СХІДНОУКРАЇНСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
ІМЕНІ ВОЛОДИМИРА ДАЛЯ

Навчально-науковий інститут (факультет) інформаційних технологій та електроніки

Кафедра інформаційних технологій та програмування

ПОЯСНЮВАЛЬНА ЗАПИСКА

до кваліфікаційної випускної роботи

освітній ступінь бакалавр
спеціальність 121 „Інженерія програмного забезпечення”
(шифр і назва спеціальності)

на тему „Розробка веб-застосунку з використанням методів NLP та інтеграції з API Spotify для аналізу емоційного забарвлення текстів пісень”

Виконав: студент групи ІПЗ-21д _____ Д. С. Турійов
(підпис) (ініціали і прізвище)

Керівник _____ Д. В. Ратов
(підпис) (ініціали і прізвище)

Завідувач кафедри _____ О.І. Захожай
(підпис) (ініціали і прізвище)

Рецензент ____Лифар В. О.____

Київ – 2025

СХІДНОУКРАЇНСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
ІМЕНІ ВОЛОДИМИРА ДАЛЯ

Навчально-науковий інститут (факультет) інформаційних технологій та електроніки
Кафедра інформаційних технологій та програмування

Освітній ступінь бакалавр
спеціальність 121 „Інженерія програмного забезпечення”
(шифр і назва спеціальності)

ЗАТВЕРДЖУЮ
Завідувач кафедри ІТП
_____ Захожай О.І.
“___” _____ 2025 року

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

Турійов Денис Сергійович
(прізвище, ім’я, по батькові)

1. Тема роботи: Розробка веб-застосунку з використанням методів NLP та інтеграції з API Spotify для аналізу емоційного забарвлення текстів пісень

Керівник роботи Ратов Денис Валентинович, к.т.н. , доцент
(прізвище, ім’я, по батькові, науковий ступінь, вчене звання)

затверджені наказом університету від “27” травня 2025 року № 67/15.15-С

2. Строк подання студентом роботи 20.06.2025

3. Вихідні дані до роботи: Об'єктом даної роботи є процес розробки вебзастосунку для аналізу емоційного забарвлення текстів пісень

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити) Аналіз предметної області та можливих шляхів реалізації інформаційної системи. Проектування інформаційної системи та шляху її життєдіяльності. Опис структури та атрибутів бази даних. Опис програмної реалізації вебзастосунку.

5. Перелік графічного матеріалу (з точним зазначенням обов’язкових креслеників) Електронні плакати

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв

7. Дата видачі завдання 01.04.2025

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів виконання кваліфікаційної випускної роботи	Строк виконання етапів	Примітка
1	Одержання завдання на виконання роботи	01.04.2025	виконано
2	Створення і погодження з керівником плану і етапів виконання роботи	03.04.2025	виконано
3	Аналіз предметної області та логіки методів NLP. Аналіз літературних джерел	23.04.2025	виконано
4	Проектування інформаційної системи	15.05.2025	виконано
5	Пошук потрібних для програмної реалізації бібліотек	20.05.2025	виконано
6	Програмна реалізація модулів	06.06.2025	виконано
7	Оформлення пояснювальної записки	15.06.2025	виконано
8	Створення презентації і доповіді	19.06.2025	виконано

Студент _____ Д. С. Турійов
підпис (ініціали і прізвище)

Керівник роботи _____ Д. В. Ратов
підпис (ініціали і прізвище)

РЕФЕРАТ

Пояснювальна записка до кваліфікаційної роботи бакалавра: 69 с., 10 рис., 1 табл., 20 бібліографічних джерел посилань, 2 додатки.

Об’єкт розробки: процес аналізу текстів пісень з метою визначення їх емоційного забарвлення.

Мета роботи: розробка вебзастосунку для автоматизованого аналізу емоційного змісту пісенних текстів із використанням методів обробки природної мови (NLP) та інтеграції з API-сервісами.

У результаті виконаної роботи проаналізовано можливості застосування NLP у музичному контексті, досліджено методи емоційного аналізу текстів (лексиконні, алгоритмічні, гібридні), вивчено API-сервіси для отримання текстів пісень. Сформовано функціональні та нефункціональні вимоги, розроблено архітектуру клієнт-серверної інформаційної системи, реалізовано модулі для автоматичного збору текстів, їх обробки та аналізу. Застосунок забезпечує візуалізацію результатів у вигляді хмар слів та графіків емоційного профілю.

Програмне забезпечення реалізовано мовою Python з використанням Flask, бібліотек NLTK, TextBlob, VADER, wordcloud, а також JavaScript-бібліотеки WordCloud2.js для фронтенд-візуалізації. Дані зберігаються у SQLite, а модулі кешування зменшують навантаження на API.

Ключові слова: NLP, емоційний аналіз, тексти пісень, хмара слів, Spotify API, Genius API, вебзастосунок, Python, Flask, візуалізація.

ЗМІСТ

ВСТУП

У сучасному цифровому середовищі музика продовжує відігравати важливу роль у культурному, емоційному та комунікативному житті суспільства. Вона є не лише формою мистецького самовираження, а й відображенням внутрішнього світу виконавців та слухачів. Зокрема, текстова складова пісень — лірика — містить значний обсяг емоційної, символічної та змістової інформації. Саме через текст пісні слухач може ідентифікувати власні переживання, побачити відображення особистих або соціальних тем, встановити емоційний зв’язок із твором.

З розвитком цифрових технологій, інтернету та потокових платформ (Spotify, Apple Music, YouTube Music), користувачі отримали доступ до мільйонів музичних композицій. Проте разом зі зростанням обсягу контенту виникла проблема його осмислення та структуризації. Стандартні механізми рекомендацій базуються переважно на жанровій належності або історії прослуховувань, тоді як емоційне забарвлення текстів залишається поза увагою.

На цьому фоні зростає інтерес до використання методів обробки природної мови (Natural Language Processing, NLP) для аналізу текстів пісень. Особливо перспективними є напрямки емоційного аналізу (sentiment analysis), визначення частотності лексем, семантичного ядра тексту, а також візуалізація отриманих даних у формі хмар слів (word clouds). Такі підходи відкривають нові можливості для дослідників, музикантів, маркетологів та звичайних слухачів — дозволяючи краще розуміти зміст пісень, виявляти емоційні патерни та створювати персоналізовані добірки за настроєм.

Актуальність дослідження зумовлена потребою у теоретичному осмисленні наявних інструментів і методів аналізу музичних текстів у цифровому середовищі, а також перспективами їх застосування у практичних розробках (зокрема у рамках майбутнього дипломного проєкту).

Мета роботи — здійснити теоретичне дослідження можливостей аналізу текстів пісень за допомогою сучасних інструментів обробки природної мови, з акцентом на емоційне забарвлення та візуалізацію змісту.

Для досягнення мети були поставлені наступні завдання:

- проаналізувати роль лірики в музичному сприйнятті та соціокультурному контексті;
- здійснити огляд публічних API для отримання текстів пісень;
- розглянути основні підходи до обробки текстових даних у NLP;
- систематизувати методи емоційного аналізу тексту;
- визначити можливості візуалізації текстової інформації (зокрема у формі хмар слів);
- окреслити можливі сфери практичного застосування отриманих результатів.

Об’єктом дослідження є процес інтерпретації музичних текстів засобами цифрових технологій. Предметом дослідження виступають методи аналізу емоційного забарвлення та лексичної структури текстів пісень.

ТЕОРЕТИЧНІ АСПЕКТИ АНАЛІЗУ МУЗИЧНИХ ТЕКСТІВ

Попри те, що музика здатна викликати емоції навіть без слів, саме текст пісні часто стає головним засобом комунікації між виконавцем і слухачем. Лірика формує змістове ядро композиції: вона передає думки, настрої, внутрішні стани та історії. У сучасному музичному просторі, де тексти є доступними в цифровому форматі, їх значення посилюється — не лише з точки зору сприйняття, а й як об’єкт аналітики.

Для багатьох слухачів слова пісень — це своєрідний емоційний компас. Музика підбирається не лише за жанром чи ритмом, а й за настроєм забарвленням лірики. Тексти, що передають почуття радості, смутку, сили чи втрати, мають прямий вплив на психоемоційний стан аудиторії. Часто пісні стають особистими рефлексіями слухача, способом емоційного співпереживання або навпаки — втечі від дійсності.

Текстовий зміст пісень відображає не лише індивідуальні переживання, а й ширші соціальні тенденції. Через пісенну лірику можна простежити зміни в домінантних темах і емоціях, які переважають у певні періоди. Наприклад, у музиці певних десятиліть відчутна зміна наративів — від оптимістичного тону й романтичних образів до складніших тем: внутрішніх конфліктів, соціального тиску, самотності чи тривоги. Усе це свідчить про зв’язок між емоційним тоном лірики та колективним психологічним станом суспільства.

Окрему увагу заслуговує питання емоційного забарвлення тексту. На відміну від лексичного значення слів, емоційне забарвлення виражає їхню здатність викликати певні почуття — наприклад, слово "ніч" у контексті пісні може викликати як спокій (у романтичній баладі), так і страх (у драматичній композиції). Емоційний підтекст формується не лише через окремі слова, а й через поєднання образів, інтонацію, повтори, контрасти. Саме завдяки цьому лірика музичних творів часто має метафоричну або символічну структуру, яка дозволяє кожному слухачеві інтерпретувати її по-своєму.

У популярній музиці цей емоційний вимір стає особливо помітним. Тексти пісень масової культури — це дзеркало колективних відчуттів: любові, втрати, пошуку сенсу, самотності або протесту. Із розвитком цифрових технологій і глобального доступу до стрімінгових платформ з'явилась можливість бачити, які саме емоції «вирують» у популярних треках певного року чи періоду. Таким чином, пісні стають не лише способом вираження себе, а й індикаторами культурних і соціальних зрушень.

Сьогодні лірика також відіграє роль у побудові бренду виконавця. Артисти, які систематично звертаються до певних тем або настроїв, формують навколо себе емоційну аудиторію — слухачів, які впізнають себе в цих текстах. Відтак емоційна тональність композицій часто стає важливою частиною стратегії позиціонування в музичному середовищі.

1.1 Основи обробки природної мови (NLP)

Обробка природної мови (NLP) — це напрямок в галузі штучного інтелекту та лінгвістики, що займається створенням алгоритмів та моделей для аналізу, інтерпретації та генерації людської мови. Основна мета NLP — навчити комп'ютери розуміти текст і мову так, як це робить людина. Важливою частиною цього процесу є автоматична обробка текстів для подальшого аналізу та видобутку корисної інформації.

У контексті дослідження, обробка природної мови набуває особливого значення при роботі з текстами пісень. Музичні тексти часто відрізняються від літературних або інших типів текстів низкою особливостей. Вони можуть включати метафори, символи, рими, химерні граматичні конструкції, повтори — все це створює унікальне середовище для обробки і аналізу.

Основними завданнями NLP є:

- **Сегментація** — процес розділення тексту на окремі елементи, або токени (слова, фрази чи інші одиниці значення). У випадку пісенних текстів

токенізація дозволяє виділити окремі слова, що є основними одиницями для подальшого аналізу.

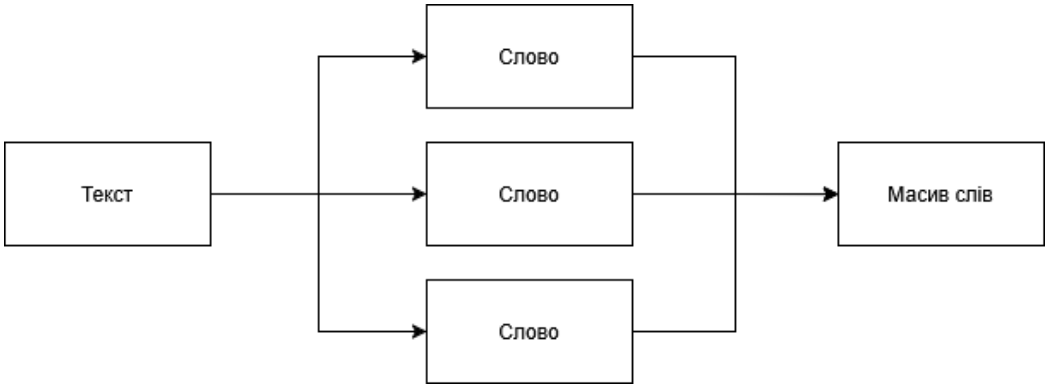

Рис. 1.1 - Приклад Сегментації(токенізації)

- **Очищення тексту** — процес видалення зайвих або непотрібних елементів тексту, таких як стоп-слова (наприклад, «і», «це», «але»), знаки пунктуації, спецсимволи, зайві пробіли. Це критично важливо, оскільки в піснях часто зустрічаються слова, що не несуть значення, але впливають на структуру та час обробки.

