Національний лісотехнічний університет України
(повне найменування вищого навчального закладу)

Навчально-науковий інститут комп'ютерних наук
та інформаційних технологій
(повне найменування інституту, назва факультету (відділення))

Кафедра комп'ютерних наук
(повна назва кафедри (предметної, циклової комісії))

Магістерська кваліфікаційна робота

другий (магістерський)
(рівень вищої освіти)

на тему: Інтерактивна інтелектуальна система музичних рекомендацій засобами Java SpringBoot

Виконав: студент 6 курсу групи КН-62м
спеціальності
122 "Комп'ютерні науки"
(шифр і назва напряму підготовки, спеціальності)

Лис Б. Л.
(прізвище та ініціали)

Керівники Габа О. О., Крошний І. М.
(прізвище та ініціали)

Рецензент Часковський О.Г.
(прізвище та ініціали)

Львів – 2025

Національний лісотехнічний університет України
(повне найменування вищого навчального закладу)

ННІ комп'ютерних наук та інформаційних технологій
Кафедра комп'ютерних наук
Рівень вищої освіти другий (магістерський)
Спеціальність 122 "Комп'ютерні науки"
(шифр і назва)

ЗАТВЕРДЖУЮ
Завідувач кафедри КН
_____ Борецька І.Б.
"10" грудня 2025 року

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

Лис Богдан Любомирович
(прізвище, ім'я, по батькові)

1. Тема роботи Інтерактивна інтелектуальна система музичних рекомендацій засобами Java SpringBoot

керівник роботи Габа Олег Орестович асистент кафедри комп'ютерних наук
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)
керівник роботи Крошний Ігор Миколайович кандидат технічних наук, доцент
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)
затверджені наказом вищого навчального закладу від "29" 04 2025 *року* *№ С-288*

2. Терміни подання студентом роботи 10.12.25 р.
3. Вихідні дані до роботи музичні метадані зі зовнішніх API; природномовний запит користувача; обсяг вихідної рекомендації: 5–7 треків;
4. Зміст пояснювальної записки (перелік питань, які потрібно розробити) провести аналіз сучасних музичних рекомендаційних систем та алгоритми формування рекомендацій; вибрати та обґрунтувати методи для реалізації; вибрати засоби розробки; визначити архітектуру ПЗ; розробити ПЗ для представлення результатів роботи;
5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)
Лист 1 – Архітектура інтелектуальної системи. Структурна схема.
Лист 2 – Функціонування системи. Схема алгоритму.
Лист 3 – Схема сховища даних. Структурна схема.
6. Дата видачі завдання 01.05.25 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломної роботи	Строк Виконання етапів роботи	Примітка
1	Аналіз предметної області та формування вимог до системи музичних рекомендацій	01.05.25 - 25.05.25	Виконано
2	Розробка архітектури системи та вибір технологічного стеку	26.05.25 - 22.06.25	Виконано
3	Інтеграція із зовнішніми API	23.06.25 - 07.08.25	Виконано
4	Розробка, оформлення та погодження графічної частини	03.11.25 - 23.11.25	Виконано
5	Оформлення та погодження пояснювальної записки	24.11.25 - 1.12.25	Виконано

Студент _____ Лис Б. Л.
(підпис) (прізвище та ініціали)

Керівник роботи _____ Габа О. О.,
(підпис) (прізвище та ініціали)

Керівник роботи _____ Крошний І. М.
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Магістерська робота містить 62 сторінки пояснювальної записки, 7 рисунків, 2 таблиць, 6 додаток, 15 джерел.

У роботі розглядається процес проєктування та розробки інтелектуальної системи музичних рекомендацій з можливістю інтерактивної взаємодії користувача в режимі природної мови. Система ґрунтується на поєднанні сучасних мовних моделей штучного інтелекту (OpenAI API) з музичним API-платформами (Spotify API) та реалізується на базі Java Spring Boot.

Ключові слова: система рекомендацій, інтерактивна взаємодія, OpenAI API, Spotify API, обробка природної мови, персоналізація, музичний контент, Spring Boot, REST API, аудіоаналіз, база даних PostgreSQL, Service-DAO-Core архітектура.

ABSTRACT

The thesis contains 62 pages of explanatory note, 7 figures, 2 tables, 6 appendix, 15 used literary sources.

This thesis explores the design and development process of an intelligent music recommendation system with support for interactive user communication in natural language. The system is based on the integration of modern artificial intelligence language models (OpenAI API) with music API platforms (Spotify API) and is implemented using the Java Spring Boot framework.

Keywords: recommendation system, interactive interaction, OpenAI API, Spotify API, natural language processing, personalization, music content, Spring Boot, REST API, audio analysis, PostgreSQL database.

ТЕХНІЧНЕ ЗАВДАННЯ

У межах магістерської кваліфікаційної роботи буде розроблено інтелектуальну систему музичних рекомендацій із можливістю інтерактивної взаємодії користувача, яка ґрунтується на використанні сучасних технологій.

Основні завдання, що повинні бути вирішені:

1. Аналіз предметної області та формування вимог до системи музичних рекомендацій.
   - Провести аналіз сучасних музичних рекомендаційних систем.
   - Визначити основні алгоритми.
   - Сформулювати основні вимоги до системи.
2. Розробка архітектури системи та вибір технологічного стеку
   - Розробити загальну архітектуру системи.
   - Побудувати UML-діаграми: компонентів, послідовностей, взаємодії.
   - Обґрунтувати вибір основних технологій:
3. Інтеграція із зовнішніми API
   - Провести огляд можливостей музичних API-платформ.
   - Дослідити OpenAI API з точки зору:
     - обробки запитів користувача, сформульованих у природній мові;
     - можливостей формування діалогів і сценаріїв взаємодії;
4. Висновки
   - Аналіз відповідності розглянутих компонентів початковим технічним вимогам.
   - Оцінка технічної життєздатності системи.
   - Визначення напрямів подальшої розробки.

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ ТА УМОВНИХ ПОЗНАЧЕНЬ

API - Application Programming Interface

CDN - Content Delivery Network

CBRS - Content-based recommendation system

CFRS - Collaborative filtering recommendation systems

DAO - Data Access Layer

LLM - Large Language Model

SQL - Structured Query Language

ШІ - Штучний інтелект

ВСТУП

Розвиток інформаційних технологій та стрімке зростання обсягів цифрових даних стимулюють активне впровадження інтелектуальних систем у різноманітні сфери людської діяльності, серед яких особливе місце займає галузь мультимедійних сервісів. Музичний контент, що представлений на глобальних платформах, таких як Spotify, Apple Music, YouTube Music, Amazon Music, стає дедалі доступнішим, однак водночас ускладнюється задача користувача щодо пошуку музики, яка відповідає його вподобанням, настрою або конкретній ситуації. У зв'язку з цим набувають все більшого значення системи рекомендацій, що здатні допомогти користувачу орієнтуватися в безмежному обсязі музичного контенту.

Сучасні системи музичних рекомендацій базуються на різноманітних підходах, серед яких найбільш поширеними є методи колаборативної та контентної фільтрації, гібридні моделі, а також алгоритми, що використовують машинне навчання та штучний інтелект. Однак більшість існуючих рішень обмежені у можливостях персоналізації, не враховують складних багатофакторних запитів користувачів, а також часто не дозволяють інтерактивно впливати на результат рекомендацій. Саме тому розробка системи, яка б дозволяла формулювати запити у природній мові, отримувати музичні рекомендації, а також взаємодіяти з системою у режимі реального часу, є актуальним і важливим завданням.

Об'єктом дослідження у роботі є процес формування музичних рекомендацій на основі текстових запитів користувача.

Предметом дослідження є методи інтеграції мовних моделей штучного інтелекту (на прикладі OpenAI API) та музичних API (на прикладі Spotify API) у процес створення персоналізованих рекомендацій.

Метою даної магістерської кваліфікаційної роботи є розробка інтелектуальної системи музичних рекомендацій із можливістю інтерактивної взаємодії користувача, що забезпечить генерацію персоналізованих плейлистів на основі текстових запитів користувача, використовуючи інтерфейси OpenAI API для обробки запитів природною мовою, Spotify API для отримання даних про музичний контент та Java,

Spring Boot як основу програмної реалізації. Збереження даних, а також управління користувацькою інформацією, планується реалізувати за допомогою системи керування базами даних PostgreSQL.

Наукова новизна роботи полягає у розробці підходу до формування музичних рекомендацій на основі текстових запитів користувачів, що обробляються за допомогою великої мовної моделі OpenAI.

На відміну від традиційних систем рекомендацій, де рекомендації ґрунтуються лише на історії прослуховувань або характеристиках треків, запропонована система дозволяє формулювати запити у вільній формі (наприклад, “підібери музику для тренування” або “заспокійливий плейлист для вечірнього відпочинку”) та отримувати персоналізовані результати, що відповідають зазначеним критеріям.

Крім того, у роботі досліджено інтеграцію зовнішніх API для поєднання мовних моделей штучного інтелекту з даними про музичний контент, що відкриває нові можливості у побудові гібридних рекомендаційних систем.

Практична значимість роботи полягає у створенні прототипу інтелектуальної системи музичних рекомендацій, яка може бути використана як основа для розробки комерційних або навчальних продуктів у сфері музичних сервісів. Розроблена система демонструє можливість інтеграції штучного інтелекту у завдання формування рекомендацій на основі запитів природною мовою, що сприяє покращенню користувацького досвіду та підвищенню ефективності підбору музики.

Результати дослідження можуть бути застосовані для побудови подібних систем у суміжних сферах, таких як рекомендації фільмів, книг, навчальних матеріалів тощо.

РОЗДІЛ 1. СТАН ПРОБЛЕМНОЇ ОБЛАСТІ

У сучасних умовах стрімкого розвитку інформаційних технологій та цифрових сервісів формування персоналізованих рекомендацій стає одним із ключових напрямів у розробці інтелектуальних систем. Особливе значення такі системи набувають у сфері мультимедійного контенту, зокрема музики, де користувачі стикаються з проблемою надмірної інформації (phenomenon of information overload) та обмеженої здатності самостійно обирати контент, який відповідає їхнім індивідуальним уподобанням, настрою чи поточній ситуації.

Системи музичних рекомендацій покликані вирішувати цю проблему, однак більшість наявних рішень базуються на класичних підходах, таких як колаборативна фільтрація або контент-орієнтовані алгоритми, які мають свої обмеження та не завжди здатні враховувати складні багатофакторні запити користувачів. Окрім того, переважна більшість таких систем не забезпечує інтерактивної взаємодії користувача із системою у режимі природномовного спілкування, що обмежує їхню функціональність та знижує зручність використання.

Водночас, останні досягнення у сфері штучного інтелекту, зокрема мовних моделей великого масштабу (LLM), відкривають нові можливості для розвитку систем рекомендацій, дозволяючи інтерпретувати запити користувачів у природній мові та формувати релевантні результати на основі глибинного аналізу контексту.

У даному розділі проведено огляд сучасного стану проблемної області систем музичних рекомендацій, класифікації існуючих підходів, розглянуто приклади реалізації таких систем, а також проаналізовано сучасні інструменти та API, що використовуються для інтеграції мовних моделей та музичних сервісів. Окрему увагу приділено визначенню проблем, що залишаються невирішеними у даній сфері, та обґрунтуванню необхідності розробки нових рішень.

1.1 Аналітичний огляд наявних реалізацій в сфері музичних рекомендаційних систем

У цьому підрозділі буде проведено порівняльний аналіз архітектурних рішень та алгоритмів персоналізації найбільших музичних стримінгових сервісів, таких як Spotify, YouTube Music, Apple Music, Deezer, Pandora та SoundCloud. Розглянуто сильні та слабкі сторони кожного підходу, а також окреслено потенційні шляхи покращення систем рекомендацій за допомогою інтеграції діалогових моделей, зокрема OpenAI API.

Spotify

Архітектура Spotify побудована на мікросервісній архітектурі, з окремими сервісами для користувачів, плейлистів, стримінгу та рекомендацій. В системі використовуються Apache Kafka для потокової обробки подій та Cassandra для розподіленого зберігання даних. Spotify спочатку був на AWS, а тепер працює на Google Cloud із Kubernetes для керування контейнерами і масштабуванням мікросервісів. Музика доставляється через CDN для низької затримки [1].

Система рекомендацій Spotify гібридна: вона комбінує колаборативну фільтрацію і аналіз контенту. Зокрема, колаборативні моделі будуються на спільних списках відтворення та уподобаннях, а контент-орієнтовані моделі аналізують аудіо і метадані. Обробка натуральної мови використовується для категоризації подкастів і текстових описів. Spotify регулярно тренує машинні моделі, щоб формувати “Discover Weekly”, “Daily Mix” та персональні станції, адаптуючись до смаку кожного користувача [2].

Переваги даної системи:

- Система забезпечує дуже персоналізовані плейлисти (Discover Weekly, Daily Mix) завдяки складному ML-підходу і аналізу поведінки.
- Висока масштабованість архітектури гарантує надійність і швидкість роботи сервісу.

Недоліки даної системи:

- Сучасні рекомендації в Spotify часто статичні – вони не передбачають інтерактивної участі користувача. Це може викликати ефект “фільтрування бульбашки”, коли система постійно пропонує схожий контент.
- Також складно пояснити користувачу, чому саме ці пісні рекомендовано.

YouTube Music

YouTube Music працює на інфраструктурі Google Cloud, збереженням даних у масштабованих NoSQL та використанням TensorFlow/PyTorch для моделювання. Інформація про історію переглядів і взаємодії з музикою зберігається у сховищах. Система виконує три етапи: пошук кандидатів, ранжування та фільтрацію результатів. Для кандидатів застосовуються різні підходи: пошук по вбудованих векторних представленнях та обхід графів взаємодій [3].

Алгоритми YouTube Music – це потужні нейромережні моделі, орієнтовані на послідовність дій користувача. Google використав Transformer-архітектуру для ранжування пісень, що аналізує послідовність уподобань, пропусків, лайків з урахуванням контексту (час, місце, активність). Ранжування комбінується з традиційними факторами (релевантність, новизна, популярність). Система ретрівує тисячі кандидатів, потім оцінює їх за допомогою нейромережі, і нарешті відфільтровує дублікати або надто схожі треки. Цей конвеєр (retrieval → ranking → filtering) – стандартний для великих систем рекомендацій [3].

Переваги даної системи:

- YouTube Music має величезний каталог та інтегровані дані YouTube Його моделі глибоко оптимізовані на взаємодію, що дає дуже релевантні рекомендації. Завдяки Google-технологіям, система легко масштабується і швидко відгукується на запити.

Недоліки даної системи:

- Виокремленим недоліком є фокус на максимізації статистик без пояснення користувачу причин рекомендацій. Приватні алгоритми також можуть формувати “ехо-камеру” трендів і популярних треків.
- Статичний підхід погано враховує зміну контексту.

Apple Music

Apple Music побудовано на закритій інфраструктурі Apple із використанням мереж доставки контенту. На відміну від iCloud, для стримінгу музики Apple використовує зовнішні CDN-партнери для глобальної доставки треків. При цьому бізнес-логіка обробляється на власних серверах Apple. Система має публічне API для розробників, але загалом сервіс є закритим екосистемним [4].

Apple Music поєднує людську кураторську роботу та алгоритмічну персоналізацію. Команда редакторів формує плейлисти і підкасти, а “For You” персоналізує музичні рекомендації на основі обробки звичок користувача. Зокрема, алгоритм враховує, як часто слухають треки, скільки разів пропускають та повторюють пісні – на це звертається увага як на сигнал зворотного зв’язку. Така система динамічно оновлює рекомендації залежно від поведінки слухача [5].

Переваги даної системи:

- Сильними сторонами Apple Music є тісна інтеграція з екосистемою Apple (Siri, HomePod), висока якість офіційних плейлистів і радіостанцій, а також великий каталог ексклюзивного контенту.
- Комбінація редакційного підходу та машинного навчання дає часто дуже влучні рекомендації, особливо для популярних жанрів.

