all-fitki-2025/paper/view/24479/20188 (дата звернення: 14.04.2025).

3. A. Kumar and R. Singh, «Trends in Intelligent Chatbots: A Review», IEEE Access, vol. 9, pp. 12345–12360, 2021.

4. M. Zhang et al., «Development of Telegram Bots for Interactive Recommendation Systems», Proc. IEEE Int. Conf. Softw. Eng. Adv., 2022.

5. L. He and W. Zhang, «Matrix Completion with Similarity Learning for Recommendation», IEEE Trans. Knowl. Data Eng., vol. 33, no. 4, pp. 1231–1242, 2021.

6. S. Bose et al., «Personalized Recommendation in Messaging Platforms Using Hybrid Algorithms», IEEE Int. Conf. Big Data, pp. 203–210, 2022.

7. Інтерактивний Telegram-бот Spotybot. URL: t.me/spotybot (дата звернення: 16.04.2025)

8. Інтерактивний Telegram-бот t.me/MusicTubeBot_Bot (дата звернення: 17.04.2025)

9. TuneMyMusic. URL: https://www.tunemymusic.com (дата звернення: 18.04.2025)

10. Fredboat. URL: https://fredboat.com/ (дата звернення: 19.04.2025)

11. Марковські ланцюги. URL: https://teorver.pp.ua/ukr/labs/MarkovSite/MarkovSite.html (дата звернення: 20.04.2025)

12. Purushotham S., Liu Y. Collaborative Topic Regression with Social Matrix Factorization for Recommendation Systems . URL: https://icml.cc/2012/papers/407.pdf. (дата звернення: 21.04.2025)

13. Бібліотека Matplotlib. URL: https://pypi.org/project/matplotlib/ (дата звернення: 22.04.2025)

14. Ланцюги Маркова. URL: https://ukrayinska.libretexts.org/Математика/Прикладна_математика/Прикладна_скінченна_математика_(Sekhon_i_Bloom)/10:_Ланцюги_Маркова (дата звернення: 23.04.2025)

15. User Inerface, UI/UX. URL: https://prjctr.com/mag/uxui-questions (дата звернення: 24.04.2025)

16. Мова Python переваги і недоліки. URL: https://justjoin.it/blog/все-що-ви-маєте-знати-про-python-які-в-нього-н (дата звернення: 25.04.2025)

17. Середовище Visual Studio переваги та недоліки. URL: https://itpro.ua/post/kakie_preimushchestva_vkhodyat_v_visual_studio_subscription_v_zavisimosti_ot_vybrannogo_urovnya_podpiski?srsltid=AfmBOoq4aL-P30_DwlaOLDFEFCHUhpZgJx6oOGQRfyTVNKOhC0AFy2PN (дата звернення: 26.04.2025)

18. Як налаштувати логування в Python. URL: https://dou.ua/forums/topic/48126/ (дата звернення: 27.04.2025)

19. Документація Networkx. URL: https://networkx.org/ (дата звернення: 28.04.2025)

20. Побудова графіків засобами мови Python. URL: https://www.miyklas.com.ua/p/informatica/9-klas/algoritmi-i-programi-python-447430/vizualizatciia-elementiv-tablichnoyi-velichini-448001/re-6106a9b3-34aa-4f60-86c5-083776965044 (дата звернення: 29.04.2025)

21. Meyer, B. Seven Principles of Software Testing. IEEE Computer, 41(8), pp. 99-101. 2008.

22. Тестування ПЗ. URL: https://university.sigma.software/what-is-software-testing/ (дата звернення: 30.04.2025)

23. Функціональне тестування. URL: https://qalight.ua/baza-znaniy/funktsionalne-testuvannya/ (дата звернення: 01.05.2025)

24. Що таке стабільність системи? URL: https://ukrayinska.libretexts.org/Інженерна/Промислове_та_системного_машинобудування/Книга:_Вступ_до_систем_управління_(Iqbal)/02:_Моделі_функцій_передачі/2.03:_Стабільність_системи (дата звернення: 02.05.2025)

25. О.Н. Романюк, Г.О. Черноволик, Методичні вказівки до виконання бакалаврських кваліфікаційних робіт для студентів спеціальності 121 «Інженерія програмного забезпечення», Вінниця, ВНТУ, 2022, с. 52.

ДОДАТКИ

Додаток А – Технічне завдання

Міністерство освіти і науки України

Вінницький національний технічний університет

Факультет інформаційних технологій та комп'ютерної інженерії

ЗАТВЕРДЖУЮ
д.т.н., проф. О. Н. Романюк
"21" 03 2025 р.

Технічне завдання

на бакалаврську кваліфікаційну роботу «Розробка інтерактивного бота для музичних рекомендацій на основі SMMC алгоритму для платформи Telegram з використанням бібліотек telebot та matplotlib»

за спеціальністю 121 Інженерія програмного забезпечення

Керівник бакалаврської кваліфікаційної роботи:
_____ д.т.н., доц. каф. ПЗ Д.І. Катєльніков
"24" 03 2025 р.

Виконав:
_____ студент гр.4ПІ-21б О. Ю. Тіслін
"24" 03 2025 р.

Вінниця – 2025 року

1. **Найменування та галузь застосування**

Бакалаврська кваліфікаційна робота: «Розробка інтерактивного бота для музичних рекомендацій на основі SMMC алгоритму для платформи Telegram з використанням бібліотек telebot та matplotlib».

Галузь застосування – онлайн-спільноти.

2. Підстава для розробки.

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

3. Мета та призначення розробки.

Метою дослідження є покращення якості існуючого й створення нового функціоналу за рахунок зменшення складності обчислень, а також локальної обробки даних без використання сторонніх сервісів.

Призначення роботи – розробка методів і засобів створення програмного забезпечення інтерактивного Telegram бота з використанням алгоритмів SMMC для формування музичних рекомендацій та статистики.

4. Вихідні дані для проведення НДР

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

1. Purushotham S., Liu Y. Collaborative Topic Regression with Social Matrix Factorization for Recommendation Systems . URL: https://icml.cc/2012/papers/407.pdf.

2. A. Kumar and R. Singh, «Trends in Intelligent Chatbots: A Review», IEEE Access, vol. 9, pp. 12345–12360, 2021.

3. M. Zhang et al., «Development of Telegram Bots for Interactive Recommendation Systems», Proc. IEEE Int. Conf. Softw. Eng. Adv., 2022.

4. L. He and W. Zhang, «Matrix Completion with Similarity Learning for Recommendation», IEEE Trans. Knowl. Data Eng., vol. 33, no. 4, pp. 1231–1242, 2021.

5. Технічні вимоги

Вихідні дані до роботи: формування персоналізованих музичних рекомендацій на основі побудованої Марковської моделі; додавання треків до системи; перегляд, оновлення та аналіз статистики переходів між треками; генерацію графа ймовірнісних переходів; обробку запитів користувача через Telegram Bot API; логування дій користувача й етапів побудови моделі; виведення інформаційних повідомлень, зокрема рекомендацій, підказок щодо використання та повідомлень про помилки; візуалізацію структури моделей за допомогою відповідних бібліотек Python; забезпечення інтерактивної взаємодії з користувачем у текстовому середовищі Telegram.

6. Конструктивні вимоги.

Графічна та текстова документація повинна відповідати діючим стандартам України.

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

– пояснювальна записка до БКР;

– технічне завдання;

– лістинги програми.

8. Вимоги до рівня уніфікації та стандартизації

При розробці програмних засобів слід дотримуватися уніфікації і ДСТУ.

9. Стадії та етапи розробки:

№ з/п	Назва етапів бакалаврської кваліфікаційної роботи	Строк виконання етапів роботи
1	Аналіз програмного забезпечення для формування музичних рекомендацій	25.03.25 – 31.03.2025
2	Розробка алгоритмів та архітектури програмного забезпечення	31.03.2025 – 05.04.2025
3	Розробка модуля музичних рекомендацій	05.04.2025 – 12.04.2025
4	Розробка модуля навчання бота	12.04.2025 – 23.04.2025
5	Розробка програмного забезпечення	23.04.2025 – 01.05.2025
6	Тестування програмного забезпечення	01.05.2025 – 10.05.2025
7	Оформлення матеріалів до захисту БКР	10.05.2025 – 30.05.25

10. Порядок контролю та прийняття.