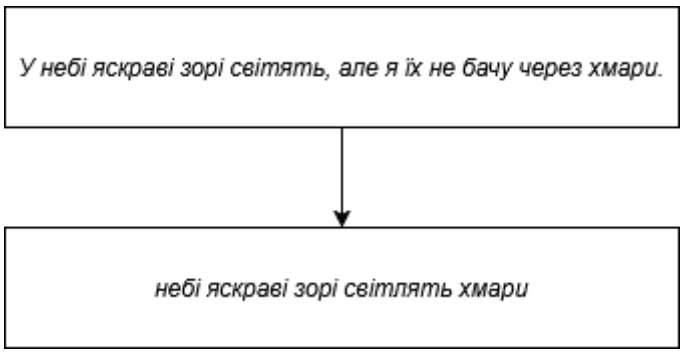

Рис. 1.2 - Приклад очищення тексту

- **Лематизація та стемінг** — це процеси, що дозволяють звести різні форми слів до їх базової чи кореневої форми. Лематизація повертає слово до його базової лексичної форми (наприклад, «біжу» → «бігти»), тоді як стемінг

просто обрізає частини слів (наприклад, «running» → «run»). Для пісенних текстів це важливий етап, оскільки часто зустрічаються різні форми одних і тих самих слів, що може спростити аналіз.

- **Частотний аналіз** — оцінка того, як часто зустрічаються певні слова чи вирази в тексті. У піснях важливу роль грають не тільки окремі слова, а й їхні комбінації, що складають рими чи повтори.

У випадку пісенних текстів особливу складність становить те, що вони часто є скороченими, емоційно насиченими і можуть використовувати нестандартні мовні форми (наприклад, діалекти, скорочення, неформальні вирази). Тому важливо враховувати контекст, в якому використовуються ті чи інші слова. Так, у піснях можна зустріти терміни або образи, що є зрозумілими лише в певному культурному чи соціальному контексті.

Ще однією важливою особливістю є наявність рим і ритмічних патернів. Це означає, що для обробки пісенних текстів потрібно враховувати їх структуру не лише з лінгвістичної точки зору, а й з точки зору музичного ритму. Тому для аналізу лірики часто використовуються моделі, які здатні розпізнавати контексти, що виходять за межі традиційного синтаксичного аналізу, та враховують музичний фон і темп пісні.

Важливою частиною NLP для пісенних текстів є також **емодіалект** — підхід, що дозволяє враховувати емоційну забарвленість тексту. В піснях часто використовуються вирази, що несуть чітке емоційне забарвлення (наприклад, «я тебе люблю», «ти залишив мене», «не можу більше терпіти»), і NLP допомагає виділити ці елементи, що є критичними для подальшого емоційного аналізу.

Усі ці методи є основою для подальшого емоційного аналізу пісенних текстів. Без належної обробки тексту важко буде провести коректний аналіз настрою чи виділити основні емоції, які він передає.

1.2 Методи емоційного аналізу текстів

Вивчення емоційного наповнення текстів — один із ключових напрямків сучасної лінгвістичної та прикладної інформатики, особливо в межах Natural Language Processing (NLP). Емоційний аналіз (також відомий як Sentiment Analysis або Opinion Mining) дозволяє виявити, яку емоційну реакцію викликає певний текст або які емоції в ньому виражено. У випадку музичних текстів, це стає особливо актуальним, адже пісні — це один із найвиразніших носіїв емоцій, які транслюються через слова, інтонацію та музичний супровід.

Емоційний аналіз зосереджується на таких двох основних рівнях:

- Визначення полярності тексту: позитивний, негативний або нейтральний настрій.
- Виявлення конкретних емоцій: радість, смуток, гнів, страх, здивування, відраза тощо.

Ці підходи можуть реалізовуватись за допомогою словникових методів, алгоритмів машинного навчання або гібридних підходів. Розглянемо основні методи та їхні особливості застосування до пісенної лірики.

1.2.1 Словникові методи

Цей підхід базується на попередньо створених словниках, у яких кожному слову приписано певне емоційне значення — полярність або приналежність до однієї чи кількох емоцій. Перевага словникових методів — їхня простота і прозорість, проте вони часто ігнорують контекст, і тому можуть бути менш точними в неоднозначних ситуаціях.

Найвідоміші лексичні ресурси:

- **AFINN** — словник, який містить понад 2,500 англійських слів із полярністю від −5 (найнегативніше) до +5 (найпозитивніше).

- **NRC Emotion Lexicon** (або EmoLex) — один із найпоширеніших словників, що пов’язує слова з вісімкою базових емоцій (радість, смуток, страх, гнів, подив, відраза, довіра, очікування) та полярністю.
- **LIWC** (Linguistic Inquiry and Word Count) — система аналізу тексту, яка дозволяє класифікувати слова за категоріями, що охоплюють емоційність, когнітивні процеси, соціальні відносини тощо.

Словникові методи добре підходять для коротких текстів, таких як тексти пісень, особливо якщо вони досить прямолінійні у вираженні емоцій. Проте проблема виникає, коли одне слово може нести різні емоційні конотації залежно від контексту. Наприклад, слово “вогонь” може означати як небезпеку, так і захоплення ("палає серце").

1.2.2 Алгоритмічні методи (машинне навчання)

Алгоритми машинного навчання дозволяють моделі самостійно навчитися виявляти емоційний настрій тексту на основі великої кількості прикладів. Для цього необхідні корпуси текстів, де кожному тексту або фрагменту вже призначено певну емоційну мітку.

Серед популярних підходів:

- **VADER** (Valence Aware Dictionary and sEntiment Reasoner) — поєднує словниковий метод із правилами обробки емоційних модифікаторів (напр. “дуже”, “трохи”), великої літери, знаків оклику тощо. Надзвичайно популярний у соціальних медіа, добре працює з короткими та неформальними текстами, а отже, підходить і для пісень.
- **TextBlob** — зручна Python-бібліотека, що використовує наївний баєсівський класифікатор для визначення полярності й суб’єктивності тексту.
- Моделі на основі трансформерів (наприклад, BERT, RoBERTa) — сучасний підхід, що дозволяє враховувати контекст і багатозначність слів. Такі

моделі вже навчено на великих корпусах, і вони показують високі результати навіть у задачах із вираженим стилістичним навантаженням.

Недолік методів машинного навчання полягає у потребі великих навчальних наборів та обчислювальних ресурсів. Однак ці моделі здатні враховувати контекст, що є надзвичайно цінним при аналізі образної та поетичної мови пісень.

1.2.3 Особливості аналізу пісень

Тексти пісень мають кілька унікальних рис, які впливають на якість емоційного аналізу:

- Коротка довжина — тексти пісень часто є надто стислими для повноцінного аналізу, особливо для моделей, які потребують багато контексту.
- Повтори — в піснях часто повторюються рядки, фрази або навіть куплети. Це може спотворювати статистичний аналіз або навпаки — підсилювати певні емоції.
- Метафори та символізм — поетичний стиль, у якому пряме значення слова може суттєво відрізнятися від того, яке мається на увазі. Наприклад, «мій світ зник» — це не буквальне твердження, а емоційна метафора.
- Нестандартна лексика — молодіжний сленг, жаргон, креативне написання слів, іншомовні вставки — усе це ускладнює традиційні методи аналізу.

Застосування методів емоційного аналізу до пісенної лірики потребує адаптації: або через попередню обробку тексту (очищення, нормалізацію), або через навчання моделей безпосередньо на подібних текстах.

1.2.4 Комбіновані підходи

Найбільш ефективним способом аналізу на практиці є гібридна стратегія, тому що вона поєднує словникові методи з машинним навчанням. Наприклад, початковий аналіз здійснюється за допомогою **NRC Lexicon** для отримання початкових емоцій, а далі результати уточнюються моделлю **VADER** або трансформером. Такий підхід дозволяє компенсувати недоліки кожного з методів та надати більш комплексного аналізу, а не тільки одну з змінних.

Емоційний аналіз текстів пісень — це складний, але надзвичайно важливий процес, який поєднує лінгвістичні методи з сучасними технологіями штучного інтелекту. Подальший розвиток NLP-моделей, адаптованих саме для музичних текстів, відкриває нові можливості для дослідження зв’язку між мовою, емоціями та музикою. Це може знайти застосування не лише в академічних дослідженнях, а й у музичній індустрії, соціальних медіа та навіть психології мистецтва.

1.3 Візуалізація результатів: хмари слів

Обробка текстової інформації у сфері музичного аналізу передбачає не лише виявлення значущих елементів змісту, але й їхню подальшу інтерпретацію — як правило, візуальну. Одним із найпростіших і водночас ефективних способів такої інтерпретації є побудова хмар слів (word clouds).

Хмара слів — це графічне представлення частоти слів у тексті, де розмір кожного слова відповідає його частоті або важливості. У контексті аналізу пісень вона дозволяє швидко і наочно побачити, які лексеми домінують у певному тексті, альбомі чи навіть у творчості виконавця загалом. Така візуалізація особливо корисна при роботі з великим обсягом текстів, коли складно здійснити повноцінний ручний огляд.

Основна сила хмари слів полягає у її здатності миттєво передати загальний настрій або тематику тексту. Наприклад, якщо у піснях певного артиста найбільшими словами у хмарі будуть «сонце», «серце», «добре», це може вказувати на сангвістичний або романтичний настрій. У свою чергу, переважання лексем типу «перемога», «супер», «грай» створює інше емоційне тло — динамічне, енергійне, мотиваційне.

Крім того, хмари слів можуть використовуватись як допоміжний інструмент для емоційного аналізу, слугуючи первинним способом «занурення» в текстовий корпус перед глибшим машинним або лексичним аналізом.

Щоб побудувати репрезентативну хмару слів, текст необхідно попередньо обробити:

- привести до нижнього регістру;
- видалити знаки пунктуації та небуквені символи;
- очистити текст від стоп-слів — найуживаніших граматичних елементів, які не несуть семантичного навантаження (наприклад, в англійській: the, and, is; в українській: і, в, на тощо);
- за потреби — провести лематизацію чи стемінг, щоб об’єднати форми одного слова (наприклад, «плакати» та «плаче» — в одне «плакати»).

У деяких випадках доцільно зберігати повтори, особливо якщо йдеться про пісні з домінантними рефренами. Такі повтори самі по собі можуть бути індикаторами емоційного напруження чи стилістичних особливостей виконавця.

Хмари слів є корисними не лише для одиничного аналізу, а й для виявлення трендів. Так, порівнюючи хмари слів у піснях різних років, можна простежити, як змінюється тематичний фокус виконавця або навіть усього музичного жанру. У дослідженнях поп-культури часто вдаються до подібної візуалізації, щоб продемонструвати, як тексти пісень відображають соціальні настрої певного періоду.

Також хмари можуть слугувати інструментом персоналізованого аналізу. Наприклад, аналізуючи тексти пісень, що входять до плейлисту конкретного користувача, можна візуалізувати загальний настрій або домінантну тематику його музичних вподобань.

Для побудови хмар слів найчастіше використовуються програмні засоби на основі мов програмування Python або JavaScript. У Python популярними є бібліотеки wordcloud та matplotlib. В JavaScript широко застосовується бібліотека WordCloud2.js. Це відкриває широкі можливості для інтеграції подібної візуалізації в інтерфейс користувача при створенні музично-аналітичних інструментів.

1.4 Інструменти для отримання текстів пісень

1.4.1 API-сервіси

Одним з найпоширеніших підходів до роботи з пісенними текстами є використання відкритих інтерфейсів прикладного програмування (API). Вони дозволяють автоматизовано отримувати тексти пісень, їх метадані, інформацію про виконавців, альбоми та інше.

