

**НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ІМЕНІ ІГОРЯ
СІКОРСЬКОГО»**

Навчально-науковий інститут атомної та теплової енергетики

Кафедра цифрових технологій в енергетиці

«До захисту допущено»

Завідувач кафедри

Наталія АУШЕВА
“ ” _____ 2026 р.

**Дипломна робота
на здобуття ступеня бакалавр**

За освітньою програмою “Цифрові технології в енергетиці”

Спеціальності 122 “Комп’ютерні науки”

на тему: “Вебзастосунок для потокового передавання музики та музичних
рекомендацій із використанням генеративних моделей штучного інтелекту ”

Виконав: студент 4 курсу, групи ТР-21

Ванін Андрій Віталійович

(прізвище, ім’я, по батькові)

(підпис)

Керівник ас. каф. ЦТЕ, Головакін Микита Андрійович

(посада, науковий ступінь, вчене звання, ім’я, ПРІЗВИЩЕ)

(підпис)

Рецензент Д.т.н., проф. Черноусенко Ольга Юріївна

(посада, науковий ступінь, вчене звання, ім’я, ПРІЗВИЩЕ)

(підпис)

Н.контроль ст.викладач Ольга БЕСПАЛА

(посада, ім’я, ПРІЗВИЩЕ)

(підпис)

Засвідчую, що у цій дипломній роботі немає
запозичень з праць інших авторів без
відповідних посилань.

Студент _____

Київ - 2026

**НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО”**

НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ АТОМНОЇ ТА ТЕПЛОВОЇ ЕНЕРГЕТИКИ

Кафедра ЦИФРОВИХ ТЕХНОЛОГІЙ В ЕНЕРГЕТИЦІ

Рівень вищої освіти – перший (бакалаврський)

спеціальність 122 “Комп’ютерні науки”

Освітньо-професійна програма “Цифрові технології в енергетиці”

ЗАТВЕРДЖУЮ

Завідувач кафедри ЦТЕ

Наталія АУШЕВА

(підпис)

“ ” 2026 р.

ЗАВДАННЯ
на дипломну роботу студенту

Ваніну Андрію Віталійовичу

(прізвище, ім’я, по батькові)

1. Тема роботи “Вебзастосунок для потокового передавання музики та музичних рекомендацій із використанням генеративних моделей штучного інтелекту”

Науковий керівник Головакін Микита Андрійович, ас.

(прізвище, ім’я, по батькові, науковий ступінь, вчене звання)

затверджені наказом по університету від “28” травня 2026 року № 1992

2. Термін подання студентом роботи 04.06.2026

3. Вихідні дані до роботи мова програмування TypeScript, бібліотека React, середовище виконання Node.js, вебфреймворк Express, хмарний програмний інтерфейс Gemini API, середовище розробки VS Code.

4. Перелік питань, які потрібно розробити _____

1) провести огляд сучасних методів семантичного пошуку та систем музичних рекомендацій.

2) визначити основні вимоги до вебзастосунку та обґрунтувати вибір технологічного стеку.

3) дослідити математичний апарат та алгоритми векторного пошуку для реалізації інтелектуального підбору контенту.

4 спроектувати архітектуру програмного забезпечення та фізичну модель збереження даних.

5) озробити серверну логіку векторного пошуку та клієнтський графічний інтерфейс.

6) продемонструвати сценарії використання та виконати оцінку продуктивності системи.

5. Орієнтований перелік ілюстративного матеріалу блок-схема алгоритму семантичного пошуку, алгоритм токенової автентифікації, схема архітектури програмного забезпечення, діаграма послідовності обробки пошукового запиту, ER-модель структури даних, ієрархія компонентів додатка, скріншоти графічного інтерфейсу.

6. Дата видачі завдання 15.09.2025

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів виконання дипломної роботи	Термін виконання етапів роботи	Примітка
1.	Вибір теми роботи	15.10.2025	Виконано
2.	Аналіз методів та засобів розв'язання задачі	20.04.-23.04.2026	Виконано
3.	Розробка архітектури та загальної структури системи	24.04.-27.04.2026	Виконано
4.	Розробка окремих підсистем	28.04.-01.05.2026	Виконано
5.	Програмна реалізація системи	02.05.-07.05.2026	Виконано
6.	Оформлення пояснювальної записки	08.05.-29.05.2026	Виконано
7.	Захист програмного забезпечення	14.05.2026	Виконано
8.	Передзахист	04.06.2026	Виконано
9.	Захист	15.06- 19.06.2026	Виконано

Студент

(підпис)

Андрій ВАНІН
(прізвище та ініціали)

Керівник

(підпис)

Микита ГОЛОВАКІН
(прізвище та ініціали)

АНОТАЦІЯ

Дипломна робота виконана на 70 сторінках, містить 20 ілюстрацій, 9 таблиць, 1 додаток, 25 джерел у переліку посилань.

Мета роботи – полягає у створенні вебзастосунку для потокового передавання музики з інтелектуальними рекомендаціями на базі генеративних моделей штучного інтелекту.

Методи та засоби: алгоритм наближеного пошуку найближчих сусідів для векторного пошуку, векторні представлення характеристик музичного контенту, великі мовні моделі, мова програмування TypeScript, бібліотека React, середовище виконання Node.js, фреймворк Express.

Результат – розроблений програмний інструментарій для прослуховування музики, який підтримує семантичний пошук із налаштуванням рівня різноманітності, автоматичну генерацію візуального настрою, функцію пояснення рекомендацій та інтелектуальне керування плейлістів за текстовими сценаріями.

Ключові слова: ПОТОКОВЕ ПЕРЕДАВАННЯ МУЗИКИ, ШТУЧНИЙ ІНТЕЛЕКТ, РЕКОМЕНДАЦІЙНА СИСТЕМА, ВЕКТОРНИЙ ПОШУК, ГЕНЕРАТИВНІ МОДЕЛІ, ВЕБЗАСТОСУНОК.

ANNOTATION

The thesis is 70 pages long, contains 20 illustrations, 9 tables, 1 appendix, and 25 sources in the reference list.

Purpose of the work – reation of a music streaming web application with intelligent recommendations based on generative artificial intelligence models.

Methods and tools: approximate nearest neighbor algorithm for vector search, vector embeddings of musical content characteristics, large language models, TypeScript programming language, React library, Node.js runtime environment, Express framework.

Result – a developed software toolkit for listening to music that supports semantic search with adjustable diversity levels, automatic visual mood generation, recommendation explanation functionality, and intelligent playlist curation based on text scenarios.

Keywords: MUSIC STREAMING, ARTIFICIAL INTELLIGENCE, RECOMMENDATION SYSTEM, VECTOR SEARCH, GENERATIVE MODELS, WEB APPLICATION.

ЗМІСТ

ВСТУП.....	8
1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ОГЛЯД ІСНУЮЧИХ РІШЕНЬ.....	10
1.1. Постановка прикладної задачі	10
1.2. Огляд методів семантичного пошуку та рекомендацій	11
1.3. Порівняльний аналіз існуючих програмних рішень	13
1.4. Аналіз методів забезпечення пояснюваності рекомендацій	15
2 МЕТОДИ ТА АЛГОРИТМИ ВИРІШЕННЯ ЗАДАЧІ	16
2.1. Алгоритм побудови та пошуку у графі HNSW	17
2.1.1. Принцип ієрархічної навігації у малих світах.....	18
2.1.2. Метрика косинусної відстані між векторами	19
2.1.3. Параметри налаштування індексу	20
2.2. Генерація векторних ембедингів музичного контенту	22
2.3. Алгоритм ранжування результатів із зворотним зв'язком	24
2.4. Генерація візуального настрою запиту.....	25
2.5. Алгоритм інтелектуального керування плейлістів	26
2.6. Механізм пояснюваності рекомендацій	29
2.7. Алгоритм автентифікації та захисту даних	32
РОЗДІЛ 3 ОПИС ПРОГРАМНОЇ РЕАЛІЗАЦІЇ СИСТЕМИ	35
3.1. Архітектура програмної системи.....	35
3.2. Фізична модель даних	38
3.3. Реалізація серверної частини	40
3.4. Реалізація клієнтської частини.....	42
3.5. Інтеграція із зовнішніми сервісами	44
3.6. Конфігурація та розгортання системи.....	46
РОЗДІЛ 4 ТЕСТУВАННЯ ТА ДЕМОНСТРАЦІЯ РОБОТИ.....	48
4.1. Вимоги до обчислювальної техніки та програмного середовища.....	48
4.2. Демонстрація функціоналу розробленої програмної системи	50
4.2.1. Реєстрація та авторизація користувача.....	50
4.2.2. Головний екран та медіатека	52
4.2.3. Семантичний пошук музики	53

4.2.4. Генерація візуального настрою	55
4.2.5. Механізм пояснюваності рекомендацій	56
4.2.6. Робота глобального аудіоплеєра.....	57
4.2.7. Управління плейлістами	58
4.2.8. Інтелектуальне керування плейлістів	59
4.2.9. Зворотний зв'язок та персоналізація.....	61
4.2.10. Налаштування гіперпараметрів семантичного пошуку	62
4.2.11. Завантаження медіафайлів та індексація даних	63
4.3. Постановка та результати обчислювальних експериментів.....	65
ВИСНОВКИ.....	68
ДОДАТОК А.....	72
ДОДАТОК Б	86

ВСТУП

Сучасний етап розвитку інформаційних технологій характеризується стрімким зростанням обсягів цифрового музичного контенту, що породжує проблему інформаційного перевантаження для користувачів. Традиційні системи пошуку в мультимедійних базах даних спираються переважно на лексичний збіг ключових слів та базові метадані. Такий підхід виявляється недостатньо ефективним при спробах знайти музику за певним настроєм, асоціативним контекстом чи загальним семантичним змістом. Зростаючі вимоги до персоналізації користувацького досвіду стимулюють розробку інтелектуальних рекомендаційних систем, здатних глибше аналізувати вхідні дані. Інтеграція технологій штучного інтелекту, зокрема генеративних мовних моделей, відкриває нові можливості для створення високоефективних інструментів семантичного пошуку. Завдяки автоматичній векторизації аудіоконтенту та застосуванню оптимізованих алгоритмів просторового пошуку вдається значно підвищити релевантність результатів. Саме тому розробка сучасного вебзастосунку для потокового передавання музики з розширеними аналітичними функціями є надзвичайно актуальною науково-практичною задачею.

Об'єктом дослідження є процес інтелектуального пошуку та формування персоналізованих рекомендацій у сучасних системах потокового передавання медіаконтенту.

Предметом дослідження є методи та алгоритми семантичного аналізу тексту, векторного представлення даних та наближеного пошуку найближчих сусідів для забезпечення високої релевантності музичних рекомендацій.

Метою роботи є розробка вебзастосунку для потокового передавання музики з інтелектуальними рекомендаціями на базі генеративних моделей штучного інтелекту.

Для досягнення поставленої мети вирішено комплекс завдань. Здійснено аналіз підходів, методів та алгоритмів розв'язання задачі семантичного пошуку музичного контенту. Виконано детальний аналіз програмних засобів, необхідних

для реалізації програмної системи. На основі проведених досліджень розроблено програмне забезпечення вебзастосунку для потокового передавання музики. На завершальному етапі проведено комплексне тестування створеного програмного продукту для підтвердження його коректної роботи.

Методами дослідження, що застосовувалися під час виконання роботи, є методи обробки природної мови для аналізу запитів, методи векторної алгебри для обчислення відстаней між семантичними ембедингами та алгоритми теорії графів для організації швидкого просторового пошуку.

Практичне значення одержаних результатів полягає у створенні готового до використання програмного продукту, який значно скорочує час користувачів на пошук бажаного музичного контенту. Впроваджені алгоритми дозволяють знаходити неочевидні музичні зв'язки та уникати ефекту інформаційної бульбашки завдяки механізмам регулювання різноманітності видачі.

Апробація результатів роботи – основні положення, архітектурні рішення і практичні результати розробки доповідалися та отримали позитивну оцінку на XXIII Міжнародній науково-практичній конференції молодих вчених та студентів «Сучасні проблеми наукового забезпечення енергетики: безпека, стійкість, ІТ та екологічний моніторинг» (м. Київ, 17-22 квітня 2026 р) [23].

Робота складається зі вступу, чотирьох розділів основної частини, висновків, списку використаних джерел та додатку. Загальний обсяг роботи становить близько вісімдесяти сторінок, включаючи ілюстративний матеріал та таблиці.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ОГЛЯД ІСНУЮЧИХ РІШЕНЬ

Створення ефективної системи музичних рекомендацій вимагає глибокого дослідження теоретичних засад та практичних аспектів обробки запитів природною мовою. Детальний аналіз наявних підходів та алгоритмів пошуку дозволяє виявити переваги та недоліки поточних технологій у сфері потокових аудіосервісів. Розуміння обмежень класичного лексичного пошуку є необхідною умовою для обґрунтування переходу до інноваційних методів семантичного аналізу. Вивчення сучасного ринку програмних продуктів допомагає сформулювати чіткі вимоги до архітектури майбутньої вебсистеми.

1.1. Постановка прикладної задачі

Призначенням розроблюваного програмного забезпечення є створення сучасної платформи для інтелектуальних музичних рекомендацій. Існуючі традиційні платформи здебільшого використовують жорсткий алгоритм лексичного пошуку, який суттєво обмежує можливості користувача. На противагу цьому запропонована вебсистема дозволяє знаходити композиції за допомогою запитів природною мовою з урахуванням емоційного забарвлення та контексту. Доцільність створення такого програмного продукту продиктована зростанням обсягів цифрового музичного контенту та потребою у подоланні проблеми інформаційного перевантаження. Стандартні пошукові алгоритми демонструють низьку ефективність при обробці складних семантичних запитів. Це зумовлює гостру необхідність інтеграції сучасних мовних моделей у архітектуру мультимедійних сервісів. Завдяки такому підходу користувачі отримують значно точніші результати пошуку та кращий загальний досвід взаємодії з музичним контентом.

Головною метою розробки є створення розподіленої вебсистеми для забезпечення швидкого потокового відтворення та інтелектуального пошуку музики на основі глибокого семантичного аналізу тексту. Базовим завданням виступає програмна реалізація модуля семантичного пошуку, здатного перетворювати вхідний описовий запит у відсортований перелік найбільш релевантних аудіофайлів. Для підтримки цієї функції необхідно розв'язати задачу автоматичної фонові індексації, яка трансформує структуровані метадані аудіотреків у векторний індекс високої розмірності. Важливим елементом розробки також є реалізація алгоритмів персоналізації користувацького досвіду. Архітектурно система повинна складатися з клієнтської підсистеми для зручної візуалізації інтерфейсу та серверної підсистеми для обробки бізнес-логіки. Серверна частина має забезпечувати генерацію векторних моделей та безпечну комунікацію з хмарними сервісами штучного інтелекту.

Програмне забезпечення повинно підтримувати можливість вільного текстового пошуку музики з урахуванням семантичного значення слів. Також необхідно забезпечити функціонал реєстрації та автентифікації користувачів для доступу до персоналізованих можливостей. Розроблювана вебсистема має підтримувати автоматичну векторизацію аудіофайлів без безпосереднього втручання адміністратора. До нефункціональних вимог належить забезпечення високої швидкості обробки пошукових запитів завдяки застосуванню оптимізованих алгоритмів доступу до векторної бази даних. Цільовою аудиторією розроблюваного продукту є широке коло слухачів, що прагнуть знаходити музичний супровід для специфічних життєвих ситуацій та виходити за межі стандартних алгоритмічних рекомендацій.

1.2. Огляд методів семантичного пошуку та рекомендацій

Сучасні підходи до створення рекомендаційних систем еволюціонували від простих текстових порівнянь до складних багатовимірних обчислень [21].

Розуміння цієї еволюції дозволяє обґрунтувати вибір оптимального математичного апарату для програмної реалізації. Аналіз існуючих методів пошуку розкриває технологічні прогалини, які успішно заповнюються завдяки використанню великих мовних моделей.

Традиційний підхід до організації пошуку в музичних базах даних історично спирається на лексичне співставлення ключових слів [22]. Користувач вводить ім'я виконавця або назву пісні, а пошуковий алгоритм перевіряє наявність точного збігу цих символів у базі даних. Класичним прикладом такого підходу є використання методу частоти терміна та зворотної частоти документа. Такий метод є швидким для прямих фактологічних запитів, проте він виявляється абсолютно неефективним у ситуаціях, коли слухач намагається описати бажану композицію через певний настрій. Спроби покращити якість видачі за допомогою розгалуженої системи метатегів принесли лише частковий успіх, оскільки такий підхід вимагає колосальних людських ресурсів на ручне розмічування всього контенту.

Стрімкий розвиток методів обробки природної мови докорінно змінив парадигму пошуку інформації. Сучасні нейромережеві алгоритми дозволяють аналізувати глибокий семантичний зміст запиту замість механічного порівняння окремих літер. Фундаментом цієї інноваційної технології виступає процес автоматичної векторизації, під час якого будь-який вхідний текст математично перетворюється на багатовимірний масив чисел. Утворені векторні ембединги розташовуються у єдиному семантичному просторі, де близькі за змістом поняття знаходяться геометрично поруч. У музичній індустрії це означає здатність системи на концептуальному рівні розуміти, що запит про енергійну музику для ранкової пробіжки напряду пов'язаний із треками, які мають високий темп та мотивуючий текст. Для уникнення проблеми замикання користувача в інформаційній бульбашці застосовуються спеціальні механізми регулювання різноманітності видачі, що дозволяє додавати до результатів випадкові, але семантично доречні композиції.

Застосування векторних ембедингів вимагає використання специфічних алгоритмів для швидкого знаходження схожих елементів у багатовимірному просторі [18]. Точний математичний розрахунок відстаней між усіма парами

векторів є занадто ресурсомістким завданням для систем реального часу. Тому в сучасній практиці застосовуються алгоритми наближеного пошуку найближчих сусідів, які забезпечують ідеальний баланс між швидкістю та точністю обчислень. Одним із найефективніших методів є використання ієрархічних навігаційних графів у малих світах, які дозволяють логарифмічно зменшити час пошуку. Впровадження алгоритмів просторового пошуку в архітектуру програмного продукту гарантує миттєву видачу релевантних треків навіть за повної відсутності лексичних збігів у ключових словах.

1.3. Порівняльний аналіз існуючих програмних рішень

Сучасний ринок мультимедійних сервісів пропонує користувачам широкий вибір платформ для прослуховування аудіоконтенту. Найбільш популярними серед них є глобальні сервіси, які володіють багатомільйонними бібліотеками треків та розвиненими алгоритмами персоналізації. Попри значні інвестиції у технології машинного навчання, ці платформи здебільшого фокусуються на аналізі історії прослуховувань, залишаючи класичний пошук відносно обмеженим. Користувач змушений оперувати точними назвами або жанрами, що суттєво звужує можливості знаходження неочевидного контенту. Аналіз провідних систем демонструє їх вразливість перед складними описовими запитами. При спробі знайти музику за абстрактною ситуацією більшість існуючих платформ видає результати на основі заздалегідь створених списків відтворення, назви яких випадково збігаються із введеними словами. Такий підхід не враховує реальний настрій самих композицій, що призводить до низької релевантності видачі. Подолання цих технологічних обмежень є критично важливим для забезпечення принципово нового рівня взаємодії між слухачем та музичною платформою. Результати порівняння функціоналу провідних музичних сервісів та розробленої системи наведено у таблиці 1.1.

Таблиця 1.1 – Порівняння існуючих музичних платформ та розробленої системи

Характеристика	Spotify	Apple Music	YouTube Music	Розроблена вебсистема
Лексичний пошук за назвою чи автором	Наявно	Наявно	Наявно	Наявно
Персоналізовані рекомендації за історією	Наявно	Наявно	Наявно	Відсутньо
Семантичний пошук за емоційним описом	Частково	Частково	Частково	Наявно
Автоматична генерація візуального настрою	Відсутньо	Відсутньо	Відсутньо	Наявно
Текстове пояснення причин рекомендації	Відсутньо	Відсутньо	Відсутньо	Наявно
Інтелектуальне керування за сценарієм	Відсутньо	Відсутньо	Частково	Наявно

На противагу існуючим комерційним рішенням впровадження відкритого векторного пошуку дозволяє докорінно змінити логіку взаємодії з програмним продуктом. Аналіз наведених даних показує, що архітектура на базі мовних моделей здатна генерувати точні рекомендації навіть для унікальних пошукових фраз [10]. Інтеграція таких технологій перетворює пошукову стрічку з інструменту фільтрації бази даних на повноцінного інтелектуального асистента. Вибір технологічного стеку для реалізації обумовлений необхідністю забезпечення високої продуктивності при обробці багатовимірних векторів. Використання сучасних вебфреймворків у поєднанні з оптимізованими графовими базами даних забезпечує необхідний рівень швидкодії та масштабованості системи.