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

Додаток Б – Протокол перевірки БКР на плагіат

ПРОТОКОЛ ПЕРЕВІРКИ КВАЛІФІКАЦІЙНОЇ РОБОТИ

Назва роботи: Розробка інтерактивного бота для музичних рекомендацій на основі SMMC алгоритму для платформи Telegram з використанням бібліотек telebot та matplotlib

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

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

Коефіцієнт подібності текстових запозичень, виявлених у роботі системою StrikePlagiarism 5,8 %

Висновок щодо перевірки кваліфікаційної роботи (відмітити потрібне)

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

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

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

Експертна комісія:

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

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

Особа, відповідальна за перевірку _____ Черноволик Г. О.

З висновком експертної комісії ознайомлений(-на)

Керівник _____ Катєльніков Д.І., ктн., доц. каф. ПЗ
(підпис) (прізвище, ініціали, посада)

Здобувач _____ Тіслін О.Ю.
(підпис) (прізвище, ініціали)

Додаток В – Лістинг програмного забезпечення

```
import telebot
import os
import random
import numpy as np
import matplotlib.pyplot as plt
import networkx as nx
from telebot import types
from telebot.types import BotCommand
import datetime
import json

API_TOKEN = '7266135853:AAH_DZFICITr8zLFhYvqHqtPjKgC_0MwjTo'
bot = telebot.TeleBot(API_TOKEN)

TRACKS_FILE = 'tracks.txt'
LOG_FILE = 'recommendation_log.txt'
USERS_FILE = 'users.json'

transition_matrix = {}
track_list = []
last_recommendation = {}  # Track per user now
user_preferences = {}  # Store user preferences

# ---------- ДОПОМІЖНІ ФУНКЦІЇ ----------
def load_tracks():
    global track_list
    if os.path.exists(TRACKS_FILE):
        with open(TRACKS_FILE, 'r', encoding='utf-8') as file:
            track_list = [line.strip() for line in file.readlines() if line.strip()]
    return track_list

def save_tracks():
    with open(TRACKS_FILE, 'w', encoding='utf-8') as f:
```

```
        for track in track_list:
            f.write(track + '\n')

def load_users():
    if os.path.exists(USERS_FILE):
        with open(USERS_FILE, 'r', encoding='utf-8') as file:
            return json.load(file)
    return {}

def save_users(users_data):
    with open(USERS_FILE, 'w', encoding='utf-8') as file:
        json.dump(users_data, file, indent=4)

def log_recommendation(user_id, track):
    with open(LOG_FILE, 'a', encoding='utf-8') as file:
        timestamp = datetime.datetime.now().strftime("%Y-%m-%d %H:%M:%S")
        file.write(f"{timestamp},{user_id},{track}\n")

def get_recommendation_stats(user_id=None):
    if not os.path.exists(LOG_FILE):
        return {}
    try:
        with open(LOG_FILE, 'r', encoding='utf-8') as file:
            recommendations = file.readlines()
        stats = {}
        for line in recommendations:
            parts = line.strip().split(',', 2)
            if len(parts) >= 3:
                timestamp, rec_user_id, track = parts
                if user_id is None or str(rec_user_id) == str(user_id):
                    stats[track] = stats.get(track, 0) + 1
        return stats
    except Exception as e:
        print(f"Error reading recommendation stats: {str(e)}")
        return {}
```

```
def initialize_matrix():
    global transition_matrix
    if not track_list:
        return
    transition_matrix = {track: {t: 1 / len(track_list) for t in track_list} for track in track_list}

def update_matrix(prev_track, new_track):
    if prev_track not in transition_matrix:
        transition_matrix[prev_track] = {t: 1 / len(track_list) for t in track_list}
    transition_matrix[prev_track][new_track] += 1
    total = sum(transition_matrix[prev_track].values())
    for key in transition_matrix[prev_track]:
        transition_matrix[prev_track][key] /= total

def markov_recommendation(user_id):
    if not track_list:
        return None

    last_track = last_recommendation.get(user_id)

    # Use user preferences if available
    user_data = load_users().get(str(user_id), {})
    favorite_genre = user_data.get('favorite_genre')

    filtered_tracks = track_list
    if favorite_genre:
        filtered_tracks = [track for track in track_list if favorite_genre.lower() in track.lower()]
        if not filtered_tracks:  # If no tracks match genre, use all tracks
            filtered_tracks = track_list

    if last_track and last_track in transition_matrix:
        next_tracks = list(transition_matrix[last_track].keys())
        probabilities = list(transition_matrix[last_track].values())
```

```
            # Filter by genre if applicable
            if favorite_genre:
                genre_indices = [i for i, track in enumerate(next_tracks) if favorite_genre.lower() in track.lower()]
                if genre_indices:  # If we have genre matches, increase their probability
                    for idx in genre_indices:
                        probabilities[idx] *= 1.5
                    # Normalize probabilities
                    total = sum(probabilities)
                    probabilities = [p/total for p in probabilities]

            return np.random.choice(next_tracks, p=probabilities)

        # Random selection with genre preference
        if filtered_tracks:
            return random.choice(filtered_tracks)
        return random.choice(track_list)

    def get_track_genres():
        """Extract unique genres from track names for filtering"""
        genres = set()
        for track in track_list:
            parts = track.split(' - ')
            if len(parts) > 1:
                # Assuming genre might be mentioned in the track description
                potential_genres = ['rock', 'pop', 'jazz', 'classical', 'hip hop', 'electronic', 'ambient', 'folk']
                for genre in potential_genres:
                    if genre in track.lower():
                        genres.add(genre)
        return list(genres)

    def extract_link(track):
        """Extract link from track text if it exists"""
        if ' - ' in track:
```

```
            parts = track.split(' - ', 1)
            if 'http' in parts[1]:
                return parts[1].strip()
        return None

    # ---------------- МЕНЮ КОМАНД ------------------
    def send_main_menu(chat_id):
        markup = types.InlineKeyboardMarkup(row_width=2)

        # Main functionality
        recommend_btn = types.InlineKeyboardButton("🎶 Рекомендація", callback_data="recommend")
        stats_btn = types.InlineKeyboardButton("📊 Статистика", callback_data="stats")

        # Track management
        list_btn = types.InlineKeyboardButton("📋 Список треків", callback_data="list")
        add_btn = types.InlineKeyboardButton("➕ Додати трек", callback_data="add")

        # Advanced features
        graph_btn = types.InlineKeyboardButton("📈 Граф", callback_data="graph")
        my_stats_btn = types.InlineKeyboardButton("🔍 Моя статистика", callback_data="my_stats")

        # Settings and preferences
        settings_btn = types.InlineKeyboardButton("⚙️ Налаштування", callback_data="settings")
        help_btn = types.InlineKeyboardButton("❓ Допомога", callback_data="help")

        markup.add(recommend_btn, stats_btn)
        markup.add(list_btn, add_btn)
        markup.add(graph_btn, my_stats_btn)
        markup.add(settings_btn, help_btn)
```

```
        bot.send_message(chat_id, "🎵 *Музичний рекомендаційний бот*\nВиберіть опцію:",
                         parse_mode="Markdown", reply_markup=markup)

    def send_settings_menu(chat_id):
        markup = types.InlineKeyboardMarkup(row_width=1)

        genre_btn = types.InlineKeyboardButton("🎸 Вибрати улюблений жанр", callback_data="set_genre")
        reset_btn = types.InlineKeyboardButton("🗑 Скинути налаштування", callback_data="reset_settings")
        clear_history_btn = types.InlineKeyboardButton("🧹 Очистити історію", callback_data="clear_history")
        back_btn = types.InlineKeyboardButton("⬅ Назад", callback_data="main_menu")

        markup.add(genre_btn, reset_btn, clear_history_btn, back_btn)

        bot.send_message(chat_id, "⚙ *Налаштування*\nНалаштуйте бота під свої уподобання:",
                         parse_mode="Markdown", reply_markup=markup)

    def send_genre_menu(chat_id):
        genres = get_track_genres()
        if not genres:
            genres = ["Rock", "Pop", "Electronic", "Hip Hop", "Jazz", "Classical"]

        markup = types.InlineKeyboardMarkup(row_width=2)
        genre_buttons = [types.InlineKeyboardButton(genre.title(), callback_data=f"genre_{genre}") for genre in genres]

        # Add buttons in pairs
        for i in range(0, len(genre_buttons), 2):
            if i + 1 < len(genre_buttons):