- **Genius API** — один із найвідоміших сервісів, який дозволяє отримувати метаінформацію про пісню (назва, альбом, виконавець, жанр), а також надає посилання на сторінку з повним текстом композиції. Особливістю є те, що тексти часто супроводжуються анотаціями, які створює спільнота користувачів.
- **Musixmatch API** — пропонує доступ до текстів пісень різними мовами, включаючи часткову підтримку перекладів. Платформа інтегрується з багатьма музичними додатками та сервісами. API орієнтований на широкий набір функцій: від пошуку рядків до ідентифікації текстів у реальному часі.

- **AudD API** — спеціалізується на розпізнаванні музики та отриманні тексту за аудіофрагментом. Крім ліричних даних, надає додаткові функції, як-от отримання ідентифікатора треку в Spotify або Apple Music.

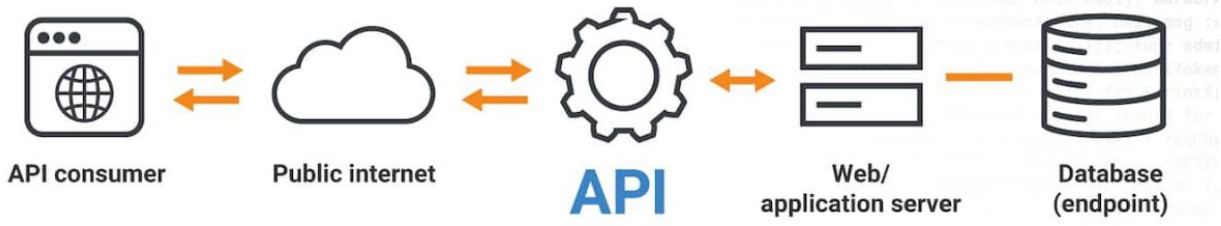

Рис. 1.3 - Візуалізація роботи API

Такі інтерфейси зручні у використанні для інтеграції в аналітичні вебсервіси, мобільні додатки та системи рекомендацій.

1.4.2 Парсинг вебсайтів

Ще один спосіб — отримання текстів безпосередньо з відкритих сайтів. Деякі платформи дозволяють переглядати тексти онлайн, що створює можливість для автоматизованого збору даних шляхом парсингу HTML-структури сторінок. При правильному підході це дозволяє отримати доступ до великої кількості матеріалів, включаючи альтернативні або неофіційні версії текстів. Такий метод часто використовується у дослідницьких цілях або для створення невеликих корпусів текстів, коли API не надає необхідної гнучкості. Парсинг вимагає більш глибоких технічних навичок (робота з HTML, обробка запитів), але водночас надає більше свободи щодо джерел.

1.4.3 Готові текстові корпуси

Існують відкриті корпуси текстів пісень, які збираються для наукових або освітніх цілей. Наприклад, деякі лінгвістичні лабораторії або дослідницькі центри публікують датасети, що включають тисячі текстів для машинного аналізу.

- **The Million Song Dataset** (у поєднанні з LyricWiki) — популярний корпус, що використовується в дослідженнях зі штучного інтелекту, обробки мови та музичної інформатики.
- **Lyrics Corpus by Kaggle** — платформи типу Kaggle часто містять датасети з текстами для публічного використання, які можна адаптувати до потреб дослідника.

Такі бази даних зручні тим, що вже підготовлені до обробки (очищені, стандартизовані), проте не завжди оновлюються регулярно.

1.4.4 Власна ручна збірка

У деяких випадках, особливо при роботі з нішевою або локальною музикою, дослідники самостійно збирають тексти пісень вручну. Це дає змогу точно контролювати зміст і формат, але потребує більше часу й ресурсів. Такий підхід корисний, коли важлива точність передавання контексту або коли тексти не представлені в загальнодоступних сервісах.

1.5 Сфери практичного застосування аналізу пісень

1.5.1 Персоналізовані музичні добірки за настроєм

Один із найпоширеніших запитів слухачів — знайти музику, що відповідає їхньому емоційному стану. Сервіси потокового мовлення вже частково використовують поведінкові алгоритми для рекомендацій, проте

аналіз текстів пісень може значно підвищити релевантність добірок. Наприклад, користувач, що шукає музику «для натхнення» або «щоб заспокоїтись», може отримати рекомендації на основі емоційного забарвлення пісенних текстів, а не лише темпу або жанру.

Застосування аналізу настрою пісень дозволяє створювати «емоційно-керовані» плейлисти, що формуються автоматично, залежно від запиту користувача.

1.5.2 Аналіз творчості виконавців і жанрів

Вивчення текстів у межах творчості окремого виконавця або жанру дозволяє виявляти повторювані теми, лексичні патерни, домінантні емоції та зміни в стилістиці. Наприклад, можна простежити, як упродовж кар'єри змінюється словник виконавця, чи стають тексти більш позитивними, соціально ангажованими або, навпаки, меланхолійними.

Аналогічно, аналіз текстів у межах певного музичного жанру (реп, інді-поп, рок тощо) дозволяє виявляти характерні риси, притаманні лексичні одиниці, а також еволюцію змісту з часом. Це може бути корисно як для музикознавців, так і для журналістів або критиків.

1.5.3 Маркетинг та аудиторна аналітика

Тексти пісень можуть слугувати маркером цільової аудиторії виконавця. Якщо у піснях часто згадуються теми самотності, підліткових переживань або протесту, це може свідчити про орієнтацію на молоду аудиторію. Натомість, наявність складної метафорики, соціальної критики чи філософських мотивів може вказувати на більш зрілу цільову групу.

Для музичних продюсерів, лейблів або стримінгових платформ такі дані є цінними: вони дозволяють краще таргетувати рекламу, розуміти потреби

слухачів і планувати стратегії просування. У поєднанні з метаданими (вік, регіон, вподобання слухачів) аналіз текстів відкриває новий вимір для музичного маркетингу.

1.5.4 Соціокультурний і суспільний аналіз

Музика часто відображає актуальні соціальні наративи, емоційний стан суспільства або навіть політичні настрої. Аналіз текстів пісень у часовій перспективі дозволяє виявити, як змінюються домінантні теми — наприклад, збільшення частоти слів, пов'язаних із тривогою, у кризові роки; популярність тем щастя або безтурботності під час економічного росту, тощо.

Такі дослідження можуть бути особливо корисними в соціологічних проектах. Пісня стає джерелом даних, що доповнює статистику, опитування або медіааналіз, надаючи більш емоційно забарвлений портрет епохи.

1.5.5 Освітні інструменти

Аналіз текстів пісень можна інтегрувати в освітній процес — зокрема, у вивченні мов, літератури або культури. Пісенні тексти, особливо в сучасних жанрах, слугують прикладом живої мови, що близька до реального вжитку. За допомогою NLP-інструментів і емоційної аналітики учні можуть не лише краще засвоювати лексику, а й аналізувати культурні особливості, стилі та емоційні контексти.

1.6 Постановка задачі

У сучасному інформаційному просторі музика виступає не лише формою мистецтва, а й значущим джерелом соціокультурної інформації. Пісенні тексти відображають емоційні стани, суспільні настрої, ідеологічні

посили та тренди певних історичних періодів. Водночас із зростанням обсягів цифрової музики зростає й потреба у системному аналізі її змісту — зокрема, з метою вивчення настрою композицій, автоматичного групування за емоційною тональністю та створення інструментів для візуального дослідження тематичних і мовних особливостей.

Особливу актуальність така задача набуває в умовах персоналізації музичного досвіду слухача: користувачі стрімінгових сервісів часто шукають пісні за настроєм або змістом, а не лише за жанром чи виконавцем. У відповідь на цей запит виникає потреба у побудові інтелектуальних систем, здатних автоматично аналізувати тексти пісень, визначати їхню емоційну спрямованість і візуалізувати результати у доступному форматі — наприклад, у вигляді хмар слів.

Дана робота присвячена вивченню підходів до аналізу текстів пісень за допомогою інструментів обробки природної мови та методів емоційного аналізу. Зокрема, досліджується можливість створення веборієнтованої системи, яка б на основі API отримувала тексти пісень, обробляла їх і представляла результати у зручному для користувача вигляді.

Для досягнення поставленої мети передбачається виконати такі завдання:

- проаналізувати сучасні методи отримання текстів пісень та засоби їх обробки;
- вивчити принципи обробки природної мови (NLP), зокрема ті, що адаптовані до художніх або поетичних текстів;
- дослідити методи емоційного аналізу текстів, включаючи словникові та алгоритмічні підходи;
- ознайомитися з підходами до візуалізації текстової інформації, зокрема побудови хмар слів;

- розробити вебзастосунок, який реалізуватиме зазначений функціонал: автоматичне отримання текстів пісень, їх емоційний аналіз, генерацію хмар слів і зручне візуальне подання результатів для користувача.

1.7 Висновки до розділу 1

Аналіз текстів пісень за допомогою методів обробки природної мови та емоційного аналізу відкриває нові можливості як для індивідуального користувача, так і для дослідників у сфері музики, культури та соціальних наук. Такі підходи дозволяють вивчати зміст пісень на глибшому рівні, виявляти домінантні настрої, тематичні патерни та лексичні особливості, що відображають культурний контекст і соціальні тренди.

Проаналізовано сучасний стан досліджень у галузі музичного контент-аналізу, розглянуто особливості пісенного тексту, окреслено принципи NLP та методи емоційного аналізу, зокрема лексиконні та алгоритмічні підходи. Було вивчено інструменти для отримання текстів пісень, досліджено способи візуалізації результатів у формі хмар слів та визначено практичні сфери застосування такого аналізу: від персоналізованих музичних добірок до соціокультурної аналітики.

Розгляд інструментів та технологій показав, що поєднання методів NLP з емоційним аналізом дозволяє створити гнучкі та ефективні рішення для інтерпретації музичного контенту. Значну роль при цьому відіграє якість джерел текстів, вибір лінгвістичних моделей та підходів до візуалізації. Подібний підхід не лише автоматизує аналіз великих обсягів даних, а й дозволяє виводити значущі висновки щодо динаміки музичних жанрів і суспільних настроїв.

1.8 Перелік джерел посилань до розділу 2

1. Brand C. O., Acerbi A., Mesoudi A. Cultural evolution of emotional expression in 50 years of song lyrics. Evolutionary human sciences. 2019. Vol. 1. URL: https://doi.org/10.1017/ehs.2019.11 (date of access: 15.04.2025).

2. How efficiency shapes human language / E. Gibson et al. Trends in cognitive sciences. 2019. Vol. 23, no. 5. P. 389–407. URL: https://doi.org/10.1016/j.tics.2019.02.003 (date of access: 15.04.2025).

3. Marshall S. R., Naumann L. P. What's your favorite music? Music preferences cue racial identity. Journal of research in personality. 2018. Vol. 76. P. 74–91. URL: https://doi.org/10.1016/j.jrp.2018.07.008 (date of access: 15.04.2025).

4. Reading song lyrics / ed. by E. Inc. Amsterdam : Editions Rodopi, 2010. 290 p.

5. Universality and diversity in human song / S. A. Mehr et al. Science. 2019. Vol. 366, no. 6468. P. eaax0868. URL: https://doi.org/10.1126/science.aax0868 (date of access: 14.04.2025).

6. Why are song lyrics becoming simpler? A time series analysis of lyrical complexity in six decades of American popular music / M. E. W. Varnum et al. PLOS ONE. 2021. Vol. 16, no. 1. P. e0244576. URL: https://doi.org/10.1371/journal.pone.0244576 (date of access: 15.04.2025).

7. Without it no music: cognition, biology and evolution of musicality / H. Honing et al. Philosophical transactions of the royal society B: biological sciences. 2015. Vol. 370, no. 1664. P. 20140088. URL: https://doi.org/10.1098/rstb.2014.0088 (date of access: 15.04.2025).