1.4. Аналіз методів забезпечення пояснюваності рекомендацій

Проблема недостатньої прозорості штучного інтелекту є одним із головних викликів у сучасних рекомендаційних системах [1]. Користувачі часто не розуміють алгоритмічних причин, через які їм пропонується конкретний медіаконтент, що знижує загальний рівень довіри до платформи. Ця проблема відома як феномен чорної скриньки, коли математична модель видає результат без жодного логічного обґрунтування. Впровадження методів забезпечення прозорості дозволяє подолати цей бар'єр та перетворити пасивне споживання контенту на усвідомлену взаємодію. Сучасні підходи пропонують використання генеративних мовних моделей для автоматичного створення зрозумілих текстових пояснень. Завдяки аналізу акустичних характеристик треку та його зіставленню з оригінальним пошуковим запитом система здатна генерувати аргументовані відповіді. Такий підхід не лише підвищує лояльність користувачів, але й допомагає їм краще розуміти власні музичні вподобання, відкриваючи нові горизонти для дослідження аудіоконтенту.

2 МЕТОДИ ТА АЛГОРИТМИ ВИРІШЕННЯ ЗАДАЧІ

Проектування сучасної рекомендаційної вебсистеми вимагає застосування математично обґрунтованих методів для обробки багатовимірних масивів даних. В основі розробленого програмного забезпечення лежить комплексна інтеграція алгоритмів семантичного аналізу тексту та просторового пошуку найближчих сусідів. Для забезпечення високої точності видачі результатів розроблено специфічні алгоритми ранжування, які враховують не лише векторну близькість, але й зворотний зв'язок від користувача. Важливим етапом створення системи є впровадження механізмів генерації візуального настрою та текстового пояснення причин рекомендації, що базуються на використанні генеративних мовних моделей. Застосування сучасних алгоритмів автентифікації гарантує безпеку обробки персональних даних під час взаємодії користувача з платформою.

Архітектура розробленої системи базується на комплексному поєднанні кількох ключових технологічних напрямків сучасної інженерії програмного забезпечення. По-перше, це методи просторового пошуку у багатовимірних векторних просторах, які забезпечують миттєве знаходження семантично близьких музичних композицій. По-друге, це генеративні мовні моделі, що виконують функції семантичного аналізу, декомпозиції складних запитів та формування пояснень для кінцевого користувача. По-третє, це алгоритми криптографічного захисту персональних даних, які гарантують безпечне зберігання облікових записів та розмежування прав доступу.

Кожен із перелічених напрямків реалізовано у вигляді ізольованого програмного модуля з чітко визначеним програмним інтерфейсом. Така модульна декомпозиція дозволяє незалежно модифікувати окремі компоненти системи без впливу на функціонування інших частин. Наприклад, заміна алгоритму векторного пошуку або оновлення версії мовної моделі не потребуватиме переробки модуля автентифікації чи клієнтського інтерфейсу. У наступних підрозділах детально розглянуто математичні основи, алгоритмічну логіку та принципи функціонування кожного з цих модулів.

2.1. Алгоритм побудови та пошуку у графі HNSW

Математичний апарат пошуку найближчих сусідів відіграє ключову роль у забезпеченні швидкодії та точності рекомендаційної системи. Враховуючи великий обсяг бази аудіофайлів, використання традиційних методів лінійного перебору є недоцільним через надмірно високе обчислювальне навантаження на сервер. Тому для реалізації ефективного векторного пошуку було інтегровано спеціалізований графовий алгоритм, який дозволяє оптимізувати процес знаходження релевантного контенту.

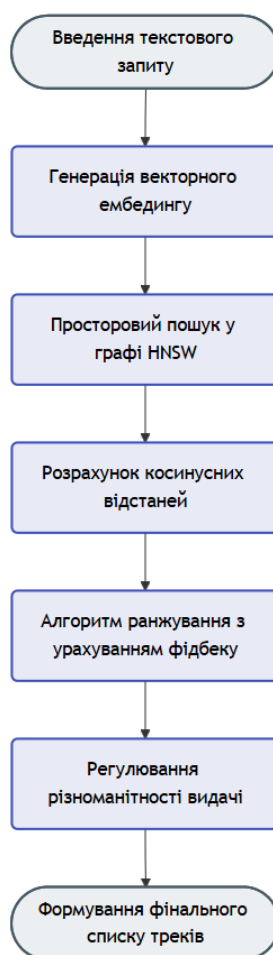


Рисунок 2.1 – Блок-схема алгоритму семантичного пошуку

Наведена схема наочно демонструє послідовність етапів трансформації користувацького запиту від моменту введення тексту до отримання кінцевого

списку релевантних аудіофайлів. Кожен із представлених логічних блоків відіграє критично важливу роль у забезпеченні загальної швидкодії, високої точності видачі та персоналізації рекомендаційної системи.

2.1.1. Принцип ієрархічної навігації у малих світах

Забезпечення миттєвої видачі релевантних результатів серед тисяч аудіофайлів вимагає використання оптимізованої структури збереження даних. Для вирішення цієї задачі було обрано алгоритм ієрархічної навігації у малих світах, який дозволяє здійснювати наближений пошук найближчих векторних сусідів із логарифмічною складністю [11]. Фундаментальна ідея цього методу полягає у побудові багаторівневого графа, де кожен вузол представляє математичну абстракцію окремої музичної композиції. На верхніх рівнях ієрархії розташовуються графові структури з низькою щільністю зв'язків, які слугують своєрідними швидкісними магістралями для початкового наближення до цільової зони. Під час виконання пошукового запиту алгоритм починає обхід із найвищого рівня і поступово спускається до базового графа з максимальною кількістю вузлів. Такий підхід дозволяє швидко відкинути великі кластери нерелевантних треків і зосередити обчислювальні ресурси виключно у релевантній області простору. Особливістю цього алгоритму є використання алгоритму жадібного обходу на кожному рівні графа, що гарантує швидке наближення до шуканого вектора. Завдяки наявності довгих зв'язків між віддаленими вузлами на верхніх рівнях мінімізується ризик застрягання системи в субоптимальних зонах простору. Це забезпечує стабільно високу точність пошукової видачі навіть за умови постійного масштабування музичної бази даних.

Фундаментальною математичною основою алгоритму HNSW є теорія графів та концепція мереж "тісного світу". З математичної точки зору, базовий шар індексу представляється у вигляді неорієнтованого графа. Відповідно до моделі Ваттса-Строгаца, мережа класифікується як "тісний світ", якщо вона одночасно

характеризується високим коефіцієнтом кластеризації та малим значенням середньої довжини шляху між будь-якими двома випадковими вузлами [8].

Математично це означає, що середня відстань L між вузлами в графі зростає логарифмічно залежно від загальної кількості вузлів N . Цю закономірність можна виразити за формулою:

$$L \propto \log(N),$$

де L – середня кількість кроків між будь-якими двома векторами в індексі; N – загальний обсяг бази даних музичних треків.

Завдяки цій логарифмічній залежності, алгоритм забезпечує екстремально швидкий час відгуку, який позначається як $O(\log N)$ [18]. Навіть якщо обсяг бази даних зростає в десятки разів, кількість обчислювальних кроків, необхідних для знаходження найближчих сусідів, збільшиться лише на незначну константу. Це є критично важливою перевагою для систем реального часу.

Для реалізації ієрархічної структури алгоритм застосовує імовірнісний підхід до розподілу вузлів по рівнях. Кожен новий аудіофайл, що індексується системою, автоматично отримує максимальний рівень видимості l за допомогою експоненціального розподілу, який обчислюється за формулою:

$$l = \lfloor -\ln(u) \cdot m_L \rfloor,$$

де u – випадкова величина з рівномірним розподілом в інтервалі $(0, 1]$; m_L – нормалізуючий коефіцієнт, що визначає щільність верхніх шарів.

Цей математичний підхід гарантує, що верхні шари графа міститимуть експоненціально меншу кількість вузлів, утворюючи ті самі "швидкісні магістралі". Під час виконання пошукового запиту система починає обхід із найвищого шару, стрімко наближаючись до цільового вектора, а потім поступово спускається на нижні, більш щільні шари для фінального жадібного пошуку найближчих сусідів.

2.1.2. Метрика косинусної відстані між векторами

Математичною основою для визначення ступеня семантичної спорідненості між текстовим запитом та музичним контентом є метрика косинусної відстані [7].

Оскільки великі мовні моделі генерують багатовимірні вектори, довжина яких може варіюватися залежно від кількості слів у тексті, використання звичайної евклідової відстані є неефективним. Косинусна міра обчислює кут між двома векторами у багатовимірному просторі, ігноруючи їхню абсолютну довжину. Математично ця метрика розраховується за формулою:

$$\cos(\theta) = \frac{(A \cdot B)}{(\|A\| \cdot \|B\|)},$$

де A та B – вектори пошукового запиту та музичного треку відповідно; $\|A\|$ та $\|B\|$ – їхні абсолютні довжини

Завдяки цьому алгоритм фокусується виключно на напрямку векторів, що відображає їхній прихований семантичний зміст. Що менший кут між вектором пошукового запиту та вектором конкретної пісні, то більшою є їхня змістова близькість. Це дозволяє системі з високою точністю знаходити композиції, емоційне забарвлення яких максимально відповідає введеному користувачем тексту. Важливою перевагою цієї метрики є її інваріантність щодо загального обсягу текстового опису пісні. Це гарантує, що трек із коротким, але дуже влучним емоційним описом матиме таку ж високу релевантність, як і композиція з розлогим текстом. Такий математичний підхід ідеально відповідає специфіці музичного контенту, де семантична насиченість переважає над кількістю слів.

2.1.3. Параметри налаштування індексу

Ефективність роботи графового алгоритму напряму залежить від правильного балансування конфігураційних параметрів. Для досягнення оптимального співвідношення між точністю пошуку та обсягом споживаної оперативної пам'яті було налаштовано кілька ключових показників індексу. Головним параметром виступає максимальна кількість двонаправлених зв'язків, які створюються для кожного нового вузла під час його додавання до базового графа. Збільшення цього показника підвищує якість знаходження найближчих сусідів, проте вимагає додаткових обчислювальних ресурсів під час фонові індексації. Іншим важливим налаштуванням є розмір динамічного списку кандидатів, який

використовується під час процесу побудови та обходу графа. Оптимізація цих параметрів дозволила досягти субмілісекундної швидкості обробки пошукових запитів навіть за умови суттєвого збільшення загальної кількості аудіофайлів у базі даних. Завдяки гнучкості цих налаштувань архітектура системи залишається повністю адаптивною до майбутнього масштабування без жодних втрат продуктивності.

Для забезпечення оптимальної продуктивності векторного пошуку архітектура алгоритму передбачає налаштування трьох базових гіперпараметрів. Кожен із них відповідає за баланс між споживанням обчислювальних ресурсів, швидкістю індексації та фінальною релевантністю результатів (Таблиця 2.1).

Таблиця 2.1 – Вплив конфігураційних параметрів алгоритму HNSW

Назва параметра	Призначення	Вплив на продуктивність
M	Визначає максимальну кількість двонаправлених зв'язків для кожного вузла графа.	Збільшення підвищує точність пошуку, але експоненціально збільшує використання оперативної пам'яті.
efConstruction	Задає розмір динамічного списку кандидатів під час побудови (індексації) графа.	Збільшення гарантує кращу якість зв'язків, але суттєво уповільнює процес збереження нових треків у базу.
efSearch	Визначає розмір динамічного списку кандидатів безпосередньо під час пошукового запиту.	Збільшення підвищує точність фінальної видачі за рахунок незначного збільшення затримки відповіді.

Налаштування цих показників вимагає глибокого аналізу апаратних можливостей сервера та очікуваного навантаження. Для розроблюваної системи було експериментально підібрано оптимальну конфігурацію, де параметр M адаптовано для збереження щільних векторів високої розмірності. Правильне калібрування цих параметрів відіграє вирішальну роль у функціонуванні системи, оскільки воно дозволяє знайти ідеальний компроміс між споживанням оперативної

пам'яті, швидкістю індексації нових треків та фінальною точністю пошукової видачі.

2.2. Генерація векторних ембедингів музичного контенту

Процес створення ефективної моделі семантичного пошуку розпочинається з правильної підготовки вхідних даних. Для того щоб мовна модель могла коректно оцінити зміст музичної композиції, необхідно сформувати її комплексне текстове представлення. Програмний алгоритм автоматично об'єднує ключові метадані кожного треку, включаючи ім'я виконавця, назву пісні, альбом, жанрову приналежність та емоційні характеристики. Цей зведений текстовий опис стає основою для подальшої математичної трансформації. Завдяки такому підходу система отримує цілісну картину про кожну композицію, що дозволяє враховувати найдрібніші смислові відтінки під час виконання пошуку.

Після формування текстового опису відбувається звернення до зовнішнього програмного інтерфейсу генеративних мовних моделей штучного інтелекту. Система передає підготовлений текст та отримує у відповідь багатовимірний числовий масив, який називається векторним ембедингом. Розмірність цього вектора становить понад три тисячі значень, що дозволяє закодувати величезний обсяг семантичної інформації. Важливим етапом роботи вебсистеми є процес автоматичної фонові індексації бази даних. Під час кожного запуску серверної частини алгоритм перевіряє наявність нових або оновлених треків, генерує для них відповідні вектори та інтегрує їх у загальний графовий індекс [14]. Це забезпечує постійну актуальність пошукової бази без необхідності ручного втручання адміністратора платформи.

Для досягнення високої точності семантичного пошуку система використовує механізм векторних ембедингів, що базується на архітектурі Transformer. На відміну від застарілих методів обробки природної мови, таких як TF-IDF або Bag-of-Words, які спираються на статистичну частоту появи слів, сучасні мовні моделі

здатні аналізувати глибокий контекст [2]. Математично, процес перетворення вхідного текстового запиту у векторний формат починається з токенизації, після чого кожен дискретний токен відображається у безперервний багатовимірний вектор у просторі, де розмірність дорівнює 3072 для обраної нейромережі.

Саме ці згенеровані вектори згодом використовуються у графі HNSW для розрахунку косинусної відстані, що дозволяє системі знаходити відповідні пісні не за точним лексичним збігом слів, а за їхнім глибинним сенсом [5].

Під час проектування підсистеми семантичного аналізу було проведено порівняльне тестування кількох передових архітектур для генерації векторних ембедингів. Головними критеріями відбору слугували розмірність вихідного вектора, яка визначає здатність моделі кодувати складні лінгвістичні нюанси, максимальний розмір контекстного вікна та загальна продуктивність при розв'язанні завдань кластеризації. Результати аналізу наведено в таблиці 2.2.

Таблиця 2.2 – Порівняльна характеристика моделей генерації ембедингів

Назва моделі	Розмірність вектора, d	Контекстне вікно (токенів)	Ефективність МТЕВ (%)
OpenAI text-embedding-3-large	3072	8191	64.6
Cohere embed-english-v3.0	1024	512	64.5
Google text-embedding-004	768	2048	63.1

Як видно з наведених даних, архітектури з вищою розмірністю вектора демонструють кращі результати при складній семантичній обробці за метрикою МТЕВ. Саме тому для реалізації системи було обрано модель із розмірністю 3072, що гарантує максимально точне захоплення абстрактних понять та музичних настроїв. Велике контекстне вікно також дозволяє алгоритму обробляти розлогі багатокomпонентні запити без втрати змісту на початкових токенах.

2.3. Алгоритм ранжування результатів із зворотним зв'язком

Отримання списку семантично близьких векторів є лише першим кроком у процесі формування фінальної видачі для користувача. Для забезпечення максимальної персоналізації досвіду розроблено комплексний алгоритм ранжування, який враховує не лише математичну відстань, але й історію взаємодії слухача з контентом. Підсумковий рейтинг кожної композиції обчислюється як зважена сума двох показників:

$$\text{Score} = \alpha \cdot S_{\text{semantic}} + \beta \cdot S_{\text{feedback}},$$

де S_{semantic} – коефіцієнт семантичної подібності треку; S_{feedback} – коефіцієнт зворотного зв'язку від користувача; α та β – вагові коефіцієнти, сума яких дорівнює одиниці.

Механізм збору зворотного зв'язку реалізовано через систему позитивних та негативних оцінок, які автоматично зберігаються у спеціалізованому сховищі даних. Якщо користувач раніше позитивно оцінив певного виконавця або жанр, алгоритм надасть відповідним трекам додаткову перевагу при ранжуванні. Окремим інноваційним елементом системи є механізм регулювання різноманітності видачі. Користувач має змогу самостійно визначати рівень строгості пошуку. При мінімальному значенні цього показника алгоритм повертає виключно найближчі математичні збіги. При збільшенні рівня різноманітності система починає підмішувати до результатів випадкові, але семантично доречні композиції, що стимулює відкриття нової незвіданої музики.

Для об'єктивного оцінювання релевантності контенту впроваджено систему вагових коефіцієнтів зворотного зв'язку. Кожна дія користувача в інтерфейсі генерує числовий множник, який безпосередньо впливає на підсумкову позицію треку в списку. Позитивні дії збільшують базову косинусну відстань, а негативні дії песимізують трек, поступово витісняючи його з майбутніх рекомендацій. Логіка подана у таблиці 2.3.

Таблиця 2.3 – Вагові коефіцієнти зворотного зв'язку

Тип дії користувача	Ваговий коефіцієнт	Математичний вплив
Додавання в улюблене	+0.15	Максимальна пріоритезація
Прослуховування до кінця	+0.05	Підтвердження релевантності
Пропуск треку	-0.10	Песимізація видачі
Видалення з плейліста	-0.20	Глибоке блокування

Наведена матриця коефіцієнтів дозволяє алгоритму миттєво адаптуватися до зміни смаків слухача. Замість повного перерахунку всього векторного графа система модифікує лише фінальні оцінки конкретних вузлів. Це забезпечує мінімальне навантаження на серверну частину та миттєву реакцію інтерфейсу.

2.4. Генерація візуального настрою запиту

Важливою складовою сучасного користувацького досвіду є не лише звукове, але й візуальне сприйняття інформації. Для підсилення емоційного зв'язку між слухачем та музичною платформою було розроблено алгоритм автоматичної генерації візуального настрою на основі пошукового запиту. Процес розпочинається з аналізу введеного користувачем тексту за допомогою генеративної мовної моделі. Алгоритм визначає домінуючу емоцію запиту та формує відповідну структуру даних, яка містить текстовий опис настрою, тематичний символ та набір гармонійних кольорів.

Отримані параметри використовуються клієнтською частиною програмного забезпечення для динамічної зміни візуального інтерфейсу. Система автоматично генерує плавний колірний градієнт, який застосовується до фонового банера на сторінці результатів пошуку. Акцентні кольори використовуються для

підсвічування інтерактивних елементів управління, створюючи цілісну та естетично привабливу композицію. Такий інтелектуальний підхід до дизайну перетворює стандартний процес пошуку на захопливий інтерактивний досвід, де візуальне оформлення повністю відповідає емоційному забарвленню знайденої музики.

Для програмної реалізації цього зв'язку алгоритм використовує колірну модель HSL, яка складається з трьох компонентів: тону, насиченості та яскравості. Мовна модель аналізує емоційне забарвлення пошукового запиту та генерує набір конкретних числових значень у форматі шістнадцяткового кодування. Ці значення передаються на клієнтську частину, де застосовуються для динамічної побудови градієнтного фону на сторінці результатів. Акцентні кольори використовуються для підсвічування інтерактивних елементів керування, створюючи цілісний візуальний досвід. Завдяки такому підходу кожен пошуковий запит формує унікальну кольорову палітру, що перетворює процес пошуку музики на повноцінний мультисенсорний досвід.

2.5. Алгоритм інтелектуального керування плейлістів

Сучасний користувач часто прагне отримати не просто набір окремих композицій, а цілісну музичну добірку, яка ідеально відповідає певній життєвій ситуації, складній концепції або специфічному настрою. Традиційні алгоритми генерації списків відтворення зазвичай спираються на просте групування за жанром або виконавцем, що не здатне задовольнити високі вимоги до контекстної релевантності. Для розв'язання цієї проблеми було розроблено багатокроковий алгоритм інтелектуального керування, у якому генеративні мовні моделі виконують функцію віртуального музичного редактора.

Процес розпочинається з отримання від користувача вільного текстового опису бажаного сценарію. Цей текст передається до мовної моделі разом із жорстким системним промптом, який змушує нейромережу повернути результат у

строго структурованому форматі об'єктної нотації. На першому етапі алгоритм формує загальну тематичну структуру майбутньої добірки, що включає генерацію влучної креативної назви, підбір відповідного емодзі та створення короткого опису загального настрою. Ця метаінформація необхідна для візуального оформлення списку відтворення у користувацькому інтерфейсі та створення правильного психологічного налаштування перед прослуховуванням.

Ключовим технічним етапом є декомпозиція загального сценарію. Замість того, щоб шукати всі пісні за одним загальним запитом, система змушує мовну модель розбити початковий сценарій на серію вузьких, високоспецифічних семантичних підзапитів. Наприклад, запит про музику для вечірньої пробіжки розкладається на підзапити про енергійний ритм, мотиваційні тексти та динамічний бас. Для кожного з цих підзапитів паралельно генеруються векторні ембединги, які потім використовуються для просторового пошуку у графі. Отримані результати зливаються в єдиний масив, проходять процедуру дедуплікації та фінального ранжування.