Недоліки даної системи:

- Відсутність відкритого діалогу з користувачем, здебільшого адаптує рекомендації за загальними показниками, але мало пояснює, чому пісні запропоновано.

1.2 Основні підходи до побудови систем рекомендацій

Системи рекомендацій — це програмні інструменти, що забезпечують користувачам персоналізовані пропозиції на основі їхніх уподобань, історії взаємодій або характеристик об’єктів. В основі побудови таких систем лежать різні підходи, кожен з яких має свої особливості, переваги та обмеження. Нижче буде розглянуто основні методи побудови даних систем.

Колаборативна фільтрація

Колаборативна фільтрація – це один з двох основних типів систем рекомендації. CF ґрунтується на припущенні, що користувачі, які мали схожі вподобання в минулому, ймовірно, матимуть схожі вподобання і в майбутньому. Система використовує історію взаємодій користувачів (оцінки, перегляди, лайки тощо) для прогнозування їхніх вподобань [6].

Колаборативна фільтрація використовує матрицю для відображення поведінки користувача для кожного елемента у своїй системі. Потім система витягує значення з цієї матриці, щоб відобразити їх у вигляді точок даних у векторному просторі. Різні метрики потім вимірюють відстань між точками як засіб розрахунку подібності між користувачами та між елементами [6].

У стандартному режимі колаборативної фільтрації ми маємо набір з n користувачів та набір з x елементів. Індивідуальні уподобання кожного користувача для кожного елемента відображаються в матриці користувач-елемент таблиця 1.1. (іноді її називають матрицею оцінок користувачів). Тут користувачі представлені рядками, а елементи – стовпцями. Ці значення можуть бути безперервними числами, наданими користувачами (наприклад, оцінки), або двійковими значеннями, які означають, чи переглянув чи придбав певний користувач елемент.

Таблиця 1.1 – Матриця оцінок користувачів

	Елемент 1	Елемент 2	Елемент 3	Елемент 4
Користувач 1	5	4	3	4
Користувач 2	1	2	4	5
Користувач 3	1	2	3	null
Користувач 4	null	4	3	1

Ця матриця відображає оцінки користувачів для різних доступних елементів. Алгоритм CF порівнює оцінки, надані користувачами для кожного елементу. Визначаючи схожих користувачів або елементи на основі цих оцінок, він прогнозує елементи, які цільовий користувач не бачив (позначені як null у матриці) та рекомендує (або не рекомендує) ці елементи цільовому користувачеві відповідно.

Приклад матриці, що використовується тут, є повним, оскільки він обмежений чотирма користувачами та чотирма елементами. Однак у реальних сценаріях відомі вподобання користувачів щодо елементів часто обмежені, що робить матрицю «користувач-елемент» розрідженою.

Переваги:

- Не потребує додаткових метаданих про об’єкти.
- Враховує приховані патерни вподобань користувачів.

Обмеження:

- Проблема холодного старту, це коли користувачі або нові об’єкти не можуть бути рекомендовані, якщо про них недостатньо даних.
- Із ростом кількості користувачів і об’єктів обчислювальні витрати зростають.
- Рідковживані об’єкти мають менший шанс бути рекомендованими.

Контентно-орієнтовані рекомендації

Фільтрація на основі контенту це другий тип рекомендаційних систем. Рекомендації формуються на основі схожості між об’єктами, визначеної їхніми характеристиками (жанр, виконавець, рік випуску, інструментальність тощо). Наприклад, якщо користувач слухає рок-музику, йому рекомендують інші пісні цього жанру [7].

Переваги:

- Може працювати навіть із невеликою кількістю користувачів
- Добре працює для нішевих або менш популярних об’єктів.
- Забезпечує прозорість, зрозуміло, чому саме обрана рекомендація.

Обмеження:

- Вузькість рекомендацій, система рекомендує об’єкти, схожі на вже переглянуті, що знижує різноманіття.
- Якість залежить від повноти та якості опису об’єктів.

Гібридні системи рекомендацій

Ця система використовує комбінацію CBRS та систем рекомендацій на основі CFRS для досягнення точної продуктивності шляхом зменшення недоліків традиційних методів рекомендацій. Додаткова інформація користувача, коментарі, оцінки, зв'язки, атрибути та відгуки використовуються разом із матрицею оцінювання в гібридній системі рекомендацій для підвищення продуктивності та точності. Наприклад, система може використовувати контентну фільтрацію для нових користувачів, а колаборативну — для досвідчених.

Переваги:

- Компенсують слабкі сторони окремих методів.
- Покращують якість рекомендацій у різних сценаріях (наприклад, при холодному старті).

Обмеження:

- Вища складність реалізації та підтримки.
- Потреба у великих обсягах даних для різних моделей.
- Може бути складно подати результати рекомендацій.

Алгоритми на основі моделей машинного навчання

Системи використовують складні моделі машинного навчання, такі як факторизація матриць (Matrix Factorization, SVD), нейронні мережі, графові нейронні мережі, моделі на основі послідовностей (RNN, Transformer) для побудови прогнозів [8].

Переваги:

- Можуть враховувати складні залежності та контексти.
- Підвищують точність прогнозів завдяки обробці великих обсягів даних.

Обмеження:

- Високі обчислювальні витрати.
- Потреба у великих і якісних наборах даних.
- Часто недостатня прозорість.

1.3 Проблеми та обмеження сучасних систем музичних рекомендацій

Сучасні платформи, такі як Spotify, YouTube Music та Apple Music, є лідерами на ринку музичних сервісів, але навіть вони мають низку обмежень, які негативно впливають на персоналізацію і задоволеність користувачів. Аналіз існуючих підходів виявив такі основні проблеми:

Обмежені можливості для інтерактивної взаємодії

Більшість сучасних систем рекомендацій працюють у форматі одностороннього потоку рекомендацій: користувач отримує список треків, але не має змоги уточнити свої вподобання, запитати пояснення або вести діалог із системою.

Spotify та YouTube Music, наприклад, не дозволяють уточнювати запити на кшталт:

– “Я хочу музику для тренування, але щоб була спокійна”,

– “Покажи щось схоже на цей трек, але більше інструментальної музики”.

Apple Music також працює за закритими алгоритмами, без пояснень вибору треків або можливості уточнити настрій і жанр.

Запропонована інтерактивна система на базі OpenAI API усуває ці обмеження. Вона дозволяє користувачеві вести діалог із системою: уточнювати жанри, настрій, стиль, отримувати пояснення до рекомендацій та формулювати запити у природній мові (наприклад, “Підбери мені музику для вечірки в стилі 80-х” або “Хочу щось на зразок цієї пісні, але більш енергійне”). Це розширює можливості персоналізації та підвищує рівень задоволеності користувачів.

Відсутність прозорості

Сучасні системи працюють як “чорні ящики” – користувач не знає, чому йому рекомендують певний трек. Це ускладнює розуміння рекомендацій.

Apple Music та Spotify не пояснюють логіку вибору пісень, що часто призводить до невдоволення користувача, який хоче розуміти, на основі чого йому пропонуються ті чи інші треки.

Проблема вузьких рекомендацій

Сучасні системи часто зациклюються на вподобаннях користувача: користувач постійно отримує музику, подібну до того, що вже слухав, і втрачає можливість відкривати новий контент.

Spotify і YouTube Music схильні рекомендувати популярні треки або ті, що вже мають високу взаємодію, обмежуючи різноманітність.

Інтерактивна система OpenAI дає змогу розширити горизонти: користувач може прямо запитати про щось нове (“Порекомендуй щось абсолютно інше, ніж я слухаю зазвичай”) або проекспериментувати з жанрами. Це допомагає долати обмеження звичайних алгоритмів і сприяє відкриттю нового музичного контенту.

Висновки до розділу 1

У результаті проведеного аналізу сучасних музичних рекомендаційних систем було встановлено, що провідні стримінгові сервіси — такі як Spotify, YouTube Music та Apple Music — активно використовують потужні алгоритмічні моделі, що поєднують підходи колаборативної фільтрації, контентного аналізу, гібридних систем та методів машинного навчання. Кожна з платформ має власні архітектурні особливості, реалізуючи стратегії рекомендацій, що забезпечують високу точність підбору музичного контенту відповідно до історії прослуховувань користувача, його вподобань та глобальних трендів.

Разом із тим, проведене дослідження виявило низку суттєвих обмежень і недоліків сучасних систем. Зокрема, спостерігається обмежена можливість пояснення рекомендацій для користувача, що створює ефект “чорної скриньки” й знижує довіру до системи. Більшість сервісів не передбачає інтерактивної взаємодії через природну мову: користувачі позбавлені можливості уточнювати запити або вести діалог для кращої персоналізації. Окрім того, обмеження гнучкості та недостатня адаптація до контексту (час, настрій, активність) знижують ефективність рекомендацій. Вузькість пропозицій та обмежений доступ до пояснень підсилюють проблему одноманітності музичного досвіду.

Аналіз основних підходів до побудови систем рекомендацій показав, що колаборативна фільтрація добре виявляє приховані закономірності, контентний підхід дозволяє враховувати характеристики об’єктів, а гібридні моделі поєднують переваги обох. Проте, усі ці підходи мають обмеження, зокрема проблему холодного старту, обчислювальні витрати та складність інтерпретації результатів. Це створює потребу у розвитку систем нового покоління, що поєднують класичні рекомендаційні алгоритми з сучасними діалоговими моделями, штучним інтелектом та гнучкими архітектурами.

Запропонована інтерактивна система музичних рекомендацій на базі OpenAI API є перспективним напрямом розвитку цієї галузі. Вона дозволяє усунути виявлені обмеження завдяки можливості ведення природної мовної взаємодії, уточнення запитів у режимі реального часу, пояснення рекомендацій, адаптації під конкретний контекст та розширення музичного досвіду за межами стандартних алгоритмів. Такий підхід формує нову парадигму користувацького досвіду — більш прозору, гнучку, персоналізовану та відкриту для творчого діалогу. Він створює фундамент для розробки інноваційних музичних сервісів, що орієнтовані на індивідуальні потреби користувачів та підвищення їхньої задоволеності.

РОЗДІЛ 2. ІНФОРМАЦІЙНЕ ЗАБЕЗПЕЧЕННЯ

Інформаційне забезпечення є ключовим елементом будь-якої інтелектуальної системи, оскільки воно визначає сукупність даних, структур, джерел інформації та процесів, необхідних для коректного функціонування програмного забезпечення. У контексті системи музичних рекомендацій, інформаційне забезпечення охоплює різноманітні аспекти: від формування та структурування бази музичних даних (треки, альбоми, жанри, характеристики композицій) до забезпечення інтеграції з зовнішніми сервісами (Spotify API) та використання моделей штучного інтелекту (OpenAI API) для генерації рекомендацій і діалогів.

У цьому розділі буде розглянуто концепцію інформаційного забезпечення розроблюваної системи музичних рекомендацій, визначено склад і структуру інформаційних потоків, описано формати даних, джерела отримання інформації, механізми інтеграції зовнішніх API, а також вимоги до побудови бази даних. Особлива увага буде приділена забезпеченню цілісності та узгодженості даних, їхньої актуалізації та ефективному використанню для забезпечення високої якості рекомендаційного процесу.

2.1 Характеристика вхідних та вихідних даних системи

Інтелектуальна система музичних рекомендацій базується на обробці великої кількості різнорідних даних. Ефективність роботи системи безпосередньо залежить від якості, структури та актуальності інформаційних потоків, що формуються у процесі взаємодії користувача із системою, зовнішніх сервісів та внутрішньої бази даних.

Вхідні дані системи

Вхідні дані — це набір інформації, що надходить у систему з різних джерел, включаючи користувача, зовнішні API (Spotify, OpenAI) та внутрішні процеси. Вхідні дані можна умовно класифікувати на кілька категорій:

Дані від користувача:

- Промт (запит, інструкція або запитання)

- Уточнення вподобань під час діалогу.
- Оцінки та реакції на рекомендації.

Дані з зовнішніх сервісів (Spotify API):

- Мета-інформація про треки: унікальний ідентифікатор (ID), назва, виконавець, альбом, жанр, дата релізу, тривалість, популярність, енергійність, танцювальність, темп (BPM).
- Дані про артистів: ім’я, жанри, популярність, опис.
- Каталоги плейлистів, альбомів і жанрів: тематичні добірки, трендові списки.
- Історія прослуховувань користувача: доступна через Spotify API для авторизованих користувачів.

Дані з OpenAI API:

- Результати обробки запитів користувача, наприклад, трансформація текстового запиту користувача у структуровану форму (жанр, настрій, активність).
- Генерація уточнюючих питань для збирання додаткової інформації про запит користувача.
- Пояснення рекомендацій, формування текстових пояснень, чому певні треки рекомендовані.

Дані внутрішньої бази даних (PostgreSQL):

- Історія взаємодій, журнал запитів, відповіді системи, тимчасові сесії діалогу.
- Налаштування користувачів, збережені вподобання, обрані жанри, історія взаємодій.
- Кешовані результати запитів, для прискорення відповідей системи.

Вихідні дані системи

Вихідні дані — це результати роботи системи, які формуються на основі обробки вхідних даних. Основні типи вихідних даних включають:

Рекомендації музичних треків:

- Список треків, що відповідають запиту користувача.

- Для кожного треку формується назва, виконавець, альбом, обкладинка, посилання на прослуховування (Spotify URL), жанр, основні характеристики.

Пояснення до рекомендацій:

- Короткі текстові коментарі, що описують причини вибору: “Ця пісня рекомендована, оскільки вона має схожий темп і атмосферу, як ваш попередній запит”.

Інтерактивні відповіді системи (діалоговий режим):

- Запит уточнень: “Ви шукаєте щось для тренування чи відпочинку?”
- Рекомендації на основі уточнень.

Відповіді на промти:

- Наприклад, “Я знайшов 5 пісень у жанрі джаз, які можуть вам сподобатися”.

Оновлені дані для інтерфейсу користувача:

- Динамічні списки рекомендацій, адаптовані під контекст.
- Графічні елементи, обкладинки альбомів, індикатори популярності.

Формати даних

JSON: основний формат обміну даними між сервісами (API OpenAI, Spotify, клієнтська частина).

SQL-структури: таблиці у PostgreSQL для збереження даних про користувачів, треки, історію сесій.

Текстові повідомлення: відповіді від OpenAI API у вигляді тексту для діалогової взаємодії.

2.2 Джерела даних та їх інтеграція

Ефективна робота інтелектуальної системи музичних рекомендацій значною мірою залежить від якісного інформаційного забезпечення рисунку 2.1 Основними джерелами даних для розроблюваної системи є зовнішні API-сервіси (зокрема, Spotify API та OpenAI API), дані користувачів, а також внутрішня база даних, що накопичує інформацію про взаємодії користувачів та результати рекомендацій. У цьому

підрозділі детально розглянуто джерела даних та особливості їх інтеграції у програмну архітектуру системи.