2 ПРОЕКТУВАННЯ ІНФОРМАЦІЙНОЇ СИСТЕМИ

2.1. Мета та завдання проектування інформаційної системи

Розробка системи аналізу емоційного забарвлення музичних текстів потребує детального визначення функціональних та нефункціональних вимог, які забезпечать ефективне виконання завдань обробки природної мови та візуалізації результатів. Система повинна поєднувати сучасні методи NLP з інтуїтивним користувацьким інтерфейсом для проведення комплексного емоційного аналізу пісенних текстів.

2.1.1 Функціональні вимоги

2.1.1.1 Автоматичне отримання текстів пісень через API

Система повинна забезпечувати автоматизоване отримання текстів пісень з зовнішніх джерел через спеціалізовані API. Основним джерелом обрано Genius API, який надає доступ до великої бази даних текстів пісень з можливістю пошуку за назвою композиції, виконавцем або альбомом. Система має підтримувати резервні джерела, зокрема функціонал кешування, для забезпечення автономності отримання даних.

Функціональність включає:

- Пошук пісень за метаданими (назва, виконавець)
- Отримання повних текстів пісень з урахуванням структури (куплети, приспіви, бридж)
- Кешування отриманих текстів для оптимізації повторних запитів
- Обробка помилок API та реалізація fallback-механізмів

2.1.1.2 Обробка текстів засобами NLP

Модуль обробки природної мови повинен реалізовувати повний пайплайн препроцесингу текстових даних, адаптований для специфіки музичних текстів. Система має ефективно обробляти неформальну мову, характерну для пісенної лірики.

Ключові функції NLP-модуля:

- **Токенізація** з урахуванням особливостей музичних текстів (скорочення, неологізми, емоційні вигуки)
- **Очищення тексту** від HTML-тегів, спеціальних символів, числових позначень структури пісні
- Нормалізація регістру
- **Фільтрація стоп-слів** з адаптованим списком для музичних текстів (виключення службових слів зі збереженням емоційно навантажених)

2.1.1.3 Емоційний аналіз текстів

Система повинна реалізовувати багаторівневий підхід до емоційного аналізу, що поєднує словникові методи з алгоритмічними підходами для досягнення максимальної точності визначення емоційного забарвлення.

Компоненти емоційного аналізу:

- **VADER (Valence Aware Dictionary and sEntiment Reasoner)** для аналізу тональності з урахуванням контексту та інтенсивності емоцій
- **TextBlob** для визначення полярності (позитивна/негативна) та суб'єктивності тексту
- **NRC Emotion Lexicon** для класифікації за базовими емоціями (гнів, страх, передбачення, довіра, подив та інше)
- **Комбінований підхід** з важеним усередненням результатів різних методів для підвищення точності аналізу

2.1.1.4 Генерація хмар слів з налаштуваннями фільтрації

Модуль візуалізації повинен створювати інтерактивні хмари слів, що відображають частотний розподіл та емоційну значущість лексики музичних текстів. Система має надавати широкі можливості налаштування для адаптації візуалізації під потреби користувача.

Функціональність генерації хмар слів:

- Автоматичне визначення найбільш значущих слів на основі частоти та емоційної ваги
- Виключення певних категорій слів (власні імена, технічні терміни, стоп-слова)

2.1.1.5 Візуалізація результатів емоційного аналізу

Система повинна надавати різноманітні форми візуалізації для наочного представлення результатів емоційного аналізу, що дозволить користувачам швидко інтерпретувати отримані дані.

Можливі види візуалізації:

- **Радар-діаграми** для відображення профілю 8 базових емоцій
- **Стовпчасті діаграми** для порівняння інтенсивності різних емоційних категорій
- **Лінійні графіки** для демонстрації динаміки настрою по структурних частинах пісні
- **Теплові карти** для візуалізації емоційної насиченості окремих рядків або куплетів
- **Порівняльні діаграми** для зіставлення емоційних профілів декількох композицій

2.1.2 Нефункціональні вимоги

2.1.2.1 Швидкість обробки текстів

Система повинна забезпечувати високу продуктивність обробки текстових даних для комфортної роботи користувачів. Максимальний час обробки одного тексту пісні не повинен перевищувати 5 секунд, включаючи всі етапи від препроцесингу до генерації візуалізації.

Параметри продуктивності:

- Час токенізації та очищення тексту: до 1.5 секунди
- Емоційний аналіз: до 2 секунд
- Генерація хмари слів: до 1.5 секунди

2.1.2.2 Точність емоційного аналізу

Система повинна демонструвати високу точність у визначенні емоційного забарвлення текстів.

Метрики точності:

- Багаторівневий підхід до емоційного аналізу

2.1.2.3 Адаптивний веб-інтерфейс

Користувацький інтерфейс повинен забезпечувати комфортну роботу на різних пристроях та роздільних здатностях екрану з дотриманням принципів responsive design.

Вимоги до інтерфейсу:

- Адаптація для настільних комп'ютерів (1920x1080 та вище)
- Оптимізація для планшетів (768x1024)
- Функціональність на мобільних пристроях (320x568 та вище)
- Час завантаження сторінки: до 3 секунд

- Сумісність з основними браузерами (Chrome, Firefox, Safari, Edge)

2.1.2.3 Обмеження частоти API-запитів

Система повинна ефективно управляти запитами до зовнішніх API та оптимізацією використання квот.

Параметри управління API:

- Інтелектуальне кешування для мінімізації повторних запитів
- Черга запитів

Визначені функціональні та нефункціональні вимоги формують основу для проектування архітектури системи та вибору відповідних технологічних рішень. Ці вимоги забезпечують баланс між функціональністю, продуктивністю та зручністю використання, необхідний для створення ефективного інструмента аналізу емоційного забарвлення музичних текстів.

2.2 Архітектура системи аналізу музичних текстів

Архітектурне проектування системи аналізу емоційного забарвлення музичних текстів вимагає ретельного балансу між продуктивністю, масштабованістю та підтримуваністю коду. Враховуючи специфіку NLP-обробки, необхідність інтеграції з зовнішніми API та вимоги до візуалізації результатів, архітектурне рішення повинно забезпечувати модульність, гнучкість та ефективне використання ресурсів.

2.2.1 Вибір архітектурного підходу

2.2.1.1 Обґрунтування вибору клієнт-серверної архітектури

Для системи аналізу музичних текстів обрано клієнт-серверну архітектуру як оптимальне рішення, що забезпечує розподіл обчислювальних

завдань та ефективне управління ресурсами. Це рішення мотивоване специфікою NLP-операцій, які потребують значних обчислювальних потужностей та спеціалізованих бібліотек, недоступних для виконання в браузері.

Переваги клієнт-серверної архітектури для даної системи:

Всі операції обробки природної мови, включаючи токенізацію, лематизацію та семантичний аналіз, виконуються на серверній стороні, що дозволяє використовувати бібліотеки машинного навчання без обмежень клієнтських середовищ. Це забезпечує оптимальне використання обчислювальних ресурсів, можливість застосування складних алгоритмів і централізоване оновлення моделей без необхідності змін у клієнтських додатках.

2.2.1.1.1 Управління зовнішніми API. Інтеграція з музичним API Spotify, реалізована через спеціалізовану бібліотеку, яку спрощує взаємодію з сервісом, обмеження запитів для дотримання лімітів зовнішніх сервісів. Кешування відповідей значно зменшує навантаження на зовнішні джерела, а трансформація даних у уніфікований формат спрощує подальшу обробку.

2.2.1.1.2 Оптимізація продуктивності за рахунок кешування. Система використовує багаторівневе кешування для зменшення часу відгуку та зниження навантаження. Результати семантичного аналізу зберігаються в реляційній базі даних з індексацією за хешем вмісту, що прискорює повторні запити. Тексти пісень кешуються з обмеженням за часом (TTL), щоб забезпечити актуальність даних. Пошукові запити оптимізуються за допомогою хешування, а проміжні результати обробки зберігаються для прискорення аналогічних операцій у майбутньому.

2.2.1.1.3 Організація системи та її компоненти. Архітектура системи реалізована як єдиний серверний застосунок, який виконує функції маршрутизації, обробки запитів, взаємодії з музичними API та NLP-аналізу. Flask-додаток працює як API Gateway — він обробляє всі запити з клієнта,

виконує маршрутизацію до відповідних обробників, забезпечує кешування даних і базову обробку помилок. Усі функціональні компоненти — включно з аналізом текстів, пошуком пісень, інтеграцією з Genius і Spotify — реалізовані в межах одного процесу.

Інтеграція з музичними API інкапсулюється в окремих функціях, які безпосередньо виконують запити до Genius і Spotify, обробляючи результати та реалізуючи fallback-механізми при помилках. Ці компоненти включають базове кешування запитів у локальній базі даних SQLite, що знижує навантаження на зовнішні сервіси та прискорює відповідь.

2.2.1.1.4 NLP-модуль та візуалізація. NLP-функціональність є ядром системи та реалізована в межах Flask-сервера. Вона включає попередню обробку текстів пісень (очищення, токенізацію, лематизацію) та емоційний аналіз на основі бібліотек VADER, TextBlob, а також власноруч зібраних емоційних лексиконів. Модуль не використовує складні ML-моделі на кшталт Word2Vec, але реалізує комбінований аналіз із використанням декількох алгоритмів для підвищення точності.

Візуалізація результатів — таких як хмари слів — генерується на клієнтській стороні за допомогою JavaScript та бібліотеки wordcloud2.js, а підготовка даних для неї (очищення тексту та підрахунок частоти слів) виконується на сервері. Це дозволяє забезпечити адаптивне представлення результатів аналізу під різні пристрої.

2.2.1.1.5 Особливості ефективності реалізованої системи. Запропонована реалізація демонструє гарну продуктивність на малих та середніх об’ємах даних, забезпечуючи швидку відповідь для більшості запитів завдяки попередньому кешуванню і спрощеній структурі. Завдяки використанню SQLite та кешу на рівні функцій, система здатна обробляти кілька запитів одночасно без суттєвого навантаження. Незважаючи на те, що горизонтальне масштабування не реалізоване, існуюча архітектура придатна для подальшого поділу на мікросервіси при необхідності. Поточна версія

системи ефективна для аналізу окремих пісень або плейлистів, виявлення загального емоційного профілю текстів та побудови тематичних візуалізацій.

2.2.1.2 Порівняння з мікросервісною архітектурою. На відміну від мікросервісного підходу, запропонована реалізація системи є монолітною — тобто всі функції, включно з API Gateway, NLP-аналізом, інтеграцією з зовнішніми API та візуалізацією, об’єднані в межах одного серверного застосунку. Це дозволяє швидко розгортати і тестувати систему, але обмежує її масштабованість та гнучкість у розробці.

Основними обмеженнями монолітної архітектури є складність розділення відповідальностей при розширенні функціональності, відсутність незалежного масштабування окремих компонентів та ризик каскадних відмов у випадку критичних помилок. Водночас для поточних задач — аналізу текстів пісень за запитом користувача, створення хмар слів і візуалізацій на клієнті — така архітектура є цілком прийнятною та ефективною. Надалі система може бути адаптована під мікросервісну модель із виділенням окремих сервісів для NLP, інтеграції з API та візуалізації результатів.

2.3 Технологічний стек

Для реалізації системи аналізу емоційного забарвлення музичних текстів було застосовано сучасний програмно-технологічний стек, який забезпечує високий рівень продуктивності, адаптивності, масштабованості та інтеграційної сумісності з зовнішніми API-сервісами. Вибір програмних засобів зумовлений специфікою завдань, пов’язаних із обробкою природної мови, формуванням REST-орієнтованих вебсервісів, забезпеченням локального кешування результатів та візуалізацією даних.

2.3.1 Серверна частина (Backend)

Бекенд (серверна частина) — це компонент програмного забезпечення, який відповідає за обробку логіки застосунку, збереження та управління даними, взаємодію з зовнішніми сервісами, а також забезпечення взаємозв’язку між користувацьким інтерфейсом (фронтендом) і серверними ресурсами. Саме на рівні бекенду реалізуються ключові алгоритми обробки даних, що забезпечують функціональну цілісність застосунку. Тут відбувається формування та виконання логіки процесів, включаючи перевірку коректності запитів, їх маршрутизацію, а також генерацію відповідей у відповідному форматі. Обробка запитів користувача включає аналіз отриманих даних, їх попередню валідацію та подальше спрямування до відповідних модулів системи. Доступ до баз даних на серверному рівні дозволяє здійснювати збереження, оновлення, видалення та запит інформації з урахуванням критеріїв безпеки, продуктивності та надійності. Окрім цього, бекенд забезпечує ефективну інтеграцію із зовнішніми прикладними інтерфейсами (API), що відкриває можливості для взаємодії із сторонніми сервісами — зокрема, для отримання, аналізу та обробки зовнішніх даних у реальному часі.