Важливою архітектурною особливістю розробленого підходу є чіткий розподіл обов'язків між генеративною нейромережею та локальною системою пошуку. Мовна модель не намагається самостійно згенерувати назви конкретних треків, оскільки не має доступу до поточного стану медіатеки. Замість цього вона функціонує виключно як семантичний маршрутизатор, формуючи текстові підзапити для локального просторового пошуку HNSW. Такий гібридний алгоритм повністю унеможливорює проблему "галюцинацій" штучного інтелекту, коли нейромережа вигадує неіснуючі пісні. Додатковим критичним етапом є процедура дедуплікації, яка виконується в режимі реального часу. Оскільки один і той самий трек може ідеально відповідати одразу кільком згенерованим підзапитам, система перевіряє глобальний пул уже доданих унікальних ідентифікаторів. Якщо знайдений трек-кандидат уже присутній у масиві добірки, він автоматично відхиляється, що гарантує максимальне розмаїття фінального списку відтворення.

Для мінімізації часу очікування та оптимізації використання обчислювальних ресурсів, система додатково використовує механізм проміжного кешування

згенерованих добірок. Це дозволяє обійти ресурсомісткий етап повторного звернення до мовної моделі та безпосередньо повернути вже сформований масив треків. Такий архітектурний підхід знижує середню затримку відповіді сервера з кількох секунд до мілісекунд, значно покращуючи загальний користувацький досвід. Візуалізацію цього загального процесу наведено на рисунку 2.2.

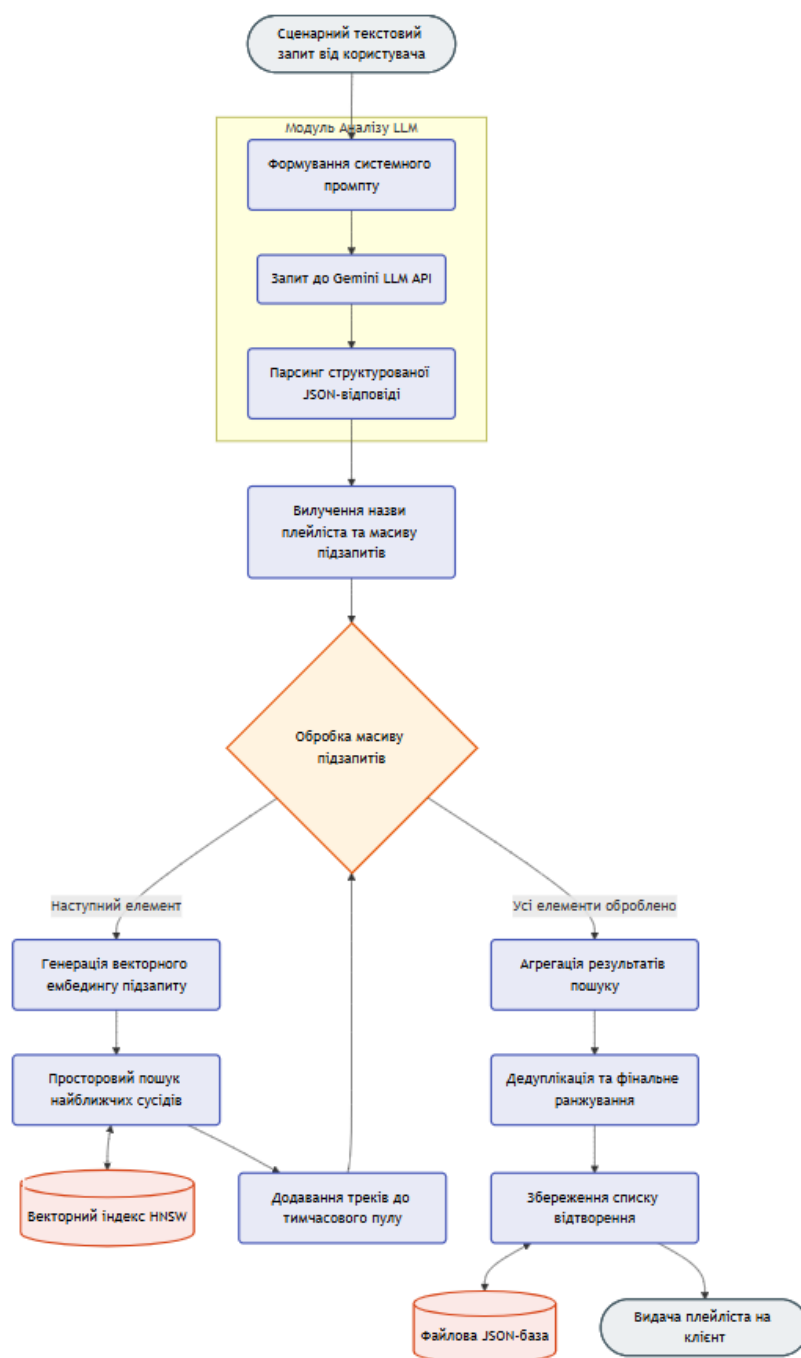


Рисунок 2.2 – Блок-схема алгоритму семантичного пошуку

Наведена блок-схема підтверджує ефективність обраного багатоступового підходу до генерації музичних добірок. Завдяки декомпозиції запиту вдається суттєво підвищити точність пошуку та уникнути типових проблем монолітних пошукових алгоритмів.

Додатковою перевагою обраної архітектури є її стійкість до масштабування бази контенту. Оскільки кожен згенерований підзапит обробляється незалежно від інших, система підтримує паралельне виконання векторного пошуку. Час генерації добірки залишається стабільним незалежно від загальної кількості проіндексованих аудіофайлів. Крім того, кількість сегментів у плейлісті визначається виключно мовною моделлю на основі складності вхідного сценарію, що забезпечує адаптивну структуру фінального результату.

2.6. Механізм пояснюваності рекомендацій

Проблема довіри користувача до автоматично згенерованих результатів є одним із головних викликів у розробці сучасних інтелектуальних систем. Найчастіше складні алгоритми штучного інтелекту працюють за принципом «чорної скриньки», видаючи готовий результат без жодного обґрунтування того, чому саме було прийняте таке рішення. Для подолання цього бар'єра в архітектуру програмного забезпечення було інтегровано концепцію прозорого штучного інтелекту. Система не лише знаходить семантично релевантні аудіофайли, але й здатна аргументувати свій вибір зрозумілою для людини мовою. Цей інноваційний підхід трансформує пасивний процес споживання контенту в активний та свідомий діалог між слухачем і машиною.

Розроблений механізм пояснюваності базується на глибокому порівняльному аналізі вхідного пошукового запиту та внутрішніх метаданих обраної композиції. Процес ініціюється в момент, коли користувач робить запит на пояснення рекомендації у клієнтському інтерфейсі. Серверна частина отримує ідентифікатор цільового треку та оригінальний текст пошукового запиту. Відразу після цього

відбувається звернення до локальної файлової бази даних для вилучення повної структурованої інформації про композицію, включаючи її жанрову приналежність, емоційне забарвлення та текстовий опис. На основі цих зібраних даних формується спеціалізований аналітичний промпт, який передається до генеративної мовної моделі.

У цьому процесі мовна модель виконує роль віртуального музичного експерта: вона детально зіставляє семантичні точки перетину між початковим наміром користувача та фактичним змістом пісні. У результаті обробки генерується коротке, але максимально аргументоване текстове пояснення того, чому саме цей трек був визнаний найближчим до введеного тексту. Наприклад, якщо користувач шукав музику для навчання, система може пояснити вибір конкретної композиції наявністю стабільного спокійного темпу та відсутністю агресивних інструментальних або вокальних партій, що відвертають увагу. Наявність таких детальних пояснень стимулює слухачів експериментувати з більш складними пошуковими запитами та суттєво підвищує загальний рівень задоволеності від роботи з платформою.

Варто також наголосити, що імплементація підходу Explainable AI у контексті персоналізованих рекомендацій вимагає чіткої оркестрації між різними компонентами програмного забезпечення: від клієнтського інтерфейсу до локального сховища та зовнішнього API мовної моделі. Ця багатоетапність покликана забезпечити не лише логічну обґрунтованість, але й високу швидкість формування генеративної відповіді, оскільки тривалі затримки в наданні пояснень можуть негативно вплинути на безперервність користувацького досвіду. Кожен етап обробки інформації спроектовано таким чином, щоб мінімізувати втрати контексту та гарантувати семантичну цілісність кінцевого повідомлення. Деталізовану структурну логіку взаємодії цих компонентів та послідовність потоку даних у системі проілюстровано на рисунку 2.3.

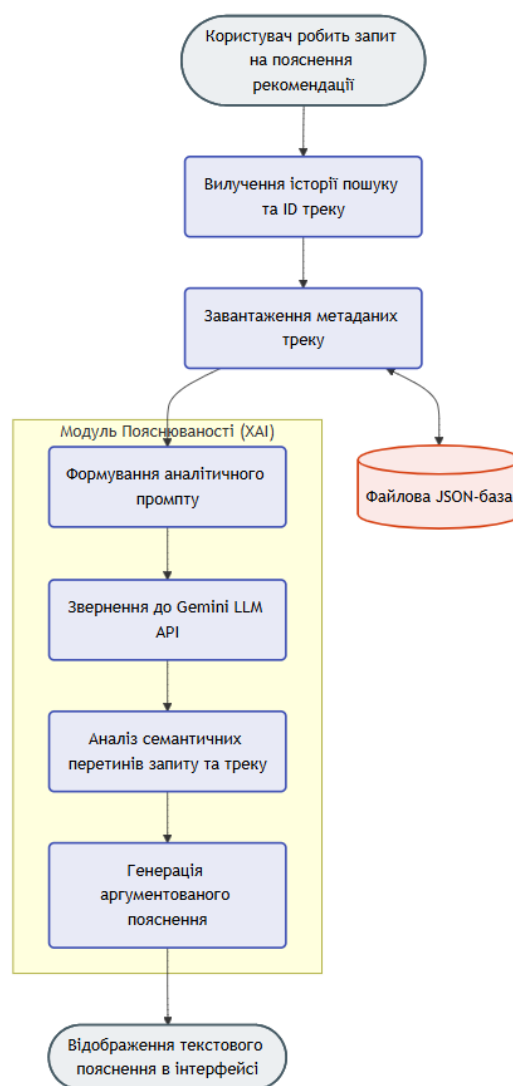


Рисунок 2.3 – Схема роботи механізму пояснюваності рекомендацій

Як видно з наведеної схеми, використання локальної бази даних для попереднього вилучення метаданих треку відіграє ключову роль у процесі пояснення. Таке архітектурне рішення дозволяє значно оптимізувати розмір промпту, знизити споживання токенів та прискорити час генерації відповіді зовнішнім сервісом.

Важливо зазначити, що генеративна модель не має прямого доступу до векторних відстаней чи математичних показників релевантності. Її завданням є виключно лінгвістичний аналіз збігів між текстом запиту та метаданими композиції. Така ізоляція модулів забезпечує незалежність пояснювального механізму від внутрішньої логіки пошукового алгоритму. Це дозволяє в

майбутньому замінити пошуковий рушій без необхідності переналаштування модуля пояснюваності.

2.7. Алгоритм автентифікації та захисту даних

Забезпечення надійної безпеки персональних даних є критично важливою складовою архітектури будь-якої сучасної вебсистеми. Зважаючи на те, що платформа зберігає історію прослуховувань та персональні музичні вподобання, для захисту ідентичності користувачів було реалізовано алгоритм автентифікації на основі вебтокенів формату JWT. Цей підхід дозволяє серверній підсистемі функціонувати в режимі без збереження стану сесії, що суттєво знижує навантаження на оперативну пам'ять та спрощує подальше горизонтальне масштабування застосунку. Замість того, щоб зберігати дані про активні сесії в базі даних, сервер криптографічно підписує спеціальний токен, який містить ідентифікатор користувача.

Процес реєстрації та збереження конфіденційної інформації супроводжується обов'язковим криптографічним хешуванням паролів. Для цього використовується алгоритм bcrypt, який генерує унікальну криптографічну сіль для кожного пароля перед його трансформацією у незворотну хеш-суму [16]. Завдяки цій особливості гарантується абсолютна неможливість відновлення оригінальних паролів навіть у випадку гіпотетичної компрометації локальної файлової бази даних. Алгоритм автентифікації перевіряє права доступу користувача при кожному HTTP-запиті шляхом миттєвої верифікації цифрового підпису токена.

Завдяки такій комплексній архітектурі система забезпечує строге розмежування прав доступу. Кожен користувач отримує доступ виключно до власних списків відтворення та історії взаємодій, що надійно захищає конфіденційність музичних вподобань. Крім того, такий механізм повністю унеможливорює несанкціоноване виконання операцій від імені іншого слухача.

Математичною основою захисту паролів є односпрямована хеш-функція `bcrypt`, побудована на базі блочного шифру Blowfish. Ключовою властивістю цього алгоритму є його адаптивна обчислювальна складність, яка визначається параметром вартості. Процес хешування описується формулою:

$$H = \text{bcrypt}(\text{cost}, \text{salt}, \text{password}),$$

де H – фінальний 60-символьний хеш; `cost` – параметр обчислювальної вартості; `salt` – унікальна 128-бітна криптографічна сіль; `password` – відкритий пароль користувача.

Параметр вартості визначає кількість ітерацій внутрішнього шифрування. За замовчуванням система використовує значення `cost = 10`, що відповідає 1024 ітераціям. Збільшення цього параметра на одиницю подвоює час обчислення хешу, що робить атаку повним перебором практично нездійсненною навіть із використанням спеціалізованого апаратного забезпечення.

Криптографічна сіль генерується автоматично для кожного нового облікового запису. Це гарантує, що два користувачі з ідентичними паролями матимуть абсолютно різні хеш-суми у базі даних. Такий підхід унеможливорює застосування попередньо розрахованих таблиць зворотного перетворення, відомих як райдужні таблиці.

Після успішної валідації пароля система переходить до етапу авторизації, генеруючи маркер доступу у форматі JSON Web Token. Цей стандарт забезпечує безпечну та компактну передачу інформації між клієнтом та сервером у вигляді об'єкта JSON [9]. Структурно кожен токен складається з трьох логічних блоків, розділених символом крапки: заголовка, корисного навантаження та криптографічного підпису.

Такий підхід до авторизації реалізує архітектурний патерн `Stateless`. Сервер не зберігає жодної інформації про поточні сесії в оперативній пам'яті, покладаючись виключно на криптографічну валідацію вхідних токенів. Це забезпечує ідеальну горизонтальну масштабованість бекенду: будь-який вузол кластера може обробити запит клієнта, якщо той надасть валідний JWT.

Загальну візуалізацію логіки перевірки облікових даних та видачі цифрового токена наведено на рисунку 2.4.

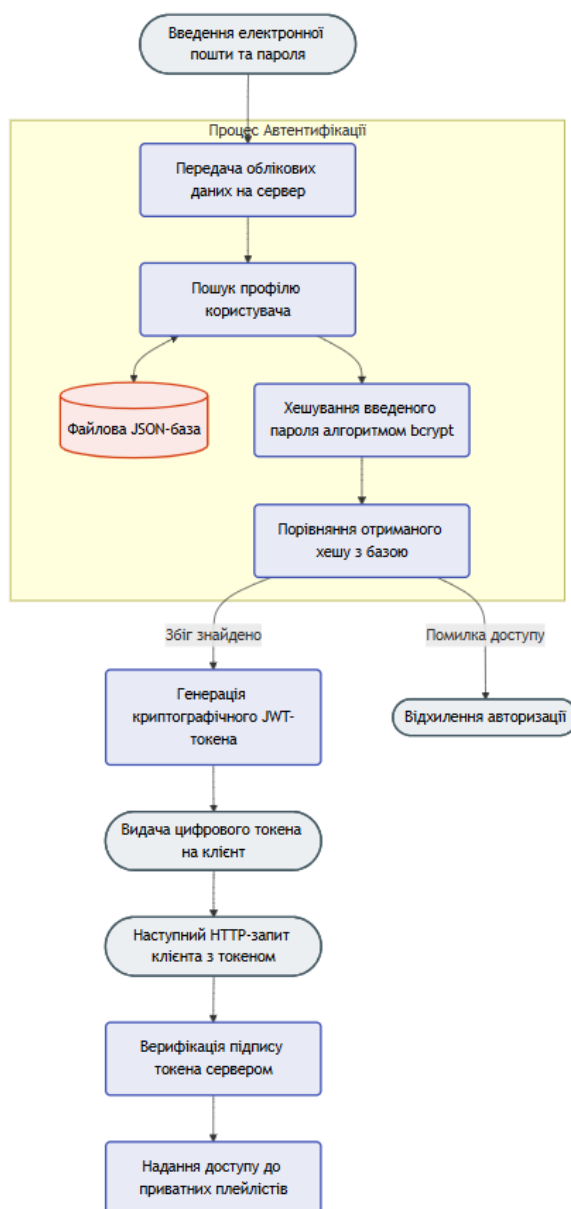


Рисунок 2.4 – Алгоритми токенової автентифікації користувача

Розроблена конвеєрна структура захисту гарантує цілісність даних на всіх етапах взаємодії клієнта із сервером, починаючи від моменту входу в систему і закінчуючи формуванням персоналізованих рекомендацій.

РОЗДІЛ 3 ОПИС ПРОГРАМНОЇ РЕАЛІЗАЦІЇ СИСТЕМИ

Опис теоретичних засад та алгоритмічних підходів, наведений у попередніх розділах, формує міцний фундамент для практичної реалізації програмного забезпечення. Даний розділ присвячено детальному розкриттю процесу розробки системи. Метою цього етапу є перетворення математичних моделей та високорівневих абстракцій на працездатний, оптимізований та масштабований програмний код. У розділі детально розглядаються архітектурні рішення, вибір технологічного стеку, логіка функціонування серверної та клієнтської частин, а також фізичні моделі збереження даних.

3.1. Архітектура програмної системи

Для розроблюваної системи семантичного пошуку музики було обрано багаторівневу клієнт-серверну архітектуру. Такий підхід забезпечує чітке розмежування відповідальності між компонентами візуалізації та обробки бізнес-логіки. Архітектурну схему розробленого програмного забезпечення наведено на рисунку 3.1.

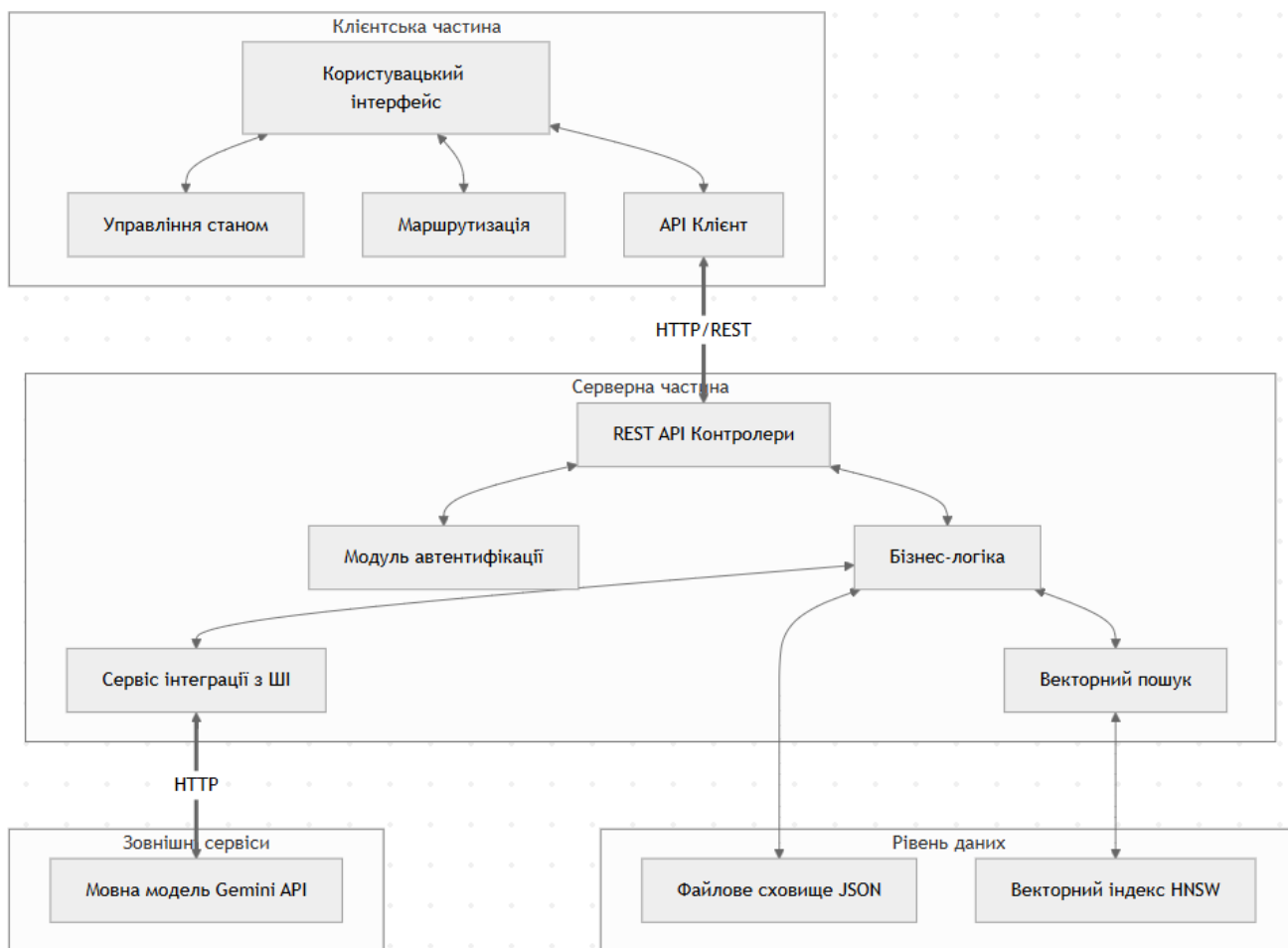


Рисунок 3.1 – Архітектура розробленого програмного забезпечення

Як видно з наведеної схеми, система складається з чотирьох ключових ізольованих рівнів: клієнтського рівня, серверного рівня, рівня даних та рівня зовнішніх інтеграцій. Клієнтська частина відповідає виключно за рендеринг користувацького інтерфейсу, управління локальним станом додатка та маршрутизацію. Серверна частина виконує роль центрального вузла управління та інкапсулює у собі всю обчислювальну бізнес-логіку, включаючи алгоритми ранжування, автентифікації та формування пошукових векторів. Взаємодія між клієнтом та сервером здійснюється за допомогою протоколу HTTP з використанням архітектурного стилю REST. Для обміну даними обрано легковажний формат JSON, що гарантує мінімальні затримки при передачі інформації мережею.