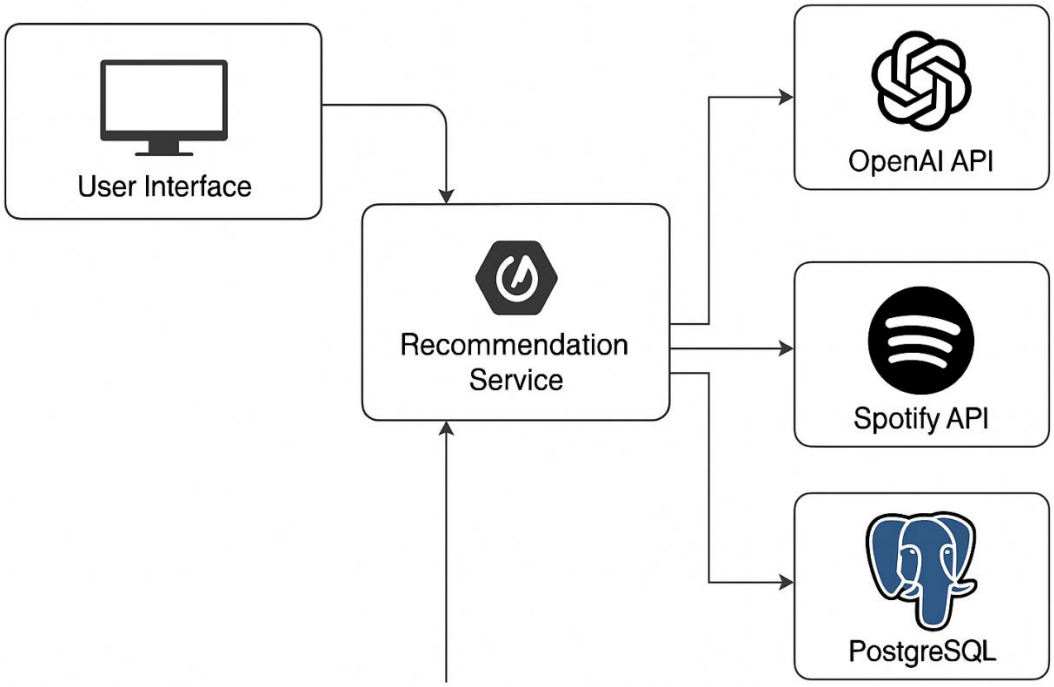


Рисунок 2.1 – Архітектурна схема інтелектуальної системи

Spotify API надає доступ до багатого каталогу музичного контенту та метаданих, необхідних для формування релевантних рекомендацій. Зокрема, з використанням Spotify API можна отримувати такі дані:

- Метаінформація про треки:
  - Унікальний ідентифікатор треку (Track ID)
  - Назва пісні
  - Ім'я виконавця
  - Альбом
  - Жанри
  - Популярність
  - Аудіоаналіз: темп (BPM), енергійність, танцювальність, акустичність, інструментальність, валентність, гучність.
- Дані про виконавців:
  - Ідентифікатор артиста

    - Жанрові приналежності
    - Популярність артиста
    - Схожі виконавці
- Дані про плейлисти та альбоми:
    - Готові добірки за темами
    - Трендові чарти та новинки.

Інтеграція:

- Використовується REST API Spotify.
- Аутентифікація — через Bearer token.
- Формат даних — JSON.
- Для отримання даних інтегрується клієнтський модуль на Java.

OpenAI API забезпечує можливості обробки запитів користувачів у природній мові та формування інтелектуальних відповідей, що підвищують рівень інтерактивності системи. З OpenAI API отримуються такі дані:

- Оброблені запити користувачів:
    - Витягнуті ключові характеристики музики (жанр, настрій, темп, контекст використання).
    - Інтерпретація текстових запитів у структуровану форму для пошуку музики.
- Генерація уточнень:
    - Автоматичне формулювання запитань, які допомагають системі краще зрозуміти потреби користувача.
- Формування пояснень до рекомендацій

Інтеграція:

- Використовується REST API OpenAI.
- Формат запитів і відповідей — JSON.
- Для інтеграції — Spring Boot REST Client для запитів до OpenAI API.

PostgreSQL використовується для зберігання внутрішніх даних системи, включаючи:

- Історію взаємодій:
    - Запити користувачів, отримані відповіді, результати рекомендацій.
- Налаштування користувачів:
    - Вибрані жанри, історія лайків/дизлайків.
- Кешовані дані:
    - Результати популярних запитів.
    - Часто запитувані треки, плейлисти.

Інтеграція:

- Доступ до даних здійснюється через JPA/Hibernate.
- Взаємодія з базою реалізується через Spring Data.
- Структура даних - реляційна модель.

2.2.1 Особливості інтеграції джерел даних

Інтеграція різнорідних джерел даних у межах інтелектуальної системи музичних рекомендацій є складним і багатокомпонентним процесом, що потребує ретельного проєктування та розробки архітектурних рішень. Основною метою інтеграції є забезпечення єдиної, узгодженої та ефективної інформаційної інфраструктури, що дозволяє отримувати, обробляти, зберігати та використовувати дані від різних сервісів і користувачів для генерації персоналізованих рекомендацій рисунок 2.2.

Рисунок 2.2 – Діаграма послідовності обробки інтеграційного запиту системи музичних рекомендацій

2.2.2 Архітектурний рівень інтеграції

Інтелектуальна система проєктується на основі багатомодульної Java-архітектури із застосуванням Spring Boot та використанням чітко розподілених логічних шарів.

- Сервіси інтеграції з зовнішніми API:
  - Окремі модулі для взаємодії з Spotify API (отримання музичних даних) та OpenAI API (обробка природної мови, генерація відповідей, уточнень, пояснень).
  - Кожен сервіс реалізується як окремий компонент із чітко визначеним інтерфейсом (наприклад, SpotifyService, OpenAIService), що полегшує масштабування та супровід.
- Сервіс керування даними:
  - Відповідає за агрегацію даних із різних джерел.
  - Виконує трансформацію форматів (наприклад, з JSON у структури Java-об'єктів) та попередню обробку даних.
- Сервіс бази даних:
  - Інтеграція з PostgreSQL через JPA/Hibernate для збереження користувацьких даних, історії взаємодій, кешованих результатів.

- Реалізація репозиторіїв (UserRepository, TrackRepository, InteractionRepository) для доступу до даних.

2.2.3 Рівень взаємодії через API

Зовнішні API:

- Spotify API:
  - Комунікація здійснюється через REST-інтерфейс за допомогою WebClient.
  - Аутентифікація — Bearer Token. Отримуються дані про треки, виконавців, альбоми та жанри.
  - Приклади запитів:
    - /v1/tracks/{id} – отримати інформацію про конкретний трек.
    - /v1/search – пошук музики за назвою або ключовими словами.
- OpenAI API:
  - Використовується REST API з авторизацією через Bearer Token.
  - Сервіс формує промти та відправляє їх до моделей OpenAI для генерації текстових відповідей, інструкцій або пояснень.
  - Отримані відповіді використовуються для подальшої обробки та уточнення запитів користувача.

Внутрішня взаємодія між сервісами:

- Всі шари комунікують через чіткі інтерфейси та DTO з модуля service-api.
- Логіка обробки запитів реалізована через сервісні класи модуля service.
- Контролери (REST) викликають методи бізнес-логіки, не порушуючи принцип розділення відповідальностей.

2.2.4 Рівень обробки даних

Трансформація та десеріалізація:

- Всі відповіді зовнішніх API надходять у форматі JSON.
- Перетворюються у Java-об’єкти завдяки DTO, що зберігаються у модулі service-api.

- Для кожного типу відповіді створений окремий DTO (наприклад, SpotifyTrackDto, OpenAIResponseDto).

Об’єднання даних із різних джерел:

  - Результати OpenAI API використовуються як параметри для формування запитів до Spotify API.
  - Наприклад, запит користувача “Музика для медитації” перетворюється на параметри: жанр = ambient, темп = повільний, настрій = спокійний.
  - Отримані треки додатково обробляються з урахуванням історії користувача з бази даних — для уникнення повторів та підвищення персоналізації.

2.2.5 Рівень безпеки та контролю доступу

- Використання OAuth 2.0 для автентифікації запитів до Spotify API.
- Захист ключів API OpenAI через конфігураційні файли.
- Реалізація обробки помилок.
- Впровадження логування запитів та відповідей для відстеження роботи системи та виявлення проблем.
- Моніторинг викликів API через Actuator або сторонні сервіси.

Висновки до розділу 2

Проведене дослідження засвідчує, що інтеграція різнорідних джерел даних є одним із ключових аспектів побудови інтелектуальної системи музичних рекомендацій. Створення такої системи потребує ретельного об’єднання зовнішніх сервісів (таких як Spotify API для доступу до музичних треків, метаданих і аудіоаналізу, OpenAI API для реалізації можливостей діалогової взаємодії та інтерпретації запитів користувачів), внутрішніх сховищ даних (PostgreSQL для збереження інформації про запити, результати рекомендацій, профілі користувачів) та обробки запитів користувача в режимі реального часу через API-інтерфейси, реалізовані на базі Java та Spring Boot.

Запропонована архітектура системи забезпечує багаторівневий підхід до інтеграції даних:

- На першому рівні відбувається взаємодія з користувачем, який формулює запити у природній мові через інтерфейс.
- На другому рівні реалізовано обробку та інтерпретацію запитів за допомогою OpenAI API, що дозволяє системі розуміти контекст, уточнювати запити та пояснювати рекомендації.
- На третьому рівні здійснюється взаємодія з зовнішніми музичними сервісами, зокрема Spotify API, для отримання актуальної інформації про музичні твори, їх характеристики та популярність.
- На четвертому рівні працює модуль обробки даних і збереження інформації у базі даних PostgreSQL для подальшого аналізу, формування статистики.
- Нарешті, п’ятий рівень відповідає за формування відповідей для користувача, що включає як рекомендації, так і пояснення вибору музичних треків.

Інтеграція цих компонентів забезпечує гнучкість системи, оскільки дозволяє динамічно підлаштовувати рекомендації під індивідуальні вподобання користувача, оперативно реагувати на зміни запитів і підтримувати діалогову взаємодію, що особливо актуально для сучасних рекомендаційних систем. Крім того, завдяки застосуванню OpenAI API, система отримує можливість формулювати пояснення, уточнювати контекст запитів і долати проблему непрозорості, характерну для багатьох існуючих алгоритмів рекомендацій.

Особливо важливо підкреслити, що така багаторівнева інтеграція дозволяє вирішити низку проблем, притаманних сучасним системам музичних рекомендацій:

Таким чином, розроблена інформаційна модель системи музичних рекомендацій створює надійне підґрунтя для реалізації сучасного сервісу, що поєднує сильні сторони зовнішніх API та можливості штучного інтелекту, забезпечуючи високу якість рекомендацій, задоволеність користувачів та інноваційний користувацький досвід через діалогову взаємодію та персоналізацію результатів. Це

формує основу для створення інтелектуальної системи нового покоління, яка здатна задовольняти зростаючі вимоги сучасних користувачів музичних сервісів.

РОЗДІЛ 3. МАТЕМАТИЧНЕ ЗАБЕЗПЕЧЕННЯ

Математичне забезпечення є ключовим елементом побудови інтелектуальної системи музичних рекомендацій, оскільки саме воно визначає принципи обробки даних, побудови моделей, аналізу інформації та формування рекомендацій. Основою системи є поєднання класичних математичних методів та сучасних алгоритмів штучного інтелекту, обробки природної мови та методів оптимізації. Особливу увагу приділено інтеграції різних підходів, таких як колаборативна фільтрація, контентний аналіз, гібридні моделі, а також використанню моделей генеративного ШІ для підтримки діалогової взаємодії.

Цей розділ присвячений опису математичних моделей та алгоритмів, що лежать в основі системи, а також формалізації завдань, які вирішує розроблювана система музичних рекомендацій. Розглянуто математичні принципи, на яких базуються ключові алгоритми системи: аналіз жанрової структури альбому, визначення релевантних виконавців, квотний механізм розподілу рекомендацій, алгоритм backfill, а також правила стійкої обробки LLM-відповідей.

3.1 Алгоритми функціонування системи

Розроблена система музичних рекомендацій взаємодіє з користувачем у режимі діалогу, аналізує запити, отримує дані з зовнішніх API, проводить інтеграцію результатів та формує персоналізовані списки треків.

Алгоритми, що складають основу системи, поділяються на декілька логічних блоків:

- обробка природномовного запиту користувача;
- генерація уточнених рекомендацій за допомогою OpenAI API;
- визначення релевантності треків;
- побудова жанрових і виконавських характеристик альбому;
- розподіл жанрових квот;
- усунення дублікатів;
- формування фінального списку.

3.2 Аналіз жанрової структури та визначення ваг жанрів

Кожен альбом має набір треків, а кожен трек — одного або кількох виконавців. Система об’єднує жанрову інформацію всіх виконавців альбому та підраховує, як часто кожен жанр з’являється серед доступних даних.

На основі цих частот формується жанрова вага — відносна "важливість" жанру у контексті конкретного альбому (Додаток Д.2).

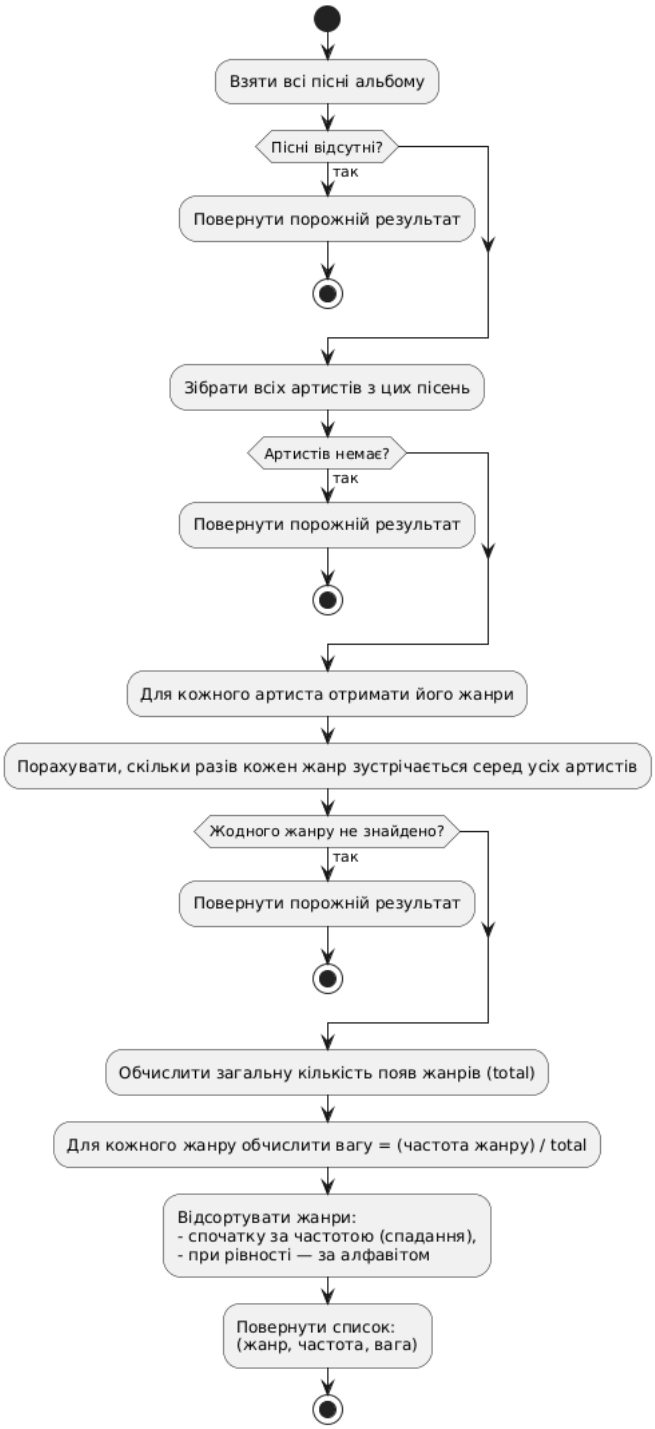


Рисунок 3.1 – Блок-схема логіки визначення ваг жанрів

Таке нормоване представлення дозволяє коректно розподіляти жанрові квоти у фінальному списку рекомендацій та забезпечує відтворення стилістичного балансу альбому в новоствореному рекомендаційному наборі.