                markup.add(genre_buttons[i], genre_buttons[i+1])
```

```
            else:
                markup.add(genre_buttons[i])

        back_btn = types.InlineKeyboardButton("⬅️ Назад", callback_data="settings")
        markup.add(back_btn)

        bot.send_message(chat_id, "🎸 *Виберіть улюблений жанр*\nЦе вплине на рекомендації:",
                         parse_mode="Markdown", reply_markup=markup)

    # ---------- ОБРОБНИКИ КОМАНД ----------

    # Обробник команди /start
    @bot.message_handler(commands=['start'])
    def start(message):
        user_id = message.from_user.id
        users = load_users()

        if str(user_id) not in users:
            users[str(user_id)] = {
                "username": message.from_user.username or "Unknown",
                "join_date": datetime.datetime.now().strftime("%Y-%m-%d"),
                "favorite_genre": None,
                "recommendations_count": 0
            }
            save_users(users)

        bot.send_message(message.chat.id, f"Привіт, {message.from_user.first_name}! 👋\n"
                         "Я музичний рекомендаційний бот з використанням ланцюгів Маркова.\n"
                         "Ви можете використовувати вбудоване меню команд бота в Telegram "
                         "(натисніть на кнопку меню або іконку / в полі введення), "
                         "або використовуйте інтерактивне меню нижче:")

    # Перезагрузка встроенного меню команд при старте (для обновления у пользователя)
```

```
    try:
        setup_bot_commands()
    except Exception as e:
        print(f"Error setting up commands: {e}")

    send_main_menu(message.chat.id)

# Обробник команди /menu
@bot.message_handler(commands=['menu'])
def menu(message):
    send_main_menu(message.chat.id)

# Обробник кнопок меню
@bot.callback_query_handler(func=lambda call: True)
def handle_query(call):
    user_id = call.from_user.id
    chat_id = call.message.chat.id

    # Main menu options
    if call.data == "recommend":
        recommend_music_callback(chat_id, user_id)
    elif call.data == "stats":
        send_stats(chat_id)
    elif call.data == "list":
        list_tracks_callback(chat_id)
    elif call.data == "add":
        request_track_add(chat_id)
    elif call.data == "graph":
        send_graph(chat_id)
    elif call.data == "my_stats":
        send_user_stats(chat_id, user_id)
    elif call.data == "settings":
        send_settings_menu(chat_id)
    elif call.data == "help":
        send_help(chat_id)
```

```
        elif call.data == "main_menu":
            send_main_menu(chat_id)

        # Settings options
        elif call.data == "set_genre":
            send_genre_menu(chat_id)
        elif call.data == "reset_settings":
            reset_user_settings(chat_id, user_id)
        elif call.data == "clear_history":
            clear_user_history(chat_id, user_id)

        # Genre selection
        elif call.data.startswith("genre_"):
            genre = call.data[6:]  # Remove "genre_" prefix
            set_user_genre(chat_id, user_id, genre)

        # Like/Dislike functionality
        elif call.data == "like_track":
            bot.send_message(chat_id, "👍 Дякую за оцінку! Будемо рекомендувати більше схожих треків.")
        elif call.data == "dislike_track":
            bot.send_message(chat_id, "👎 Дякую за зворотний зв'язок! Ми врахуємо це в майбутніх рекомендаціях.")

        # Try to edit the message to remove the inline keyboard
        try:
            bot.answer_callback_query(call.id)
        except Exception as e:
            print(f"Error answering callback: {e}")

    def recommend_music_callback(chat_id, user_id):
        load_tracks()
        initialize_matrix()

        new_track = markov_recommendation(user_id)
```

```
        if not new_track:
            bot.send_message(chat_id, "⚠️ Немає треків у списку.")
            return

        # Update user stats
        users = load_users()
        if str(user_id) in users:
            users[str(user_id)]["recommendations_count"] = users[str(user_id)].get("recommendations_count", 0) + 1
            save_users(users)

        # Create a message with a link if available
        link = extract_link(new_track)
        message_text = f"🎶 *Сьогоднішня рекомендація:*\n{new_track}"

        # Add button to request another recommendation
        markup = types.InlineKeyboardMarkup(row_width=2)
        another_btn = types.InlineKeyboardButton("🎲 Ще одна", callback_data="recommend")
        back_btn = types.InlineKeyboardButton("🔙 Меню", callback_data="main_menu")
        like_btn = types.InlineKeyboardButton("👍 Подобається", callback_data="like_track")
        dislike_btn = types.InlineKeyboardButton("👎 Не подобається", callback_data="dislike_track")

        markup.add(like_btn, dislike_btn)
        markup.add(another_btn, back_btn)

        bot.send_message(chat_id, message_text, parse_mode="Markdown", reply_markup=markup)

        log_recommendation(user_id, new_track)
        last_rec = last_recommendation.get(user_id)
        if last_rec:
            update_matrix(last_rec, new_track)
        last_recommendation[user_id] = new_track
```

```
        def list_tracks_callback(chat_id):
            load_tracks()
            if not track_list:
                bot.send_message(chat_id, "📂 Список треків порожній.")
                return

            # Create paginated track list
            markup = types.InlineKeyboardMarkup(row_width=2)
            back_btn = types.InlineKeyboardButton("🔙 Назад", callback_data="main_menu")
            remove_btn = types.InlineKeyboardButton("🗑 Видалити трек",
callback_data="remove_track")
            markup.add(remove_btn, back_btn)

            msg = "🎶 *Треки у списку:*\n" + "\n".join(f"{i+1}. {track}" for i, track in
enumerate(track_list))
            bot.send_message(chat_id, msg, parse_mode="Markdown", reply_markup=markup)

        def send_user_stats(chat_id, user_id):
            stats = get_recommendation_stats(user_id)
            users = load_users()
            user_data = users.get(str(user_id), {})

            if not stats:
                msg = "⚠️ У вас ще немає історії прослуховувань."
            else:
                # Get top 5 tracks
                top_tracks = sorted(stats.items(), key=lambda x: x[1], reverse=True)[:5]
                top_tracks_str = "\n".join(f"• {track} ({count} разів)" for track, count in top_tracks)

                msg = f"👤 *Ваша статистика*\n\n"
                msg += f"Всього рекомендацій: {user_data.get('recommendations_count', 0)}\n"
                msg += f"Улюблений жанр: {user_data.get('favorite_genre', 'Не вказано')}\n\n"
                msg += f"Топ треків:\n{top_tracks_str}"
```

```
            markup = types.InlineKeyboardMarkup()
            back_btn = types.InlineKeyboardButton("⬅️ Назад", callback_data="main_menu")
            markup.add(back_btn)

            try:
                bot.send_message(chat_id, msg, parse_mode="Markdown", reply_markup=markup)
            except Exception as e:
                # В случае проблем с форматированием Markdown, отправляем без него
                bot.send_message(chat_id, msg.replace('*', ''), reply_markup=markup)

        def set_user_genre(chat_id, user_id, genre):
            users = load_users()
            if str(user_id) not in users:
                users[str(user_id)] = {"username": "Unknown", "join_date":
datetime.datetime.now().strftime("%Y-%m-%d")}

            users[str(user_id)]["favorite_genre"] = genre
            save_users(users)

            bot.send_message(chat_id, f"✅ Ваш улюблений жанр встановлено: *{genre}*",
parse_mode="Markdown")
            send_settings_menu(chat_id)

        def reset_user_settings(chat_id, user_id):
            users = load_users()
            if str(user_id) in users:
                users[str(user_id)]["favorite_genre"] = None
                save_users(users)

            bot.send_message(chat_id, "🔄 Ваші налаштування скинуто")
            send_settings_menu(chat_id)

        def clear_user_history(chat_id, user_id):
            if not os.path.exists(LOG_FILE):
```

```
        bot.send_message(chat_id, "⚠️ Історія вже порожня.")
        return

    # Create new log without this user's entries
    with open(LOG_FILE, 'r', encoding='utf-8') as file:
        all_logs = file.readlines()

    with open(LOG_FILE, 'w', encoding='utf-8') as file:
        for log in all_logs:
            if str(user_id) not in log:
                file.write(log)

    bot.send_message(chat_id, "🗑 Вашу історію рекомендацій очищено.")
    send_settings_menu(chat_id)

def send_help(chat_id):
    help_text = (
        "*🎧 Музичний рекомендаційний бот*\n\n"
        "*Основні команди:*\n"
        "• /start - Запустити бота\n"
        "• /menu - Показати головне меню\n\n"

        "*Як це працює:*\n"