Базовою мовою програмування серверної частини обрано Python версії 3.10 або вище, що обумовлено її широкими можливостями в області Natural Language Processing (NLP), наявністю розвиненої екосистеми бібліотек для лінгвістичного аналізу, роботи з API, обробки HTML-структур, а також її придатністю для побудови швидкодіючих прототипів. Мова забезпечує ефективну реалізацію усіх етапів обробки текстової інформації — від збору даних до емоційного моделювання.

Фреймворк Flask було обрано як основне середовище для побудови веб-додатку. Він забезпечує реалізацію RESTful-архітектури, ефективну маршрутизацію запитів, можливість гнучкого налаштування серверної логіки,

підтримку шаблонів, а також розширення функціональності за допомогою спеціалізованих модулів. Простота структури Flask, разом із мінімалістичним підходом до побудови застосунків, дозволяє зосередитися на розробці прикладної логіки без надмірного ускладнення архітектури. Завдяки модульності, Flask легко адаптується до потреб конкретного проєкту, забезпечуючи масштабованість, розширюваність та можливість інтеграції з іншими інструментами — такими як бібліотеки для роботи з базами даних, механізми авторизації, або сервіси для обробки природної мови. У контексті реалізації системи аналізу текстів пісень цей фреймворк надає необхідні засоби для побудови серверного середовища, яке відповідає як функціональним, так і нефункціональним вимогам нашого вебзастосунку.

У межах реалізації запропонованої системи важливим компонентом є інтеграція із зовнішніми інформаційними сервісами за допомогою прикладних програмних інтерфейсів (Application Programming Interfaces, API). Застосування API забезпечує можливість налагодження стандартизованої взаємодії з віддаленими ресурсами, що містять структуровані дані, без необхідності зберігати ці дані локально або підтримувати власну базу музичного контенту. У даному контексті використовуються API двох авторитетних платформ — **Genius** та **Spotify**, які надають відкритий доступ до своїх музичних баз даних у межах визначених політик використання.

2.3.1.1 Genius API. Надає інтерфейс до обширного репозитарію текстів пісень, а також пов’язаних метаданих, зокрема назв композицій, імен виконавців, жанрової належності та візуального контенту. Через API реалізовано можливість здійснення пошукових запитів за ключовими словами та отримання у відповідь релевантних результатів з посиланням на повну версію тексту пісні, розміщену на вебсторінці сервісу. Незважаючи на те, що безпосередній доступ до текстів пісень через API обмежено з міркувань авторського права, структура відповіді містить URL-адресу, яка дозволяє програмно обробити HTML-вміст сторінки та витягти необхідний

лінгвістичний матеріал для подальшого аналізу. Таким чином, API виконує функцію пошукового посередника, який забезпечує доступ до автентичних джерел текстової інформації.

2.3.1.2 Spotify Web API, у свою чергу, є офіційним інструментом для взаємодії з музичним контентом платформи Spotify. Він дозволяє отримувати інформацію про публічні плейлисти, окремі музичні треки, альбоми та виконавців. API підтримує механізми автентифікації та авторизації відповідно до протоколу OAuth 2.0, що забезпечує безпечний та контрольований доступ до ресурсів. Отримані дані представлені у форматі JSON і можуть включати назву треку, імена виконавців, обкладинки альбомів, тривалість композиції, рівень популярності тощо. Така інформація є необхідною для формування списків композицій, які підлягають подальшому текстовому та емоційному аналізу в межах функціональності системи.

Для проведення лінгвістичного та емоційного аналізу текстів пісень застосовується комплекс спеціалізованих програмних бібліотек, серед яких ключове значення мають **NLTK**, **TextBlob** та **VADER**. Бібліотека **NLTK** (Natural Language Toolkit) виконує функції попередньої обробки текстових даних, зокрема здійснює нормалізацію, лематизацію та вилучення синонімічних рядів за допомогою ресурсу **WordNet**, що сприяє підвищенню якості подальшого аналізу. Інструмент **TextBlob** використовується для оцінювання полярності та суб’єктивності текстів, що дає змогу класифікувати пісні за ступенем емоційного навантаження, визначаючи позитивні, негативні або нейтральні відтінки емоційної експресії. Модуль **VADER** (Valence Aware Dictionary and sEntiment Reasoner), інтегрований у **NLTK**, спеціалізується на аналізі коротких англомовних текстів і базується на експертно розробленому лексиконі, що забезпечує високоточну детекцію позитивної, нейтральної та негативної тональності. Результати, отримані за допомогою зазначених інструментів, інтегруються для формування комплексного емоційного

профілю текстів, що дозволяє більш глибоко та комплексно оцінити їхню емоційну структуру.

Таблиця 2.1 – Порівняння NLP-систем

Критерій	NLTK	TextBlob	VADER
Призначення	Загальна лінгвістична обробка тексту	Аналіз полярності та суб'єктивності текстів	Емоційний аналіз коротких англомовних текстів
Типи обробки	Токенізація, лематизація, вилучення синонімів	Оцінка полярності та суб'єктивності	Оцінка позитивної, негативної та нейтральної тональності

Процес векторизації музичних композицій здійснюється на основі алгоритмічного підходу TF-IDF (Term Frequency–Inverse Document Frequency), реалізованого засобами програмної бібліотеки Scikit-learn. Зазначений метод забезпечує трансформацію неструктурованих текстових даних у багатовимірний числовий простір, що створює передумови для застосування математичних операцій над текстовими корпусами.

Для визначення ступеня семантичної близькості між текстовими документами застосовується метрика косинусної подібності (cosine similarity), яка обчислює кут між векторними представленнями документів у багатовимірному просторі ознак. Косинусна подібність між двома векторами A та B визначається формулою (2.1):

$$\cos(\theta) = (A \cdot B) / (||A|| \times ||B||) \quad (2.1)$$

де:

- $A \cdot B$ — скалярний добуток векторів A та B
- $||A||$ та $||B||$ — евклідові норми векторів A та B відповідно
- θ — кут між векторами

У розгорнутому вигляді для n-вимірних векторів (2.2):

$$\cos(\theta) = \Sigma_{i=1}^{n}(A_i \times B_i) / (\sqrt{\Sigma_{i=1}^{n}(A_i)^2} \times \sqrt{\Sigma_{i=1}^{n}(B_i)^2}) \quad (2.2)$$

Результуюче значення міри подібності знаходиться в діапазоні [-1, 1], де:

- 1 означає повну подібність векторів
- 0 означає ортогональність (відсутність подібності)
- -1 означає протилежну спрямованість векторів

Значення TF-IDF для терміна t у документі d визначається як (2.3):

$$\text{TF-IDF}(t,d) = \text{TF}(t,d) \times \text{IDF}(t) \quad (2.3)$$

де:

- $\text{TF}(t,d) = f(t,d) / \max\{f(w,d) : w \in d\}$ — нормалізована частота терміна
- $\text{IDF}(t) = \log(N / |\{d \in D : t \in d\}|)$ — обернена частота документа

Математичне обґрунтування використання косинусної міри полягає в її здатності нівелювати вплив довжини документа на результуючу оцінку подібності, зосереджуючись на кутовому відхиленні векторів.

Інтеграція описаних методів векторизації та обчислення подібності формує основу інтелектуальної системи рекомендацій, яка здійснює автоматизований відбір найбільш релевантних музичних композицій відповідно до специфікованих користувачем емоційно-семантичних критеріїв. Функціонування такої системи базується на принципі максимізації коефіцієнта семантичної відповідності між цільовим запитом та наявним корпусом текстових даних.

2.3.2 Клієнтська частина (Frontend)

Фронтенд (клієнтська частина) вебзастосунку — це компонент програмної системи, який відповідає за взаємодію з кінцевим користувачем через графічний інтерфейс. На цьому рівні відбувається відображення вмісту, обробка подій користувача, динамічна зміна елементів сторінки, а також передача запитів до серверної частини. Фронтенд відіграє ключову роль у забезпеченні доступності, зручності використання та інтуїтивної зрозумілості функціоналу, що є важливими аспектами користувацького досвіду (UX). З технічної точки зору, клієнтська частина реалізується за допомогою мов розмітки, стилізації та сценарного програмування, які спільно забезпечують адаптивність, інтерактивність та естетичну привабливість інтерфейсу.

Для глибшого розуміння логіки взаємодії користувача з інформаційною системою та визначення основних сценаріїв її використання доцільним є побудова діаграми прецедентів (**Use Case Diagram**). Діаграма прецедентів - це один із структурних інструментів моделювання в нотації UML (**Unified Modeling Language**), який використовується для візуального представлення функціональних вимог до програмної системи. Вона відображає взаємодію між зовнішніми користувачами системи (акторами) та основними функціями, які система повинна виконувати (прецедентами, або варіантами використання). Основною метою такої діаграми є формалізація поведінки системи з точки зору користувача, визначення ролей, сценаріїв використання та меж системи. Кожен прецедент описує певну логічно завершену послідовність дій, що виконується у відповідь на зовнішній запит користувача з метою досягнення конкретного результату.

На побудованій діаграмі прецедентів (рис. 2.1) представлено основні сценарії взаємодії користувача з клієнтською частиною інформаційної системи, що охоплюють такі дії, як *«Пошук пісень»*, *«Пошук плейлиста»*, *«Перегляд результатів»* та *«Пошук плейлиста за настроєм»*. Кожен із

зазначених варіантів відображає окрему функціональну активність, яка ініціюється користувачем з метою отримання релевантної музичної інформації.

Функціональні прецеденти *«Пошук пісень»* та *«Пошук плейлиста»* передбачають проведення автоматизованого аналізу текстового та метаданого контенту за допомогою методів обробки природної мови (NLP-аналізу). Збір відповідних даних здійснюється через інтеграцію з зовнішніми прикладними інтерфейсами API сервісів Spotify та Genius. Результати аналізу проходять подальшу обробку, після чого зберігаються у базі даних системи для забезпечення цілісності, повторного використання та підвищення продуктивності при наступних запитах.

Прецедент *«Пошук плейлиста за настроєм»* реалізує доступ до вже збережених у базі даних результатів емоційного аналізу музичного контенту, здійснюючи вибірку відповідних плейлистів на основі заданих емоційних параметрів. Усі згадані сценарії логічно з’єднані з функціональним елементом *«Перегляд результатів»*, який забезпечує відображення обробленої інформації у клієнтському інтерфейсі у зручному та візуально структурованому вигляді.