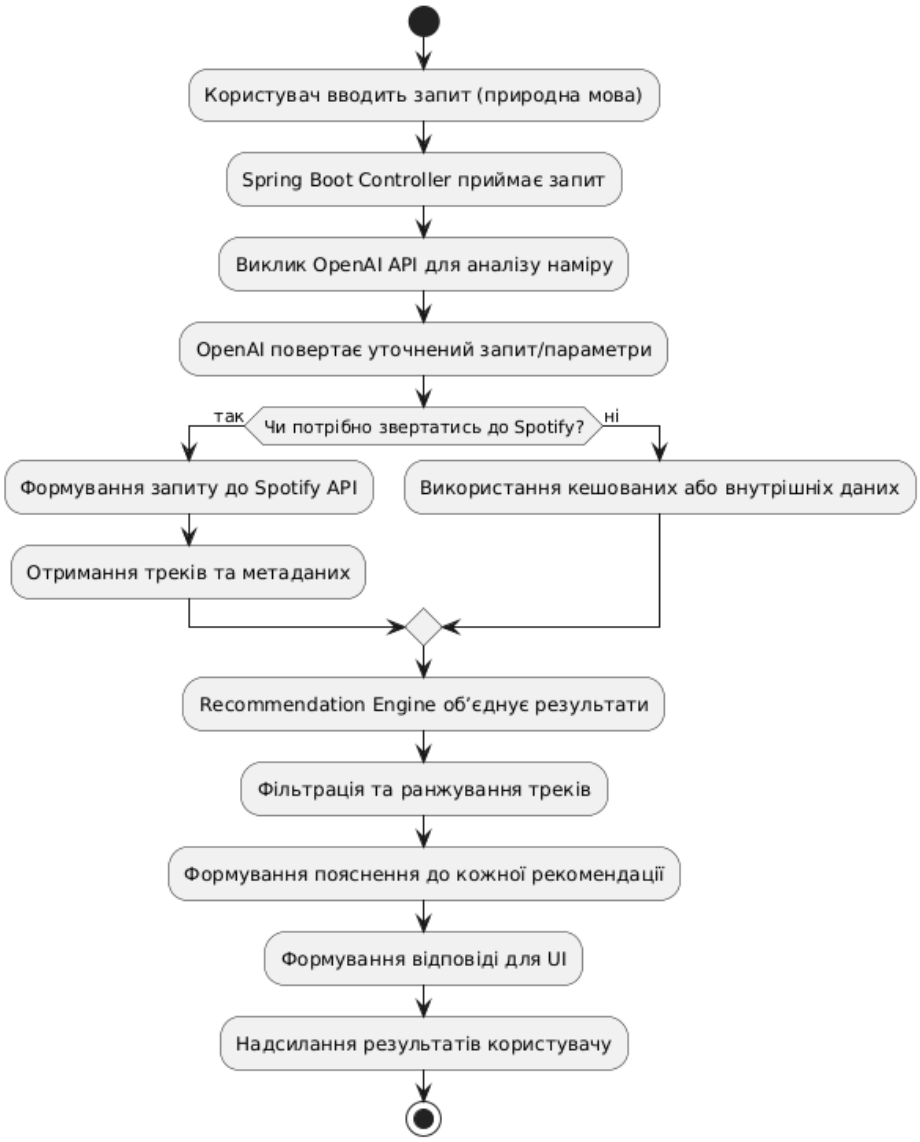


Рисунок 3.2 – Блок-схема алгоритму функціонування системи

Система повинна сформувати список рекомендацій фіксованого розміру. Щоб досягти пропорційності між жанрами, використовується метод квотування на основі ваг жанрів.

Після обчислення "ідеальної" кількості треків у кожній категорії застосовується механізм найбільших залишків, який дозволяє точно виділити необхідну кількість рекомендацій і зберегти співвідношення жанрів.

Цей метод широко використовується в задачах розподілу ресурсів та виборчих системах, що підтверджує його надійність і математичну коректність.

Важливою складовою є визначення так званих “focus artists” — виконавців, які найбільше асоціюються з альбомом.

Для цього система підраховує, у скількох треках кожен виконавець бере участь, після чого ранжує їх за частотою появи. У разі однакових значень застосовується лексикографічне впорядкування.

OpenAI API використовується для генерації початкових кандидатів на основі запиту користувача та (у випадку альбомних рекомендацій) контекстних параметрів — жанрових квот і ключових виконавців.

Система застосовує механізми очищення та реконструкції JSON-відповідей, які усувають:

- зайві символи;
- “галюцинації” мовної моделі;
- неправильне форматування JSON;
- код-блоки;
- надлишковий текст поза масивами.

Якщо розпізнати JSON неможливо, система переходить у детермінований режим та повністю покладається на алгоритм backfill - детермінований добір треків.

Якщо мовна модель повернула недостатньо елементів, система виконує добір у кілька етапів:

- Пошук треків від ключових виконавців у найбільш важливих жанрах.
- Пошук за жанром незалежно від виконавця.
- Поступове заповнення списку до необхідного розміру.

Алгоритм працює жадібно, однак гарантує, що результуючий список буде повним і відповідатиме жанровому профілю альбому.

Ключові виконавці використовуються мовною моделлю та backfill-алгоритмом для підвищення точності результатів.

Для різних етапів системи характерні такі закономірності:

- обробка жанрів займає пропорційний час щодо кількості виконавців;
- квотний механізм має логарифмічну складність;
- основна затримка системи — зовнішні запити до Spotify та OpenAI;

- backfill-алгоритм керується кількістю необхідних спроб пошуку.

Таким чином система масштабується лінійно щодо кількості викликів API.

Обмеження та напрями вдосконалення. Серед відомих проблем:

- жанри оцінюються за виконавцями, а не за мета даними аудіо;
- backfill працює лише локально оптимально;
- стохастичність LLM може впливати на стабільність.

Можливі покращення:

- урахування косинусної подібності між треками;
- додавання фільтрації за мета даними аудіо;
- використання нечітких текстових метрик;

Висновки до розділу 3

У цьому розділі було проведено аналіз який показує, що система спирається на узгоджену сукупність алгоритмів, які поєднують статистичні методи, елементи оптимізації, механізми обробки природної мови, а також сучасні підходи генеративного штучного інтелекту. Така багаторівнева інтеграція дозволила сформувати математичну модель, здатну працювати в умовах невизначеності, неповноти даних і стохастичного характеру мовних відповідей.

У межах розділу було показано, що ключовими структурними елементами математичного забезпечення є: аналіз жанрової структури альбому, визначення найбільш значущих виконавців, квотний розподіл рекомендацій, алгоритми дедуплікації, жадібний backfill та стійке опрацювання результатів генеративної мовної моделі. Визначення ваг жанрів на основі частот їх появи в альбомі дозволяє формувати імовірнісний стильовий профіль, який відтворює характерні жанрові особливості конкретної музичної вибірки. Подібний підхід забезпечує адаптивність і можливість формування рекомендацій, що відповідають художньому контексту початкового матеріалу.

Окремо було виділено механізм визначення “focus artists” — виконавців, чия присутність у треках альбому є найбільш домінантною. Такий механізм дозволяє алгоритмам рекомендацій орієнтуватися не лише на жанрову структуру, а й на

персоніфіковані музичні ознаки, властиві конкретним артистам. Поєднання цих підходів лежить в основі гібридної моделі, здатної враховувати як контентні характеристики (жанри, виконавці), так і поведінкові сигнали.

З математичної точки зору одним із найбільш важливих елементів системи є квотний механізм формування списку рекомендацій. Використання методу найбільших залишків забезпечує пропорційний розподіл обмеженого набору місць між жанрами відповідно до їх ваги. Цей метод широко застосовується у задачах розподілу ресурсів, зокрема у виборчих системах, що підтверджує його надійність, стабільність та здатність зберігати пропорційність у кінцевому результаті.

Надзвичайно важливим для коректності системи є алгоритм усунення дублікатів, який працює на двох рівнях — за унікальними ідентифікаторами і за нормалізованими текстовими ключами. Такий підхід дозволяє уникнути повторів, враховуючи варіативність назв треків, версій записів та різні формати представлення виконавців, що неминуче виникають за умов роботи з відкритими даними та генеративними моделями.

Також було виконано аналіз обчислювальної складності основних алгоритмів системи. Показано, що більшість внутрішніх операцій мають лінійну або логарифмічну складність, а основним джерелом затримок є виклики зовнішніх API (Spotify і OpenAI), що відповідає природі інтегрованої системи з інтенсивним використанням зовнішніх джерел даних. Це дозволяє масштабувати систему лінійно відносно кількості звернень до сервісів та забезпечує її ефективність у реальних умовах.

Наприкінці розділу було розглянуто обмеження моделі та вказано потенційні напрями розвитку. До таких належать: можливість підсилення контентної моделі за рахунок аналізу мета даних аудіо, використання подібності треків у векторному просторі ознак, застосування нечітких текстових метрик для точнішої дедуплікації, а також удосконалення механізмів стійкості до варіативності LLM-відповідей. Запропоновані напрями розвитку демонструють, що система має потенціал для подальшого вдосконалення в напрямку більш глибокого врахування музичних властивостей, контексту та індивідуальних уподобань користувача.

РОЗДІЛ 4. ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ

Програмне забезпечення є центральним компонентом реалізації інтелектуальної системи музичних рекомендацій і забезпечує повну підтримку функціональних, інтеграційних та інтерфейсних вимог, сформульованих у попередніх розділах. Саме на рівні програмного забезпечення здійснюється практичне втілення математичних моделей, алгоритмів обробки музичних даних, механізмів взаємодії з користувачем, а також цілісна інтеграція з зовнішніми сервісами — зокрема Spotify Web API, Spotify Web Playback SDK та OpenAI API.

У межах цього розділу розглядаються архітектурні та технологічні рішення, які визначають структуру та поведінку системи. Зокрема, backend-частина побудована на базі Java 21 та Spring Boot із використанням модульного підходу (core, service, service-api, dao), що забезпечує чітке розмежування відповідальностей між рівнями доступу до даних, бізнес-логіки та веб-інтерфейсів. Постійність та відтворюваність схеми бази даних гарантується за допомогою Liquibase, а інтеграції з зовнішніми сервісами реалізовано на основі REST-архітектури, WebClient (WebFlux) і спеціалізованих сервісних модулів. Окрему увагу приділено безпековим аспектам, таким як JWT-автентифікація, рольова авторизація та керування доступом до API.

Frontend частина системи розроблена засобами Next.js 15 та React 19, що дозволяє поєднувати рендеринг на стороні сервера (RSC) та інтерактивні клієнтські компоненти. Вона забезпечує реалізацію користувацького інтерфейсу, включно з пошуком треків, керуванням музичними “альбомами”, чатом рекомендацій, відтворенням музики та інтерактивними функціями на основі Spotify Web Playback SDK. Для керування станом та взаємодією з backend-серверами використовуються SWR, Axios та архітектурні контексти React. Аутентифікація реалізована через NextAuth, що підтримує єдиний механізм автентифікації на фронтенді та бекенді з використанням JWT-токенів.

Таким чином, програмне забезпечення системи є інтегрованим комплексом, який об’єднує серверну логіку, клієнтську взаємодію, обчислювальне ядро рекомендацій та механізми доступу до зовнішніх сервісів. У цьому розділі детально

описано архітектуру застосунку, структуру модулів, принципи реалізації рекомендаційної логіки, особливості функціонування фронтенду, організацію API-шару, а також рішення щодо безпеки, продуктивності та тестування. Це забезпечує цілісне бачення програмної системи як інтелектуального, масштабованого та надійного інструменту для персоналізованої музичної рекомендації.

4.1 Вибір середовища та засобів реалізації

Процес створення інтелектуальної системи музичних рекомендацій потребує застосування сучасного, надійного та масштабованого програмного середовища, яке забезпечує продуктивну розробку, можливість інтеграції з зовнішніми сервісами, модульність, стійкість до навантажень, а також підтримку сучасних методів взаємодії з користувачем. Обраний технологічний стек охоплює як серверну, так і клієнтську частини системи, утворюючи цілісну архітектурну платформу для реалізації складних обчислювальних процесів, рекомендаційних алгоритмів та інтерактивної взаємодії з користувачем.

Середовище розробки: IntelliJ IDEA та WebStorm

Для реалізації бекенд-частини застосовано середовище розробки IntelliJ IDEA, яке забезпечує:

- повну підтримку Java-екосистеми (Spring Boot, Maven, JPA, WebFlux);
- зручну навігацію та інструменти рефакторингу;
- інтеграцію зі сховищами GitHub;
- вбудовані можливості розробки REST API (HTTP client);
- засоби роботи з PostgreSQL і Docker;
- підтримку тестування (JUnit, Mockito).

Фронтенд розроблявся у середовищі WebStorm, яке дозволяє:

- ефективно працювати з TypeScript і React;
- використовувати інструменти перевірки якості коду (ESLint, Prettier);
- забезпечувати hot-reload і швидку збірку із застосуванням Turbopack;
- інтегрувати UI-бібліотеки та налаштовувати Tailwind CSS.

Мови програмування: Java 21, TypeScript

Java була обрана як основна мова розробки з огляду на:

- високу продуктивність у вебсервісах;
- типізацію, що зменшує ризик помилок;
- багату екосистему бібліотек та фреймворків;
- активну підтримку та наявність великої спільноти.

Використання Java 21 забезпечує доступ до сучасних синтаксичних можливостей, оптимізації та LTS-підтримки.

Фреймворки: Spring Boot, React та Next.js

Spring Boot був обраний як основний фреймворк для побудови бекенд-сервісу системи через такі властивості:

- швидкий старт (мінімум конфігурацій);
- вбудований вебсервер (Tomcat);
- підтримка REST-контролерів та інтерфейсів;
- модульність (Spring Web, Spring Data, Spring Validation тощо);
- зручна інтеграція з базами даних (через JPA);
- готові рішення для обробки помилок, валідації, безпеки;

React дає можливість побудови реактивних інтерфейсів та компонентної архітектури.

Next.js 15 з App Router поєднує:

- серверний рендеринг (RSC);
- клієнтські інтерактивні компоненти;
- оптимізований роутинг;
- розширені можливості продуктивності;

Інтеграція з зовнішніми API

Для обміну даними з сервісами Spotify API та OpenAI API буде використано:

- Spring WebClient для асинхронних викликів до Spotify та OpenAI;
- Власні DTO-класи для обробки JSON-відповідей;
- AOP та Slf4j для логування запитів;

Frontend має додаткові інтеграції:

- ініціалізація Spotify плеєра у браузері;
- отримання user-access-token через бекенд;
- функції відтворення: play, pause, next, previous, volume, seek;
- відтворення треків у режимі реального часу.

База даних: PostgreSQL

PostgreSQL було обрано як реляційну СУБД завдяки:

- надійності, стабільності, відкритому коду;
- підтримці складних зв'язків між таблицями;
- сумісності з ORM-рішеннями (Hibernate/JPA);
- підтримці JSON-типів, які можуть бути використані для збереження динамічних налаштувань або відповідей API.

Інструменти:

- pgAdmin — для адміністрування БД;
- IntelliJ SQL Tool — для візуального перегляду даних;
- Liquibase — для керування міграціями БД.

Інші допоміжні засоби

- Maven — для управління залежностями та складання проєкту;
- JUnit 5, Mockito — для модульного тестування;
- Lombok — для зменшення шаблонного коду;
- Jackson — для роботи з JSON (серіалізація/десеріалізація відповідей API);
- Postman — для ручного тестування REST-запитів.
- Axios — HTTP-клієнт для роботи з backend API;
- SWR — кешування та автоматичний рефетчинг даних;
- React Hook Form + Zod — формування форм та валідація;
- ESLint + Prettier — лінтинг і форматування коду.

4.2 Реалізація архітектури ПЗ

Програмне забезпечення реалізовано як багатомодульний Maven-проєкт зі структурним розмежуванням шарів за принципом core → service → dao, при цьому контракти та DTO винесені в окремий модуль service-api. Така організація зменшує зв'язаність, спрощує тестування й повторне використання компонентів.