        "1. Бот рекомендує музику на основі ланцюгів Маркова\n"
        "2. Кожна рекомендація враховує ваші попередні вподобання\n"
        "3. Встановіть улюблений жанр для кращих рекомендацій\n\n"

        "*Додаткові функції:*\n"
        "• Статистика прослуховувань\n"
        "• Візуалізація графу переходів\n"
        "• Управління списком треків\n"
        "• Персональні налаштування\n\n"

        "Якщо у вас є питання або пропозиції, зв'яжіться з адміністратором."
    )
```

```
    markup = types.InlineKeyboardMarkup()
    back_btn = types.InlineKeyboardButton("⬅️ Назад", callback_data="main_menu")
    markup.add(back_btn)

    bot.send_message(chat_id, help_text, parse_mode="Markdown", reply_markup=markup)

# ---------- СТАРІ ОБРОБНИКИ КОМАНД (для сумісності) ----------

@bot.message_handler(commands=['recommend'])
def recommend_music(message):
    recommend_music_callback(message.chat.id, message.from_user.id)

@bot.message_handler(commands=['stats'])
def stats_command(message):
    send_stats(message.chat.id)

def send_stats(chat_id):
    stats = get_recommendation_stats()
    if not stats:
        bot.send_message(chat_id, "⚠️ Поки немає статистики.")
        return

    # Sort tracks by popularity
    sorted_stats = sorted(stats.items(), key=lambda x: x[1], reverse=True)

    plt.figure(figsize=(10, 5))
    plt.bar([item[0].split(' - ')[0] if ' - ' in item[0] else item[0] for item in sorted_stats][:10],
            [item[1] for item in sorted_stats][:10], color='lightgreen')
    plt.xticks(rotation=45, ha="right")
    plt.xlabel("Треки")
    plt.ylabel("Кількість рекомендацій")
    plt.title("📊 Статистика музичних рекомендацій")
    filename =
f"stats_chart_{datetime.datetime.now().strftime('%Y%m%d%H%M%S')}.png"
```

```
        plt.savefig(filename, bbox_inches='tight')
        plt.close()

        markup = types.InlineKeyboardMarkup()
        back_btn = types.InlineKeyboardButton("⬅ Назад", callback_data="main_menu")
        markup.add(back_btn)

        try:
            with open(filename, 'rb') as img:
                bot.send_photo(chat_id, img, reply_markup=markup)
            os.remove(filename)
        except Exception as e:
            bot.send_message(chat_id, f"⚠ Помилка при відправці графіка: {str(e)}")
            os.remove(filename) if os.path.exists(filename) else None

    @bot.message_handler(commands=['graph'])
    def graph_command(message):
        send_graph(message.chat.id)

    def send_graph(chat_id):
        if not transition_matrix:
            initialize_matrix()

        if not transition_matrix:
            bot.send_message(chat_id, "⚠ Недостатньо даних для графу.")
            return

        G = nx.DiGraph()
        for track, transitions in transition_matrix.items():
            # Получаем полное название трека для отладки
            track_name = track if len(track.split(' - ')) <= 1 else track.split(' - ')[0][:20]
            for next_track, weight in transitions.items():
                next_name = next_track if len(next_track.split(' - ')) <= 1 else next_track.split(' -
')[0][:20]
                # Уменьшаем порог для отображения ребер, чтобы показать больше данных
```

```
            if weight > 0.05:  # Порог 0.05 вместо 0.1 для большей детализации
                G.add_edge(track_name, next_name, weight=round(weight, 2))

        if not G.nodes():
            bot.send_message(chat_id, "⚠️ Граф порожній. Можливо, недостатньо даних або всі ваги занадто малі.")
            return

        # Ограничиваем количество узлов для читаемости
        if len(G.nodes()) > 15:
            top_tracks = sorted([(n, G.degree(n)) for n in G.nodes()], key=lambda x: x[1], reverse=True)[:15]
            top_nodes = [n[0] for n in top_tracks]
            G = G.subgraph(top_nodes)

        # Настраиваем визуализацию
        plt.figure(figsize=(14, 10))
        pos = nx.spring_layout(G, seed=42, k=0.9)
        nx.draw(G, pos,
                with_labels=True,
                node_size=4000,
                node_color="lightblue",
                edge_color="gray",
                font_size=10,
                font_weight="bold",
                arrows=True,
                arrowsize=20)

        # Добавляем подписи к ребрам
        edge_labels = {(u, v): f"{d['weight']:.2f}" for u, v, d in G.edges(data=True)}
        nx.draw_networkx_edge_labels(G, pos, edge_labels=edge_labels, font_size=8)

        plt.title("🔄 Граф переходів між треками", fontsize=14, pad=20)
        graph_img = f"graph_{datetime.datetime.now().strftime('%Y%m%d%H%M%S')}.png"
```

```
            try:
                plt.savefig(graph_img, bbox_inches='tight', dpi=100)
                plt.close()

                markup = types.InlineKeyboardMarkup()
                back_btn = types.InlineKeyboardButton("⬅️ Назад", callback_data="main_menu")
                markup.add(back_btn)

                with open(graph_img, 'rb') as img:
                    bot.send_photo(chat_id, img, reply_markup=markup)
                os.remove(graph_img)
            except Exception as e:
                bot.send_message(chat_id, f"⚠️ Помилка при відправці графіка: {str(e)}")
                if os.path.exists(graph_img):
                    os.remove(graph_img)

        @bot.message_handler(commands=['list'])
        def list_tracks(message):
            list_tracks_callback(message.chat.id)

        @bot.message_handler(commands=['add'])
        def request_track_add(chat_id):
            markup = types.ForceReply(selective=True)
            msg = bot.send_message(chat_id, "Введи новий трек у форматі:\nНазва - посилання",
reply_markup=markup)
            bot.register_next_step_handler(msg, add_track)

        def add_track(message):
            text = message.text.strip()
            if "-" not in text:
                bot.send_message(message.chat.id, "⚠️ Невірний формат. Використай: Назва -
посилання")
                send_main_menu(message.chat.id)
                return
```

```
        load_tracks()
        track_list.append(text)
        save_tracks()

        bot.send_message(message.chat.id, f"✅ Трек додано: {text}")
        send_main_menu(message.chat.id)

    @bot.message_handler(commands=['clear'])
    def clear_history(message):
        if os.path.exists(LOG_FILE):
            os.remove(LOG_FILE)
        bot.send_message(message.chat.id, "🗑 Історію рекомендацій очищено.")
        send_main_menu(message.chat.id)

    @bot.message_handler(commands=['remove'])
    def request_track_remove(message):
        markup = types.ForceReply(selective=True)
        msg = bot.send_message(message.chat.id, "Введіть номер треку, який хочете видалити:", reply_markup=markup)
        bot.register_next_step_handler(msg, remove_track)

    def remove_track(message):
        try:
            track_num = int(message.text) - 1
            load_tracks()

            if 0 <= track_num < len(track_list):
                removed = track_list.pop(track_num)
                save_tracks()
                bot.send_message(message.chat.id, f"✅ Трек видалено: {removed}")
            else:
                bot.send_message(message.chat.id, "⚠️ Невірний номер треку.")
        except ValueError:
            bot.send_message(message.chat.id, "⚠️ Будь ласка, введіть номер треку.")
```

```
    send_main_menu(message.chat.id)

# ---------- ВСТРОЕННОЕ МЕНЮ КОМАНД ----------
def setup_bot_commands():
    """Настройка встроенного меню команд бота Telegram"""
    commands = [
        BotCommand(command="start", description="Запустити бота"),
        BotCommand(command="menu", description="Показати головне меню"),
        BotCommand(command="recommend", description="Отримати рекомендацію"),
        BotCommand(command="list", description="Список треків"),
        BotCommand(command="add", description="Додати трек"),
        BotCommand(command="stats", description="Загальна статистика"),
        BotCommand(command="mystats", description="Моя статистика"),
        BotCommand(command="settings", description="Налаштування"),
        BotCommand(command="graph", description="Показати граф переходів"),  # Добавляем команду
        BotCommand(command="help", description="Допомога")
    ]
    bot.set_my_commands(commands)

# Добавляем обработчики для новых команд
@bot.message_handler(commands=['mystats'])
def my_stats_command(message):
    send_user_stats(message.chat.id, message.from_user.id)

@bot.message_handler(commands=['settings'])
def settings_command(message):
    send_settings_menu(message.chat.id)

@bot.message_handler(commands=['help'])
def help_command(message):
    send_help(message.chat.id)

# ---------- ЗАПУСК ----------
```

```
if __name__ == "__main__":
    load_tracks()
    initialize_matrix()
    # Настройка встроенного меню команд
    setup_bot_commands()
    print("Bot started with command menu...")
    bot.infinity_polling()
```