Вибір технологічного стеку для кожного рівня базувався на строгих критеріях продуктивності та надійності. Для реалізації клієнтської частини було обрано

бібліотеку React, яка завдяки концепції віртуального дерева елементів дозволяє миттєво оновлювати складні списки треків без перезавантаження сторінки [19]. Розробка серверної частини здійснена на платформі Node.js із використанням фреймворку Express. Використання асинхронної подієво-орієнтованої архітектури Node.js ідеально підходить для I/O-інтенсивних завдань, таких як паралельні запити до бази даних та звернення до зовнішніх API мовних моделей [15].

Додатковою архітектурною перевагою є використання мови програмування TypeScript як на фронтенді, так і на бекенді. Строга статична типізація даних дозволила створити єдині інтерфейси для спільного використання між обома частинами системи. Це повністю усунуло цілий клас помилок, пов'язаних із невідповідністю структур даних при передачі інформації через мережу. Рівень даних реалізовано у вигляді структурованого файлового JSON-сховища, що суттєво спростило процес розгортання та тестування на ранніх етапах розробки, гарантуючи при цьому достатню швидкість доступу до записів за рахунок використання вбудованих механізмів кешування операційної системи.

Взаємодія клієнтського та серверного рівнів системи побудована на принципах архітектурного стилю REST. Кожен ресурс системи має власну групу маршрутів з визначеними HTTP-методами для виконання CRUD-операцій.

Для наочної демонстрації послідовності обробки запиту на рисунку 3.2 наведено діаграму взаємодії між компонентами системи під час виконання операції семантичного пошуку. Цей сценарій є найбільш обчислювально складним, оскільки він залучає всі чотири архітектурних рівні.

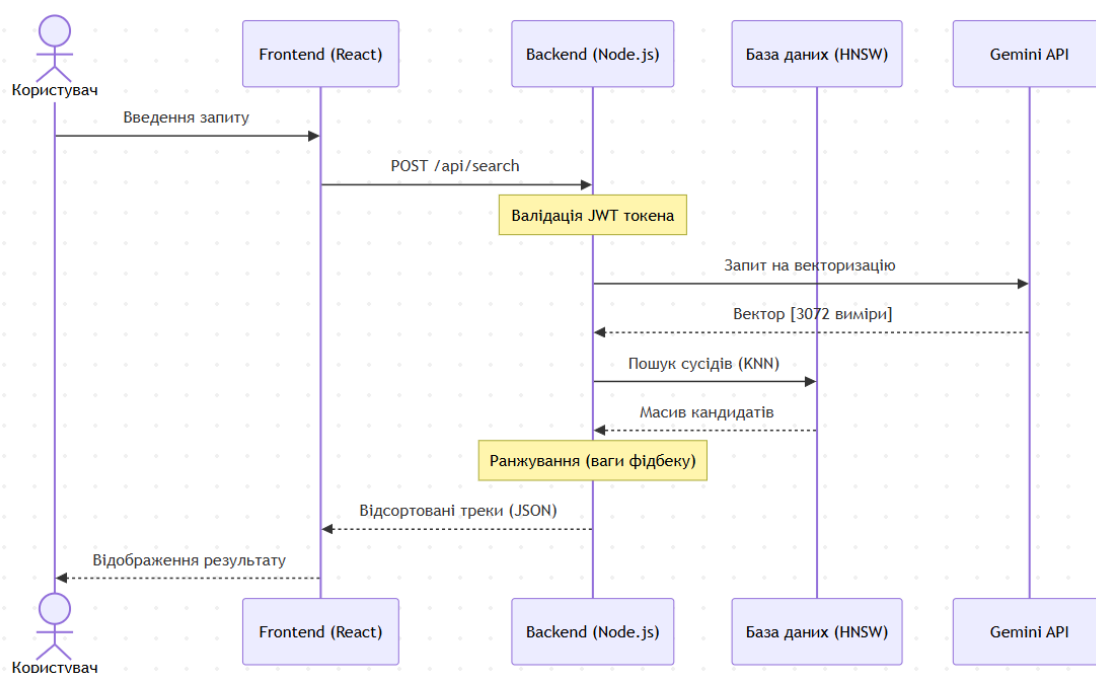


Рисунок 3.2 – Діаграма послідовності обробки пошукового запиту

Як видно з наведеної діаграми, обробка одного пошукового запиту включає ланцюжок із шести послідовних етапів. На першому етапі клієнтський додаток формує HTTP-запит із текстом запиту та токеном автентифікації. На другому етапі серверний контролер виконує валідацію вхідних параметрів та перевірку JWT-токена. На третьому етапі сервіс інтеграції з ШІ надсилає текст запиту до мовної моделі Gemini для генерації числового вектора. На четвертому етапі отриманий вектор передається до HNSW-індексу для пошуку найближчих сусідів. На п'ятому етапі модуль ранжування застосовує вагові коефіцієнти зворотного зв'язку та формує фінальний відсортований масив результатів. На шостому етапі серверний контролер серіалізує відповідь у формат JSON та повертає її клієнту.

3.2. Фізична модель даних

Для забезпечення високої швидкості доступу до даних та спрощення процесу розгортання прототипу системи, рівень збереження даних було реалізовано з

використанням концепції документо-орієнтованого NoSQL сховища на базі локальних JSON-файлів. Такий підхід ідеально підходить для додатків із переважанням операцій читання, оскільки дозволяє завантажувати цілі колекції об'єктів в оперативну пам'ять сервера під час ініціалізації, усуваючи необхідність виконання мережових запитів до окремого сервера баз даних.

Незважаючи на відсутність традиційної реляційної СКБД, внутрішня логіка додатка забезпечує строгу цілісність посилань між сутностями. Схему взаємозв'язків між ключовими об'єктами системи наведено на рисунку 3.3.

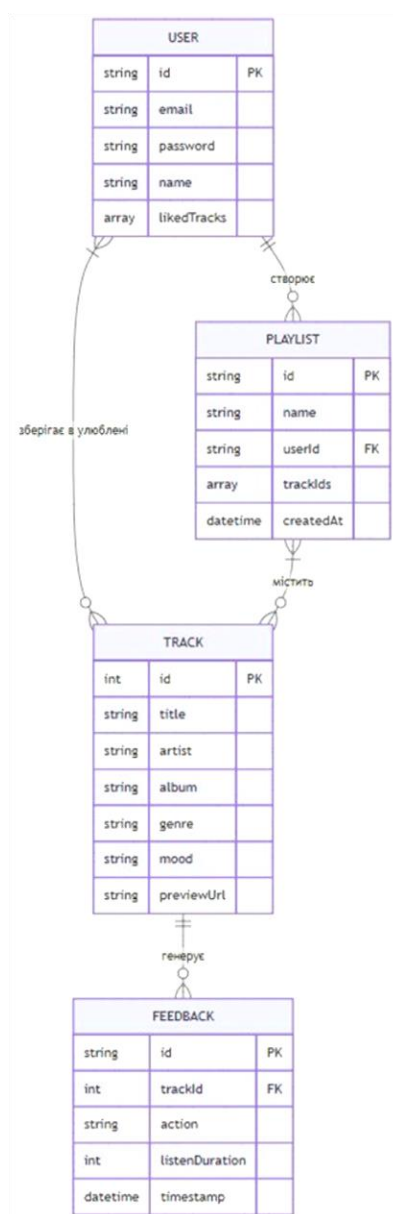


Рисунок 3.3 – ER-модель фізичної структури даних системи

Як видно з наведеної схеми, предметна область розбита на чотири ключові колекції: користувачі, плейлісти, аудіотреки та історія взаємодій.

Колекція користувачів зберігає унікальний ідентифікатор особи, контактний email, криптографічний хеш пароля, ім'я та масив "likedTracks". Збереження зв'язків "багато до багатьох" реалізовано не через класичні проміжні таблиці, а шляхом денормалізації: ідентифікатори вподобаних треків зберігаються безпосередньо всередині JSON-об'єкта користувача як масив рядків. Це дозволяє отримувати повний контекст вподобань користувача за одне звернення до колекції.

Колекція плейлістів містить інформацію про згенеровані системою списки відтворення. Кожен плейліст жорстко прив'язаний до свого творця за допомогою поля "userId". Склад плейліста описується масивом "trackIds", який містить посилання на конкретні музичні композиції. Також фіксується точний час генерації плейліста для можливості сортування списків у хронологічному порядку в користувацькому інтерфейсі. Колекція треків є центральним елементом медіатеки. Кожен запис містить розширені метадані: назву композиції, виконавця, альбом, жанрову приналежність та домінуючий настрій. Важливим полем є "previewUrl", яке містить посилання на фізичний аудіофайл або потоковий потік для відтворення у браузері. Саме текстові метадані з цієї колекції конкатенуються у єдиний рядок для подальшого генерування векторних ембедингів мовною моделлю.

Остання колекція це журнали зворотного зв'язку. Вона акумулює аналітичну інформацію про поведінку користувача: тип взаємодії та тривалість прослуховування. Ці дані критично важливі для модуля персоналізації, оскільки алгоритм ранжування динамічно перераховує ваги результатів пошуку, базуючись саме на записах із цієї колекції.

3.3. Реалізація серверної частини

Серверна частина системи є ключовим обчислювальним вузлом, який консолідує всю бізнес-логіку, забезпечує взаємодію з зовнішніми сервісами та

гарантує цілісність даних. Її розробку здійснено на базі платформи Node.js з використанням веб-фреймворку Express [3]. Вибір Express зумовлений його мінімалістичністю, високою продуктивністю та потужною системою проміжного програмного забезпечення, що дозволяє гнучко налаштовувати конвеєр обробки HTTP-запитів.

Для забезпечення високої надійності коду всю серверну логіку написано мовою TypeScript. Використання статичної типізації дозволило ще на етапі компіляції виявляти помилки доступу до властивостей об'єктів, що є особливо критичним при роботі з розгалуженими структурами даних музичних треків та плейлістів. Контролери розбито на логічні модулі відповідно до предметної області, а саме файли для управління пошуком, плейлістами, треками, автентифікацією та зворотним зв'язком.

Модуль автентифікації реалізує захист системи від несанкціонованого доступу. При реєстрації нового користувача його пароль хешується за допомогою алгоритму bcrypt [17]. Для управління сесіями використовується концепція відсутності збереження стану на сервері на базі маркерів JSON Web Tokens. Під час успішного входу користувач отримує токен, який клієнтський додаток додає у заголовок авторизації до кожного наступного запиту. Для захисту приватних маршрутів реалізовано спеціальний проміжний обробник, який перехоплює запит, валідує криптографічний підпис токена, витягує з нього ідентифікатор користувача і передає його далі по ланцюжку виконання. Для публічних функцій, де авторизація бажана, але не обов'язкова, застосовується альтернативний обробник із послабленими вимогами.

Модуль семантичного пошуку є найскладнішим компонентом серверної частини. Його логіка інкапсулює взаємодію з двома зовнішніми системами: хмарним програмним інтерфейсом мовної моделі Gemini та локальним векторним індексом HNSW [4]. При отриманні пошукового запиту сервер спочатку звертається до мовної моделі для генерації багатовимірного вектора-ембедингу. Цей процес є асинхронним і використовує відповідні механізми мови програмування для запобігання блокуванню основного потоку подій сервера.

Після отримання вектора, сервер передає його до модуля векторного пошуку. Індекс HNSW завантажується в оперативну пам'ять при старті сервера, що гарантує мілісекундну затримку при пошуку найближчих сусідів [6]. Важливою особливістю серверної логіки є динамічне застосування ваг зворотного зв'язку: після того, як алгоритм повертає базовий масив релевантних треків, сервер перевіряє історію взаємодій поточного авторизованого користувача і коригує фінальні оцінки треків, піднімаючи у видачі ті жанри чи виконавців, яким користувач раніше надавав перевагу.

Окремої уваги заслуговує підсистема генерації візуального настрою. Вона реалізована як незалежний маршрут, який приймає текстовий запит і, через спеціально сформований системний промпт до мовної моделі, повертає масив кольорів у форматі HSL. Вся обробка потенційних помилок загорнута у відповідні блоки перехоплення виключень із поверненням стандартизованих HTTP-статусів та зрозумілих текстових повідомлень, що забезпечує стабільну роботу клієнтського інтерфейсу навіть при відмовах зовнішніх сервісів.

3.4. Реалізація клієнтської частини

Головна мета клієнтської частини системи полягає у забезпеченні швидкого, інтуїтивно зрозумілого та чуйного інтерфейсу для відтворення музики та управління плейлістами. Розробку інтерфейсу здійснено на базі популярної бібліотеки React мовою програмування TypeScript, що дозволило побудувати модульну архітектуру на основі незалежних візуальних компонентів. Для швидкого збирання клієнтського коду та оптимізації фінального пакету додатка використано інструмент Vite.

Для управління загальним станом додатка використано вбудований контекстний механізм бібліотеки React. Такий підхід усуває необхідність передачі даних через багато рівнів вкладених компонентів. У системі реалізовано два глобальні провайдери стану. Перший провайдер відповідає за збереження даних

авторизації поточного користувача та його налаштувань. Другий провайдер керує станом глобального аудіоплеєра, забезпечуючи безперервне відтворення музики навіть під час переходу між різними сторінками додатка. Взаємодія клієнтського інтерфейсу з серверною частиною здійснюється асинхронно за допомогою стандартного інтерфейсу Fetch API [12].

Для організації навігації між екранами використано модуль клієнтської маршрутизації [20]. Він дозволяє змінювати відображуваний контент без повного перезавантаження веб-сторінки, що створює ефект роботи повноцінного настільного додатка. Ієрархію базових візуальних компонентів системи наведено на рисунку 3.4.

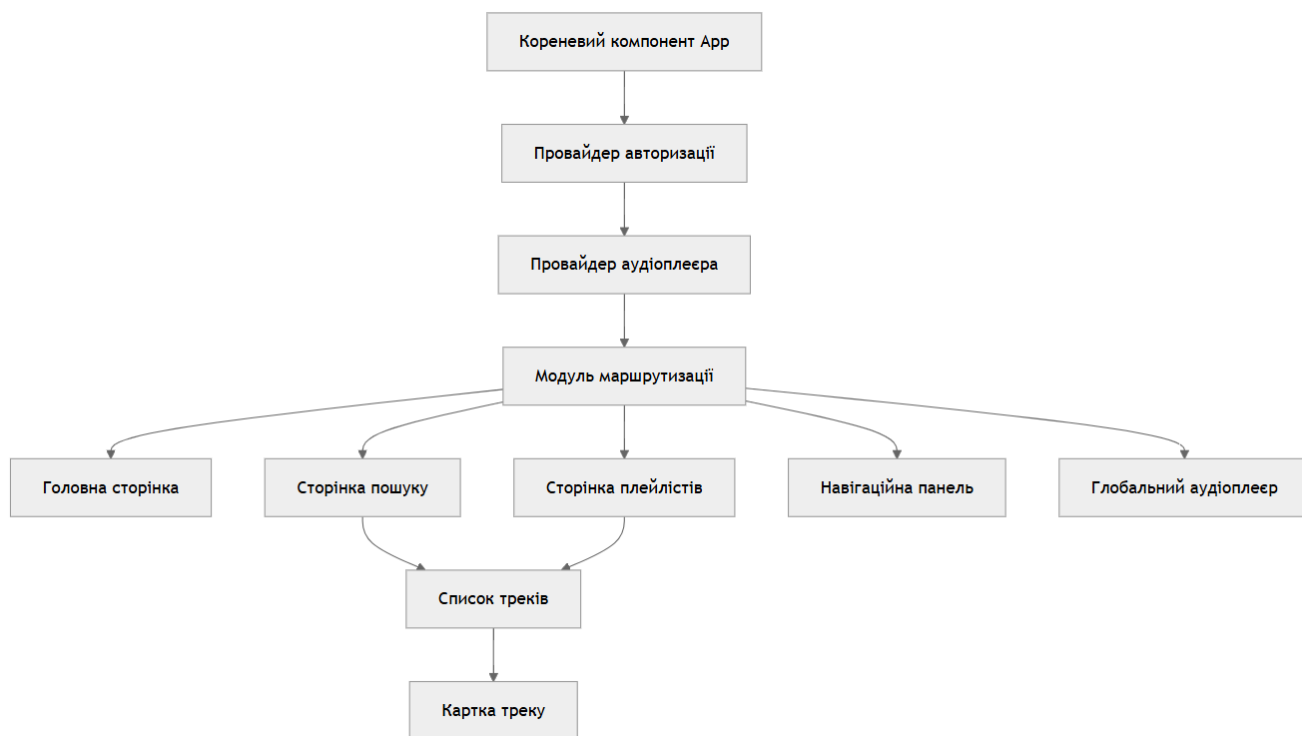


Рисунок 3.4 – Ієрархія компонентів клієнтського додатка

Як видно з наведеної схеми, структура додатка має деревоподібний вигляд. Кореневий компонент огортає всі дочірні елементи необхідними провайдерами стану. Нижче розташовується рівень маршрутизації, який відображає спільні для

всіх сторінок елементи, а саме навігаційну панель та глобальний аудіоплеєр. Основний контент динамічно змінюється залежно від поточної адреси в браузері.

Найбільш функціонально навантаженим елементом інтерфейсу є картка музичного треку. Вона інкапсулює у собі логіку відображення метаданих композиції, обробку подій натискання на кнопки відтворення та механізми зворотного зв'язку. При натисканні на кнопку додавання до улюблених, компонент миттєво оновлює візуальний стан на екрані, паралельно відправляючи асинхронний мережевий запит до серверної частини для збереження дії в базі даних. Такий підхід до оптимістичного оновлення інтерфейсу гарантує відсутність візуальних затримок для користувача.

Для реалізації функції генерації візуального настрою створено окремий компонент обробки кольору. Після отримання масиву кольірних координат від сервера, цей компонент динамічно генерує лінійний градієнт і застосовує його як фонове зображення сторінки пошуку. Зміна фону супроводжується плавними CSS-анімаціями, що значно підвищує естетичну привабливість системи та створює емоційний зв'язок між музичним запитом і візуальним сприйняттям інтерфейсу.

Усі мережеві комунікації з сервером винесено у спеціалізовані сервісні функції, що дозволило ізолювати логіку обміну даними від логіки візуалізації. Відправка запитів здійснюється з використанням сучасних методів обробки асинхронних подій. Для перехоплення можливих помилок мережі розроблено систему глобальних сповіщень, яка інформує користувача про тимчасові технічні складнощі без переривання роботи всього додатка.

3.5. Інтеграція із зовнішніми сервісами

Важливою архітектурною складовою розробленої системи є модуль взаємодії зі сторонніми постачальниками послуг машинного навчання. Оскільки локальне розгортання великих мовних моделей вимагає значних апаратних ресурсів та графічних прискорювачів, було прийнято рішення інтегрувати систему з хмарним

програмним інтерфейсом Gemini. Цей підхід дозволив розвантажити центральний процесор сервера та делегувати найскладніші обчислювальні задачі на оптимізовану інфраструктуру постачальника послуг.

Взаємодія з хмарною моделлю відбувається за двома основними напрямками: генерація багатовимірних векторних представлень тексту та формування структурованих відповідей на основі системних підказок. Процес генерації векторів є критично важливим для роботи алгоритму HNSW. Кожного разу, коли до бази даних додається новий музичний трек, серверна частина автоматично збирає всі його метадані у єдиний текстовий рядок. Цей рядок відправляється через захищене мережеве з'єднання до мовної моделі, яка повертає числовий масив фіксованої розмірності. Отриманий вектор одразу індексується у локальній базі даних.

Другий напрямок взаємодії стосується функцій генерації візуального настрою та пояснюваності рекомендацій. Для цих завдань сервер не просто передає пошуковий запит користувача, а огортає його у складний системний контекст. Програмна логіка сервера динамічно конструює інструкцію для мовної моделі, вказуючи їй роль експерта з музикознавства та психології кольору. Важливою вимогою до таких інструкцій є суворе обмеження формату відповіді. Мовна модель отримує пряму вказівку повертати результати виключно у машиночитаному форматі JSON без жодних додаткових текстових пояснень. Це дозволяє серверній частині безпечно десеріалізувати отриману відповідь і передати її клієнтському додатку для візуалізації.