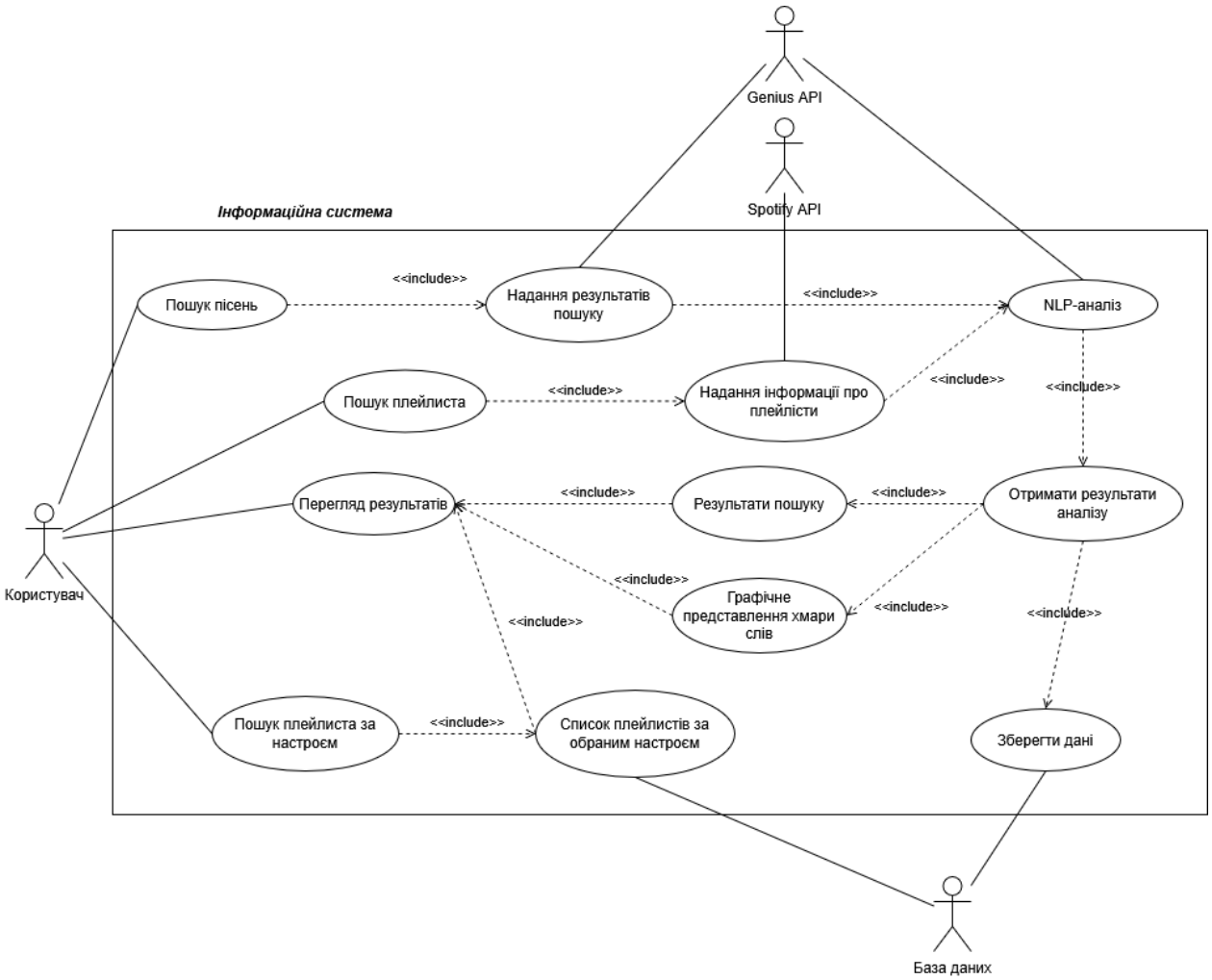

Рис. 2.1 – Сценарій поведінки користувача

Завдяки діаграмі прецедентів з’являється можливість систематизувати очікувану поведінку кінцевого користувача, виокремити ключові функціональні вимоги до системи, а також сформулювати перелік дій, які повинні бути реалізовані на рівні клієнтської частини. Діаграма наочно відображає відношення між користувачем і функціональними компонентами системи, фокусуючись на цільових завданнях, які він прагне виконати. Це, у свою чергу, сприяє обґрунтованому прийняттю рішень щодо структури інтерфейсу, вибору механізмів взаємодії, необхідного рівня інтерактивності та зручності у користуванні.

Базовими технологіями для реалізації клієнтської частини вебзастосунку обрано HTML5, CSS3 та JavaScript версії ES6+, що обумовлено їх

універсальністю, сумісністю з усіма сучасними веб-браузерами, а також широкими можливостями для створення інтерактивних і адаптивних інтерфейсів. Використання монолітного підходу дозволяє досягти високої швидкості завантаження, зменшити залежність від зовнішніх бібліотек і спростити розгортання застосунку, що є особливо доцільним у випадках середньої складності функціоналу. Така архітектурна модель забезпечує компактність структури, зручність супроводу на ранніх етапах розробки, а також повний контроль над усіма елементами відображення, що сприяє точній відповідності інтерфейсу функціональним вимогам системи.

2.3.2.1 HTML5 обрано у якості структурної основи клієнтської частини веб-застосунку розмітки, що зумовлено її широким функціональним потенціалом у сфері створення семантично організованих, доступних та ефективно оптимізованих інтерфейсів. HTML5 надає розробнику розширений набір семантичних елементів, зокрема <*header*>, <*main*>, <*nav*>, <*section*>, які дозволяють логічно структурувати вміст документа згідно з принципами доступності та стандартизованого представлення інформації. Такий підхід сприяє покращенню сумісності з допоміжними технологіями для користувачів з особливими потребами, а також підвищує ефективність індексації контенту пошуковими системами. Також, HTML5 забезпечує можливість використання нативних елементів форм з вбудованими механізмами валідації даних на клієнтському рівні, що дозволяє мінімізувати залежність від зовнішніх бібліотек, зменшує обсяг клієнтського коду та підвищує загальну продуктивність застосунку. Застосування стандартних атрибутів, таких як *required*, *placeholder*, а також визначення типів полів (наприклад, *text*, *url*) дає змогу реалізувати базову перевірку введених даних без додаткового навантаження на інтерфейс.

2.3.2.2 CSS3 (Cascading Style Sheets, Level 3) – це третє покоління мови каскадних таблиць стилів, яка використовується для опису зовнішнього вигляду HTML-документів, зокрема кольорів, шрифтів, розташування

елементів, а також анімацій та адаптивності. Основне призначення CSS3 полягає в тому, щоб відокремити структуру документа (визначену HTML) від його візуального оформлення, що сприяє більш чистій архітектурі веб-інтерфейсу та полегшує його підтримку. Вибір продиктований прагненням забезпечити сучасний, візуально привабливий, адаптивний інтерфейс, який відповідає актуальним вимогам до використання, кросбраузерності та швидкодії. CSS3 надає потрібний інструментарій для гнучкого управління візуальним представленням елементів, дозволяючи досягти високої якості відображення без залучення додаткових сторонніх бібліотек або фреймворків. Застосування цієї технології сприяє ефективній реалізації інтерфейсів, здатних забезпечити як естетичну привабливість, так і функціональну зручність для кінцевого користувача.

2.3.2.3 JavaScript стандарт ES8. На відміну від багатьох сучасних фреймворків, нативний підхід не створює додаткового рівня абстракції, що дозволяє досягти максимальної швидкодії та зменшити розмір фінального програмного пакета. Це особливо важливо в умовах обмежених ресурсів або при необхідності оптимізації швидкості завантаження та реакції інтерфейсу. Прямий доступ до **DOM API** надає можливості для точного налаштування динамічної поведінки елементів та оптимізації критичних ділянок коду. Окрему увагу приділено реалізації асинхронної взаємодії з серверною частиною. Асинхронна модель дозволяє здійснювати обмін даними між клієнтом і сервером без необхідності перезавантаження сторінки, що відповідає сучасним вимогам до швидкодії, адаптивності та зручності вебінтерфейсів Для обробки HTTP-запитів застосовується **Fetch API**, що відповідає сучасним стандартам обробки асинхронних операцій, забезпечує зручну роботу з промісами та дозволяє формувати гнучкі механізми обміну даними з сервером.

JavaScript виконує ключову роль у забезпеченні повноцінної та безперервної взаємодії між клієнтською та серверною частинами

інформаційної системи. Його функціональні можливості в повному обсязі покривають усі вимоги до реалізації комунікаційного шару, починаючи з формування, надсилання та обробки HTTP-запитів, і завершуючи асинхронною обробкою отриманих відповідей, управлінням помилками, а також динамічним оновленням інтерфейсу на основі даних, що надходять від серверу.

2.3.2.4 Клієнтські бібліотеки. Для реалізації функціональних вимог вебзастосунку особливу увагу приділено вибору зовнішніх бібліотек, які не лише оптимізують реалізацію окремих функціональних компонентів, але й відповідають вимогам надійності, підтримуваності та гнучкості інтеграції у загальну архітектуру застосунку. Вибір таких рішень обумовлено доцільністю делегування частини задач перевіреним інструментам, що дозволяє підвищити ефективність розробки без втрати контролю над логікою інтерфейсу.

Засобом для спрощення роботи з DOM-структурою HTML-документа було обрано **jQuery.** Забезпечуючи високий рівень абстракції над нативним DOM API браузера, вона значно спрощує реалізацію маніпуляцій із вмістом сторінки, подієвої моделі та анімацій. Також автоматично нейтралізує відмінності в реалізації браузерами, що робить її особливо корисною у випадках, коли застосунок повинен підтримувати широкий спектр клієнтських платформ.

WordCloud2.js використовується як спеціалізований інструмент для візуалізації текстових даних у форматі хмар слів. Бібліотека реалізує складні алгоритми просторового розміщення слів з урахуванням їх частотності, ваги та обраної геометрії області відображення. Завдяки використанню графічного контексту HTML5 Canvas, вона забезпечує високу продуктивність навіть при візуалізації великих масивів текстової інформації. WordCloud2.js також підтримує гнучке налаштування стилістичних параметрів — таких як розмір, колір, шрифт, орієнтація слів — що дає змогу створювати візуально привабливі та інформативні представлення результатів аналізу.

Toastify.js інтегрується для реалізації механізму повідомлень про події — так званих toast-сповіщень — які оперативно інформують користувача про успішне виконання або помилки при здійсненні дій. Бібліотека вирізняється невеликим розміром, відсутністю зовнішніх залежностей, а також високим ступенем кастомізації. Вона підтримує зміни кольору, позиціювання, тривалість відображення повідомлень, а також доступність для користувачів з обмеженнями (зокрема через підтримку screen readers). Це робить її доцільним вибором для створення ненав'язливого, але інформативного зворотного зв'язку.

2.3.2.5 Вибір клієнтського рендерингу (SPA). Згідно з функціональних вимог та вимог до інтерактивності інтерфейсу, вебзастосунок буде реалізовано у вигляді односторінкового застосунку (Single Page Application, SPA). Такий підхід передбачає динамічне оновлення вмісту сторінки без повного перезавантаження, що забезпечує підвищену швидкодію, зменшення мережевого навантаження та покращення користувацького досвіду.

Архітектура SPA дозволяє централізовано керувати станом інтерфейсу на клієнтській стороні, що, у поєднанні з використанням HTML5, CSS3 та JavaScript ES8+, забезпечує плавну навігацію між логічними секціями застосунку, мінімізуючи час відгуку системи. У межах даної моделі основна взаємодія з серверною частиною відбувається через API-запити, які повертають лише необхідні дані, без надмірної передачі HTML-контенту, що додатково оптимізує продуктивність.

Застосування SPA також сприяє більшій контролюваності над інтерфейсними подіями та дозволяє реалізувати розширену логіку клієнтської взаємодії, зокрема для роботи з результатами запитів, фільтрації, попереднього рендерингу або анімаційних ефектів.

2.3.3 Система кешування

Бази даних є невід'ємною складовою сучасних інформаційних систем, оскільки забезпечують централізоване збереження, структурований доступ і ефективне управління даними. Їх використання дозволяє не лише акумулювати значні обсяги інформації, а й організовувати її у впорядковану, логічно зв'язану форму, що відповідає потребам конкретної предметної області. Завдяки реляційній природі, бази даних забезпечують чітке визначення взаємозв'язків між сутностями (наприклад, між піснями, плейлистами та результатами аналізу), що, в свою чергу, сприяє побудові цілісної інформаційної моделі.

Однією з ключових переваг використання баз даних є **можливість індексованого доступу** до даних. Індекси, які створюються на основі значень часто використовуваних полів (наприклад, унікальних ідентифікаторів чи ключових слів пошуку), суттєво зменшують час виконання операцій вибірки, сортування та фільтрації. Це особливо важливо у випадках, коли обсяг даних зростає, а кількість звернень до них збільшується — наприклад, у багатокористувацьких середовищах або при повторному виконанні однотипних запитів.

Крім цього, бази даних гарантують **збереження цілісності даних**, тобто відповідність між таблицями, відсутність дублювання та узгодженість усіх записів у межах транзакцій. Завдяки підтримці обмежень цілісності (наприклад, первинних і зовнішніх ключів), виключається можливість логічних суперечностей або помилок у структурі даних, що має критичне значення для достовірності результатів обробки та подальшого аналізу.