Додаток Г – Графічна частина

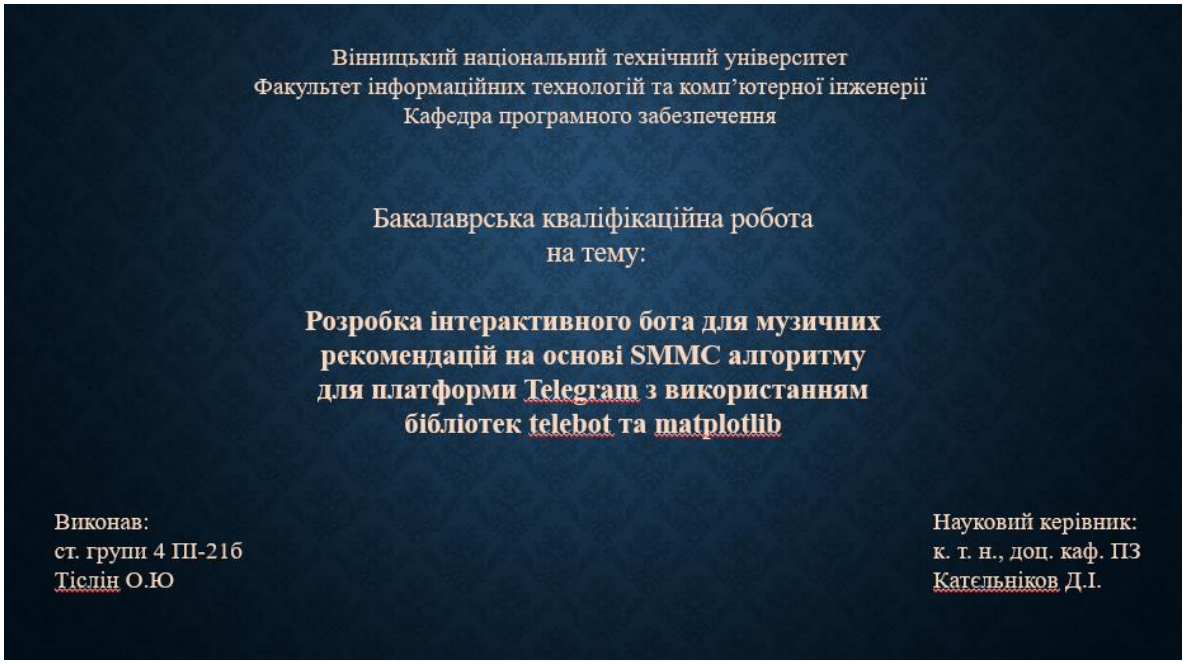

Рисунок Г.1 – Титульний слайд

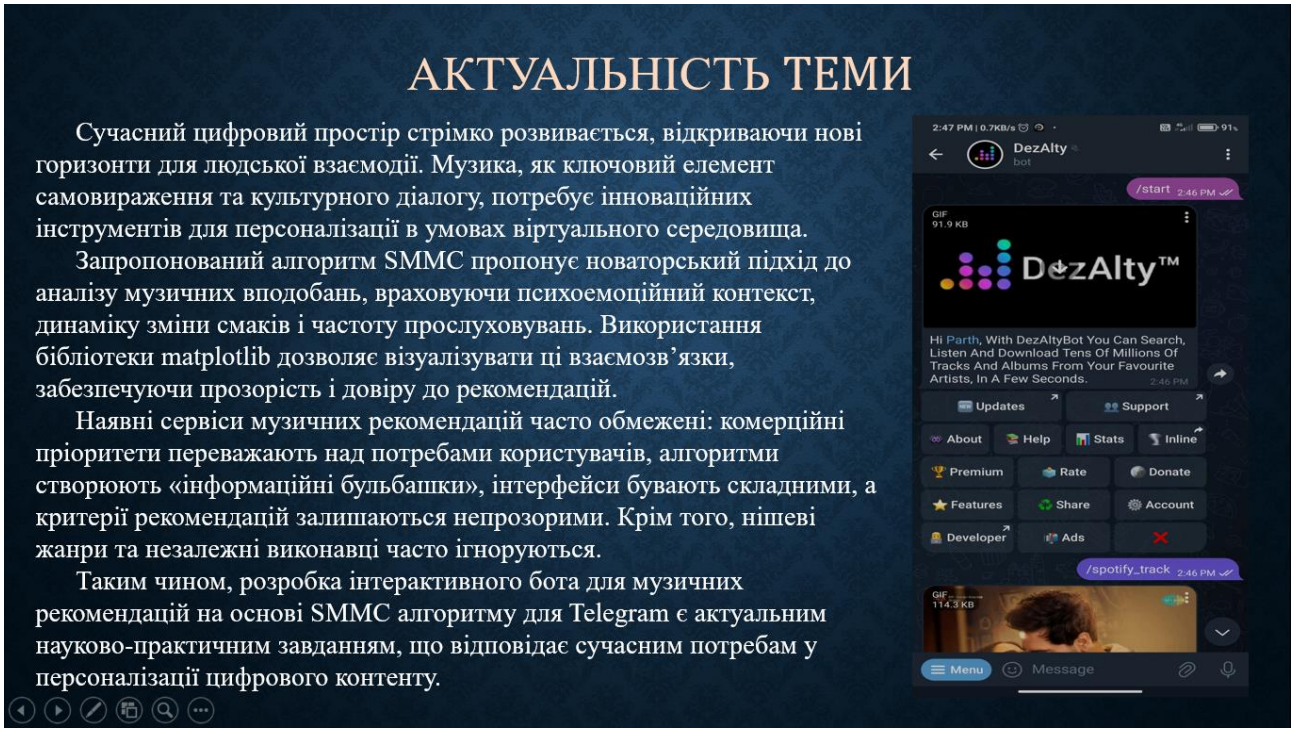

Рисунок Г.2 – Актуальність теми

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

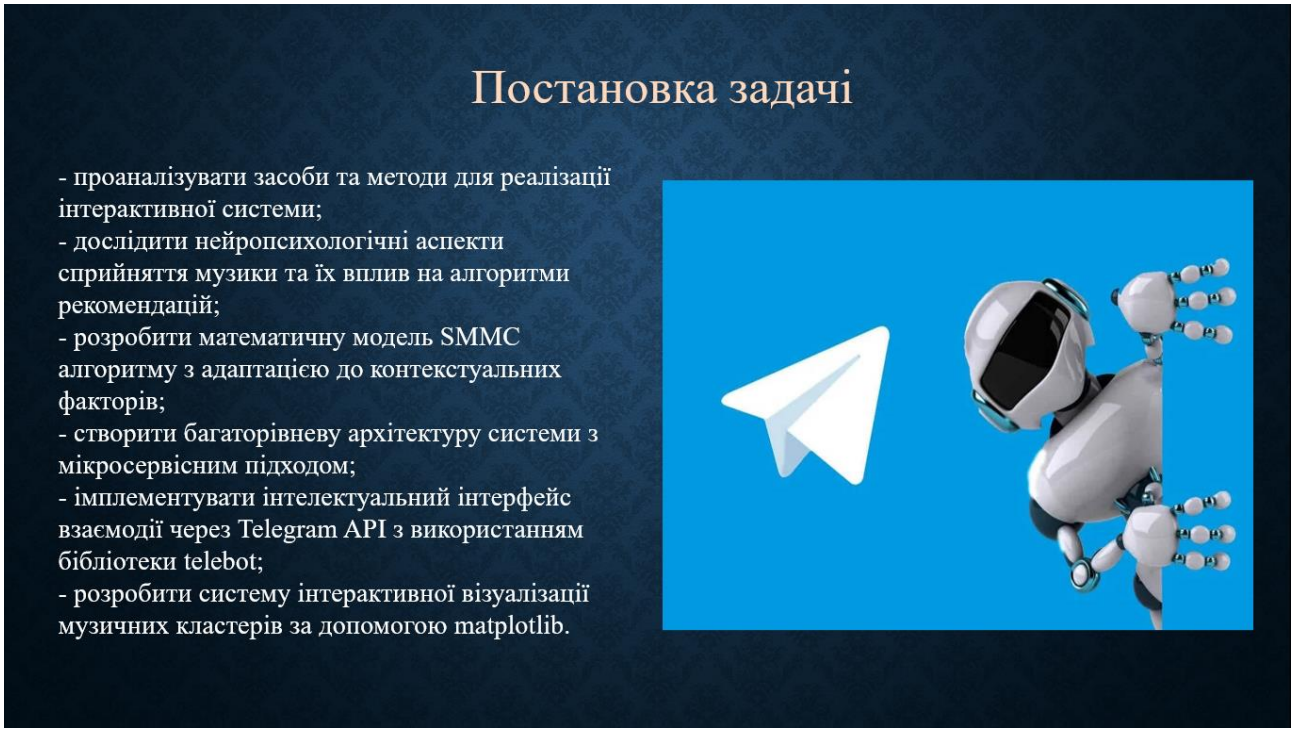

Рисунок Г.4 – Постановка задачі

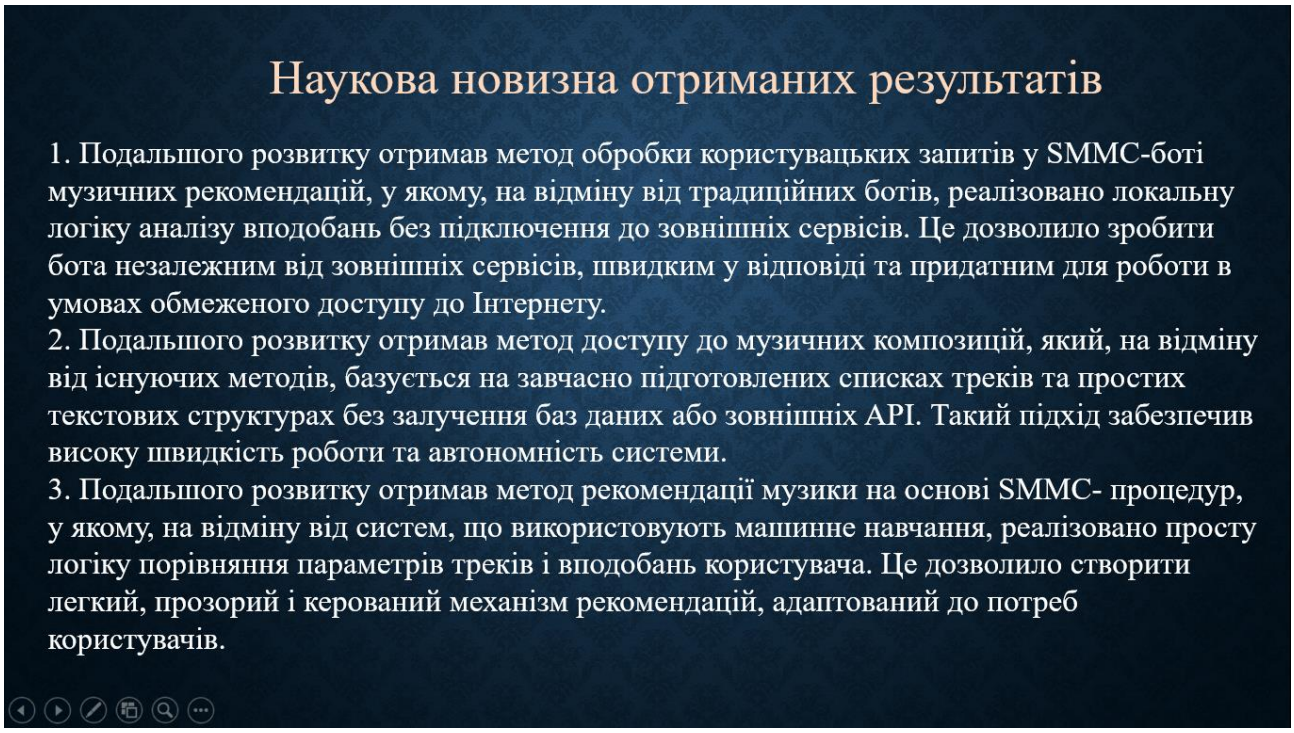

Рисунок Г.5 – Наукова новизна отриманих результатів

Назва функціоналу	Spotybot	MusicTubeBot Bot	TuneMyMusic	FredBoat	YoutubeConverter власна розробка
Сумісність з YouTube та Spotify	1	1	1	1	1
Рекомендації на основі смаків	1	1	1	0	0
Перегляд попередніх треків	0	1	0	1	1
Побудова графічних моделей для рекомендацій	0	0	0	0	1
Функція офлайн-збереження музики	0	0	0	0	1
Сумарний коефіцієнт	2	3	2	2	4

Рисунок Г.6 – Порівняльний аналіз аналогів

Порівняльний аналіз аналогів

У процесі розробки інтерактивного Telegram-бота для рекомендацій на основі алгоритму SMMC було проаналізовано наявні рішення у цій сфері, що дало змогу краще зрозуміти наявні інноваційні тренди. Зокрема, такі застосунки, Spotybot, MusicTubeBot_Bot, TuneMyMusic, FredBoat, продемонстрували ключові