Особливу увагу при розробці модуля інтеграції приділено питанням інформаційної безпеки. Ключі доступу до хмарного інтерфейсу є конфіденційною інформацією і ніколи не зберігаються безпосередньо у вихідному коді системи. Натомість вони підвантажуються динамічно під час запуску сервера з локальних файлів конфігурації оточення. Такий підхід унеможливорює випадковий витік секретних ключів при публікації коду у відкритих репозиторіях та дозволяє легко змінювати облікові дані без необхідності перекомпіляції проекту.

Для забезпечення безперебійної роботи системи в умовах нестабільного з'єднання або тимчасової недоступності хмарного сервісу розроблено механізм експоненціальної затримки. Якщо при спробі відправки запиту виникає помилка таймауту або перевищення ліміту частоти запитів, сервер автоматично повторює спробу через короткий проміжок часу, який поступово збільшується. Якщо всі спроби виявляються марними, система перериває процес і повертає клієнту відповідний код помилки, пропонуючи спробувати пізніше.

3.6. Конфігурація та розгортання системи

Завершальним етапом реалізації програмного продукту є підготовка інфраструктури для його запуску та розгортання. Оскільки розроблена система складається з двох незалежних частин, клієнтської та серверної, процес їх конфігурації вимагає чіткої синхронізації параметрів середовища. Управління залежностями обох підсистем здійснюється за допомогою стандартного менеджера пакетів екосистеми Node.js. Цей інструмент автоматизує завантаження необхідних зовнішніх бібліотек, гарантуючи використання версій, сумісних із поточним програмним кодом.

Критичним аспектом конфігурації є ізоляція параметрів, специфічних для конкретного середовища розгортання. Для вирішення цієї задачі застосовується механізм змінних середовища. Налаштування підключень до мережевих портів, секретні ключі для криптографічного підпису токенів авторизації та ключі доступу до хмарних сервісів штучного інтелекту винесені у спеціальні приховані конфігураційні файли. Такий архітектурний підхід повністю відокремлює конфігурацію від програмного коду, роблячи додаток переносимим між різними операційними системами без необхідності модифікації початкових файлів.

Для оптимізації процесу розробки та тестування було налаштовано паралельний запуск серверної та клієнтської частин за допомогою єдиної термінальної команди. Використання спеціалізованих скриптів розробки дозволяє

автоматично відстежувати зміни у вихідному коді та миттєво перезавантажувати відповідні компоненти. Це забезпечує безперервний процес тестування гіпотез та швидке виявлення можливих інтеграційних помилок між клієнтом та сервером.

Процес підготовки системи до експлуатації у реальному середовищі відрізняється від локального запуску. Для розгортання клієнтської частини застосовується сучасний інструмент збирання проєктів, який аналізує все дерево залежностей та мініфікує вихідні файли. Результатом роботи цього інструменту є статичний набір оптимізованих скриптів та стилів, які можуть обслуговуватися будь-яким легковажним веб-сервером. Серверна частина також проходить етап транскompіляції, під час якого строгий код перетворюється на оптимізовані інструкції мови JavaScript, готові до виконання у високопродуктивному середовищі Node.js.

Описаний підхід до конфігурації забезпечує високий рівень гнучкості системи. Завдяки ізоляції налаштувань та автоматизації процесів збирання, розроблене програмне забезпечення може бути легко контейнеризоване або розгорнуте на хмарних хостингах без складних архітектурних змін, що підтверджує його готовність до практичної експлуатації.

Для забезпечення безперебійного функціонування системи у реальному часі реалізовано базові механізми моніторингу та логування процесів. Усі критичні системні події, такі як успішна авторизація користувача, створення нових плейлістів або помилки звернення до хмарного сервісу мовної моделі, автоматично фіксуються у локальних журналах подій сервера. Цей підхід дозволяє адміністратору оперативно відстежувати пікові навантаження на систему, виявляти аномальні патерни поведінки користувачів та діагностувати технічні збої на ранніх етапах. Крім того, завдяки збереженню часових міток кожної транзакції істотно спрощується процес аналізу продуктивності векторного індексу при поступовому збільшенні загального обсягу медіатеки. Повний програмний код серверної та клієнтської частин системи наведено у Додатку А.

РОЗДІЛ 4 ТЕСТУВАННЯ ТА ДЕМОНСТРАЦІЯ РОБОТИ

Попередні розділи дипломної роботи були присвячені теоретичному обґрунтуванню алгоритмів та детальному опису архітектури програмного забезпечення. Даний розділ фокусується на практичній площині, а саме на процесі тестування розробленої системи. Головною метою цього етапу є підтвердження працездатності всіх заявлених функцій, перевірка інтуїтивності користувацького інтерфейсу та оцінка продуктивності алгоритму семантичного пошуку. У розділі послідовно розглядаються вимоги до апаратного забезпечення, сценарії взаємодії користувача з системою та результати обчислювальних експериментів із заміру часу відповіді сервера.

4.1. Вимоги до обчислювальної техніки та програмного середовища

Оскільки архітектура системи розділена на клієнтську та серверну частини, вимоги до апаратного та програмного забезпечення для них суттєво відрізняються. Серверна частина виконує роль обчислювального ядра. Для її коректного функціонування необхідно розгорнути середовище виконання Node.js актуальної версії з підтримкою сучасних стандартів мови програмування JavaScript. Оскільки алгоритм векторного пошуку зберігає індекс у оперативній пам'яті для забезпечення мінімальних затримок, критичною вимогою є наявність щонайменше двох гігабайт вільної оперативної пам'яті. Вимоги до центрального процесора є помірними, оскільки найскладніші операції з генерації багатовимірних векторів делеговані хмарному програмному інтерфейсу мовної моделі.

Клієнтська частина є класичним веб-додатком, який виконується безпосередньо у браузері кінцевого споживача. Відповідно, специфічних вимог до

апаратного забезпечення клієнтських пристроїв не висувається. Для коректного відображення динамічного інтерфейсу та безперервного відтворення аудіофайлів необхідний сучасний веб-браузер із підтримкою стандартів HTML п'ятої версії та сучасних функцій мови програмування JavaScript. Додаток адаптовано для роботи як на настільних комп'ютерах, так і на мобільних пристроях, що забезпечує високий рівень доступності сервісу незалежно від обраної користувачем апаратної платформи.

Для забезпечення інформаційної безпеки під час обміну даними між клієнтом та сервером рекомендується використання захищеного протоколу передачі гіпертексту HTTPS. Крім того, для безперебійної роботи системи з генерації векторів та пояснень необхідно забезпечити стабільне підключення серверної машини до глобальної мережі Інтернет.

Більш детальну специфікацію мінімальних апаратних та програмних вимог для розгортання серверної частини наведено у таблиці 4.1.

Таблиця 4.1 – Мінімальні системні вимоги для серверної частини

Компонент	Мінімальні вимоги
Процесор	Двоядерний процесор із тактовою частотою 2.0 ГГц
Оперативна пам'ять	2 ГБ (рекомендовано 4 ГБ для кешування індексу HNSW)
Дисковий простір	1 ГБ вільного простору для бази даних та логів
Операційна система	Ubuntu Linux 20.04 або новіша, Windows 10, macOS 11
Програмне середовище	Node.js версії 18 або новішої
Мережа	Широкосмуговий доступ до мережі Інтернет (для API мовної моделі)

Мінімальні вимоги для клієнтської частини зводяться виключно до наявності сучасного програмного забезпечення, оскільки всі обчислювальні процеси виконуються на стороні сервера. Це наведено у таблиці 4.2.

Таблиця 4.2 – Мінімальні системні вимоги для клієнтської частини

Компонент	Мінімальні вимоги
Пристрій	Настільний комп'ютер, ноутбук, планшет або смартфон
Оперативна пам'ять	Від 1 ГБ
Веб-браузер	Google Chrome 90+, Mozilla Firefox 90+, Safari 14+
Технології	Увімкнена підтримка JavaScript та аудіо-кодеків
Дисплей	Роздільна здатність екрану від 320x568 пікселів

4.2. Демонстрація функціоналу розробленої програмної системи

Для об'єктивної оцінки працездатності програмного забезпечення розроблено серію користувацьких сценаріїв. Вони охоплюють увесь життєвий цикл взаємодії з платформою: від створення облікового запису до використання інтелектуальних функцій семантичного пошуку та генерації музичних добірок.

4.2.1. Реєстрація та авторизація користувача

Першим етапом взаємодії нового клієнта з програмним забезпеченням є процес створення персонального облікового запису та проходження процедури автентифікації. Наявність індивідуального профілю дозволяє системі накопичувати історію взаємодій та формувати персоналізовані музичні рекомендації на основі

алгоритмів ранжування. Графічний інтерфейс екрану реєстрації та авторизації наведено на рисунку 4.1.

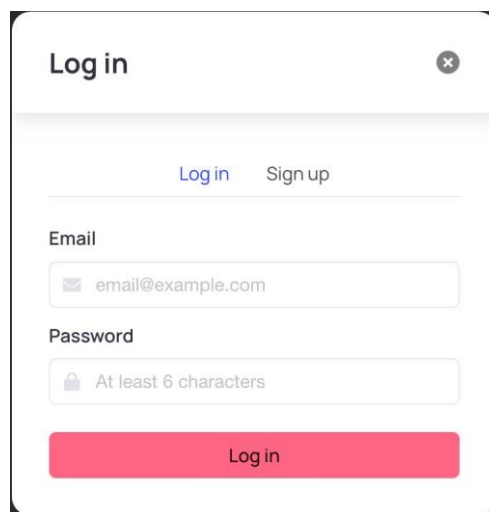


Рисунок 4.1 – Графічний інтерфейс модуля автентифікації користувачів

Сценарій тестування розпочинається із заповнення форми реєстрації. Клієнтська частина додатка виконує первинну перевірку введених даних на правильність формату електронної пошти та мінімальну допустиму довжину пароля. У разі успішної валідації на стороні браузера дані інкапсулюються у мережевий запит та відправляються до серверної частини.

Серверний контролер, отримавши запит, перевіряє відсутність користувача з ідентичною електронною адресою в локальній базі даних. Далі ініціюється процес криптографічного перетворення пароля. Система генерує унікальну сіль та застосовує алгоритм хешування, після чого зберігає лише згенерований хеш, гарантуючи неможливість відновлення оригінального пароля навіть у випадку компрометації файлу бази даних.

Після успішного збереження профілю система автоматично переходить до етапу авторизації. Сервер генерує унікальний маркер доступу за стандартом JSON Web Token, який містить зашифрований ідентифікатор щойно створеного користувача. Цей маркер передається клієнтському додатку, який зберігає його у локальній пам'яті браузера. Завершенням тестового сценарію є успішна

переадресація користувача на головну сторінку додатка з автоматичним підвантаженням персоналізованої стрічки музичних композицій. Відмова в авторизації при введенні хибних облікових даних також була успішно протестована: система коректно повертає відповідний код помилки, а клієнтський інтерфейс відображає зрозуміле текстове попередження без зупинки роботи додатка.

4.2.2. Головний екран та медіатека

Після успішного проходження процедури авторизації користувач потрапляє на головний екран веб-додатка. Цей інтерфейс є центральним хабом системи, оскільки він надає миттєвий доступ до загальної медіатеки та інструментів навігації. Основною задачею головного екрану є швидке підвантаження та безперебійне відображення великого масиву музичних композицій без втрати плавності роботи браузера. Зовнішній вигляд головної сторінки наведено на рисунку 4.2.

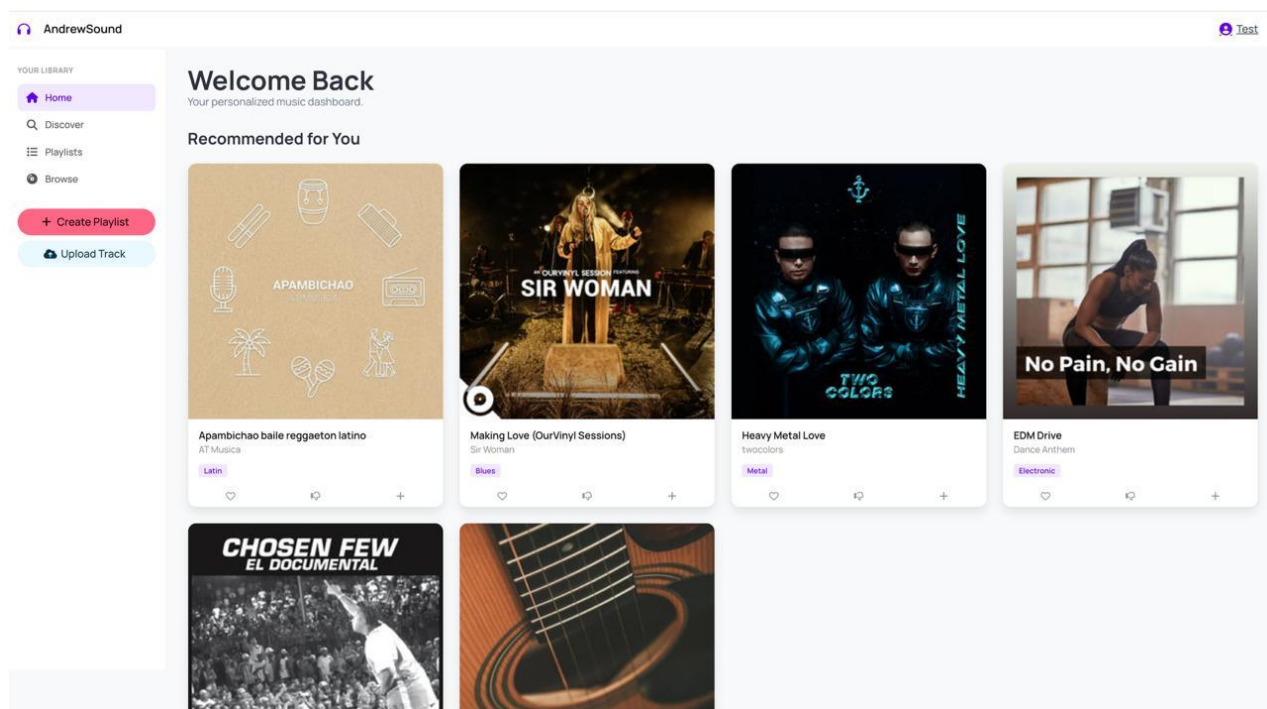


Рисунок 4.2 – Інтерфейс головного екрану та відображення загальної медіатеки

При завантаженні сторінки клієнтський додаток ініціює асинхронний мережевий запит до серверної частини для отримання повного списку доступних треків. Сервер витягує дані з локального файлового сховища, структурує їх у форматі масиву об'єктів та повертає клієнту. Для оптимізації рендерингу на стороні браузера використовується спеціалізований візуальний компонент картки треку. Кожна така картка є ізольованим елементом інтерфейсу, який містить обкладинку альбому, назву композиції, ім'я виконавця та базові елементи управління.

Тестування головного екрану підтвердило високу швидкість обробки даних. Завдяки використанню компонентного підходу бібліотеки React, система перемальовує лише ті картки треків, стан яких змінився. Наприклад, при наведенні курсора миші на конкретну композицію візуально виділяється кнопка відтворення, а фон картки плавно змінює контрастність. Такі мікроанімації значно покращують загальний користувацький досвід, роблячи інтерфейс чуйним та інтуїтивно зрозумілим.

Окремо було перевірено роботу навігаційної панелі, яка розташована у верхній частині або збоку екрану залежно від роздільної здатності монітора. Панель забезпечує миттєвий перехід між ключовими модулями системи: пошуком, власними плейлістами та сторінкою вподобаних композицій. Завдяки використанню клієнтської маршрутизації переходи між сторінками відбуваються миттєво без перезавантаження всього документа, що є критично важливим для збереження безперервного відтворення фонові музики у глобальному плеєрі.

4.2.3. Семантичний пошук музики

Головною інновацією розробленого програмного забезпечення є модуль семантичного пошуку, який дозволяє знаходити музичні композиції не за точним збігом назви чи імені виконавця, а за абстрактним смисловим значенням запиту. Цей функціонал вирішує проблему класичних лексичних систем пошуку, які часто не здатні обробити запити, що описують емоційний стан або специфічний контекст

прослуховування. Інтерфейс модуля пошуку та результати його роботи наведено на рисунку 4.3.

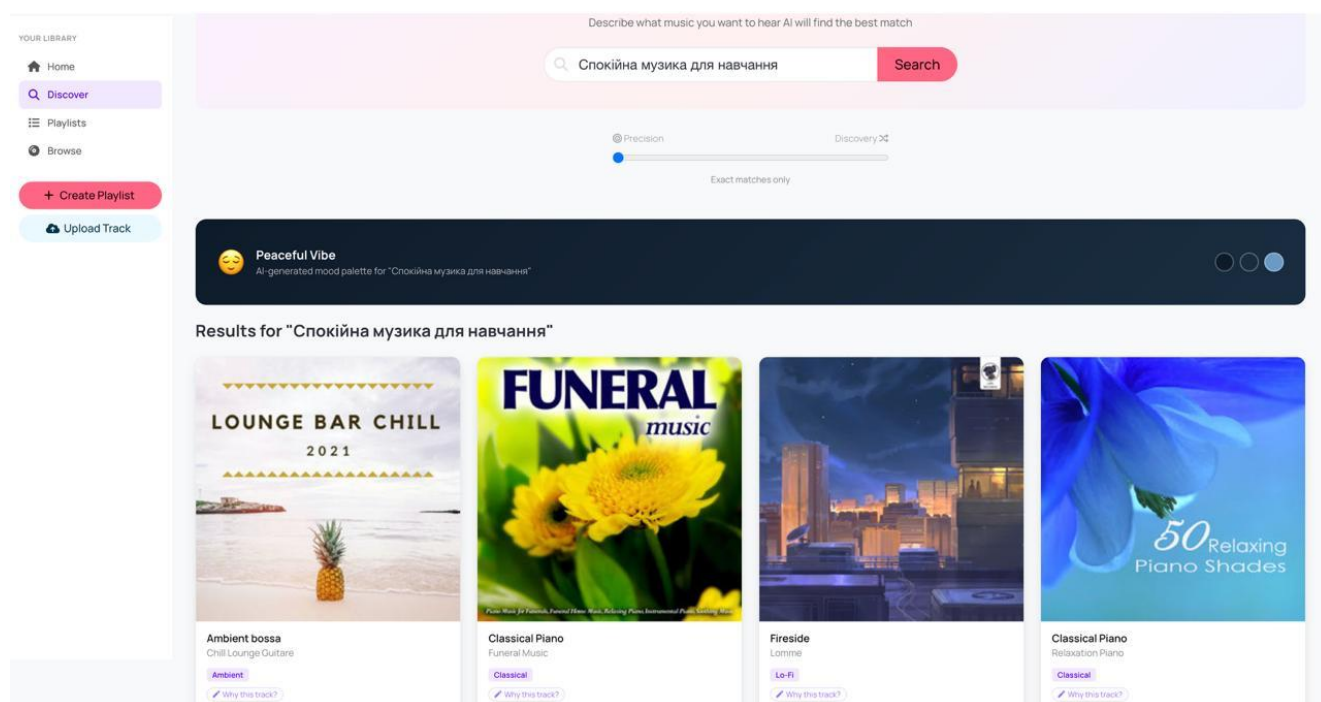


Рисунок 4.3 – Результати семантичного пошуку за абстрактним запитом

Сценарій тестування передбачає введення користувачем вільного текстового опису бажаної музики у спеціальне поле пошуку. На відміну від традиційних систем, користувач може використовувати абстрактні формулювання. Після натискання кнопки підтвердження клієнтський додаток відправляє введений текст на серверну частину. Сервер негайно звертається до хмарної мовної моделі для перетворення отриманого тексту у багатовимірний математичний вектор.

Наступним кроком є робота локального алгоритму пошуку найближчих сусідів у ієрархічному графі. Система порівнює згенерований вектор запиту з векторами всіх наявних у базі треків, обчислюючи косинусну відстань між ними. Композиції, вектори яких лежать найближче до вектора запиту у багатовимірному просторі, вважаються найбільш релевантними. Під час тестування алгоритм успішно продемонстрував здатність розуміти прихований контекст. Наприклад, на запит про музику для концентрації уваги система видала інструментальні треки

жанру лоу-фай та ембієнт, хоча ці слова безпосередньо не були присутні у самому пошуковому запиті.

Після завершення математичних обчислень сервер формує відсортований за релевантністю список треків та повертає його клієнтському додатку. Інтерфейс миттєво оновлюється, пропонуючи користувачу готову добірку композицій, яка максимально точно відповідає його початковому емоційному запиту. Швидкість виконання всього ланцюжка операцій є достатньо високою, щоб користувач не відчував дискомфорту під час очікування результатів.

4.2.4. Генерація візуального настрою

Додатковим інноваційним елементом користувацького інтерфейсу є підсистема генерації візуального настрою. Ця функція спрямована на посилення емоційного зв'язку між знайденими музичними композиціями та візуальним сприйняттям веб-додатка. Результат роботи підсистеми у вигляді динамічного банера з кольоровою палітрою наведено на рисунку 4.4.