Ще однією важливою характеристикою є **масштабованість** — здатність бази даних ефективно працювати за умов зростання обсягів інформації та збільшення складності запитів. Навіть у випадку використання вбудованих рішень, таких як SQLite, система може бути масштабована як на рівні схеми

БД (через розширення структури), так і на рівні застосунку (наприклад, переходом до повноцінної клієнт-серверної СУБД при необхідності).

Як механізм зберігання даних обрано **SQLite** — легковагову вбудовану реляційну СУБД, яка є складовою стандартної бібліотеки Python. Такий вибір обумовлено її простотою у використанні, відсутністю потреби в окремому серверному середовищі та можливістю швидкої інтеграції в прототип або готовий вебзастосунок. SQLite забезпечує повну функціональність класичної СУБД при мінімальних витратах на розгортання, що є особливо актуальним у контексті середніх за обсягом застосунків, орієнтованих на обробку текстових даних та кешування запитів.

У структурі бази даних реалізовано кілька логічно незалежних, проте взаємопов’язаних таблиць:

Таблиця **playlist** містить узагальнену інформацію про музичні плейлисти. До її складу входять такі поля:

- **id** — унікальний ідентифікатор плейлиста;
- **url** — посилання на плейлист;
- **hash** — хеш-представлення посилання;
- **name** — назва плейлиста;
- **owner** — ім’я користувача або автора;
- **description** — текстовий опис;
- **cover** — URL зображення обкладинки;
- **tracks_count** — кількість треків у плейлисті;
- **overall_sentiment** — інтегральна емоційна оцінка;
- **created_at** — дата й час створення;
- **updated_at** — дата й час останнього оновлення.

Між таблицями **playlist** та **tracks** реалізовано зв’язок типу **"один до багатьох" (1:N)**: один плейлист може містити багато треків, тоді як кожен трек належить лише одному плейлисту.

Таблиця **tracks** призначена для збереження інформації про окремі музичні композиції. Вона включає:

- **id** — унікальний ідентифікатор треку;
- **playlist_id** — зовнішній ключ, що вказує на пов’язаний плейлист;
- **title** — назва треку;
- **artist** — ім’я виконавця;
- **cover** — зображення обкладинки композиції;
- **url** — посилання на джерело треку.

Таким чином, між таблицями **tracks** і **playlist** існує зв’язок типу **"багато до одного" (N:1)**.

Таблиця **search_cache** використовується для кешування результатів пошукових запитів, що дозволяє зменшити навантаження на API та покращити продуктивність. У ній зберігаються такі атрибути:

- **id** — унікальний ідентифікатор запиту;
- **query_hash** — хеш пошукового запиту;
- **query** — оригінальний текст запиту;
- **results** — результати пошуку у вигляді структурованого рядка;
- **created_at** — час створення запису.

Таблиця має **один до одного (1:1)** зв’язок із таблицею **tracks**, що забезпечує відповідність між запитом і знайденим треком. Крім того, **search_cache** пов’язана з **songs_cache** також у співвідношенні **1:1**, оскільки кожен запит може породжувати унікальний текст пісні, що підлягає подальшому аналізу.

Songs_cache виконує функцію збереження текстових і аналітичних даних про пісні. До її складу входять:

- **id** — унікальний внутрішній ідентифікатор запису;
- **song_id** — зовнішній ідентифікатор пісні (наприклад, з API);

- **title** — назва пісні;
- **artist** — виконавець;
- **lyrics** — повний текст пісні;
- **preprocessed_lyrics** — очищений текст від HTML-кодів;
- **cleared_lyrics** — текст без стоп-слів та зайвих символів;
- **sentiment_data** — результати емоційного аналізу у форматі JSON або структурованого рядка;
- **created_at** — дата створення запису;
- **updated_at** — дата останнього оновлення.

Ця таблиця перебуває у зв'язку **1:1 з таблицею search_cache**, оскільки кожен результат пошуку має свій текстовий та аналітичний опис. Окрім цього, **songs_cache надає інформацію для таблиці playlist**, оскільки результати аналізу пісень використовуються для обчислення загального емоційного показника плейлиста (**overall_sentiment**), хоча прямий зовнішній ключ не реалізовано.

З метою покращення сприйняття логіко-структурних взаємозв'язків між основними сутностями інформаційної системи, було створено діаграму класів (рис. 2.2), яка візуалізує архітектуру бази даних, розробленої для зберігання, обробки та аналітики музичних композицій. Діаграма побудована відповідно до стандартів нотації UML (Unified Modeling Language). Діаграма класів виконує низку важливих функцій: вона узагальнює структуру таблиць бази даних, представляючи кожну з них як окремий клас із відповідними атрибутами; візуалізує типи зв'язків між таблицями, зокрема асоціації типу «один до одного» (1:1), «один до багатьох» (1:N) та «багато до одного» (N:1); сприяє глибшому розумінню логіки взаємодії між ключовими сутностями системи, такими як треки, плейлисти, результати пошукових запитів і тексти пісень; а також слугує основою для побудови схем фізичної реалізації бази даних.

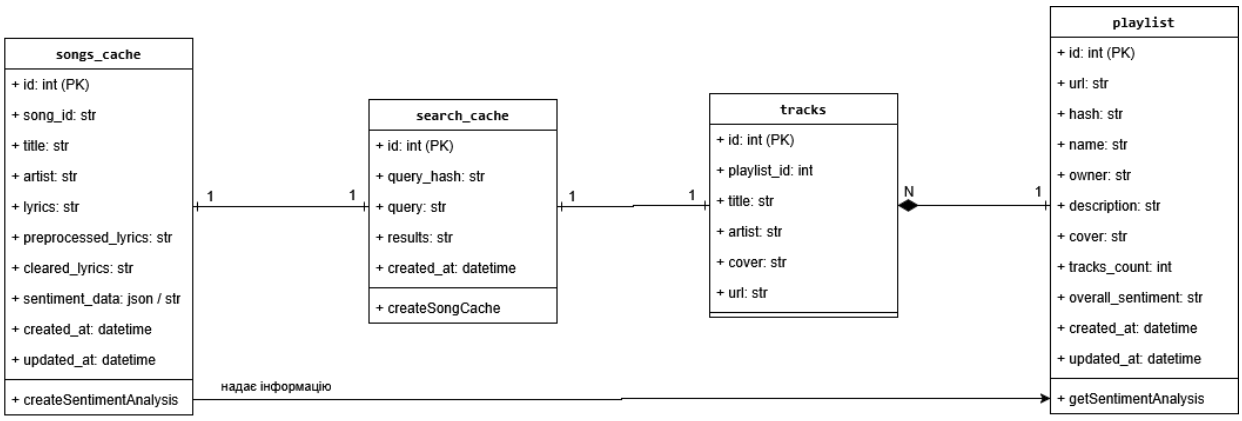

Рис. 2.2 – Діаграма класів проектованої БД

Для підвищення продуктивності при виконанні запитів до БД впроваджено **індексування** ключових полів — зокрема, унікальних ідентифікаторів пісень, хешів запитів і зовнішніх ключів. Індекси суттєво зменшують час виконання SELECT-операцій, особливо у випадках складної фільтрації або сортування великих обсягів даних.

2.3.4 Життєвий цикл інформаційної системи

Життєвий цикл інформаційної системи (ІС) — це структурована сукупність послідовних етапів, що охоплюють усі фази існування програмного продукту: від початкового виявлення потреб користувача до завершення експлуатації або виведення з обігу. Метою побудови життєвого циклу є забезпечення систематизованого підходу до розробки, впровадження та супроводу інформаційних рішень, що відповідають вимогам як бізнесу, так і кінцевих користувачів.

Існує низка класичних моделей життєвого циклу, кожна з яких має свої особливості реалізації. Серед найбільш поширених — каскадна (водоспадна), V-подібна, інкрементна, спіральна, а також гнучкі підходи (Agile, Scrum). Вибір тієї чи іншої моделі обумовлюється низкою чинників: складністю

проєкту, стабільністю вимог, очікуваними термінами, ресурсами та рівнем взаємодії з користувачами.

У рамках розробки даної ІС доцільним виявився **інкрементний (спіральний) життєвий цикл**, який поєднує переваги поетапної розробки та ітеративного удосконалення. Суть цього підходу полягає в тому, що розробка ведеться серіями (інкрементами), кожен з яких є завершеним функціональним блоком, що може бути протестований, інтегрований та використаний незалежно. Кожна ітерація дозволяє детальніше опрацювати вимоги, виявити можливі ризики, протестувати функціональність та отримати зворотний зв'язок, що є основою для наступного вдосконалення.

Спіральна модель, яка є узагальненням інкрементного підходу, доповнює цикл повторюваними фазами аналізу ризиків, планування та прототипування. Таким чином, розробка просувається по спіралі з поступовим наближенням до фінального продукту, що забезпечує гнучкість і мінімізацію критичних помилок на ранніх етапах.

Рис. 2.3 – Схема реалізації «спіральної» моделі розробки

Вибір інкрементно-спіральної моделі для реалізації інформаційної системи у сфері аналізу текстів пісень обґрунтовується низкою причин:

- **Змінність вимог**. У процесі реалізації функціоналу (аналіз текстів, генерація хмар слів, інтеграція з API) можуть виникати нові вимоги або

коригування вже визначених. Ітеративна модель дозволяє легко адаптувати систему до таких змін.

- **Можливість поетапного розгортання**. Оскільки система складається з взаємопов'язаних компонентів (інтерфейс, аналізатор, база даних), доцільним є поступове впровадження функціональності з можливістю часткового використання або тестування.
- **Зворотний зв'язок від користувача**. Отримання проміжних результатів дає змогу залучати кінцевих користувачів до процесу оцінювання, що дозволяє виявити недоліки або побажання ще до завершення всієї системи.
- **Ризик-орієнтоване планування**. Завдяки спіральній моделі, ключові ризики (нестабільність API, обробка великих обсягів тексту, якість аналізу) можуть бути виявлені та нейтралізовані ще до етапу масштабної реалізації.

Однією суттєвою перевагою інкрементно-спіральної моделі є її здатність підтримувати безперервне вдосконалення функціональності вебзастосунку шляхом поступового оновлення. Завдяки ітеративній природі розробки, кожне нове оновлення може включати розширення існуючих можливостей або впровадження нових функціональних модулів без необхідності повного перероблення всієї системи. Це дозволяє не лише оперативно реагувати на зміну потреб користувачів або зовнішніх умов (наприклад, оновлення API або технологічні тренди), але й забезпечити поступове розширення функціоналу, зберігаючи при цьому стабільність роботи вже реалізованих компонентів.

Користувачі отримують оновлення в інтерактивному режимі, що сприяє покращенню користувацького досвіду, підвищенню лояльності до системи та забезпечує відповідність вебзастосунку сучасним вимогам. Таким чином, обрана модель не лише підтримує стабільність і контроль над розвитком проєкту, але й забезпечує гнучкий механізм еволюційного росту

функціональності, що є надзвичайно важливим у контексті веборієнтованих рішень.

2.4 Висновки до розділу 2

Було спроектовано інформаційну систему для емоційного аналізу музичних текстів, що включає визначення функціональних і нефункціональних вимог, вибір архітектурного підходу та технологічного стека. Обрано клієнт-серверну модель на основі Python і Flask, яка забезпечує ефективну обробку текстів, взаємодію з API (Genius, Spotify), реалізацію NLP-аналізу та візуалізацію результатів.

Архітектура системи включає модулі для збору, очищення, емоційного аналізу та кешування текстів пісень. Використання бібліотек NLTK, VADER, TextBlob, а також методів TF-IDF і cosine similarity дозволяє точно оцінювати емоційне забарвлення композицій. Фронтенд реалізовано засобами HTML5, CSS3 та JavaScript з підтримкою SPA-підходу. В якості моделі життєвого циклу обрано інкрементно-спіральну.

2.5 Перелік джерел посилань до розділу 2

1. Contributors to Wikimedia projects. Frontend and backend - Wikipedia. Wikipedia, the free encyclopedia. URL: https://en.wikipedia.org/wiki/Frontend_and_backend (date of access: 17.06.2025).

2. Contributors to Wikimedia projects. JavaScript - wikipedia. Wikipedia, the free encyclopedia. URL: https://en.wikipedia.org/wiki/JavaScript (date of access: 17.06.2025).