- core — центральний модуль, що містить точку входу до застосунку, REST-контролери, налаштування безпеки (Spring Security + JWT), конфігураційні класи, механізми обробки помилок та опис API через Swagger/OpenAPI. У цьому модулі реалізовано шар взаємодії з клієнтським застосунком, включно з проксі-маршрутами для Spotify OAuth та засобами керування сесіями користувачів.
- service — модуль бізнес-логіки, який реалізує функціональні сценарії системи: інтеграцію зі Spotify Web API, OAuth-аутентифікацію користувача, логіку роботи з "альбомами" та лайками, обчислення жанрових ваг і фокус-артистів, взаємодію з OpenAI, а також алгоритм формування рекомендацій. У цьому модулі реалізовано шар системи, що пов'язує математичні моделі зі зовнішніми ресурсами та базою даних.
- service-api — контрактний модуль, який містить інтерфейси сервісів, транспортні моделі даних для доменної логіки та безпеки, утиліти генерації та валідації JWT, а також спільні константи. Цей модуль не містить бізнес-логіки, але визначає стандартизовану форму обміну даними між модулями та між фронтендом і бекендом, що знижує ризики неузгодженості.
- dao — модуль доступу до даних, який включає JPA-сутності, Spring Data репозиторії та опис міграцій PostgreSQL через Liquibase. Це забезпечує стійку еволюцію схеми бази даних, відтворюваність змін, а також відокремлення роботи з даними від бізнес-логіки.

Архітектура підтримує класичний шаровий підхід, де кожен модуль має строго окреслену відповідальність, а взаємодія між шарами виконується лише через інтерфейси. Такий підхід спрощує модульне тестування, дозволяє ізолювати зміни й забезпечує легше масштабування системи в майбутньому.

Крім того, бекенд-архітектура орієнтована на інтеграцію із сучасним клієнтським застосунком на основі Next.js 15 та React 19. Серверна частина виступає в ролі єдиного API-шару, який об’єднує:

- запити від клієнта, що надходять через Axios/SWR;
- аутентифікаційну взаємодію з NextAuth (Credentials Provider → JWT);
- Spotify OAuth-авторизацію та генерацію Spotify user-access-token;
- асинхронну взаємодію з OpenAI API.

Таким чином архітектурна модель ПЗ формує цілісну систему, у якій frontend та backend працюють як скоординовані незалежні компоненти: Frontend відповідає за інтерфейс і взаємодію користувача, а Backend — за бізнес-логіку, безпеку, інтеграції та збереження даних.

4.2.1 Конфігурація та профілі середовищ

Конфігурацію параметрів виконання зосереджено у файлах application.properties з підтримкою профілів. Окремо задаються:

- підключення до БД (URL, користувач, пароль);
- параметри Liquibase (автоматичні міграції схеми);
- налаштування Swagger/OpenAPI;
- параметри JWT (секрет, час життя токена);
- інтеграції зі Spotify (client id/secret, редіректи) та OpenAI/LangChain4j.

Для покращення бесзпеки є можливість винести секрети зі звичайних properties у змінні середовища або secrets manager (Azure KeyVault).

Застосунок працює в stateless-режимі з JWT-автентифікацією (Bearer-token). Паролі зберігаються у вигляді BCrypt-хешів. Авторизація — рольова (ROLE_USER, ROLE_ADMIN), публічними лишаються Swagger та ендпойнти автентифікації (Додаток Д.1).

4.3 Інтеграція зі зовнішніми сервісами

Інтеграція із зовнішніми сервісами є ключовим елементом функціонування інтелектуальної системи музичних рекомендацій, оскільки саме зовнішні API забезпечують доступ до релевантних музичних даних, можливість керування відтворенням треків та використання моделей штучного інтелекту для генерації текстових рекомендацій. У цьому підрозділі розглядається комплексна архітектура взаємодії з сервісами Spotify Web API, Spotify Web Playback SDK та OpenAI API, що охоплює як серверну, так і клієнтську частини застосунку.

4.3.1 Інтеграція з Spotify Web API

Spotify Web API виконує роль основного джерела інформації про треки, альбоми, виконавців та жанри, а також використовується для виконання операцій у контексті облікового запису користувача. Інтеграція реалізована у серверному модулі через Spring WebClient, що забезпечує асинхронну та неблокуючу взаємодію з API.

Архітектурна структура інтеграції передбачає:

- Отримання токенів доступу через реалізацію Spotify OAuth (Authorization Code Flow).
- Збереження access/refresh token у базі даних (таблиця UsersSpotifyTokens).
- Автоматичне оновлення access token за допомогою refresh token.
- Інкапсульовані сервіси для виконання запитів до Spotify:
- пошук треків за ключовими словами;
- отримання доступних пристроїв користувача та їх статусу.

Комунікація з API виконується через стандартизовані DTO, що знижує ризик несумісності між сервісами та підвищує читабельність бізнес-логіки. Перевагою такого підходу є можливість централізованої обробки помилок Spotify API, повторного відправлення запитів після оновлення токена та логування HTTP-взаємодій із метою аналізу продуктивності.

Дана інтеграція забезпечує систему повним набором необхідних музичних даних для побудови рекомендаційних моделей та інтерактивної взаємодії користувача із власною медіатекою.

4.3.2 Інтеграція з Spotify Web Playback SDK

Spotify Web Playback SDK використовується на клієнтській частині для забезпечення можливості локального відтворення музики в браузері користувача. На відміну від Web API, що надає доступ до даних, Web Playback SDK забезпечує:

- керування відтворенням (play, pause, resume);
- перемикання треків (next/previous);
- зміну гучності;
- отримання інформації про поточний трек;
- роботу з аудіопотоком через локальний віртуальний плеєр.

Архітектурна схема інтеграції SDK передбачає:

1. Ініціалізацію плеєра під час завантаження застосунку та реєстрацію callback-функцій.
2. Отримання Spotify user access token від бекенду (через /spotify/user-token).
3. Реєстрацію пристрою в обліковому записі користувача.
4. Передачу управління плеєром через бекенд-проксі:
   - /spotify/play
   - /spotify/pause
   - /spotify/volume
   - /spotify/transfer
   - /spotify/next, /previous

Це дозволяє забезпечити захист від прямої взаємодії фронтенду з Spotify API, оскільки всі ключові операції виконуються через авторизований бекенд.

На фронтенді інтеграція структурована через окремі React-контексти:

- PlayerContext — стан відтворення, поточний трек, контрольні методи.
- PlayUriContext — функція швидкого запуску треку.
- VolumeSlider компонент — управління гучністю.

У поєднанні зі SWR (кеш та оновлення даних) це формує високопродуктивну, реактивну та безпечну архітектуру роботи плеєра.

4.3.3 Інтеграція з OpenAI API

OpenAI API використовується для побудови інтелектуальної підсистеми рекомендацій музики. Інтеграція реалізована у модулі service через LangChain4j, який забезпечує зручне формування запитів та обробку природної мови.

Основні функціональні можливості інтеграції:

- генерація рекомендацій на основі текстового запиту користувача;
- формування JSON-структур з парами «title–artist»;
- уточнення запитів та моделювання діалогової взаємодії;
- формування рекомендаційного набору на основі аналізу жанрової структури альбому.

Система побудована таким чином, що мовна модель виконує лише первинну генерацію кандидатів, а остаточна валідація та добір треків здійснюються локальними алгоритмами. Це дозволяє:

- фільтрувати некоректні або неоднозначні відповіді;
- усувати структурні похибки JSON;
- уникати дублювання треків;
- забезпечувати відповідність жанровим квотам.

Взаємодія з OpenAI API інтегрована в рекомендаційний пайплайн, який складається з:

1. аналізу запиту та побудови промпту;
2. генерації музичних кандидатів;
3. аналізу жанрових ваг;
4. визначення “focus artists”;
5. алгоритму backfill у випадку неповних відповідей.

Таким чином OpenAI не замінює внутрішню логіку, а доповнює її, виступаючи як семантичний інтерфейс між вимогами користувача та рекомендаційним механізмом.

4.4 Підсистема роботи з даними, обробки помилок та тестування

Сучасні інтелектуальні системи потребують не лише функціонально багатої бізнес-логіки, але й надійної інфраструктури обробки даних, механізмів контролю помилок та систематичного тестування. У цьому підрозділі розглянуто інтегровану підсистему, що поєднує:

1. Доступ до даних і керування їхньою структурою,
2. Механізми обробки виключних ситуацій та логування,
3. Комплексний підхід до тестування програмного забезпечення.

Дана підсистема забезпечує стабільність, відмовостійкість і передбачуваність роботи рекомендаційної платформи.

4.4.1 Доступ до даних та керування міграціями

Доступ до даних реалізовано через модуль dao, який будується на технологіях Spring Data JPA, Hibernate та PostgreSQL. Така архітектура забезпечує абстракцію від SQL-запитів, автоматичне формування репозиторіїв та прозоре перетворення доменних сутностей у записи бази даних.

Структура доступу до даних

- JPA-сутності описують доменну модель користувачів, Spotify-токенів, альбомів, треків, лайків та інших компонентів системи.
- Spring Data Repository забезпечує CRUD-операції, формування динамічних запитів, пагінацію та оптимізоване завантаження пов’язаних об’єктів.
- Сервісний шар звертається до репозиторіїв через інтерфейс service-api, що підтримує слабке зв’язування та полегшує модульне тестування.

Для керування змінами структури бази застосовується Liquibase, який забезпечує:

- контроль версій схеми БД через XML/SQL/JSON changelog-файли;
- повторювану та безпечну ініціалізацію таблиць;
- можливість відкату змін;
- автоматичний запуск міграцій при старті застосунку.

Рисунок 4.1 – Структурна схема сховища даних

Версіонування схеми дозволяє гарантувати, що будь-який сервер розгортання матиме ідентичну структуру БД, що є критичним для інтелектуальних систем, де узгодженість даних впливає на алгоритмічні результати.

Оптимізація та цілісність даних

- використання індексів для полів, що часто запитуються (spotifyId, userId, albumId);
- відкладене завантаження для зменшення кількості SQL-запитів;
- каскадні операції для узгодженого видалення/оновлення пов’язаних даних;
- перевірки цілісності на рівні сутностей (nullable, length, unique).

Таким чином, підсистема зберігання даних забезпечує стабільний фундамент для бізнес-логіки рекомендацій.

4.4.2 Обробка помилок та логування

У складних системах, що взаємодіють зі сторонніми API (Spotify, OpenAI), надійна система обробки помилок є критичною. У даному проєкті реалізовано багаторівневий підхід до ерор-хендлінгу та логування, який охоплює backend, frontend та інтеграційні сервіси.

У серверній частині Spring Boot використовується централізований механізм:

- @ControllerAdvice + @ExceptionHandler — для перехоплення бізнесових, валідаційних та системних помилок;
- уніфікована структура відповіді для помилок (статус, код, повідомлення, timestamp);
- класи винятків поділяються на:
    - бізнесові (NotFoundException, ConflictException),
    - технічні (IntegrationException),
    - помилки автентифікації (JwtException).

Особлива увага приділена помилкам інтеграцій:

- недоступність Spotify API;
- завершення строку дії токена;
- помилки у відповіді LLM;
- некоректні JSON-структури.

У таких ситуаціях реалізується стратегія повторних запитів, оновлення токена, або перехід у детермінований режим, що забезпечує відмовостійкість.

Система логування побудована на основі Slf4j + Logback і охоплює:

- інформаційні логи бізнес-процесів;
- детальні інтеграційні логи HTTP-запитів до Spotify та OpenAI;
- логування помилок і стек-трейсів;
- часові метрики запитів та відповідей.

На фронтенді застосовується:

- перехоплення помилок у Axios Interceptor,
- обробка HTTP-кодів,

- відображення помилок користувачу через toast-компоненти.

Такий підхід забезпечує діагностованість і прозорість роботи системи.

4.4.3 Тестування програмного забезпечення

Тестування виконується тільки на модульному рівні (JUnit 5 + Mockito), а саме охоплює наступні випадки:

- сервісні методи DAO;
- бізнес-логіку рекомендацій;
- Spotify OAuth-авторизацію;
- обробку жанрів і обчислення жанрових ваг;
- валідацію даних користувача;
- алгоритми backfill та фільтрацію дублікатів.

Mockito використовується для:

емулювання сторонніх API;

заміни репозиторіїв під час тестування сервісів;

створення тестових сценаріїв із визначеними відповідями.

Це забезпечує ізольованість тестів та контрольованість вхідних умов.

Висновки до розділу 4

У цьому розділі було комплексно розглянуто програмне забезпечення інтелектуальної системи музичних рекомендацій, що включає вибір середовища розробки, архітектурні рішення, інтеграцію з зовнішніми сервісами, організацію доступу до даних, механізми обробки помилок та підходи до тестування. На основі проведеного аналізу можна зробити висновок, що обрана технологічна платформа повністю відповідає вимогам функціональності, продуктивності, масштабованості та надійності системи.

Вибір середовища та засобів реалізації (розділ 4.1) є обґрунтованим з позицій інженерної ефективності. IntelliJ IDEA забезпечує повну інтеграцію зі стеком Java та зручні інструменти для роботи зі Spring Boot, Maven, базами даних, засобами налагодження й тестування. Використання Java 21 гарантує стабільність, сучасні

мовні можливості та довготривалу підтримку. На рівні бекенда Spring Boot виступає універсальною платформою для реалізації REST-орієнтованих сервісів, забезпечуючи модульність, розширюваність та чітке розмежування відповідальностей. На клієнтському боці застосування Next.js 15 і React 19 формує сучасну, продуктивну та інтерактивну SPA-архітектуру з підтримкою серверних і клієнтських компонентів.

У рамках реалізації архітектури програмного забезпечення (розділ 4.2) впроваджено багатомодульну структуру Maven-проєкту, де модулі core, service, service-api та dao взаємодіють через чітко визначені контракти. Такий підхід забезпечує низьку зв'язаність, зручність рефакторингу, можливість незалежного тестування та масштабування проєкту. Конфігурація профілів середовищ дає змогу адаптувати поведінку системи до різних етапів розгортання (dev, test, prod), забезпечивши керованість параметрами безпеки, з'єднання з БД та інтеграційних сервісів.

Інтеграція із зовнішніми сервісами (розділ 4.3) є однією з ключових складових системи. Інтеграція зі Spotify Web API забезпечує доступ до музичного контенту, керування автентифікацією користувача через OAuth 2.0 та можливість виконання операцій у межах облікового запису. Spotify Web Playback SDK реалізує повноцінне відтворення музики у браузері користувача, тоді як бекенд виконує роль проксі для захищеної передачі команд. OpenAI API виконує роль інтелектуального модуля для генерації рекомендацій і семантичної обробки запитів користувачів. Завдяки LangChain4j забезпечено стандартизовану взаємодію з LLM та контрольоване формування відповідей. Усі три сервісні інтеграції працюють як єдина інтелектуальна екосистема рекомендацій.

Підсистема доступу до даних, обробки помилок та тестування (розділ 4.4) забезпечує надійність і стійкість функціонування застосунку. PostgreSQL у поєднанні зі Spring Data JPA та Liquibase дозволяє гарантувати узгодженість і цілісність даних, а також відтворюваність міграцій схеми БД. Централізована обробка помилок і системи логування забезпечують діагностованість і стійкість до збоїв зовнішніх сервісів, а використання детермінованих fallback-алгоритмів (наприклад, backfill)

гарантує повноту результатів навіть у разі непередбачуваних відповідей API. Тестування, що включає модульні, інтеграційні й поведінкові тести на обох частинах (backend і frontend), забезпечує стабільність і коректність системи в різних умовах експлуатації.

Отже, розроблене програмне забезпечення являє собою технологічно цілісну, модульну та інтелектуально насичену систему, що включає комплекс зовнішніх інтеграцій, стабільну архітектуру з багаторівневим розподілом відповідальностей, сучасний клієнтський застосунок і надійну інфраструктуру для роботи з даними та тестування. Такий підхід забезпечує відповідність проєкту поставленим вимогам, підтримує можливість його подальшого розширення і є міцним підґрунтям для розвитку інтелектуальних функцій у майбутньому.