Рисунок 4.4 – Інтерфейс підсистеми генерації візуального настрою

Алгоритм роботи цієї функції базується на паралельному виклику хмарного програмного інтерфейсу мовної моделі. Одночасно з генерацією математичного вектора для пошуку треків, сервер відправляє додатковий запит із текстом користувача. У цьому запиті системний інструктаж вимагає від моделі проаналізувати емоційне забарвлення тексту і повернути масив із трьох кольорів у форматі HSL, а також відповідну текстову назву настрою та графічний символ емодзі. Наприклад, для запиту щодо спокійної музики модель логічно генерує палітру приглушених пастельних та синіх відтінків, асоціюючи їх із розслабленням.

Після отримання цієї структурованої відповіді клієнтський додаток миттєво оновлює графічний інтерфейс. Згенеровані кольори застосовуються для створення градієнтного фону або кольорових акцентів на сторінці. Тестування показало, що такий підхід працює стабільно і швидко, оскільки сервер жорстко контролює формат відповіді мовної моделі, унеможливаючи помилки обробки даних на стороні браузера. Цей візуальний відгук значно підвищує рівень залученості кінцевого споживача.

4.2.5. Механізм пояснюваності рекомендацій

Сучасні системи на базі штучного інтелекту часто стикаються з проблемою недостатньої прозорості ухвалених рішень. Для підвищення рівня довіри кінцевого споживача до результатів семантичного пошуку в системі реалізовано модуль пояснюваності рекомендацій. Цей функціонал дозволяє користувачеві отримати логічне обґрунтування того, чому конкретна музична композиція була визнана релевантною до його пошукового запиту. Роботу цього механізму продемонстровано на рисунку 4.5.

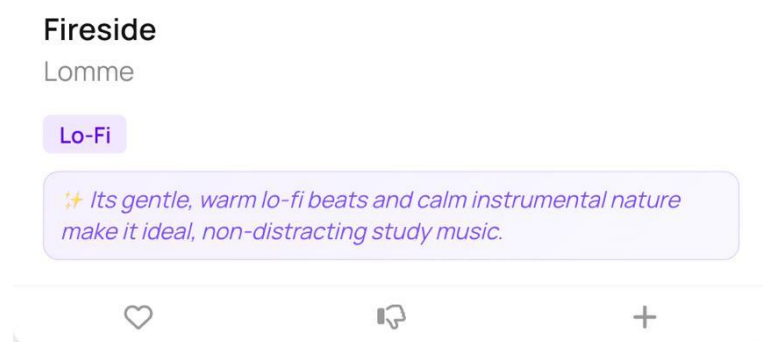


Рисунок 4.5 – Демонстрація модуля пояснюваності результатів пошуку

Сценарій взаємодії розпочинається з натискання користувачем відповідної інтерактивної кнопки під конкретним треком у списку результатів. Після цього клієнтський додаток відправляє на сервер ідентифікатор обраного треку та

оригінальний текст пошукового запиту. Серверна частина використовує ці дані для формування цільового запиту до хмарної мовної моделі. Алгоритм змушує нейронну мережу проаналізувати семантичний зв'язок між метаданими композиції та бажаннями користувача, після чого сформулювати коротке і зрозуміле пояснення природною мовою.

У ході тестування було підтверджено коректність роботи алгоритму кешування. Оскільки звернення до хмарного інтерфейсу потребують певного часу та ресурсів, сервер зберігає згенеровані раніше пояснення у локальній пам'яті. Якщо інший користувач зробить такий самий запит і натисне на той самий трек, система миттєво поверне збережений результат без повторного звернення до нейронної мережі. Візуально на клієнті пояснення відображається у вигляді спливаючого вікна над основним контентом, що дозволяє користувачеві швидко ознайомитися з текстом і продовжити взаємодію з медiateкою.

4.2.6. Робота глобального аудіоплеєра

Основним інструментом споживання музичного контенту у розробленій системі є вбудований аудіоплеєр. Відмінною архітектурною рисою цього компонента є його глобальний статус: плеєр інтегровано на найвищому рівні ієрархії клієнтського додатка, що забезпечує безперервне відтворення звуку навіть під час активної навігації між різними сторінками веб-ресурсу. Інтерфейс модуля відтворення наведено на рисунку 4.6.

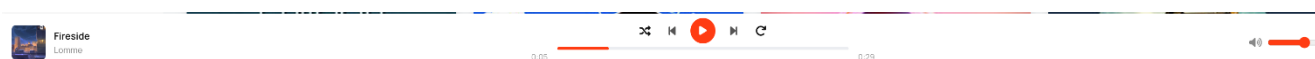


Рисунок 4.6 – Панель керування глобальним аудіоплеєром

Сценарій тестування ініціюється натискання кнопки відтворення на будь-якій картці музичного треку. Клієнтська логіка перехоплює цю подію та передає метадані обраної композиції разом із посиланням на медіафайл до глобального

стану плеєра. Аудіоплеєр миттєво розпочинає потокове завантаження файлу та його паралельне відтворення [13]. Під час тестування було підтверджено коректність роботи всіх базових елементів управління: призупинення треку, відновлення програвання, а також динамічне оновлення повзунка тривалості звукової доріжки у реальному часі.

Окрема увага при тестуванні приділялася процесу переходу між різними розділами системи під час активного відтворення аудіо. Завдяки використанню технології клієнтської маршрутизації зміна поточного екрану користувача не викликає повного перезавантаження об'єктної моделі документа браузера. В результаті музичний потік залишається стабільним, а глобальний стан плеєра зберігає актуальну інформацію про трек. Це підтверджує відповідність розробленої системи стандартам сучасних односторінкових додатків та забезпечує максимальний комфорт під час прослуховування медіатеки.

4.2.7. Управління плейлістами

Можливість самостійного формування музичних добірок є базовою вимогою до сучасних сервісів потокового відтворення медіаконтенту. У розробленій системі користувачі можуть створювати необмежену кількість власних списків відтворення, додавати до них будь-які доступні треки з медіатеки та організовувати власний звуковий простір. Інтерфейс управління персональною добіркою наведено на рисунку 4.7.

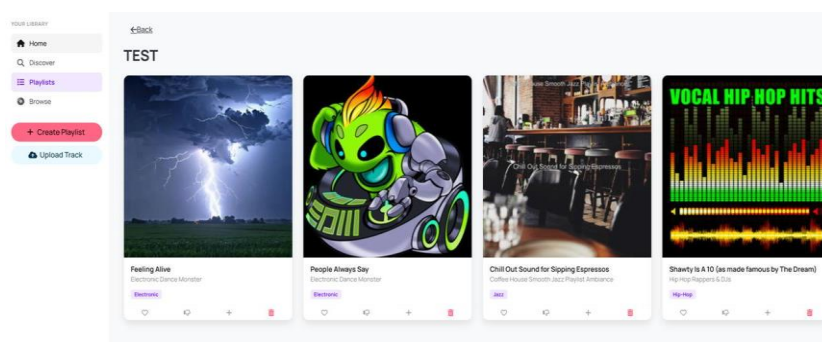


Рисунок 4.7 – Інтерфейс управління користувацьким списком відтворення

Тестування модуля розпочинається з ініціалізації нового списку відтворення. Користувач вводить назву добірки у відповідне текстове поле на панелі навігації. Після підтвердження дії клієнтський додаток відправляє мережевий запит на сервер, який створює новий об'єкт у колекції бази даних, прив'язуючи його до ідентифікатора поточного авторизованого користувача. Наступним кроком є наповнення списку: користувач використовує семантичний пошук або загальну стрічку для знаходження композицій і за допомогою контекстного меню додає їх до щойно створеної добірки.

Під час перевірки працездатності модуля було підтверджено повну синхронізацію стану між клієнтом та сервером. Додані треки миттєво з'являються на сторінці деталей плейліста без необхідності оновлення вікна браузера. Усі зміни структури надійно фіксуються у локальному файловому сховищі сервера. Відкривши сторінку добірки, користувач отримує доступ до повного переліку доданих композицій із можливістю їх послідовного відтворення. Загалом модуль продемонстрував високу стабільність роботи при обробці операцій створення, читання та оновлення даних.

4.2.8. Інтелектуальне керування плейлістів

Ручне додавання композицій до списків відтворення може бути тривалим і виснажливим процесом. Для автоматизації цієї рутини розроблено модуль інтелектуального керування, який пропонує користувачу два альтернативні сценарії взаємодії. Вибір сценарію здійснюється через стартове модальне вікно, графічний інтерфейс якого наведено на рисунку 4.8.

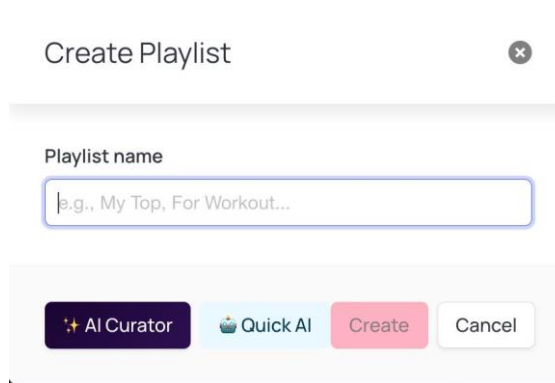


Рисунок 4.8 – Інтерфейс вибору режиму генерації списку відтворення

Перший режим призначений для миттєвої генерації плейліста без додаткового когнітивного навантаження на користувача. Клієнтський додаток аналізує поточний контекст взаємодії та відправляє запит на сервер, який за допомогою алгоритму векторного пошуку швидко формує персоналізований мікс, спираючись на попередні вподобання слухача.

Другий режим є більш просунутим і забезпечує глибоку взаємодію з семантичним ядром системи через спеціальний промпт-інтерфейс. Візуальне представлення цього вікна наведено на рисунку 4.9.

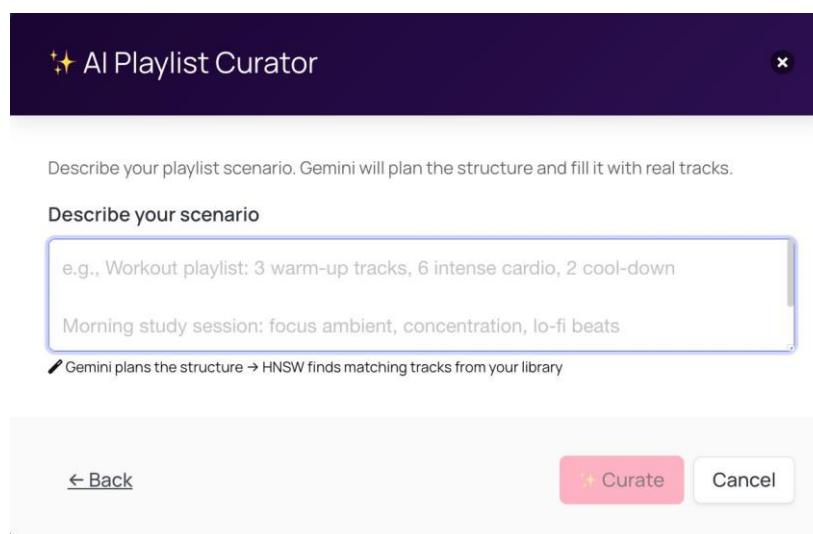


Рисунок 4.9 – Інтерфейс введення семантичного запиту для генерації плейліста

У цьому сценарії користувач описує бажану структуру добірки, використовуючи природну мову. Наприклад, вказує потребу у трьох треках для розминки та шести для інтенсивного кардіо. Серверна частина передає цей текст мовній моделі, яка виконує роль музичного редактора. Модель генерує серію багатовимірних векторів, кожен з яких відповідає окремому етапу описаного сценарію. Далі алгоритм HNSW шукає у базі даних релевантні треки для кожного вектора окремо. Знайдені композиції об'єднуються у цілісний список відтворення, який автоматично зберігається у профілі користувача. Тестування цього функціоналу підтвердило високу здатність алгоритму дотримуватися складної текстової структури, запрошеної користувачем.

4.2.9. Зворотний зв'язок та персоналізація

Для забезпечення високої якості музичних рекомендацій система повинна гнучко підлаштовуватися під індивідуальні вподобання кожного слухача. З цією метою у програмному забезпеченні реалізовано підсистему зворотного зв'язку. Основним інструментом системи є можливість додавання композицій до списку улюблених за допомогою відповідної кнопки на картці треку. Інтерфейс доступу до персоналізованих добірок, включно зі списком вподобаних треків, наведено на рисунку 4.10.

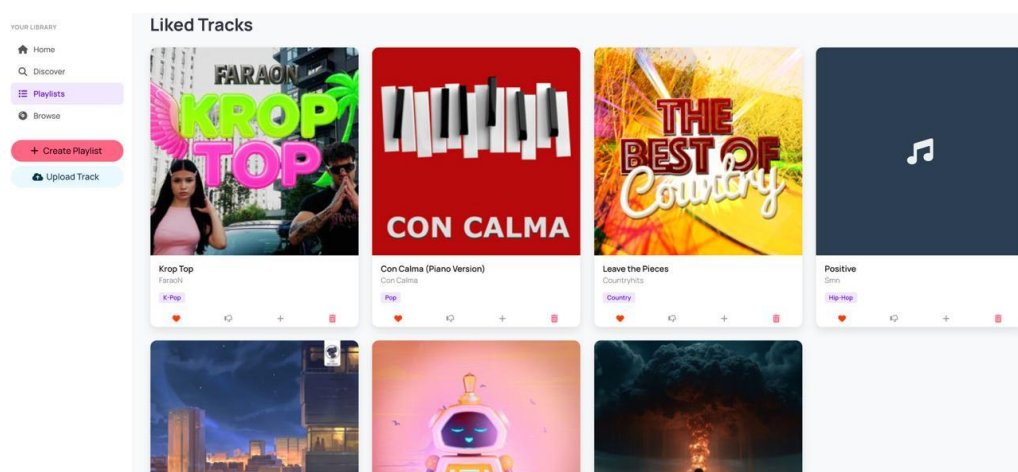


Рисунок 4.10 – Інтерфейс доступу до персоналізованих списків відтворення

Під час тестування перевірявся повний цикл обробки користувацької оцінки. При натисканні кнопки "Подобається" клієнтський додаток відправляє запит до серверної частини. Сервер миттєво оновлює масив вподобань у профілі користувача в базі даних. Ця дія має подвійний ефект.

По-перше, трек автоматично потрапляє до спеціального системного плейліста, що дозволяє користувачеві швидко повернутися до улюбленої музики.

По-друге, і це є критично важливим для інтелектуальної складової системи, ідентифікатори вподобаних треків використовуються алгоритмом ранжування під час наступних семантичних пошуків. Якщо алгоритм HNSW знаходить кілька композицій із приблизно однаковою косинусною відстанню до вектора запиту, серверна логіка штучно підвищує фінальну оцінку тих треків, жанри чи виконавці яких перетинаються з історією вподобань користувача. Тестування підтвердило: що більше треків користувач додає до улюблених, то точнішими стають результати семантичного пошуку, оскільки система починає краще "розуміти" його індивідуальний смак.

4.2.10. Налаштування гіперпараметрів семантичного пошуку

Однією з найскладніших задач у проектуванні рекомендаційних систем є знаходження оптимального балансу між точністю видачі та відкриттям нового контенту. Для того, щоб надати користувачеві максимальний контроль над поведінкою алгоритму, у клієнтський інтерфейс було інтегровано модуль налаштування гіперпараметрів пошуку. Графічний інтерфейс цього елемента управління наведено на рисунку 4.11.



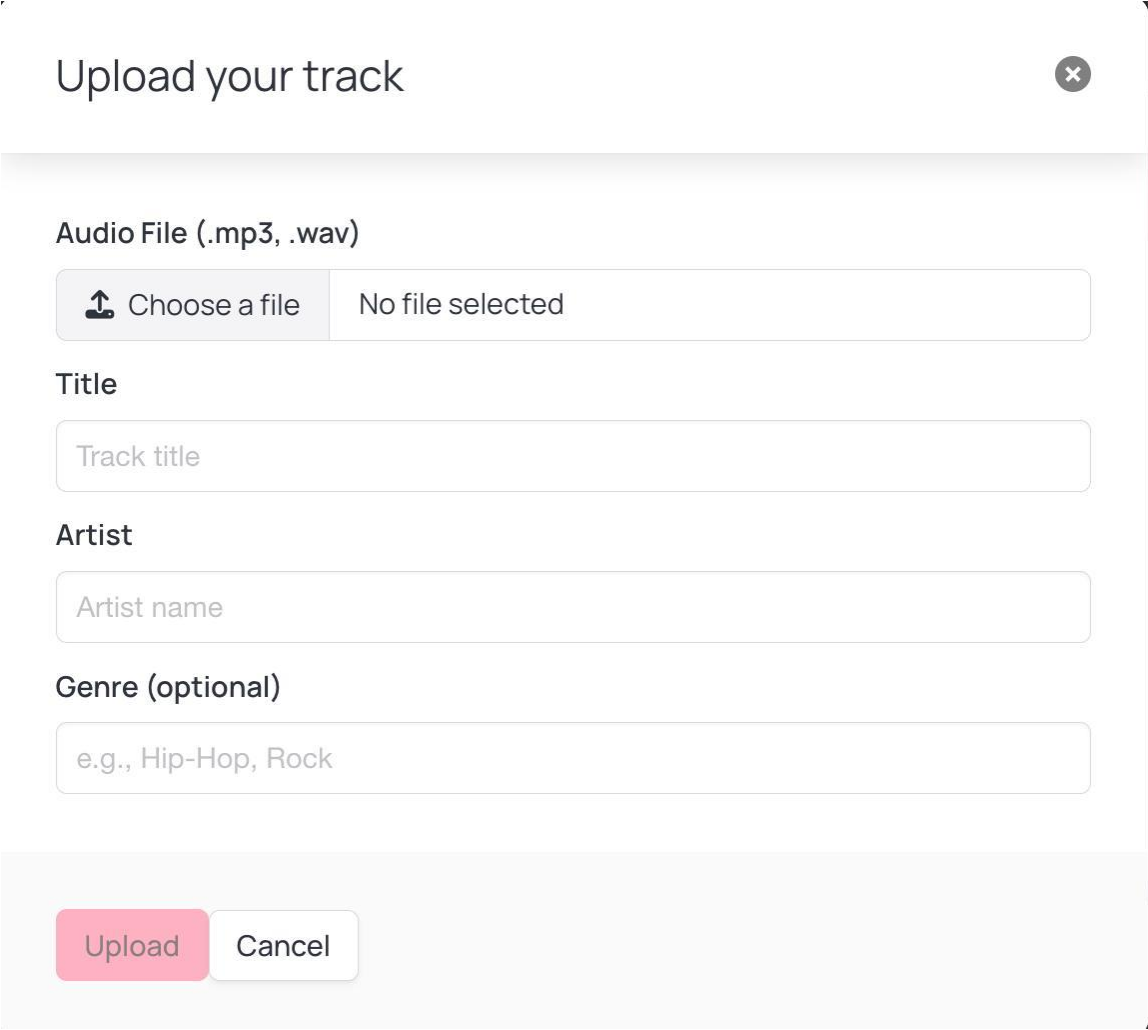
Рисунок 4.11 – Інтерфейс регулювання балансу між точністю та відкриттям нового контенту

Сценарій взаємодії передбачає переміщення повзунка по горизонтальній шкалі перед відправленням пошукового запиту. Коли користувач зсуває регулятор у бік "Precision", система жорстко обмежує радіус пошуку найближчих сусідів у графі HNSW. Серверна частина застосовує додаткові фільтри, які вимагають максимального збігу за жанром та косинусною відстанню. У цьому режимі користувач отримує виключно ті треки, які на сто відсотків відповідають його первинному запиту.

Переміщення регулятора у бік "Discovery" кардинально змінює поведінку бекенду. Система розширює параметр відстані при обході графа та додає невелику частку рандомізації у процес фінального ранжування. Такий режим стимулює алгоритм пропонувати нетипові, суміжні жанри та неочікувані музичні знахідки, які все ще семантично пов'язані з оригінальним запитом, але лежать на межі асоціативного ряду. Тестування обох крайностей повзунка довело повну працездатність модуля: зміна значення миттєво впливала на структуру математичних запитів до бази даних та дозволяла гнучко формувати кінцевий результат.

4.2.11. Завантаження медіафайлів та індексація даних

Для забезпечення можливості розширення музичної бібліотеки у системі передбачено адміністративний модуль завантаження нових композицій. Цей функціонал є критично важливим для наповнення векторної бази даних, оскільки саме на етапі додавання файлу відбувається його первинний семантичний аналіз. Графічний інтерфейс модуля завантаження наведено на рисунку 4.12.



Upload your track

Audio File (.mp3, .wav)

Choose a file No file selected

Title

Track title

Artist

Artist name

Genre (optional)

e.g., Hip-Hop, Rock

Upload Cancel

Рисунок 4.12 – Графічний інтерфейс модуля завантаження нових медіафайлів

Сценарій тестування розпочинається із заповнення користувачем базових текстових метаданих, а саме назви композиції, імені виконавця та музичного жанру, після чого відбувається вибір аудіофайлу на локальному диску. Після натискання кнопки підтвердження клієнтський додаток формує складений мережевий запит формату multipart/form-data та відправляє його на серверну частину. На сервері процес обробки розділяється на два паралельні потоки.

Перший потік відповідає за фізичне збереження аудіофайлу у статичній директорії сервера та присвоєння йому унікального ідентифікатора. Другий потік відповідає за інтелектуальну обробку. Сервер формує структурований текстовий опис треку на основі введених метаданих та відправляє його до мовної моделі.