функції, які очікують сучасні користувачі: інтеграцію з потоковими сервісами, доступ до історії прослуховувань, побудову персоналізованих порад тощо. Проте жоден із них не об'єднує в собі повного функціонального спектру, включно з побудовою візуалізацій для аналізу рекомендацій або збереженням музики для офлайн-доступу, як це реалізовано у власній розробці YoutubeConverter. Це дозволяє не лише запропонувати нові підходи до рекомендацій, але й створити зручний інтерфейс на базі бібліотеки telebot із використанням matplotlib для графічного представлення результатів.

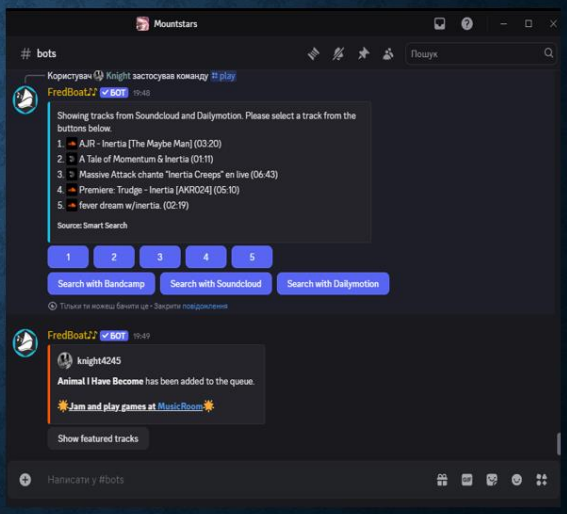

Рисунок Г.7 – Порівняльний аналіз аналогів

Послідовний змішаний ланцюг Маркова

Метод **SMMC (послідовний змішаний ланцюг Маркова)** — це підхід, який використовується для аналізу послідовностей подій. Його основна ідея полягає в тому, щоб передбачати наступний крок у певній послідовності, враховуючи кілька попередніх дій. Наприклад, у Telegram-боті, який рекомендує музику, SMMC може використовуватись для передбачення наступного треку, який може сподобатись користувачу, на основі того, що він слухав раніше.