РОЗДІЛ 5. РОЗРОБЛЕННЯ СТАРТАП-ПРОЄКТУ

5.1 Опис ідеї стартап-проєкту

Ідея стартап-проєкту полягає у створенні інтелектуальної системи музичних рекомендацій, що поєднує можливості класичних алгоритмів рекомендаційних систем, інтеграцію з музичним сервісом Spotify та використання моделей штучного інтелекту для обробки природномовних запитів користувачів. На відміну від стандартних механізмів рекомендацій, орієнтованих переважно на схожість треків або історію прослуховувань, запропонований підхід поєднує контентний аналіз (жанрова структура, характеристика альбомів і виконавців), поведінкові дані користувача та семантичну інтерпретацію його запитів у вільній формі.

Кінцевий продукт розглядається як сервіс, що виконує роль «персонального музичного куратора», здатного:

- формувати добірки треків на основі конкретних альбомів, жанрів, виконавців або настрою;
- здійснювати діалогову взаємодію з користувачем у форматі чат-асистента;
- враховувати історію вподобань користувача в Spotify;
- адаптувати рекомендації до виду діяльності (робота, спорт, відпочинок, навчання);
- генерувати пропозиції, що поєднують відомі та нові треки, зберігаючи стилістичну цілісність.

Потенційні напрями застосування стартап-проєкту включають:

- індивідуальних користувачів потокових сервісів, які бажають отримувати більш “людяні” рекомендації;
- невеликі бізнеси (кав’ярні, магазини тощо), що потребують фонового музичного супроводу;
- інтеграцію як окремого рекомендаційного модуля в інші застосунки через API.

Унікальність стартап-ідеї полягає в поєднанні декількох рівнів персоналізації: від аналізу конкретного альбому користувача (побудова жанрового профілю та

визначення ключових виконавців) до врахування його поточного запиту у природній мові. Модель не лише підбирає схожі треки, а й намагається зрозуміти контекст: для чого потрібна музика, які емоції або атмосфера бажані, які жанри є прийнятними, а які варто уникати.

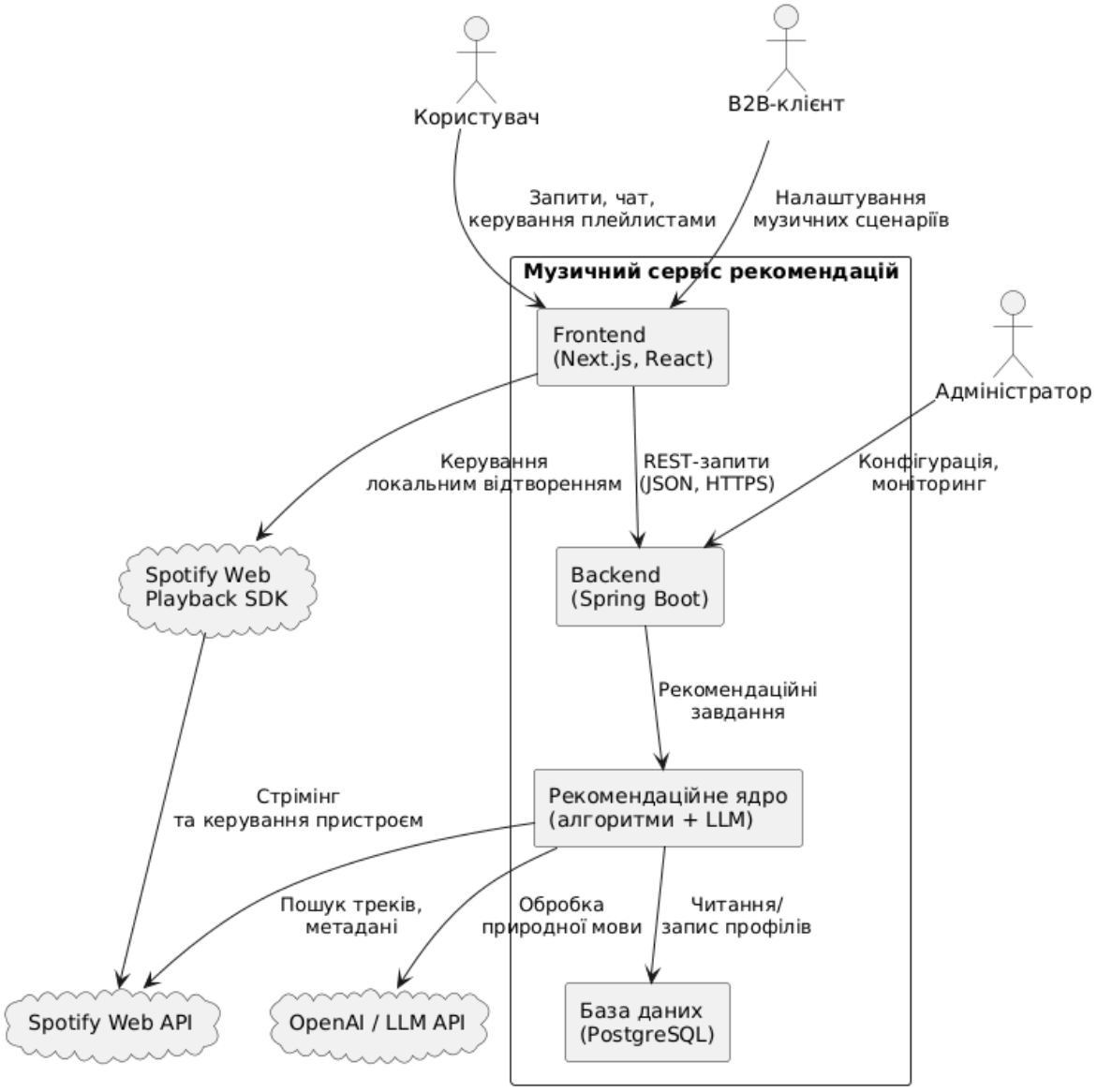


Рисунок 5.1 – UML-діаграма узагальненої архітектури стартап-проєкту

5.2 Аналіз технологічних можливостей реалізації ідей проєкту

Технологічна реалізація стартап-проєкту спирається на стек технологій, який уже був використаний у розробленій інтелектуальній системі музичних рекомендацій: Java 21, Spring Boot, PostgreSQL, Next.js 15, React 19, а також інтеграції з Spotify Web API, Spotify Web Playback SDK та OpenAI API. Однак з позицій стартап-підходу необхідно окремо оцінити не лише наявність, але й реальну доступність

критично важливих технологій та обмеження, пов’язані з політикою використання API зовнішніх сервісів.

Усі базові компоненти – мова програмування, фреймворки, середовище виконання, механізми доступу до БД – є повноцінними, відкрито доступними та добре задокументованими. Вони не створюють бар’єрів для реалізації серверної та клієнтської частин застосунку. На рівні веб-платформи, REST API, схеми збереження даних та клієнтського інтерфейсу, наявний повний контроль над життєвим циклом розробки.

Суттєві технологічні обмеження виникають у частині доступу до музичних даних та формування рекомендацій на їхній основі. По-перше, політика Spotify Web API у 2024–2025 роках була змінена в бік посилення контролю за доступом до певних категорій метаданих. Повноцінний доступ до інформації про треки (розширені метадані, жанрові описи, аудіо аналітика) та частини рекомендаційного функціоналу надається лише додаткам, що пройшли ретельну модерацію та отримали схвалення платформи. Заявки розглядаються індивідуально, а схвалення залежить від прозорості бізнес-моделі, дотримання політик конфіденційності та цілей використання даних користувачів[12].

Таким чином, на етапі прототипу та MVP не має гарантій доступу до повного набору функцій Spotify API, що обмежує можливість масштабування рекомендаційної логіки, особливо якщо йдеться про комерційну експлуатацію продукту. Фактично, технічно API існує, але з практичного погляду є частково недоступним без формальної процедури погодження із Spotify.

По-друге, використання OpenAI API як ядра рекомендацій має низку обмежень. LLM попри їхню гнучкість у роботі з природною мовою, не є спеціалізованими системами рекомендацій. Вони можуть генерувати списки треків, але:

- схильні до повторення однакових рекомендацій у схожих контекстах;
- можуть упереджено віддавати перевагу популярним трекам;
- не мають доступу до персоналізованої історії прослуховувань користувача в реальному часі;
- не гарантують стабільної якості рекомендацій із запиту в запит.

Отже, OpenAI API доцільно розглядати як інструмент для семантичної інтерпретації запитів і генерації первинних кандидатів, проте не як самодостатнє рекомендаційне ядро.

5.2.1 Можливі шляхи подолання технологічних обмежень

Попри наведені обмеження, існує низка напрямів, які дають змогу зменшити залежність від політики окремих API та підвищити технологічну стійкість стартап-проєкту.

1. Офіційне отримання розширеного доступу до Spotify API. Стратегічно стартап може орієнтуватися на проходження процедури схвалення з боку Spotify. Це передбачає:

- підготовку прозорої бізнес-моделі та політики обробки персональних даних;
- оформлення публічної документації та політики конфіденційності;
- подання заявки на розширений доступ до Spotify API;
- поступове розширення функціоналу відповідно до вимог платформи.

2. Побудова власної рекомендаційної моделі. Можлива розробка власного рекомендаційного ядра, що базується на:

відкритих музичних дата сетах (із жанровими мітками та описами);

локальній обробці аудіосигналу (темп, енергія, спектральні характеристики тощо);

анонімних даних користувачів, які дають згоду на використання своїх вподобань для покращення сервісу.

У цьому випадку Spotify використовується переважно як транспортний канал відтворення (через Playback SDK) та джерело базових ідентифікаторів треків, а самі рішення щодо схожості треків і релевантності добірок приймаються локально, без критичної залежності від доступу до закритих метаданих.

3. Гібридна схема з використанням LLM як семантичного шару. OpenAI може виконувати роль:

- інтерфейсу обробки природномовних запитів;
- генератора початкових кандидатів;

- інструмента для пояснення рекомендацій (генерація текстових пояснень, описів).

Натомість остаточний вибір треків, перевірка на дублікати, відповідність жанровому профілю та історії користувача здійснюються локальними алгоритмами. Це дозволяє компенсувати нестабільність LLM, використовуючи її переваги для frontend семантики, а не для ядра прийняття рішень.

Узагальнюючи, можна стверджувати, що технологічна здійсненність стартап-проєкту є підтвердженою за рахунок доступності базових платформних технологій, але потребує додаткових стратегічних та інженерних рішень для подолання зовнішніх обмежень, пов’язаних із доступом до Spotify API та використанням OpenAI як рекомендаційного ядра.

5.3 Аналіз ринкових можливостей запуску стартап-проєкту

Ринкові можливості стартап-проєкту визначаються як загальними тенденціями розвитку ринку потокової музики, так і зростанням актуальності індивідуальних рекомендацій і сервісів, що поєднують штучний інтелект із повсякденним користувацьким досвідом. Домінуючими сервісами на ринку є Spotify, Apple Music, YouTube Music, однак їхні рекомендаційні системи здебільшого функціонують як закриті “чорні ящики”, де користувач має мінімальні можливості впливу на логіку роботи.

Попит на подібні до запропонованого стартапу рішення проявляється у кількох сегментах:

- індивідуальні меломани, які активно досліджують нову музику;
- творці контенту, що використовують музику як частину свого бренду;
- малі підприємства сфери послуг, де музика впливає на атмосферу;
- платформи й сервіси, яким потрібен зовнішній рекомендаційний модуль (через API).

Аналіз конкурентів показує, що основні сервіси пропонують:

- автоматичні плейлисти;
- рекомендації на основі історії прослуховувань і схожості треків;

- базові mood/genre-плейлисти.

Натомість запропонований стартап робить акцент на:

- діалоговій взаємодії (чат-асистент);
- прозорому поясненні логіки рекомендацій (через текстові описи);
- кращій керованості (користувач може задавати контекст, сценарій, обмеження).

Таблиця 5.1 – SWOT-аналіз стартап-проєкту

Сильні сторони	**Можливості**
Поєднання класичних алгоритмів і LLM; Глибока інтеграція з особистою музичною бібліотекою користувача; Гнучкий діалоговий інтерфейс; Можливість масштабування через API;	Зростаючий попит на персоналізовані AI-сервіси; Вихід на B2B-сегмент; Інтеграція з іншими платформами (відео, подкасти, ігрові сервіси);
Слабкі сторони	**Загрози**
Залежність від політики зовнішніх API; Потреба в додаткових обчислювальних ресурсах для AI-компонентів; Складність пояснення користувачу внутрішньої логіки рекомендацій;	Посилення регуляторних вимог до використання персональних даних; Активні дії конкурентів, які можуть швидко впровадити подібний функціонал; Зміна умов використання зовнішніх API;

Узагальнюючи, ринок музичних рекомендацій характеризується високою конкуренцією, але водночас і значним простором для нішевих рішень, що підвищують якість, прозорість та керованість рекомендаційного досвіду.

5.4 Розроблення ринкової стратегії

Ринкова стратегія стартап-проєкту базується на поєднанні концентрованого та диференційованого маркетингу. На початковому етапі доцільно таргетувати вузький сегмент – активних користувачів Spotify, які:

- мають достатню історію прослуховувань;

- вже користуються базовими рекомендаційними функціями;
- відкриті до експериментів з AI-сервісами.

У подальшому, після стабілізації ядра продукту та накопичення досвіду, можливе розширення на:

B2B-сегмент (готові музичні сценарії для бізнесу);

інтеграційні рішення (API-доступ для інших сервісів);

преміум-підписки для активних користувачів.

У термінах моделі 4P:

Product - AI-сервіс музичних рекомендацій з функціями персональних добірок, чат-асистента, інтерактивного плеєра, сценаріїв для різних видів діяльності.

Price - базовий функціонал безкоштовний, розширені можливості та B2B-інтеграції доступні за підпискою.

Place - веб-платформа, у перспективі – мобільні застосунки і API-доступ.

Promotion - цільова реклама серед користувачів Spotify, активність у соціальних мережах, партнерства з музичними діячами.

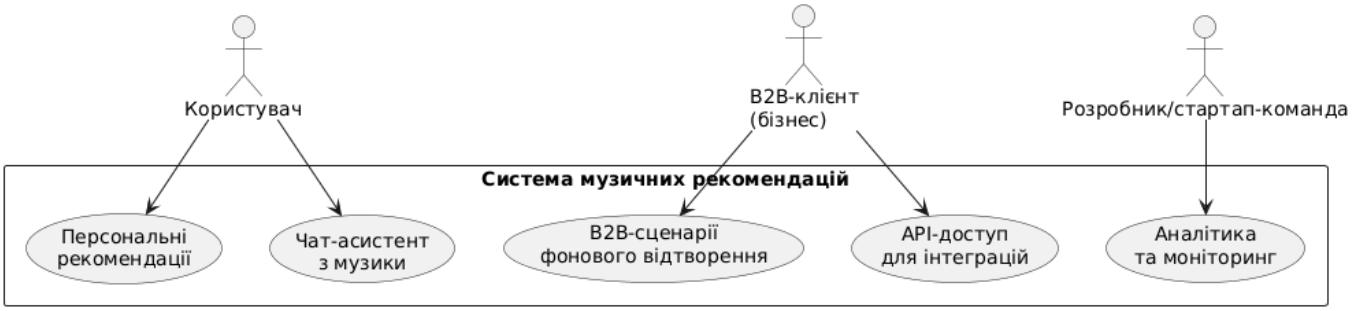


Рисунок 5.2 – UML-діаграма варіантів використання бізнес-моделі

Монетизаційні сценарії можуть включати:

- індивідуальні підписки (місячна/річна оплата за розширений функціонал);
- B2B-тарифи (ліцензування сервісу для бізнесів із можливістю налаштування сценаріїв);
- плату за використання API у сторонніх додатках.

Важливим елементом стратегії є поступовість: на ранніх етапах головний акцент робиться на якості рекомендацій і короткому циклі зворотного зв’язку з

користувачами, а інструменти монетизації вводяться тоді, коли продукт продемонструє стійку цінність для цільової аудиторії.