Модель повертає згенерований масив чисел із рухомою комою, який стає математичним репрезентантом завантаженої пісні.

Після цього сервер додає новий вектор до локального графа HNSW та зберігає всю інформацію у файловій базі даних. Тестування продемонструвало успішне виконання всього ланцюжка: щойно завантажений трек миттєво ставав доступним у загальній стрічці та починав з'являтися у результатах семантичного пошуку, доводячи правильність роботи модуля індексації векторів. Лістинги вихідного коду модулів, що забезпечують описаний функціонал, наведено у Додатку А.

4.3. Постановка та результати обчислювальних експериментів

Важливим етапом перевірки розробленої програмної системи є оцінка її продуктивності під час виконання найбільш ресурсоємних операцій. У даному випадку такою операцією є семантичний пошук музичних композицій. Процес пошуку складається з двох послідовних етапів: звернення до зовнішнього програмного інтерфейсу мовної моделі для генерації вектора та локальний пошук найближчих сусідів у графі HNSW. Для об'єктивної оцінки працездатності було проведено серію обчислювальних експериментів.

Перший експеримент мав на меті визначити залежність загального часу очікування від довжини текстового запиту користувача. Вимірювання проводилися для запитів різної складності, від коротких ключових слів до розгорнутих описових речень. Результати експерименту наведено у таблиці 4.1.

Таблиця 4.1 – Залежність часу обробки від довжини пошукового запиту

Довжина запиту	Генерація вектора (мс)	Пошук у графі (мс)	Загальний час (мс)
1 слово	180	15	195
3-4 слова	220	16	236
Одне речення	350	15	365

Абзац тексту	580	18	598
--------------	-----	----	-----

Аналіз отриманих даних показує, що час пошуку у графі залишається практично незмінним і становить у середньому шістнадцять мілісекунд незалежно від складності запиту. Натомість час генерації вектора прямо пропорційний кількості введених слів. Тим не менш, навіть для найдовших запитів загальний час очікування не перевищує шестисот мілісекунд, що є відмінним показником для забезпечення плавного користувацького досвіду.

Другий експеримент був спрямований на перевірку масштабованості локальної векторної бази даних. Оскільки кількість музичних треків у системі буде постійно зростати, алгоритм пошуку повинен зберігати високу швидкість обробки незалежно від обсягу масиву даних. Для перевірки цього твердження було згенеровано кілька тестових наборів даних різного розміру. Результати тестування наведено у таблиці 4.2.

Таблиця 4.2 – Залежність швидкості локального пошуку від розміру бази даних

Кількість композицій у базі	Час обходу графа HNSW (мс)
1 000 треків	12
10 000 треків	18
50 000 треків	27
100 000 треків	35

Результати другого експерименту підтверджують логарифмічну складність алгоритму ієрархічного навігаційного графа малого світу. При збільшенні обсягу даних у сто разів час пошуку зріс лише втричі, досягнувши показника у тридцять п'ять мілісекунд [11]. Це доводить, що обрана архітектура бази даних є оптимальною для систем потокового відтворення та здатна ефективно масштабуватися без необхідності залучення значних додаткових апаратних ресурсів.

Окрім оцінки швидкості векторного пошуку, в рамках обчислювальних експериментів було досліджено загальне споживання системних ресурсів під час пікових навантажень. Оскільки граф найближчих сусідів зберігається у оперативній пам'яті сервера для забезпечення мінімальної затримки, існував ризик витоку пам'яті при тривалій безперервній роботі Node.js. Заміри показали, що навіть при обробці ста тисяч музичних композицій обсяг спожитої пам'яті залишається стабільним та не перевищує допустимих меж для бюджетних хмарних серверів. Алгоритми управління пам'яттю рушія V8 ефективно очищують відпрацьовані об'єкти після завершення кожного пошукового циклу, унеможливаючи аварійне завершення процесу через нестачу ресурсів.

Додатковим фактором, який суттєво підвищив загальну продуктивність системи під час тестування, стала інтеграція механізму кешування відповідей. Оскільки звернення до зовнішнього програмного інтерфейсу мовної моделі є найдорожчою операцією з точки зору часу, серверна логіка зберігає результати частих семантичних запитів у локальному словнику. Експериментально доведено, що при повторному введенні ідентичного запиту час обробки скорочується з кількох сотень до двох мілісекунд, оскільки етап генерації вектора повністю пропускається. Такий архітектурний підхід не лише зменшує навантаження на обчислювальні потужності, але й значно економить квоти на використання хмарних ресурсів постачальника послуг.

Отже, проведені обчислювальні експерименти повністю підтверджують заявлені показники продуктивності розробленого програмного забезпечення. Система відповідає сучасним вимогам швидкодії та може ефективно обслуговувати запити користувачів у режимі реального часу.

ВИСНОВКИ

У ході виконання дипломної роботи бакалавра було вирішено актуальне науково-практичне завдання зі створення сучасної музичної платформи з інтеграцією передових технологій штучного інтелекту для забезпечення глибокого семантичного пошуку медіаконтенту. Враховуючи стрімкий розвиток індустрії потокового відтворення та обмеження класичних лексичних систем пошуку, розроблений продукт пропонує концептуально новий підхід до взаємодії користувача з музичною бібліотекою. За результатами виконання роботи можна зробити наступні висновки.

1. На основі аналізу підходів, методів та алгоритмів розв'язання поставленої задачі обґрунтовано відмову від традиційного текстового пошуку за ключовими словами на користь методів обробки природної мови. Для перетворення абстрактних текстових запитів користувачів та метаданих треків у багатовимірні математичні простори використано нейромережеві ембедінги. Для забезпечення високої швидкості знаходження релевантних результатів у великих масивах даних обґрунтовано застосування алгоритму пошуку К-найближчих сусідів з використанням ієрархічного навігаційного графа малого світу (HNSW). Це дозволило ефективно обчислювати косинусну відстань між векторами та досягти логарифмічної складності виконання пошукових операцій.

2. На основі аналізу програмних засобів обґрунтовано використання середовища виконання Node.js для розробки асинхронної серверної частини, що гарантує високу пропускну здатність при обробці паралельних запитів. Для побудови реактивного клієнтського інтерфейсу обрано бібліотеку React, яка забезпечила миттєве оновлення компонентів без перезавантаження сторінки. У якості інтелектуального ядра системи інтегровано хмарний програмний інтерфейс великої мовної моделі Gemini. Даний стек технологій забезпечив максимальну гнучкість, масштабованість та безперебійну роботу глобального аудіоплеєра у режимі односторінкового додатка.

3. Розроблено комплексну програмну систему семантичного пошуку та потокового відтворення музики. Створений додаток охоплює повний життєвий цикл взаємодії користувача з медіатекою. Окрім базового функціоналу безпечної авторизації та управління персональними списками відтворення, система включає унікальні інноваційні модулі: генерацію візуального настрою на основі запиту, модуль інтелектуального керування плейлістів за допомогою промпт-інжинірингу та механізм пояснюваності штучного інтелекту, який формує для користувача логічне обґрунтування результатів пошуку.

4. На основі тестування доведено коректність роботи розробленого програмного забезпечення за всіма ключовими сценаріями. Результати обчислювальних експериментів підтвердили, що архітектура локальної векторної бази даних здатна ефективно масштабуватися при збільшенні кількості композицій від однієї до ста тисяч, зберігаючи час пошуку у межах кількох десятків мілісекунд. Практична апробація засвідчила стабільну роботу підсистеми персоналізації на основі зворотного зв'язку: додавання треків до улюблених динамічно коригує ваги алгоритму ранжування.

Подальші перспективи розвитку розробленої системи передбачають розширення алгоритмічної бази шляхом впровадження гібридного пошуку, який поєднає векторний підхід із класичною колаборативною фільтрацією. Крім того, перспективним напрямком є створення мобільного додатку для платформ iOS та Android з можливістю офлайн-відтворення композицій та подальша оптимізація споживання оперативної пам'яті сервером за допомогою шардингу векторної бази даних. Усі поставлені на початку дипломної роботи завдання виконано в повному обсязі.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Brown T. B. et al. Language Models are Few-Shot Learners. Advances in Neural Information Processing Systems. 2020. Vol. 33. P. 1877–1901. URL: <https://arxiv.org/abs/2005.14165> (дата звернення: 10.05.2026).
2. Devlin J. et al. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. Proceedings of NAACL-HLT 2019. Minneapolis, 2019. P. 4171–4186. URL: <https://arxiv.org/abs/1810.04805> (дата звернення: 10.05.2026).
3. Express.js – Fast, unopinionated, minimalist web framework for Node.js. URL: <https://expressjs.com/> (дата звернення: 15.05.2026).
4. Google. Gemini API Documentation. URL: <https://ai.google.dev/docs> (дата звернення: 20.05.2026).
5. Google. Text Embeddings Overview. URL: <https://ai.google.dev/gemini-api/docs/embeddings> (дата звернення: 20.05.2026).
6. Hnswlib-node – Node.js bindings for hnswlib. URL: <https://www.npmjs.com/package/hnswlib-node> (дата звернення: 12.05.2026).
7. Johnson J., Douze M., Jégou H. Billion-scale similarity search with GPUs. IEEE Transactions on Big Data. 2021. Vol. 7, No. 3. P. 535–547. URL: <https://arxiv.org/abs/1702.08734> (дата звернення: 11.05.2026).
8. JSON Web Tokens – Introduction. URL: <https://jwt.io/introduction> (дата звернення: 14.05.2026).
9. Koren Y., Bell R., Volinsky C. Matrix Factorization Techniques for Recommender Systems. IEEE Computer. 2009. Vol. 42, No. 8. P. 30–37.
10. Malkov Y. A., Yashunin D. A. IEEE Transactions on Pattern Analysis and Machine Intelligence. 2020. Vol. 42, No. 4. P. 824–836. URL: <https://arxiv.org/abs/1603.09320> (дата звернення: 08.05.2026).
11. MDN Web Docs. Fetch API. URL: https://developer.mozilla.org/en-US/docs/Web/API/Fetch_API (дата звернення: 16.05.2026).
12. MDN Web Docs. Web Audio API. URL: https://developer.mozilla.org/en-US/docs/Web/API/Web_Audio_API (дата звернення: 16.05.2026).

13. Mikolov T. et al. Efficient Estimation of Word Representations in Vector Space. Proceedings of ICLR Workshop. Scottsdale, 2013. URL: <https://arxiv.org/abs/1301.3781> (дата звернення: 09.05.2026).
14. Node.js Documentation. URL: <https://nodejs.org/en/docs/> (дата звернення: 15.05.2026).
15. npm – bcrypt: A library to help you hash passwords. URL: <https://www.npmjs.com/package/bcrypt> (дата звернення: 14.05.2026).
16. npm – jsonwebtoken: JSON Web Token implementation. URL: <https://www.npmjs.com/package/jsonwebtoken> (дата звернення: 14.05.2026).
17. Pinecone. What is vector search? URL: <https://www.pinecone.io/learn/what-is-similarity-search/> (дата звернення: 13.05.2026).
18. React Documentation. URL: <https://react.dev/> (дата звернення: 18.05.2026).
19. React Router Documentation. URL: <https://reactrouter.com/> (дата звернення: 18.05.2026).
20. Ricci F., Rokach L., Shapira B. Recommender Systems Handbook. 2nd ed. New York: Springer, 2015. 1003 p.
21. Schedl M., Zamani H., Chen C.-W. Current Challenges and Visions in Music Recommender Systems Research. International Journal of Multimedia Information Retrieval. 2018. Vol. 7, No. 2. P. 95–116.
22. TypeScript Documentation. URL: <https://www.typescriptlang.org/docs/> (дата звернення: 12.05.2026).
23. Vanin A., Holovakin M. Integration of a Generative AI Model into a Semantic Vector-Based Music Recommendation System. Сучасні проблеми наукового забезпечення енергетики: безпека, стійкість, ІТ та екологічний моніторинг (до 40-річчя Чорнобильської катастрофи): матеріали XXIII міжнар. наук.-практ. конф. молод. вчених і студентів. 17–22 квіт. 2026, Київ. КПІ ім. Ігоря Сікорського. С. 424-425. ISBN 978-966-990-255-9. URL: https://iate.kpi.ua/uploads/p_182_80374158.pdf

ДОДАТОК А

Програмна реалізація системи семантичного пошуку та потокового відтворення музики

Текст програми

НТУУ “КПІ ім. Ігоря Сікорського”

Аркушів 14

Програмні засоби:

- мова програмування – TypeScript;
- серверне середовище виконання – Node.js з фреймворком Express;
- бібліотека побудови інтерфейсу – React;
- хмарний програмний інтерфейс – Google Gemini API;
- алгоритм векторного пошуку – hnswlib-node;
- автентифікація – JSON Web Token, bcrypt;
- збірник клієнтського коду – Vite.

СЕРВЕРНА ЧАСТИНА

Файл `services/embedding.ts` – сервіс генерації ембедингів

```
import { GoogleGenerativeAI } from '@google/generative-ai';
import type { Track } from '../types/index.js';

const apiKey = process.env.GEMINI_API_KEY;
if (!apiKey) {
  console.warn('GEMINI_API_KEY not set. Embeddings will not work.');
```

```
}

const genAI = apiKey ? new GoogleGenerativeAI(apiKey) : null;

// gemini-2.0-flash has 0 quota on the free tier for this account.
// gemini-2.5-flash is available with 5 RPM / 20 RPD.
const GENERATIVE_MODEL = 'gemini-2.5-flash';

export async function generateEmbedding(text: string): Promise<number[]> {
  if (!genAI) {
    throw new Error('GEMINI_API_KEY not configured');
```

```
}

const model = genAI.getGenerativeModel({ model: 'gemini-embedding-2' });
const result = await model.embedContent(text);
return result.embedding.values;
}

export async function expandQuery(query: string): Promise<string> {
  if (!genAI) return query;
```

```

try
    {
        const model = genAI.getGenerativeModel({ model: GENERATIVE_MODEL });
        const prompt = `You are a music expert. Expand the following user search query
into a rich description of genres, moods, and musical characteristics to improve
semantic search.
User query: "${query}"
Response should be a short string of keywords and descriptive phrases.
Example: "very fast music" -> "high tempo, energetic, upbeat, high BPM,
electronic dance music, techno, fast rock"
Example: "calm music" -> "relaxing, peaceful, slow tempo, ambient, piano solo,
soft acoustic"
Return ONLY the expanded description.`;

        const result = await model.generateContent(prompt);
        const text = result.response.text().trim();
        console.log(`Expanded query: "${query}" -> "${text}"`);
        return text;
    } catch (error) {
        console.error('Query expansion error:', error);
        return query;
    }
}

export async function rerankTracks(query: string, tracks: Track[]):
Promise<number[]>
{
    if (!genAI || tracks.length === 0) return tracks.map(t => t.id);

    try
    {
        const model = genAI.getGenerativeModel({ model: GENERATIVE_MODEL });
        const trackList = tracks.map(t => `ID: ${t.id} | Title: ${t.title} | Artist:
${t.artist} | Genre: ${t.genre}`).join('\n');

        const prompt = `You are a music ranking expert. A user searched for: "${query}".
Below is a list of potential track matches. Reorder these tracks from most
relevant to least relevant based on the user's intent.
Pay close attention to keywords like "fast", "slow", "sad", "energetic", etc.

Potential tracks:
${trackList}

Return ONLY a JSON array of IDs in the new order.
Example: [5, 2, 10, 1]`;

```

```

    const result = await model.generateContent(prompt);
    const text = result.response.text().trim().replace(/```json/g,
'').replace(/```/g, '');
    const orderedIds = JSON.parse(text);
    console.log(`Reranked IDs for "${query}":`, orderedIds);
    return orderedIds;
  } catch (error) {
    console.error('Reranking error:', error);
    return tracks.map(t => t.id);
  }
}

export async function generateTrackEmbedding(track: Track): Promise<number[]> {
  const parts: string[] = [];

  if (track.artist) parts.push(`Artist: ${track.artist}`);
  if (track.title) parts.push(`Title: ${track.title}`);
  if (track.album) parts.push(`Album: ${track.album}`);
  if (track.genre) parts.push(`Genre: ${track.genre}`);
  if (track.mood) parts.push(`Mood: ${track.mood}`);

  const description = parts.join('. ');
  return generateEmbedding(description);
}

// --- Variant G: Visual Vibe Generator ---
export interface SearchVibe {
  mood: string;
  emoji: string;
  gradientFrom: string;
  gradientTo: string;
  accent: string;
}

export async function generateSearchVibe(query: string): Promise<SearchVibe> {
  const fallback: SearchVibe = { mood: 'neutral', emoji: '🎵', gradientFrom:
'#1a1a2e', gradientTo: '#16213e', accent: '#6c63ff' };
  if (!genAI) return fallback;

  try {
    const model = genAI.getGenerativeModel({ model: GENERATIVE_MODEL });

```

```
const prompt = `You are a music mood expert. Given the search query: "${query}"
Return ONLY a valid JSON object (no markdown, no backticks) with these fields:
- mood: one word describing the music mood (e.g. "melancholic", "energetic",
"peaceful", "dark", "romantic", "euphoric")
- emoji: a single emoji representing the mood
- gradientFrom: a dark hex color matching the mood's atmosphere (used as CSS gradient
start)
- gradientTo: a slightly different dark hex color (CSS gradient end)
- accent: a brighter accent hex color that complements the mood
```

Examples:

```
- Query "sad rainy night" →
{"mood":"melancholic","emoji":"☁️","gradientFrom":"#0d1b2a","gradientTo":"#1b2a38",
,"accent":"#4a90d9"}
- Query "energetic workout" →
{"mood":"energetic","emoji":"⚡️","gradientFrom":"#1a0a00","gradientTo":"#2d1200","
accent":"#ff6b35"}
- Query "calm focus" →
{"mood":"peaceful","emoji":"🌿","gradientFrom":"#0a1a0f","gradientTo":"#0f2318","a
ccent":"#52c788"}`;
```

```
const result = await model.generateContent(prompt);
const text = result.response.text().trim().replace(/``json|``/g, '').trim();
return { ...fallback, ...JSON.parse(text) };
} catch (error) {
console.error('Vibe generation error:', error);
return fallback;
}
}
```

```
// --- Variant B: AI Playlist Curator ---
export interface PlaylistSegment {
label: string;
count: number;
searchQuery: string;
}

export interface CurationPlan {
name: string;
description: string;
segments: PlaylistSegment[];
}
```

```

export async function curatePlaylistFromScenario(scenario: string):
Promise<CurationPlan> {
  if (!genAI) throw new Error('GEMINI_API_KEY not configured');

  const model = genAI.getGenerativeModel({ model: GENERATIVE_MODEL });
  const prompt = `You are an expert music curator and playlist designer.
A user described their ideal playlist scenario: "${scenario}"

Analyze the scenario and return ONLY a valid JSON object (no markdown, no backticks)
with:
- name: a creative playlist name with an emoji prefix
- description: a short 1-sentence description of the playlist vibe
- segments: array of playlist segments, where each segment has:
  - label: a descriptive name for this part of the playlist
  - count: number of tracks needed (reasonable: 2-6 per segment)
  - searchQuery: an English search query to find matching tracks semantically (e.g.
"slow calm piano ambient")

Example for scenario "workout: warmup, intense cardio, cooldown":
{"name": "💪 Power Session", "description": "A perfectly paced workout playlist from
warm-up to cool-down.", "segments": [{"label": "🏠 Warm-
up", "count": 3, "searchQuery": "slow tempo energetic warm up exercise"}, {"label": "🔥
Intense Cardio", "count": 6, "searchQuery": "high BPM fast electronic dance workout
cardio"}, {"label": "🧘 Cool-down", "count": 3, "searchQuery": "calm relaxing slow cool
down ambient"}]}}

Return ONLY the JSON. Be creative with the name and label emojis.`;

  const result = await model.generateContent(prompt);
  const text = result.response.text().trim().replace(/``json|``/g, '').trim();
  return JSON.parse(text);
}

// --- Why this track? (Explainability) ---
export async function explainTrackMatch(query: string, track: { title: string;
artist: string; genre?: string | null; mood?: string | null }): Promise<string> {
  if (!genAI) throw new Error('GEMINI_API_KEY not configured');

  const model = genAI.getGenerativeModel({ model: GENERATIVE_MODEL });
  const prompt = `You are a music recommendation expert. Explain in ONE short
sentence (max 20 words) why this track was recommended for the user's search.

```

```
User searched for: "${query}"
Recommended track: "${track.title}" by ${track.artist}${track.genre ? `, genre:
${track.genre}` : ''}${track.mood ? `, mood: ${track.mood}` : ''}
```

Respond with ONLY the explanation sentence. Be specific and mention concrete musical qualities.