На відміну від звичайного ланцюга Маркова, де рішення ґрунтується тільки на останньому елементі, SMMC враховує декілька останніх дій користувача. Це дозволяє отримати глибше розуміння його вподобань. Наприклад, якщо користувач прослухав кілька пісень певного жанру, модель врахує всю цю серію, а не тільки останню пісню, щоб підібрати наступну.

Ще однією важливою особливістю SMMC є «змішування» моделей різної глибини. Якщо модель не може знайти підходяще продовження на основі трьох останніх дій, вона може спробувати зробити прогноз, використовуючи лише дві або одну дію. Це робить метод більш гнучким і здатним працювати навіть тоді, коли даних недостатньо.

$$P = \begin{pmatrix} 1/2 & 0 & 1/2 & 0 & 0 \\ 1/4 & 1/2 & 1/4 & 0 & 0 \\ 1/2 & 0 & 1/2 & 0 & 0 \\ 0 & 0 & 0 & 1/2 & 1/2 \\ 0 & 0 & 0 & 1/2 & 1/2 \end{pmatrix}$$

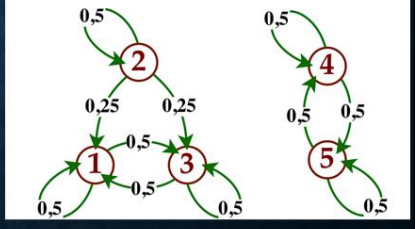

Рисунок Г.8 – Послідовний змішаний ланцюг Маркова

Послідовний змішаний ланцюг Маркова

У методі SMMC основним інструментом для збереження інформації про ймовірності переходів є матриці переходів. Для кожної моделі певного порядку створюється окрема матриця. Наприклад, у моделі першого порядку матриця переходів показує, з якою ймовірністю кожен стан переходить в інший. Така матриця має вигляд таблиці, де рядки відповідають поточним станам (наприклад, трекам), а стовпці можливим наступним станам. Кожен елемент у цій таблиці відображає частоту або ймовірність того, що після одного треку буде прослухано інший.
Щоб здійснити передбачення, модель використовує ці матриці переходів для обчислення вектора ймовірностей наступного стану. Поточна історія користувача інтерпретується як вектор стану, який дозволяє звернутися до відповідного рядка в матриці і дістати ймовірності для всіх можливих наступних дій. Якщо для наявної історії не вдається знайти точного відповідника, модель зменшує порядок (тобто використовує меншу кількість попередніх станів) і пробує ще раз. Таким чином, навіть при нестачі даних SMMC все одно може згенерувати припущення, опираючись на доступну інформацію з моделей нижчого порядку.

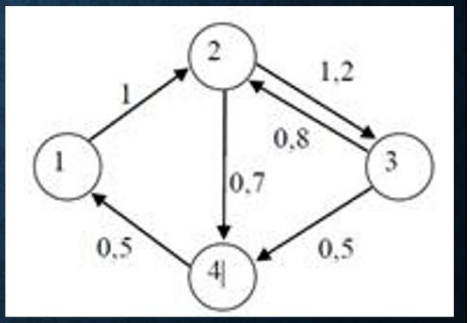

Рисунок Г.9 – Послідовний змішаний ланцюг Маркова

Блок схема роботи алгоритму Markov_ recommendation

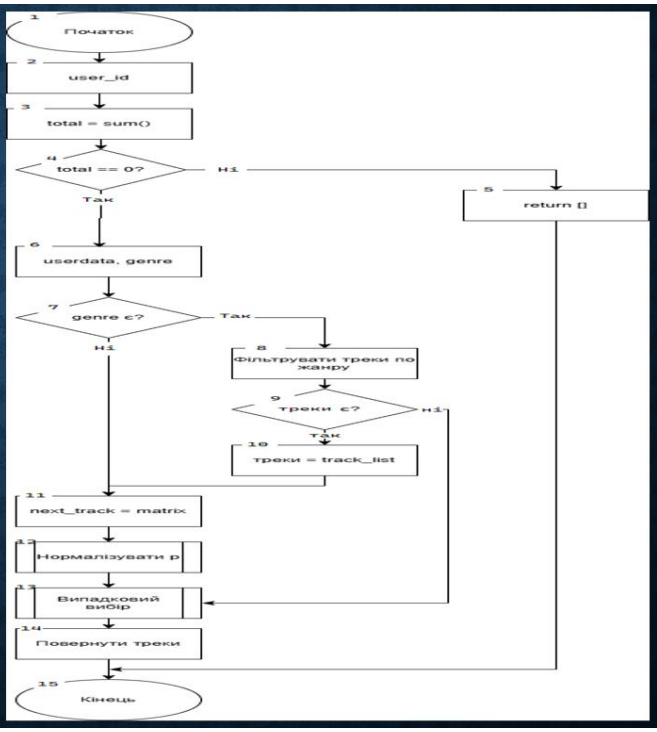

Рисунок Г.10 – Блок схема методу Markov_recommendation

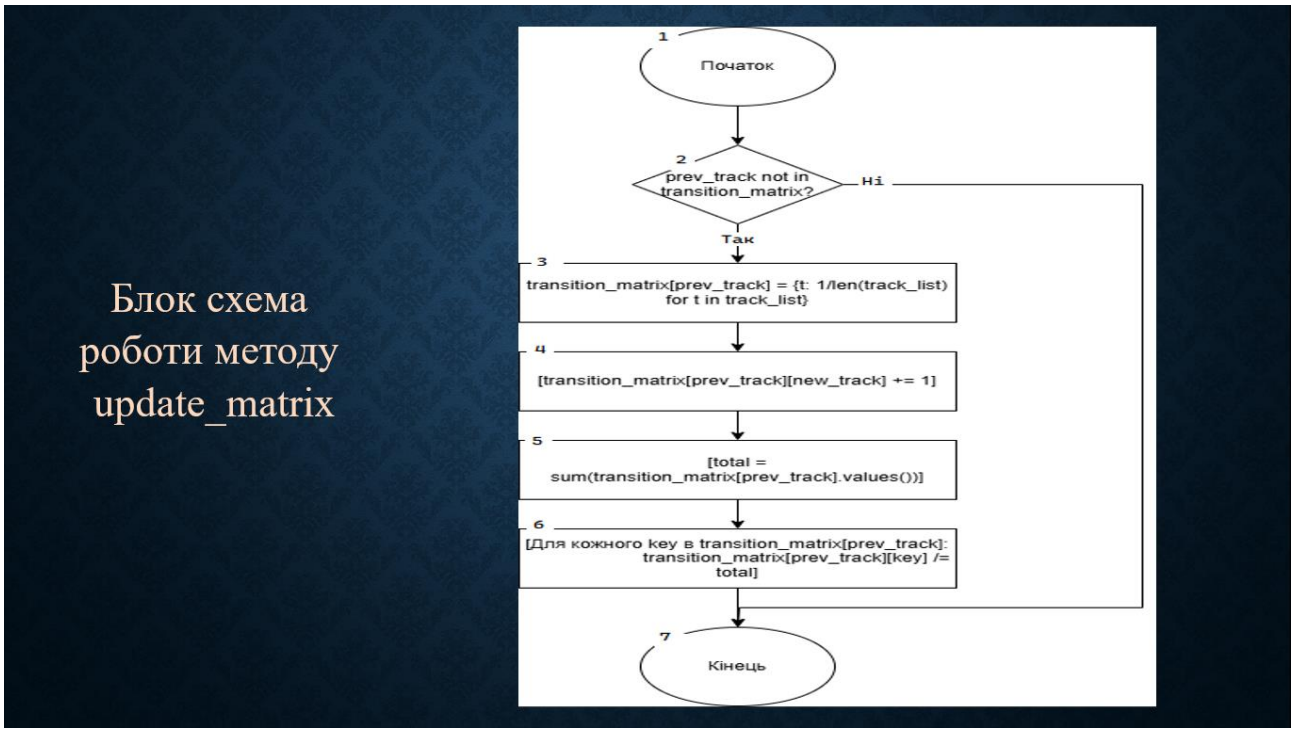

Рисунок Г.11 – Блок схема роботи методу update_matrix

Параметр	Python	JavaScript	Java	C#
Швидка розробка прототипу	1	0	0	0
Високодоступні бібліотеки для обробки даних	1	0	0	0
Проста інтеграція з API месенджерів	1	1	0	0
Висока читабельність коду	1	0	0	1
Висока продуктивність виконання	0	1	1	1
Розширена підтримка спільноти	1	1	1	1
Вбудоване асинхронне програмування	1	1	0	1
Сумарний коефіцієнт	6	4	2	4