Висновки до розділу 5

У даному розділі було розглянуто стартап-аналіз розробленої системи музичних рекомендацій. Описано ідею продукту, його унікальність, потенційні сфери застосування, а також проведено аналіз технологічних і ринкових можливостей реалізації. Показано, що, попри наявність повноцінного стеку платформних технологій, доступ до критично важливих функцій Spotify API та обмеження OpenAI-моделей як рекомендаційного ядра вимагають додаткових інженерних і стратегічних рішень, зокрема розроблення власної моделі рекомендацій та отримання розширеного доступу до музичних даних.

Проведений аналіз ринку засвідчив, що, незважаючи на високу конкуренцію з боку глобальних стрімінгових платформ, існує помітний запит на більш прозорі, керовані й контекстно чутливі рекомендаційні сервіси. Стартап-проєкт має потенціал посісти нішу інтелектуального надбудованого шару, що покращує досвід користувача без необхідності відмови від його основного музичного сервісу.

Сформульована ринкова стратегія, що поєднує free-модель, орієнтацію на ранніх послідовників і поступовий вихід у B2B-сегмент, відповідає сучасним тенденціям у сфері цифрових стартапів. Таким чином, проєкт має як технологічні, так і ринкові передумови для еволюції від прототипу до повноцінного комерційного продукту за умови врахування виявлених ризиків та обмежень.

ВИСНОВКИ

У магістерській кваліфікаційній роботі розроблено концепцію, математичне та програмне забезпечення інтелектуальної системи музичних рекомендацій, що поєднує сучасні методи аналізу даних, алгоритми обробки природної мови та інтеграцію з зовнішніми API. Проведений огляд предметної області засвідчив, що існуючі рекомендаційні системи Spotify, Apple Music та інших платформ характеризуються недостатньою прозорістю, обмеженою можливістю впливу користувача на логіку рекомендацій та відсутністю діалогового механізму уточнення вподобань. На основі цього обґрунтовано потребу у створенні гібридної системи нового покоління.

У роботі сформульовано вимоги до функціонування системи, визначено структуру вхідних і вихідних даних та описано сценарії використання. Розроблено математичну модель рекомендаційного ядра, яка включає аналіз жанрової структури альбомів, обчислення жанрових ваг, визначення ключових виконавців, застосування квотного розподілу та алгоритму заповнення добірки (backfill), а також механізми стабілізації відповідей мовних моделей. Побудоване математичне забезпечення створює основу для формування персоналізованих добірок треків, які враховують як поведінкові дані користувача, так і контекст його природномовних запитів.

У межах програмної реалізації створено багатомодульний серверний застосунок на Spring Boot та клієнтський веб-застосунок на Next.js 15. Реалізовано інтеграцію зі Spotify Web API, OpenAI API та Spotify Web Playback SDK, механізми доступу до даних через PostgreSQL, системи логування, обробки помилок і тестування. Практична реалізація підтвердила коректність розроблених алгоритмів та архітектурних рішень.

У рамках стартап-аналізу оцінено ринковий потенціал системи, визначено основні конкурентні переваги, сформовано цільові сегменти та ринкову стратегію. Проведений SWOT-аналіз показав, що проєкт має перспективи комерційного розвитку за умови подолання технологічних обмежень, зокрема щодо політики доступу до музичних метаданих Spotify та підвищення стабільності генеративних

моделей. Окреслено подальші напрями розвитку — створення власної моделі рекомендацій, оптимізація алгоритмів, розширення функціональності та розвиток B2B-сценаріїв.

Таким чином, у роботі досягнуто поставленої мети: розроблено концептуальну, математичну та програмну основу інтелектуальної системи музичних рекомендацій, здатної забезпечувати високий рівень персоналізації та інтерактивної взаємодії. Отримані результати мають практичну цінність і можуть бути використані як фундамент для створення повноцінного інноваційного музичного сервісу.

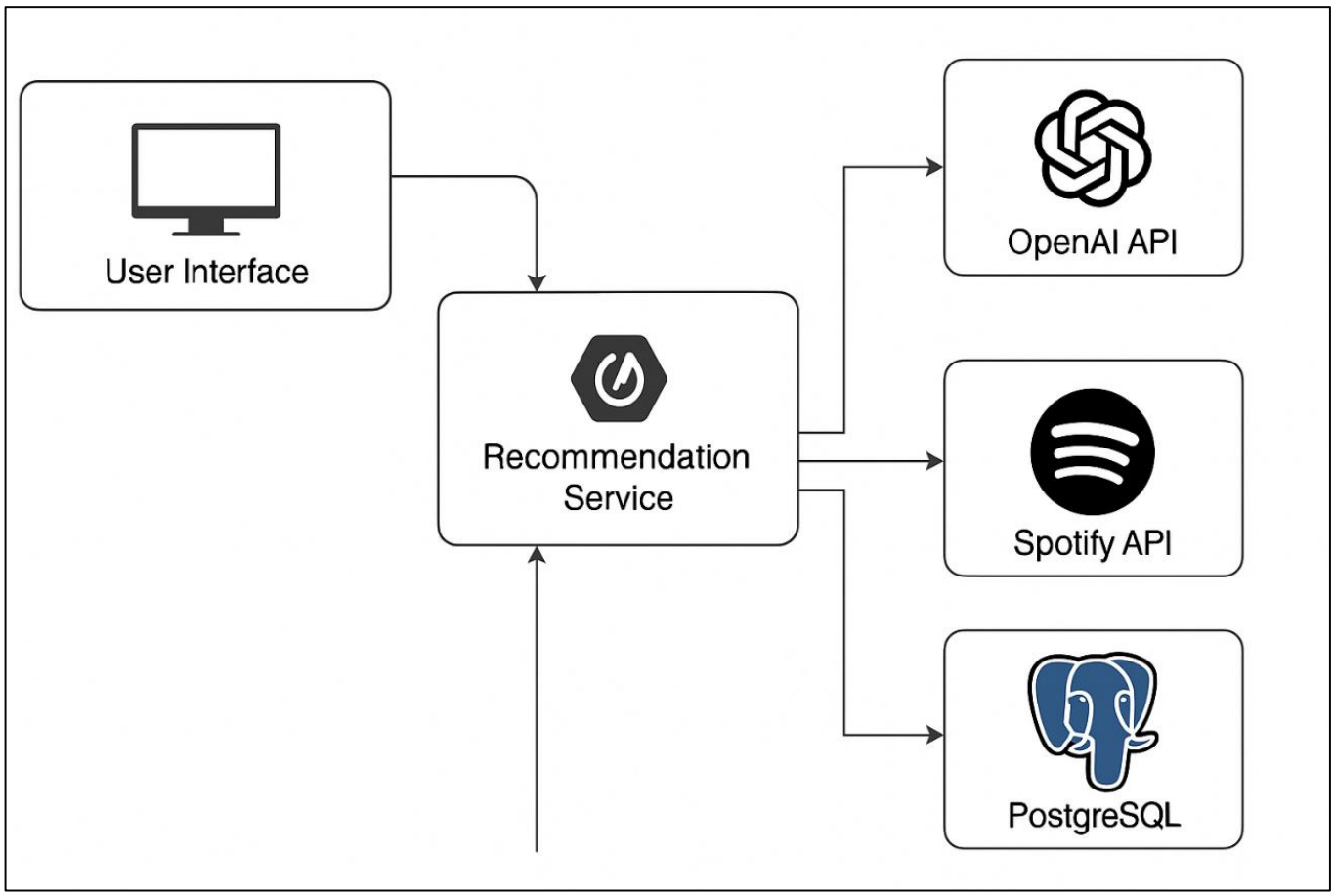
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. BairesDev [Електроний ресурс] – Режим доступу до ресурсу: https://www.bairesdev.com/blog/spotify-engineering/
2. Music Tomorrow [Електроний ресурс] – Режим доступу до ресурсу: https://www.music-tomorrow.com/blog/how-spotify-recommendation-system-works-a-complete-guide-2022
3. Google Research [Електроний ресурс] – Режим доступу до ресурсу: https://research.google/blog/transformers-in-music-recommendation
4. Apple`s Multi-Cloud Strategy [Електроний ресурс] – Режим доступу до ресурсу: https://webpuppies.com.sg/what-cloud-infrastructure-does-apple-use
5. Artist Push [Електроний ресурс] – Режим доступу до ресурсу: https://artistpush.me/blogs/news/apple-music-recommendation-algorithm
6. IBM CFRS [Електроний ресурс] – Режим доступу до ресурсу: https://www.ibm.com/think/topics/collaborative-filtering
7. IBM CBRS [Електроний ресурс] – Режим доступу до ресурсу: https://www.ibm.com/think/topics/content-based-filtering
8. Nvidia Recommendation System [Електроний ресурс] – Режим доступу до ресурсу: https://www.nvidia.com/en-us/glossary/recommendation-system/
9. Robert C. M. Functional Design: Principles, Patterns, and Practices. 1th ed. 2023. 384p.
10. H. C. Thomas et al. Introduction to Algorithms. 2023. 1288p.
11. W3School [Електроний ресурс] - Режим доступу до ресурсу: https://www.w3schools.com/java/default.asp
12. Spotify Community [Електроний ресурс] - Режим доступу до ресурсу: https://community.spotify.com/t5/Spotify-for-Developers/Web-API-Get-Track-s-Audio-Features-403-error/td-p/6654507
13. Kief Morris, Infrastructure as Code: Dynamic Systems for the Cloud Age 2nd ed. 2021. 350p.

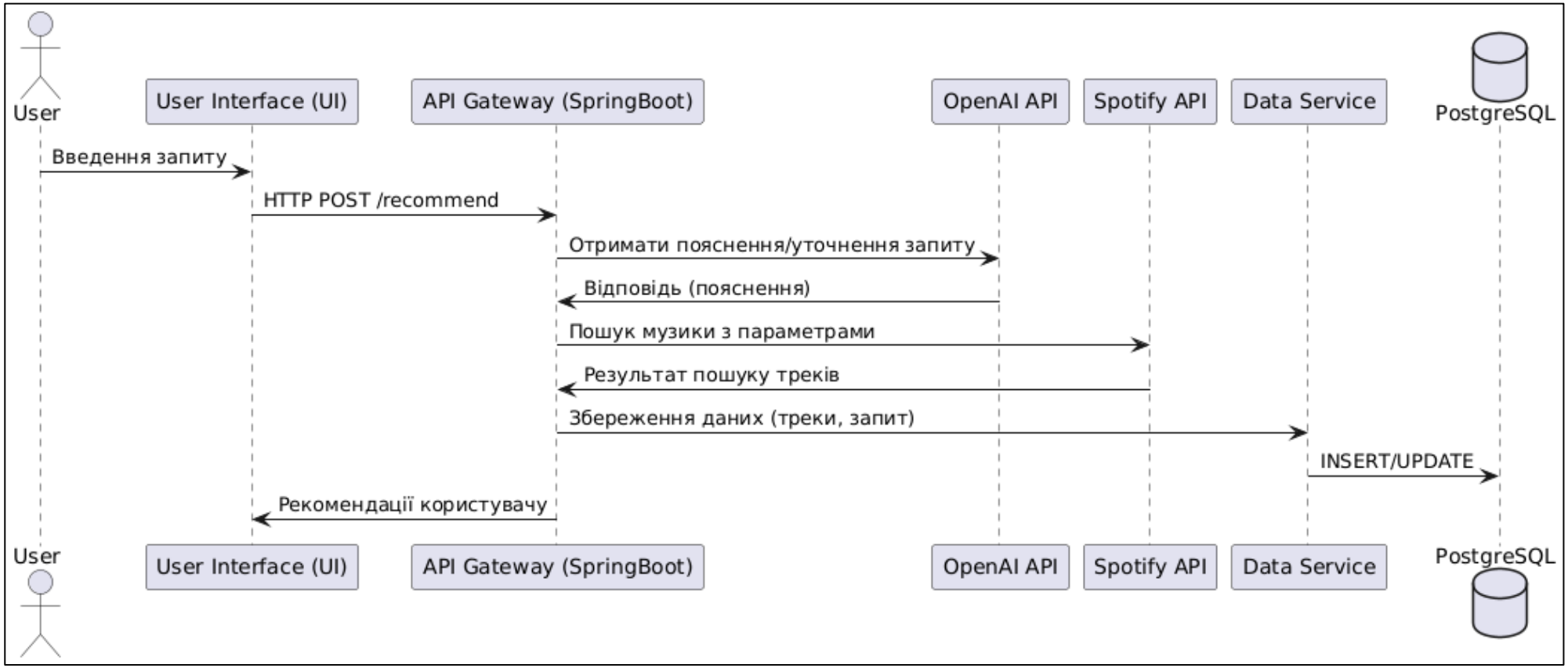
14. David F., Benjamin E., Jason C. Java in a Nutshell: A Desktop Quick Reference 8th ed. 2023. 480p.
15. Greg L. Turnquist, Learning Spring Boot 3.0 3rd ed. 2022. 270p.

ДОДАТКИ

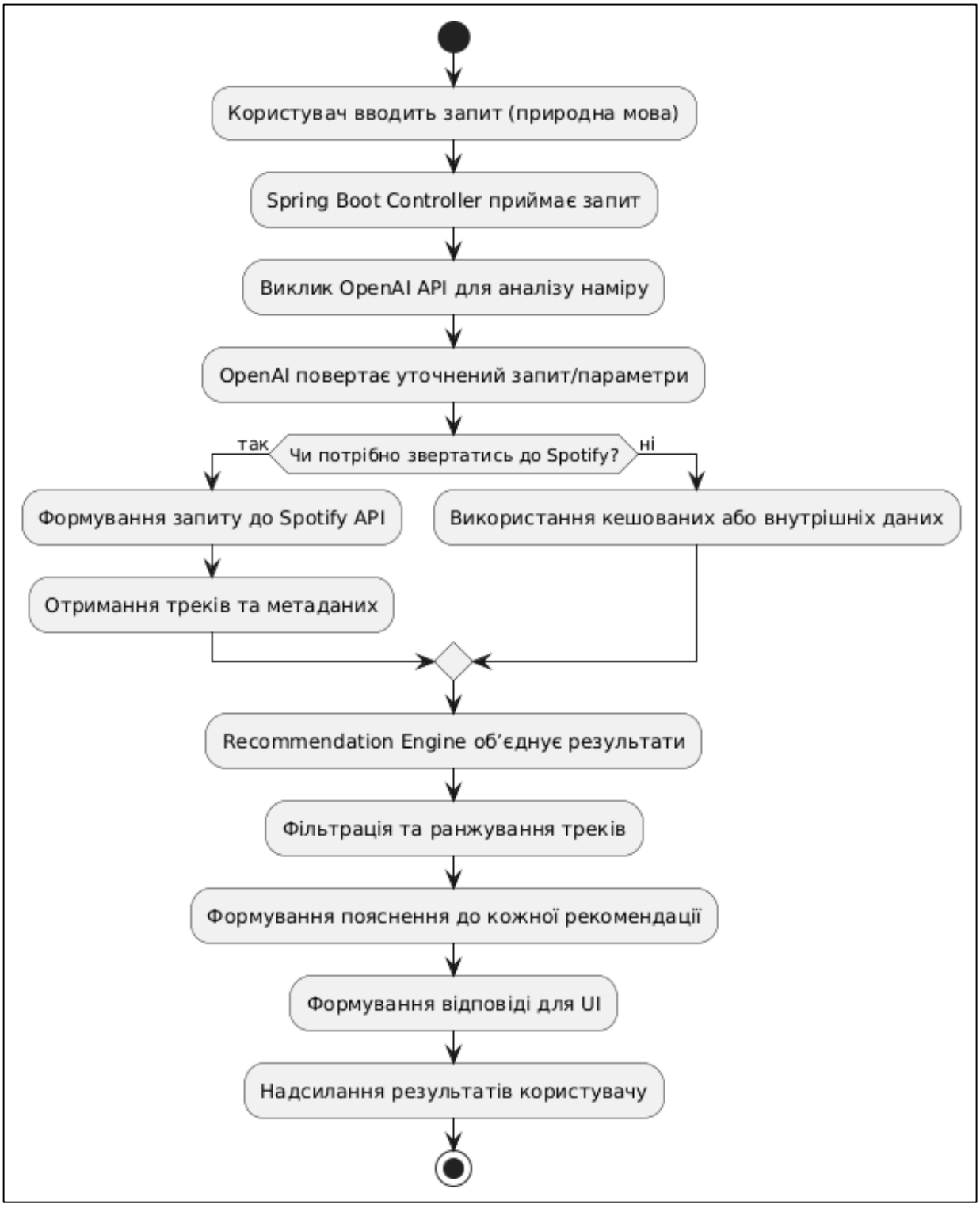
Додаток А. Архітектура інтелектуальної системи. Схема структурна