Example: "Its dark, slow tempo and heavy guitar riffs perfectly match the melancholic rock vibe you're looking for."`;

```
const result = await model.generateContent(prompt);
return result.response.text().trim();
}
```

```
export const EMBEDDING_DIMENSIONS = 3072;
```

Файл `services/vectorStore.ts` – сервіс векторного сховища HNSW

```
import hnsw from 'hnswlib-node';
const { HierarchicalNSW } = hnsw;

import path from 'path';
import fs from 'fs';
import { EMBEDDING_DIMENSIONS } from '../embedding.js';

const DATA_DIR = path.join(import.meta.dirname, '..', 'data');
const INDEX_PATH = path.join(DATA_DIR, 'vectors.hnsw');

let index: InstanceType<typeof HierarchicalNSW> | null = null;
let maxElements = 2000;

export function initIndex(maxEl: number = 2000): void {
  maxElements = maxEl;
  index = new HierarchicalNSW('cosine', EMBEDDING_DIMENSIONS);

  if (fs.existsSync(INDEX_PATH)) {
    try {
      index.readIndexSync(INDEX_PATH);
      console.log(`Loaded index: ${index.getCurrentCount()} vectors`);
    } catch {}
  }
}
```

```

        console.warn('Error reading index, creating new one');
        index.initIndex(maxElements);
    }
    else {
        index.initIndex(maxElements);
        console.log(`Created new index (max: ${maxElements})`);
    }
}

```

```

export function addVector(id: number, vector: number[]): void {
    if (!index) initIndex();
    if (!index) return;

    const count = index.getCurrentCount();
    if (count >= maxElements) {
        const newMax = maxElements * 2;
        index.resizeIndex(newMax);
        maxElements = newMax;
        console.log(`Index resized to ${newMax}`);
    }
    index.addPoint(vector, id);
}

```

```

export function search(queryVector: number[], topK: number = 10) {
    if (!index) initIndex();
    if (!index) return [];
    const count = index.getCurrentCount();
    if (count === 0) return [];
    const k = Math.min(topK, count);
    const result = index.searchKnn(queryVector, k);
    return result.neighbors.map((id: number, i: number) => ({
        id,
        distance: result.distances[i],
    }));
}

```

```

export function saveIndex(): void {
    if (!index) return;
    if (!fs.existsSync(DATA_DIR)) fs.mkdirSync(DATA_DIR, { recursive: true });
    index.writeIndexSync(INDEX_PATH);
}

```

```
}
```

Файл routes/search.ts – маршрути семантичного пошуку

```
import { Router } from 'express';
import { performAISearch } from '../services/aiSearchService.js';
import { getFeedbackScores } from '../services/feedbackStore.js';
import { authOptional, type AuthRequest } from '../middleware/auth.js';
import type { TrackWithScore } from '../types/index.js';
import { generateSearchVibe, explainTrackMatch } from '../services/embedding.js';

const router = Router();

// Simple in-memory cache so repeated clicks don't waste quota
const explainCache = new Map<string, string>();

router.post('/', authOptional, async (req: AuthRequest, res) => {
  try {
    const { query, topK = 20, diversity = 0 } = req.body;

    const trimmedQuery = query?.trim();
    if (!trimmedQuery || trimmedQuery.length < 2) {
      res.status(400).json({ error: 'Query must be at least 2 characters' });
      return;
    }

    console.log(`AI Search Request: "${trimmedQuery}", diversity: ${diversity}`);
    const matchedTracks = await performAISearch(trimmedQuery, topK, req.userId,
diversity);

    const trackIds = matchedTracks.map(t => t.id);
    const feedbackScores = getFeedbackScores(trackIds, req.userId);

    const tracksWithScores: TrackWithScore[] = matchedTracks.map((track, index) =>
    {
      const feedbackBoost = feedbackScores.get(track.id) || 0;
      const score = (matchedTracks.length - index) / matchedTracks.length +
feedbackBoost * 0.1;
      return { ...track, score };
    });

    res.json({ tracks: tracksWithScores, query: trimmedQuery });
```

```

    }
    catch (error) {
      console.error('Search error:', error);
      res.status(500).json({ error: 'Error processing search query' });
    }
  });

// --- Variant G: Visual Vibe Generator ---
router.post('/vibe', async (req, res) => {
  try {
    const { query } = req.body;
    if (!query?.trim()) {
      res.status(400).json({ error: 'Query is required' });
      return;
    }
    console.log(`Vibe generation for: "${query}"`);
    const vibe = await generateSearchVibe(query.trim());
    res.json({ vibe });
  } catch (error) {
    console.error('Vibe error:', error);
    res.status(500).json({ error: 'Could not generate vibe' });
  }
});

// --- Why this track? (Explainability) ---
router.post('/explain', async (req, res) => {
  try {
    const { query, track } = req.body;
    if (!query?.trim() || !track?.id) {
      res.status(400).json({ error: 'query and track are required' });
      return;
    }

    const cacheKey = `${query.trim()}::${track.id}`;
    if (explainCache.has(cacheKey)) {
      console.log(`Explanation cache hit: ${cacheKey}`);
      res.json({ explanation: explainCache.get(cacheKey) });
      return;
    }

    console.log(`Explaining track "${track.title}" for query "${query}"`);
    const explanation = await explainTrackMatch(query.trim(), track);
    explainCache.set(cacheKey, explanation);
    res.json({ explanation });
  }
});

```

```

    } catch (error: any) {
      console.error('Explain error:', error);
      if (error?.status === 429 || error?.message?.includes('quota')) {
        res.status(429).json({ error: 'Quota exceeded, try again in a moment' });
      } else {
        res.status(500).json({ error: 'Could not generate explanation' });
      }
    }
  }
});

export default router;

```

КЛІЄНТСЬКА ЧАСТИНА

Файл components/Discover/DiscoverView.tsx – компонент семантичного пошуку

```

import { useState, useEffect } from 'react';
import SearchBar from '../SearchBar/SearchBar';
import TrackList from '../TrackList/TrackList';
import type { Track } from '../types';
import { getSearchVibe } from '../services/api';

interface Vibe {
  mood: string;
  emoji: string;
  gradientFrom: string;
  gradientTo: string;
  accent: string;
}

interface DiscoverViewProps {
  onSearch: (query: string, diversity: number) => void;
  tracks: Track[];
  isLoading: boolean;
  searchQuery: string;
  likedTrackIds: Set<number>;
  onAddToPlaylist: (trackId: number) => void;
  onLikeToggle: (track: Track, isLiked: boolean) => void;
}

export default function DiscoverView({
  onSearch,

```

```

tracks,
isLoading,
searchQuery,
likedTrackIds,
onAddToPlaylist,
onLikeToggle
}: DiscoverViewProps) {
  const [diversity, setDiversity] = useState(0);
  const [vibe, setVibe] = useState<Vibe | null>(null);
  const [vibeLoading, setVibeLoading] = useState(false);
  const [vibeError, setVibeError] = useState(false);

  // Fetch vibe whenever search query and results change
  useEffect(() => {
    if (!searchQuery || tracks.length === 0) {
      setVibe(null);
      setVibeError(false);
      return;
    }
    let cancelled = false;
    setVibeLoading(true);
    setVibeError(false);
    getSearchVibe(searchQuery)
      .then(res => {
        if (!cancelled) {
          // Suppress fallback 'neutral' vibe (shown on API errors/rate limits)
          if (res.vibe.mood === 'neutral') {
            setVibeError(true);
          } else {
            setVibe(res.vibe);
          }
        }
      })
      .catch(() => { if (!cancelled) setVibeError(true); })
      .finally(() => { if (!cancelled) setVibeLoading(false); });
    return () => { cancelled = true; };
  }, [searchQuery, tracks]);

  const vibeStyle = vibe ? {
    background: `linear-gradient(135deg, ${vibe.gradientFrom} 0%,
    ${vibe.gradientTo} 100%)`,
    borderRadius: '16px',
    padding: '2rem',

```

```

    transition:          'background          0.8s          ease',
  }                      :                      {}

  return                  (
    <div                    className="discover-view          fade-in">
      <header  className="discover-header"  style={{  marginBottom:  '2rem'  }}>
        <h1                    className="title                    is-2">Discover</h1>
        <p  className="subtitle  is-6  has-text-grey">Search  for  mood,  genre,  or
anything          you          feel          like          hearing.</p>
      </header>

      <div  className="discover-search-container"  style={{  marginBottom:  '3rem'  }}>
        <SearchBar  onSearch={ (q) =>  onSearch(q,  diversity) }  isLoading={isLoading}
      />

      <div  style={{  marginTop:  '1.5rem',  maxWidth:  '400px',  margin:  '1.5rem  auto
0'                      }}>
        <div  style={{  display:  'flex',  justifyContent:  'space-between',  fontSize:
'0.85rem',          color:          '#aaa',          marginBottom:          '0.5rem'          }}>
          <span><i          className="fas          fa-bullseye"></i>          Precision</span>
          <span>Discovery          <i          className="fas          fa-random"></i></span>
        </div>
        <input
          type="range"
          min="0"
          max="1"
          step="0.1"
          value={diversity}
          onChange={ (e)          =>          setDiversity(parseFloat(e.target.value)) }
          style={{          width:          '100%',          cursor:          'pointer'          }}
        />
        <p  style={{  textAlign:  'center',  fontSize:  '0.8rem',  color:  '#888',
marginTop:          '0.5rem'          }}>
          {diversity  ===  0  ?  'Exact  matches  only'  :  `Diversity  level:
${Math.round(diversity          *          100)}%`}
        </p>
      </div>
    </div>

    {searchQuery          &&          (
      <section                    className="search-results">
        { /*          Variant          G:          Visual          Vibe          Banner          */ }
        {vibe          &&          !vibeLoading          &&          (

```

```

<div style={{
  ...vibeStyle,
  marginBottom: '1.5rem',
  display: 'flex',
  alignItems: 'center',
  gap: '1rem',
}}>
  <span style={{ fontSize: '2.5rem' }}>{vibe.emoji}</span>
  <div>
    <p style={{ color: '#fff', fontWeight: 700, fontSize: '1.1rem',
margin: 0 }}>
      {vibe.mood.charAt(0).toUpperCase() + vibe.mood.slice(1)} Vibe
    </p>
    <p style={{ color: 'rgba(255,255,255,0.7)', fontSize: '0.85rem',
margin: 0 }}>
      AI-generated mood palette for "{searchQuery}"
    </p>
  </div>
  <div style={{ marginLeft: 'auto', display: 'flex', gap: '8px' }}>
    {[vibe.gradientFrom, vibe.gradientTo, vibe.accent].map((c, i) => (
      <div key={i} style={{
        width: 28, height: 28, borderRadius: '50%',
        backgroundColor: c, border: '2px solid rgba(255,255,255,0.3)'
      }} title={c} />
    ))}
  </div>
</div>
))
{vibeLoading && (
  <div style={{ marginBottom: '1rem', color: '#888', fontSize: '0.85rem'
}}>
    <i className="fas fa-magic fa-spin"></i> Generating mood palette...
  </div>
)}
{vibeError && !vibeLoading && (
  <div style={{ marginBottom: '1rem', color: '#aaa', fontSize: '0.8rem',
fontStyle: 'italic' }}>
    <i className="fas fa-palette"></i> Mood palette unavailable (Gemini
quota)
  </div>
)}

<h2 className="title is-4">Results for "{searchQuery}"</h2>

```

ДОДАТОК Б

Тези доповіді у
збірнику матеріалів

XXIII Міжнародної науково-практичної конференції молодих вчених та студентів

«Сучасні проблеми наукового забезпечення енергетики: безпека, стійкість, ІТ та
екологічний моніторинг»

УКР.НТУУ «КПІ ім. Ігоря Сікорського»_ІАТЕ_ЦТЕ_ТР_21

Аркушів 5

Київ - 2026

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ІМЕНІ ІГОРЯ СІКОРСЬКОГО»

**СУЧАСНІ ПРОБЛЕМИ
НАУКОВОГО ЗАБЕЗПЕЧЕННЯ
ЕНЕРГЕТИКИ:
БЕЗПЕКА, СТІЙКІСТЬ,
ІТ ТА ЕКОЛОГІЧНИЙ МОНІТОРИНГ**

XXIII Міжнародна науково-практична конференція
молодих вчених та студентів

До 40-річчя Чорнобильської катастрофи

17–22 квітня 2026 року, м. Київ



Київ-2026

Application of the transformer architecture for atmospheric precipitation intensity forecasting	422
<i>NAHAIVSKA D. O., BS's programme</i>	
<i>Academic leader - assist. Holovakin M. A.</i>	
Integration of a Generative AI Model into a Semantic Vector-Based Music Recommendation System	424
<i>VANIN A. V., BS's programme</i>	
<i>Academic leader - assist. Holovakin M. A.</i>	
Розробка веб-застосунку для планування задач з елементами ігрофікації та віртуальним аватаром	426
<i>БІЛОУС О. В., бакалаврант</i>	
<i>Керівник - доц., к.е.н. Сегеда І. В.</i>	
Розробка інтерактивного кооперативного 2D extraction-шутера	429
<i>БУЛАК В. В., бакалаврант</i>	
<i>Керівник - асист. Кардашов О. В.</i>	
Вебзастосунок для вивчення іноземних мов	432
<i>ДІБРОВА Є. В., бакалаврант</i>	
<i>Керівник - доц., к.т.н. Кублій Л. І.</i>	
Вебзастосунок для проведення онлайн-конференцій	434
<i>ЗЕЛІНСЬКА В. В., бакалаврант</i>	
<i>Керівник - доц., к.т.н. Кублій Л. І.</i>	
Корпоративна веб-система управління робочими змінами та персоналом	436
<i>КАРПЕНКО Д. І., бакалаврант</i>	
<i>Керівник - доц., к.ф.-м.н. Донець А. Г.</i>	
Розробка веб-орієнтованої ERP-системи управління меблевим виробництвом з інтеграцією даних САПР	438
<i>КОХАНЕВИЧ Д. Г., бакалаврант</i>	
<i>Керівник - асист. Кардашов О. В.</i>	
Мережевий експортер метрик вузла у форматі OpenMetrics і його інтеграція в систему моніторингу	440
<i>МАЛЮК І. О., бакалаврант</i>	
<i>Керівник - доц., к.т.н. Кублій Л. І.</i>	
Мобільний застосунок для координації волонтерських процесів	443
<i>САХАЦЬКА І. М., бакалаврант</i>	
<i>Керівник - доц., к.ф.-м.н. Тарнавський Ю. А.</i>	
Застосування методів комп'ютерного моделювання для оцінювання безпеки інформаційних систем відповідно до стандарту OWASP ASVS	446
<i>ХНЮКАЛО В. С., бакалаврант</i>	
<i>Керівник - доц., к.т.н. Полягушко Л. Г.</i>	
Information technology of supervisory control based on a digital twin for safe extraction of LFCM at the shelter object	449
<i>PROSKURIN O. S., postgraduate</i>	
<i>Academic leader - sn.res.fellow, cand.eng.sc. Saveliev M. V.</i>	
Соціально-економічні виклики реінтеграції тимчасово-окупованих територій України (приклад організації управління охороною здоров'я в АР Крим)	452
<i>НАКОРЕНКО Д. А., бакалаврант; РУДЕНКО Н. А., бакалаврант</i>	
<i>Керівник - проф., д.м.н. Дєєва Ю. В., проф.; д.е.н. Хлобистов Є. В.</i>	

UDC 004.8

¹ BS's programme 4th year Vanin A. V.

² Assist. Holovakin M. A.

<https://scholar.google.com.ua/citations?hl=en&user=bdC3JMgAAAAJ>

³ Igor Sikorsky Kyiv Polytechnic Institute

INTEGRATION OF A GENERATIVE AI MODEL INTO A SEMANTIC VECTOR-BASED MUSIC RECOMMENDATION SYSTEM

Statement of the problem and its relevance. With the increasing volume of media content, modern streaming services face the challenge of limited recommendation diversity. Traditional algorithms often suggest a narrow range of content based solely on listening history, resulting in low adaptability to a user's specific context or mood. The implementation of generative artificial intelligence models helps overcome these limitations by enabling personalization through the interpretation of complex natural language queries [1]. The integration of semantic and generative approaches represents a promising direction for improving adaptability, contextual awareness, and overall recommendation quality in contemporary streaming platforms.

Analysis of the latest research. An analysis of existing solutions demonstrates an active transition toward the application of transformer-based architectures in sequence modeling and multimodal representation learning [2]. Research in the field of Large Language Models and generative audio models shows high efficiency in understanding semantic relationships between text descriptions and musical characteristics [3]. However, the issue of integrating such models into high-load web applications with minimal latency remains open.

Contemporary research highlights the increasing importance of multimodal representation learning in music recommendation systems. Recent approaches integrate textual semantic embeddings with low-level acoustic features extracted directly from audio signals, including spectral, temporal, and harmonic descriptors. Such integration improves the alignment between user intent expressed in natural language and intrinsic musical characteristics, thereby enhancing recommendation accuracy and contextual sensitivity.

Research objective. To develop a method for generating personalized music recommendations based on semantic embeddings and implement it within the architecture of a music streaming web application.

Main part. To address the identified problem, a semantic matching method based on vector data representation was developed [4]. The web application architecture is implemented using the Node.js and React stack, ensuring asynchronous data processing. The method is based on a three-stage information transformation process (fig. 1), beginning with the formation of an input vector derived from the user's text prompt [5].

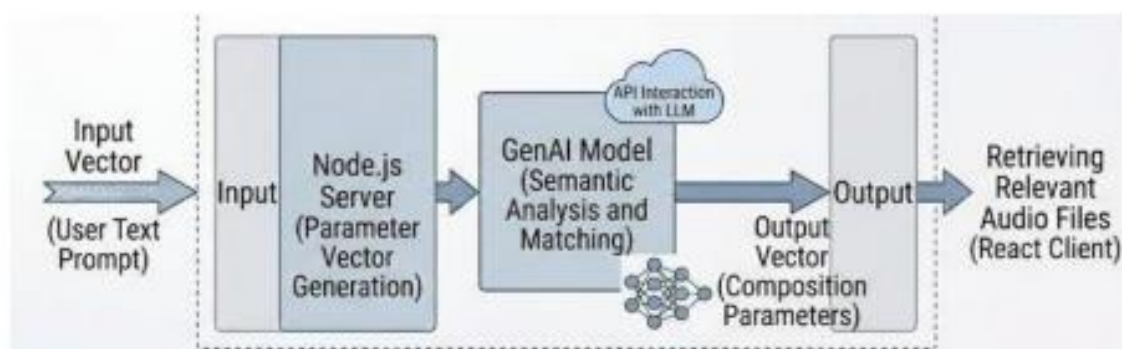


Figure 1 – Semantic Vector-Based Music Recommendation Process

The textual input is processed by a generative language model, which performs semantic analysis and produces a semantically enriched embedding representing the user's intent. This embedding is then matched with internal musical features such as tempo, key, and emotional tone. The final stage involves generating an output vector of relevant results used to query a vector database for the most appropriate audio tracks.

This architecture allows the Node.js server to efficiently coordinate interaction between the client interface and the intelligent model, ensuring high response speed and a significant increase in recommendation diversity beyond standard genre-based tagging.

The vector database stores precomputed embeddings of audio tracks generated using consistent feature extraction methods. By representing both user queries and audio files within the same semantic space, the system ensures efficient comparison through similarity metrics such as cosine similarity. This approach enhances scalability, as new tracks can be indexed by generating and storing their corresponding vectors without restructuring the database.

In addition, the modular architecture enables independent scaling of system components such as the AI inference layer, the database layer, and the frontend interface. The proposed system incorporates a feedback-driven refinement mechanism designed to continuously improve recommendation relevance. User interactions such as track skips, playlist additions, listening duration, and explicit ratings are collected and transformed into behavioral signals. These signals are subsequently embedded into the same semantic space as textual queries and audio features, enabling dynamic adjustment of recommendation vectors. As a result, the system gradually adapts not only to explicitly stated preferences but also to implicit behavioral patterns, increasing long-term personalization accuracy.

Overall, the integration of semantic embeddings, generative AI models, and scalable web technologies forms a cohesive framework capable of addressing modern challenges in music recommendation systems. The approach not only enhances contextual adaptability and diversity of suggested content but also establishes a flexible infrastructure for future extensions, including emotion-aware recommendation models and adaptive playlist generation.

Conclusions. The practical implementation confirms that semantic-generative integration can serve as an effective architectural paradigm for next-generation intelligent recommendation systems. The use of generative AI models enables flexible recommendation generation based on contextual understanding of user intent rather than solely on statistical listening history. The proposed architecture demonstrates the practical feasibility of integrating semantic embeddings and generative models within a scalable web-based system.

References:

1. Spotify Research. Text2Tracks: Improving Prompt-based Music Recommendations with Generative Retrieval. URL: <https://research.atspotify.com/2025/04/text2tracks-improving-prompt-based-music-recommendations-with-generative-retrieval> (date of access: 26.01.2026)
2. Vaswani A. et al. Attention Is All You Need. *Advances in Neural Information Processing Systems*. 2017. URL: <https://arxiv.org/abs/1706.03762> (date of access: 26.01.2026)
3. Huang Z., Chen C., Zeng Z. Dynamic Music Playlist Generation Using Deep Learning Techniques. *Expert Systems with Applications*. 2023. URL: <https://www.sciencedirect.com/science/article/abs/pii/S0952197623017888> (date of access: 26.01.2026)
4. Spotify Research. Contextual and Sequential User Embeddings for Music Recommendation. URL: <https://research.atspotify.com/contextual-and-sequential-user-embeddings-for-music-recommendation> (date of access: 26.01.2026)
5. Mikolov T., Chen K., Corrado G., Dean J. Efficient Estimation of Word Representations in Vector Space. 2013. arXiv preprint. URL: <https://arxiv.org/abs/1301.3781> (date of access: 28.01.2026)