Рисунок Г.12 – Обґрунтування вибору мови програмування Python

Функціональність	Visual Studio	PyCharm	VS Code	Jupyter Notebook
Високоякісне автодоповнення Python	1	1	0	0
Комплексне інтегроване налагодження	1	1	0	0
Повна підтримка віртуальних середовищ	1	1	0	0
Вбудоване профілювання коду	1	0	0	0
Повна інтеграція з Git	1	1	0	0
Висока підтримка мультимовних проектів	1	0	0	0
Вбудовані інструменти аналізу даних	0	0	0	1
Висока продуктивність при великих проектах	1	0	0	0
Повна крос-платформна сумісність	0	1	1	1
Сумарний бал	7	5	1	2

Рисунок Г.13 – Обґрунтування вибору середовища Visual Studio

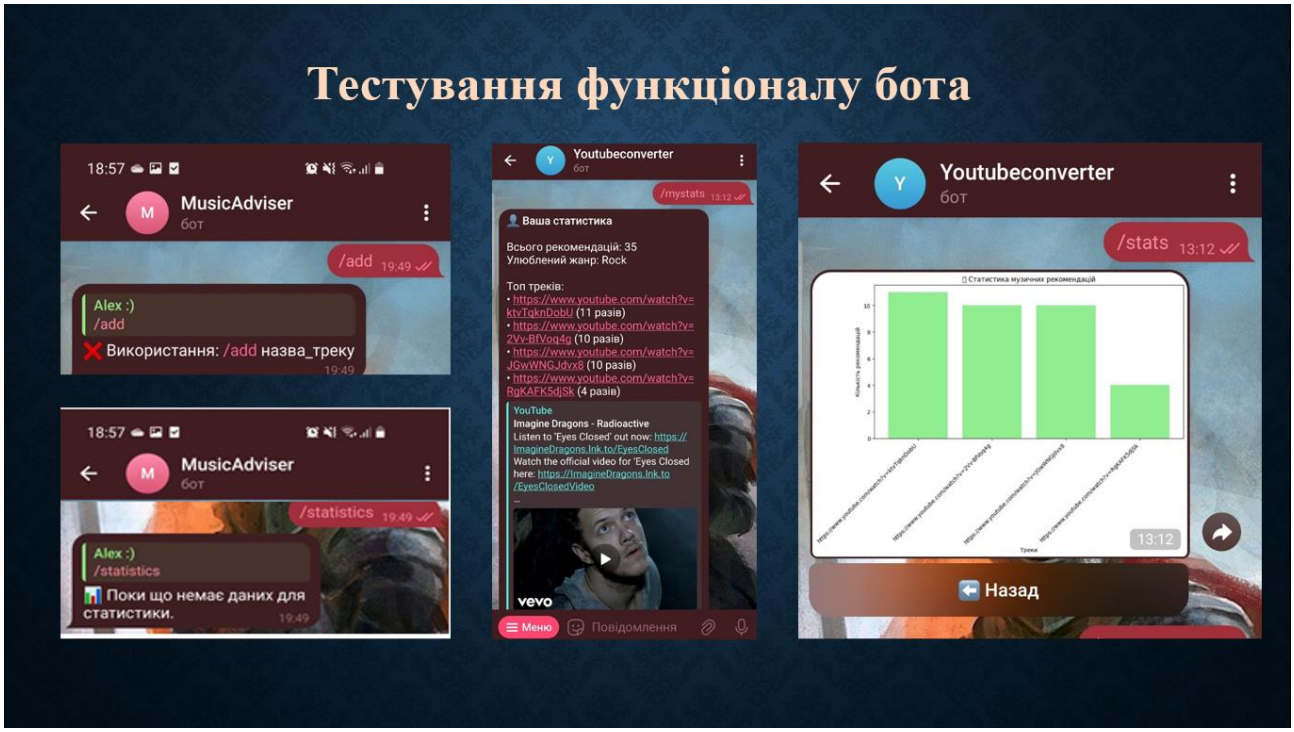

Рисунок Г.14 – Тестування функціоналу бота

Рисунок Г.15 – Апробація матеріалів бакалаврської роботи

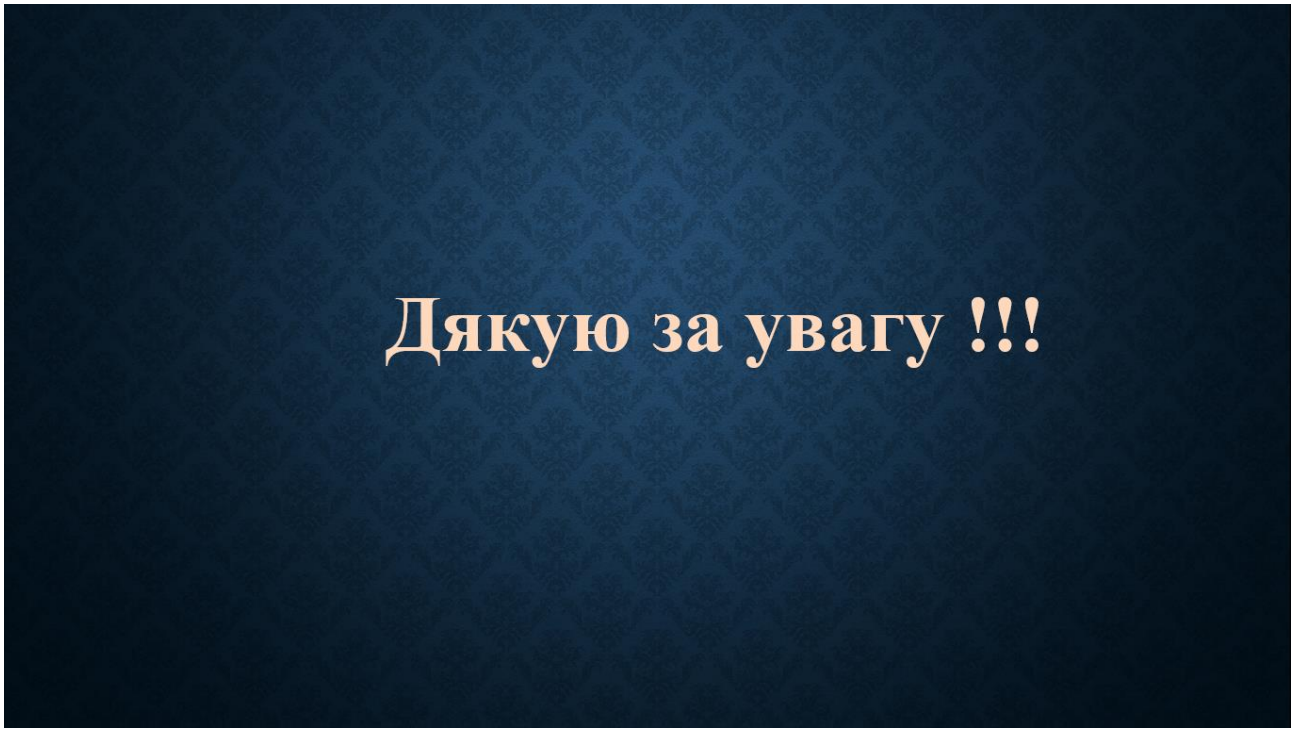

Рисунок Г.16 – Подяка за увагу