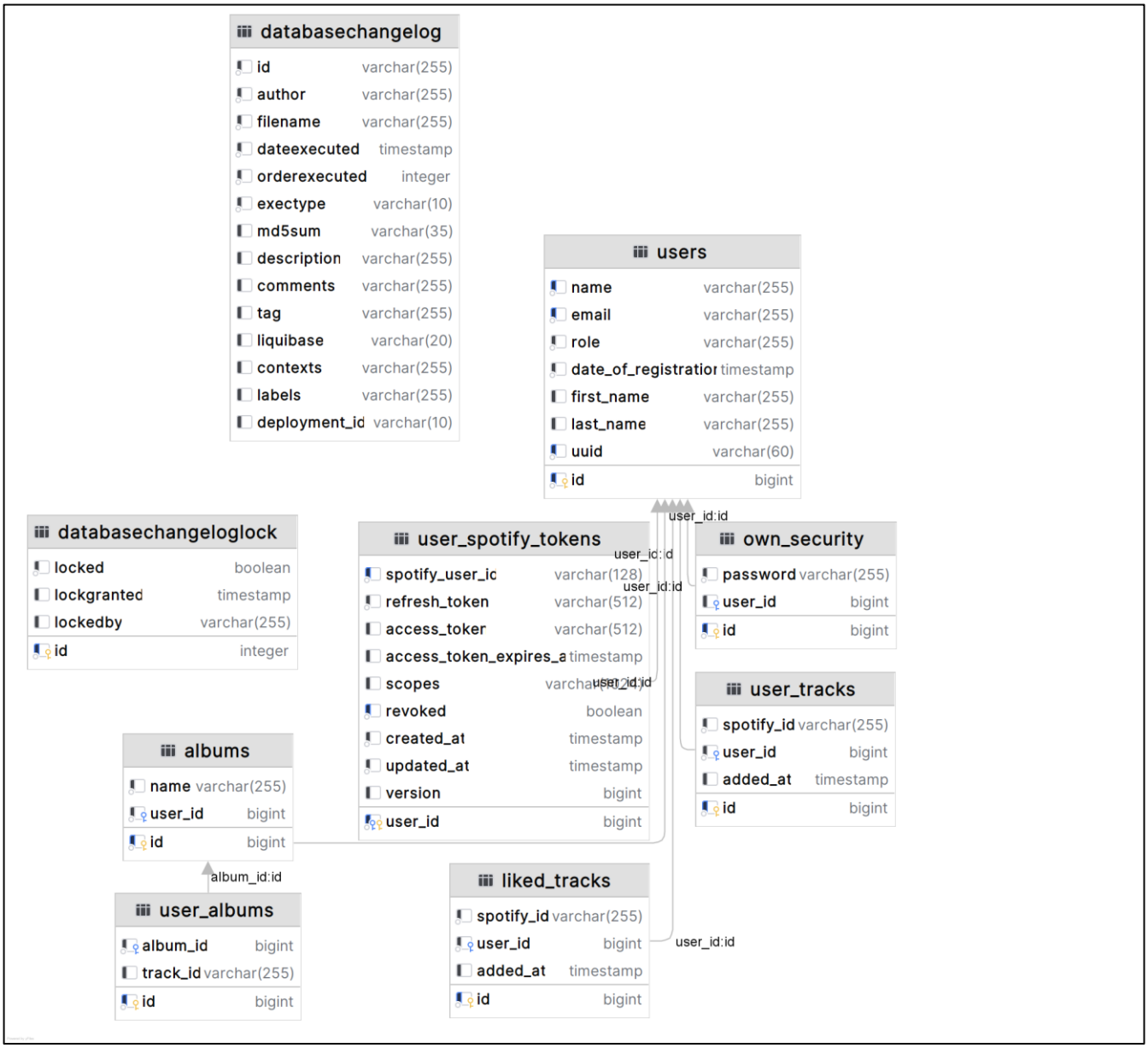
Додаток Б. Виконання інтеграційного запиту системи музичних рекомендацій. Діаграма послідовності



Додаток В. Функціонування системи. Схема алгоритму



Додаток Г. Сховище даних. Структурна схема



Додаток Д. Код програмної реалізації

Додаток Д.1. JWT-авторизація

```
@RequiredArgsConstructor
public class JwtAuthenticationProvider implements AuthenticationProvider {
   private final JwtTool jwtTool;

   @Override
   public Authentication authenticate(Authentication authentication) throws AuthenticationException {
      SecretKey key = Keys.hmacShaKeyFor(jwtTool.getAccessTokenKey().getBytes());

      String email = Jwts.parser()
         .verifyWith(key).build()
         .parseSignedClaims(authentication.getName())
         .getPayload()
         .getSubject();
      @SuppressWarnings({"unchecked, rawtype"})
      List<String> authorities = (List<String>) Jwts.parser()
         .verifyWith(key).build()
         .parseSignedClaims(authentication.getName())
         .getPayload()
         .get(ROLE);

      return new UsernamePasswordAuthenticationToken(
         email,
         "",
         authorities.stream().map(SimpleGrantedAuthority::new).toList());
   }

   @Override
   public boolean supports(Class<?> authentication) {
      return authentication.equals(UsernamePasswordAuthenticationToken.class);
   }
}

@Slf4j
@RequiredArgsConstructor
public class AccessTokenAuthenticationFilter extends OncePerRequestFilter {
   private final JwtTool jwtTool;
   private final AuthenticationManager authenticationManager;
   private final UserService userService;


   private String getTokenFromCookies(Cookie[] cookies) {
      return Arrays.stream(cookies)
         .filter(c -> c.getName().equals("accessToken"))
         .findFirst()
         .map(Cookie::getValue).orElse(null);
   }

   private String extractToken(HttpServletRequest request) {
      Cookie[] cookies = request.getCookies();
```

```
        String uri = request.getRequestURI();
        if (cookies != null && uri.startsWith("/management")) {
            return getTokenFromCookies(cookies);
        }

        return jwtTool.getTokenFromHttpServletRequest(request);
    }
    @Override
    protected void doFilterInternal(HttpServletRequest request,
                                    HttpServletResponse response,
                                    FilterChain filterChain) throws ServletException, IOException {
        String token = extractToken(request);

        if (token != null) {
            try {
                Authentication authentication = authenticationManager
                    .authenticate(new UsernamePasswordAuthenticationToken(token, null));
                Optional<UserVOShort> user =
                    userService.findRegisteredUserByEmail((String) authentication.getPrincipal());
                if (user.isPresent()) {
                    log.debug("User successfully authenticate - {}", authentication.getPrincipal());
                    SecurityContextHolder.getContext().setAuthentication(authentication);
                }
            } catch (ExpiredJwtException e) {
                log.info("Token has expired");
            } catch (Exception e) {
                log.info("Access denied during token authentication: {}", e.getMessage());
            }
        }
        filterChain.doFilter(request, response);
    }
}

@Component
public class JwtTool {
    private final Integer accessTokenValidTimeInMinutes;
    private final String accessTokenKey;
    private final JwtService jwtService;

    public JwtTool(
        @Value("${jwt.accesstoken.lifetime}") Integer accessTokenValidTimeInMinutes,
        @Value("${jwt.accesstoken.key}") String accessTokenKey,
        JwtService jwtService) {
        this.accessTokenValidTimeInMinutes = accessTokenValidTimeInMinutes;
        this.accessTokenKey = accessTokenKey;
        this.jwtService = jwtService;
    }

    public String createAccessToken(String email, Role role) {
        return createAccessToken(email, List.of(role));
    }
```

```
    public String createAccessToken(String email, List<Role> roles) {
        ClaimsBuilder claims = userClaimsBuilder(email, roles);

        return createAccessToken(claims.build(), accessTokenKey.getBytes(StandardCharsets.UTF_8),
            accessTokenValidTimeInMinutes);
    }

    private String createAccessToken(Claims claims, byte[] signature, Integer validTimeMinutes) {
        Date now = new Date();
        Calendar calendar = Calendar.getInstance();
        calendar.setTime(now);
        calendar.add(Calendar.MINUTE, validTimeMinutes);
        return Jwts.builder()
            .claims(claims)
            .issuedAt(now)
            .expiration(calendar.getTime())
            .signWith(
                Keys.hmacShaKeyFor(signature),
                Jwts.SIG.HS256)
            .compact();
    }

    private ClaimsBuilder userClaimsBuilder(String userEmail, List<Role> roles) {
        Long userId = jwtService.findUserIdByEmail(userEmail);
        List<String> roleNames = roles.stream().map(Role::name).toList();

        ClaimsBuilder claims = Jwts.claims().subject(userEmail);
        claims.add("role", roleNames);
        claims.add("userId", userId);

        return claims;
    }

    public String getAccessTokenKey() {
        return accessTokenKey;
    }

    public String getTokenFromHttpServletRequest(HttpServletRequest servletRequest) {
        return Optional
            .ofNullable(servletRequest.getHeader("Authorization"))
            .filter(authHeader -> authHeader.startsWith("Bearer "))
            .map(token -> token.substring(7))
            .orElse(null);
    }
}
```

Додаток Д.2. Код формування рекомендації за вагами жанрів

```
@Override
   public List<SpotifyTrackSearchDto> recommendSongsByAlbum(Long albumId) {
      final int K = 7;

      List<GenreWeightDto> weights = spotifyApiService.getGenreWeightsForAlbum(albumId);
      if (weights == null || weights.isEmpty()) {
         throw new NotFoundException("No genre weights found for album " + albumId);
      }

      Set<String> existingIds = new
LinkedHashSet<>(userAlbumsService.getTracksFromAlbum(albumId));

      List<String> focusArtists = spotifyApiService.getTopArtistsForAlbum(albumId, 5);

      List<Quota> quotas = computeQuotas(weights, K, Math.min(6, weights.size()));
      StringBuilder prompt = new StringBuilder("""
         You are a music assistant.
         Recommend %d tracks as a JSON array of objects with fields "title" and "artist".
         Balance picks by these genre quotas:
         """.formatted(K));
      for (Quota q : quotas) prompt.append("- ").append(q.genre()).append(":
").append(q.count()).append("\n");
      prompt.append("""

         Output JSON only (no markdown), like:
         [
           {"title": "...", "artist": "..."}
         ]
         """);
      if (focusArtists != null && !focusArtists.isEmpty()) {
         prompt.append("\nAdditionally, keep these focus artists in mind (you may include related
collaborations or similar tracks, but do not repeat tracks from the album):\n");
         for (String a : focusArtists) prompt.append("- ").append(a).append("\n");
      }

      String content = chatModel.generate(prompt.toString());
      content = content.replaceAll("(?s)```json|```", "").replaceAll(",\\s*]", "]").trim();
      if (!content.startsWith("[")) {
         int s = content.indexOf('['), e = content.lastIndexOf(']');
         if (s >= 0 && e > s) content = content.substring(s, e + 1);
      }

      List<SongRecommendation> recs;
      try {
```

```
            recs = objectMapper.readValue(content, new com.fasterxml.jackson.core.type.TypeReference<>()
{});
        } catch (Exception e) {
            recs = List.of();
        }

        LinkedHashSet<String> seenIds = new LinkedHashSet<>(existingIds);
        LinkedHashSet<String> seenText = new LinkedHashSet<>();
        List<SpotifyTrackSearchDto> picked = new ArrayList<>();

        for (var r : recs) {
            if (r == null || r.getTitle() == null || r.getArtist() == null) continue;
            String key = normalizeKey(r.getTitle(), r.getArtist());
            if (key == null || !seenText.add(key)) continue;

            SpotifyTrackSearchDto dto = spotifyApiService.searchTrackByTitleAndArtist(r.getTitle(),
r.getArtist());
            if (dto == null || dto.getId() == null) continue;
            if (!seenIds.add(dto.getId())) continue;
            String dtoKey = normalizeKey(dto.getName(), first(dto.getArtists()));
            if (dtoKey == null || !seenText.add(dtoKey)) continue;
            picked.add(dto);
            if (picked.size() == K) break;
        }

        if (picked.size() < K) {
            int need = K - picked.size();
            int top = Math.min(5, weights.size());
            for (int i = 0; i < top && need > 0; i++) {
                String g = weights.get(i).getGenre();
                if (focusArtists != null && !focusArtists.isEmpty()) {
                    for (String artist : focusArtists) {
                        if (need == 0) break;
                        String q = "artist:\"" + artist + "\" genre:\"" + g + "\"";
                        List<SpotifyTrackSearchDto> pool = spotifyApiService.searchTrack(q);
                        for (SpotifyTrackSearchDto t : pool) {
                            if (t == null || t.getId() == null) continue;
                            if (!seenIds.add(t.getId())) continue;
                            String key = normalizeKey(t.getName(), first(t.getArtists()));
                            if (key == null || !seenText.add(key)) continue;
                            picked.add(t);
                            if (--need == 0) break;
                        }
                    }
                }
                if (need > 0) {
```

```
            List<SpotifyTrackSearchDto> pool = spotifyApiService.searchTrack("genre:\"" + g + "\"");
            for (SpotifyTrackSearchDto t : pool) {
                if (t == null || t.getId() == null) continue;
                if (!seenIds.add(t.getId())) continue;
                String key = normalizeKey(t.getName(), first(t.getArtists()));
                if (key == null || !seenText.add(key)) continue;
                picked.add(t);
                if (--need == 0) break;
            }
        }
    }

    return picked.size() > K ? picked.subList(0, K) : picked;
}

private record Quota(String genre, int count) {}

private List<Quota> computeQuotas(List<GenreWeightDto> weights, int K, int topN) {
    List<GenreWeightDto> top = weights.subList(0, topN);
    double sum = top.stream().mapToDouble(GenreWeightDto::getWeight).sum();
    if (sum <= 0) sum = topN;

    int[] base = new int[topN];
    double[] rem = new double[topN];
    int assigned = 0;
    for (int i = 0; i < topN; i++) {
        double exact = (top.get(i).getWeight() / sum) * K;
        base[i] = (int) Math.floor(exact);
        rem[i]  = exact - base[i];
        assigned += base[i];
    }
    int left = K - assigned;
    Integer[] idx = new Integer[topN];
    for (int i = 0; i < topN; i++) idx[i] = i;
    Arrays.sort(idx, (a, b) -> Double.compare(rem[b], rem[a]));
    for (int i = 0; i < left; i++) base[idx[i % topN]]++;

    List<Quota> qs = new ArrayList<>();
    for (int i = 0; i < topN; i++) if (base[i] > 0) qs.add(new Quota(top.get(i).getGenre(), base[i]));
    return qs;
}

private String first(List<String> list) {
    return (list != null && !list.isEmpty()) ? list.get(0) : null;
}
```

```
private String normalizeKey(String title, String artist) {
    if (title == null || artist == null) return null;
    String t = title.toLowerCase().trim();
    String a = artist.toLowerCase().trim();
    t = t.replaceAll("\\s*\\(.*?remaster.*?\\)", "");
    t = t.replaceAll("\\s*\\(.*?version.*?\\)", "");
    t = t.replaceAll("\\s*\\[.*?\\]", "");
    t = t.replaceAll("\\s+", " ");
    a = a.replaceAll("\\s+", " ");
    return t + "|" + a;
}
```