3. Contributors to Wikimedia projects. Natural language processing - Wikipedia. Wikipedia, the free encyclopedia. URL:

https://en.wikipedia.org/wiki/Natural_language_processing (date of access: 17.06.2025).

4. Contributors to Wikimedia projects. Tf–idf - wikipedia. Wikipedia, the free encyclopedia. URL: https://en.wikipedia.org/wiki/Tf–idf (date of access: 17.06.2025).

5. Rajaraman A. Mining of massive datasets. Cambridge University Press.

6. Welcome to flask – flask documentation (3.1.x). Welcome to Flask – Flask Documentation (3.1.x). URL: https://flask.palletsprojects.com/en/stable/ (date of access: 17.06.2025).

7. SPA (single-page application) - glossary | MDN. MDN Web Docs. URL: https://developer.mozilla.org/en-US/docs/Glossary/SPA (date of access: 17.06.2025).

8. Contributors to Wikimedia projects. Iterative and incremental development - Wikipedia. Wikipedia, the free encyclopedia. URL: https://en.wikipedia.org/wiki/Iterative_and_incremental_development (date of access: 17.06.2025).

3 РЕАЛІЗАЦІЯ ІНФОРМАЦІЙНОЇ СИСТЕМИ

3.1 Вибір засобів розробки

Згідно з вимог до проєкту, основними критеріями вибору засобів стали: підтримка сучасних мов програмування та бібліотек, зручність інтеграції з зовнішніми сервісами (зокрема API), а також наявність інструментів для обробки й зберігання структурованих даних.

Для написання програмного коду було обрано середовище розробки Microsoft Visual Studio Code (VS Code) — сучасний багатоплатформовий редактор, який підтримує велику кількість мов програмування, розширень та інтеграцій. Серед його переваг — легкий інтерфейс, можливість налаштування під конкретні потреби розробника, а також інтеграція з системами контролю версій, середовищами виконання та вбудованими терміналами. У межах даного проєкту VS Code було використано для розробки як серверної (бекенд), так і клієнтської (фронтенд) частин системи.

Для роботи з базою даних було застосовано програму SQLiteStudio — інструмент з відкритим кодом для візуального керування базами даних на основі СКБД SQLite. Цей засіб забезпечує зручне створення та редагування таблиць, виконання SQL-запитів, перегляд і модифікацію даних, що є необхідним для налагодження збереження текстів пісень, результатів їхнього аналізу та іншої пов’язаної інформації.

З огляду на потребу у швидкому доступі до даних, простоті впровадження та відсутності необхідності в розгортанні серверної СКБД, база даних SQLite була обрана як основне рішення для збереження інформації. Її інтеграція з мовою програмування Python, що використовувалася для реалізації логіки обробки текстів, є повністю підтримуваною й не потребує встановлення додаткових серверних компонентів.

3.2 Реалізація вебзастосунку

3.2.1 Реалізація структури бази даних та кешування

Для забезпечення ефективної роботи системи аналізу музичних композицій було розроблено механізм кешування даних, який значно зменшує навантаження на зовнішні API та прискорює відповідь сервера. Структура бази даних була спроектована відповідно до діаграми класів системи.

Кешування пісень реалізовано через таблицю **songs_cache**, яка зберігає повну інформацію про пісні, включаючи текст, результати обробки та аналізу. Метод **get_song_from_cache** перевіряє наявність пісні в кеші за ідентифікатором або парою "назва-виконавець". Якщо запис знайдено, дані повертаються негайно, що дозволяє уникнути зайвих запитів до Genius API. Для збереження нових даних використовується метод **save_song_to_cache**, який або додає новий запис, або оновлює існуючий. Кожен запис містить мітки часу створення та оновлення, що забезпечує контроль актуальності даних.

Кешування пошукових запитів здійснюється через таблицю **search_cache**, яка оптимізує роботу з пошуком пісень. Перед виконанням запиту до API Genius система перевіряє кеш через метод **get_search_from_cache**, використовуючи MD5-хеш запиту (**query_hash**) для швидкого пошуку. Результати зберігаються на 24 години (перевіряється за допомогою **datetime(created_at) > datetime('now', '-1 day')**), після чого автоматично оновлюються. Метод **save_search_to_cache** забезпечує збереження нових результатів пошуку, підтримуючи актуальність даних.

Кешування плейлистів реалізовано через таблиці **playlists** та **tracks** для роботи з плейлистами Spotify. Метод **get_playlist_from_db** дозволяє отримувати повну інформацію про плейлист за його URL, уникаючи повторних запитів до Spotify API. Збереження даних здійснюється через **save_playlist_to_db**, який автоматично оновлює існуючі записи та пов'язані

треки. Важливою особливістю є збереження агрегованих результатів емоційного аналізу (**overall_sentiment**) для подальшого швидкого доступу.

3.2.2 Реалізація API-запитів до сервісів Genius і Spotify

Для роботи з Genius API система застосовує два основних кінцевих точки: ендпоінт /search для пошуку пісень за ключовими словами або настроями, та ендпоінт /songs/:id для отримання посилання на веб-сторінку з текстом пісні. Оскільки API не надає безпосереднього доступу до текстів, реалізовано спеціалізований механізм парсингу HTML-контенту з використанням бібліотеки BeautifulSoup. Даний механізм включає кілька альтернативних стратегій аналізу розмітки, що дозволяє обробляти як традиційну структуру сторінок (елементи div з класом **lyrics**), так і новіший формат представлення даних (контейнери з класами **Lyrics__Container**). Авторизаційні запити до API здійснюються з використанням токена доступу, який передається через стандартний заголовок Authorization у форматі Bearer Token. У разі виникнення помилок під час виконання запитів або відсутності тексту пісні на сторінці, система генерує зрозумілі повідомлення про помилку, що відповідають принципам юзабілітету.

Інтеграція з Spotify API реалізована через офіційну бібліотеку Spotipy, яка є стандартним Python-інструментом для взаємодії з даним сервісом. Автентифікація відбувається за схемою Client Credentials Flow, що передбачає використання пари **client_id** та **client_secret** для отримання токена доступу. Такий підхід дозволяє отримувати публічні дані без необхідності участі кінцевого користувача в процесі авторизації. Функціональність роботи з плейлистами включає аналіз URL-адрес для виділення унікальних ідентифікаторів, запит метаданих про плейлисти (назва, автор, опис, зображення обкладинки, кількість треків), а також поетапне отримання інформації про окремі композиції. Для ефективного управління великими

наборами даних реалізовано механізм посторінкового завантаження треків через функції sp.playlist() та sp.playlist_tracks(). Отримані дані трансформуються в уніфіковану структуру, яка містить усі необхідні атрибути для подальшого аналізу: назву треку, ім'я виконавця, посилання на композицію та зображення альбому.

3.2.3 Реалізація попередньої обробки та очищення текстів пісень

Перед застосуванням методів емоційного аналізу та візуалізації даних необхідно провести комплексну попередню обробку текстів пісень.

На етапі попередньої обробки тексту (функція **preprocess_lyrics**()) здійснюється низка трансформацій, спрямованих на виділення семантично навантаженого контенту. Першим кроком є ізоляція основного тексту пісні шляхом видалення всіх фрагментів, що передують першій структурній позначці у квадратних дужках (наприклад, [Verse 1] чи [Chorus]). Наступним кроком виконується видалення всіх службових позначок, що містяться у квадратних дужках (таких як [Intro], [Bridge] чи [Outro]), які не несуть значущої семантичної навантаження для аналізу. Завершальним етапом попередньої обробки є нормалізація текстової структури - видалення зайвих пробілів, порожніх рядків та інших нефункціональних елементів, що призводить до формування компактного та структурованого тексту, готового до подальшого аналізу.

Етап очищення тексту (функція **clean_text**()) має на меті підготувати текст до побудови хмари слів та інших форм візуалізації. Процес починається з видалення стоп-слів - загальновживаних мовних одиниць (таких як артиклі, прийменники чи окремі літери), які не мають значущої семантичної ваги. Подальша нормалізація передбачає приведення всіх символів до нижнього регістру, видалення пунктуації та спеціальних символів, а також усунення зайвих пробільних символів. Важливим етапом є перевірка наявності

смислових одиниць у тексті після очищення - у разі відсутності значущих слів система припиняє подальшу обробку та інформує про це користувача. Фінальним кроком є збереження очищеного тексту у форматі .txt у спеціальній директорії **intermediate_texts**, що дозволяє фіксувати проміжні результати обробки для можливого подальшого аналізу чи налагодження системи.

3.2.4 Реалізація аналізу текстів пісень на основі емоційної лексики

Аналіз емоційного забарвлення текстів пісень є центральною функціональністю розробленої інформаційної системи. Основною метою цього модуля є автоматичне визначення домінуючих емоцій у текстах пісень, що дає змогу класифікувати музичні композиції за настроєм та створювати відповідні візуальні представлення. Реалізація модуля базується на поєднанні лексиконного та статистичного підходів до емоційного аналізу.

Функція емоційного аналізу реалізована у вигляді окремого модуля з назвою **advanced_english_sentiment_analysis**(). Її робота починається з перевірки мови тексту — визначається, чи є текст переважно англомовним, і лише тоді допускається до подальшої обробки. Далі проводиться розрахунок базових емоційних метрик за допомогою VADER, після чого текст очищується від пунктуації та розбивається на окремі слова. Кожне з цих слів звіряється зі словником емоцій, і на основі кількості збігів формується попередній емоційний профіль. Отримані значення нормалізуються — для кожної емоційної категорії обчислюється її відносна частка. Це дозволяє визначити домінуючу емоцію, яка буде використана у візуалізації та для подальшої класифікації пісні. Також на основі кількості знайдених емоційних слів оцінюється рівень впевненості аналізу, що може бути низьким, середнім або високим.

Після завершення аналізу функція повертає структурований результат, який включає числові показники полярності та суб’єктивності, деталізований

емоційний профіль, домінуючу емоцію, щільність емоційних слів та загальну кількість слів у тексті. Усі ці дані зберігаються в базі даних у форматі JSON у полі відповідної таблиці **songs_cache**.

3.2.5 Реалізація пошуку пісень за ключовими словами та синонімами

Коли користувач надсилає запит до ендпоінта **/find-songs**, сервер запускає виконання функції **find_songs**(), яка відповідає за реалізацію логіки пошуку музичних композицій відповідно до заданої емоційної характеристики. Ключове слово, що відображає настрій, передається як параметр **mood** у форматі GET-запиту, після чого сервер починає послідовну обробку цього запиту.

У разі, якщо додатково в параметрі **synonyms** передано значення **true**, активується режим розширеного пошуку з урахуванням синонімічного ряду. Система викликає функцію **get_synonyms**(), яка формує перелік синонімів. Отримані лексеми додаються до списку пошукових термінів, який буде використано на наступному етапі.

Для кожного слова зі списку **search_terms** система виконує пошук відповідних пісень за допомогою функції **search_songs_by_keywords**(), яка взаємодіє з зовнішнім API. З отриманого результату обираються не більше десяти найбільш релевантних записів. По кожній знайденій пісні визначаються її назва, ім’я виконавця, обкладинка, URL, а також унікальний ідентифікатор.

Після цього здійснюється перевірка наявності тексту пісні в локальному кеші за допомогою функції **get_cached_song_data**(). Якщо текст відсутній у кеші, виконується його отримання через API, попередня обробка та емоційний аналіз. У разі успішного вилучення тексту пісні, він очищується й додається до списку **songs_lyrics** для подальшого аналізу схожості. Одночасно

формується структура **songs_info**, яка включає всю супровідну інформацію про кожну пісню.

Далі проводиться семантичне порівняння змісту пісень із запитом користувача. Для цього застосовується функція **find_similar_songs**(), яка реалізує метод TF-IDF у поєднанні з метрикою косинусної подібності. На цьому етапі кожному об’єкту **song_info** додається показник схожості, після чого результати накопичуються у загальному списку **all_songs**.

У випадку, якщо не знайдено жодної релевантної пісні, система повертає повідомлення про помилку з відповідним HTTP-статусом 404. Якщо ж результати наявні, виконується їх фільтрація з метою видалення дублікатів (ідентифікація здійснюється за парою назва–виконавець) та сортування за спаданням коефіцієнта схожості.

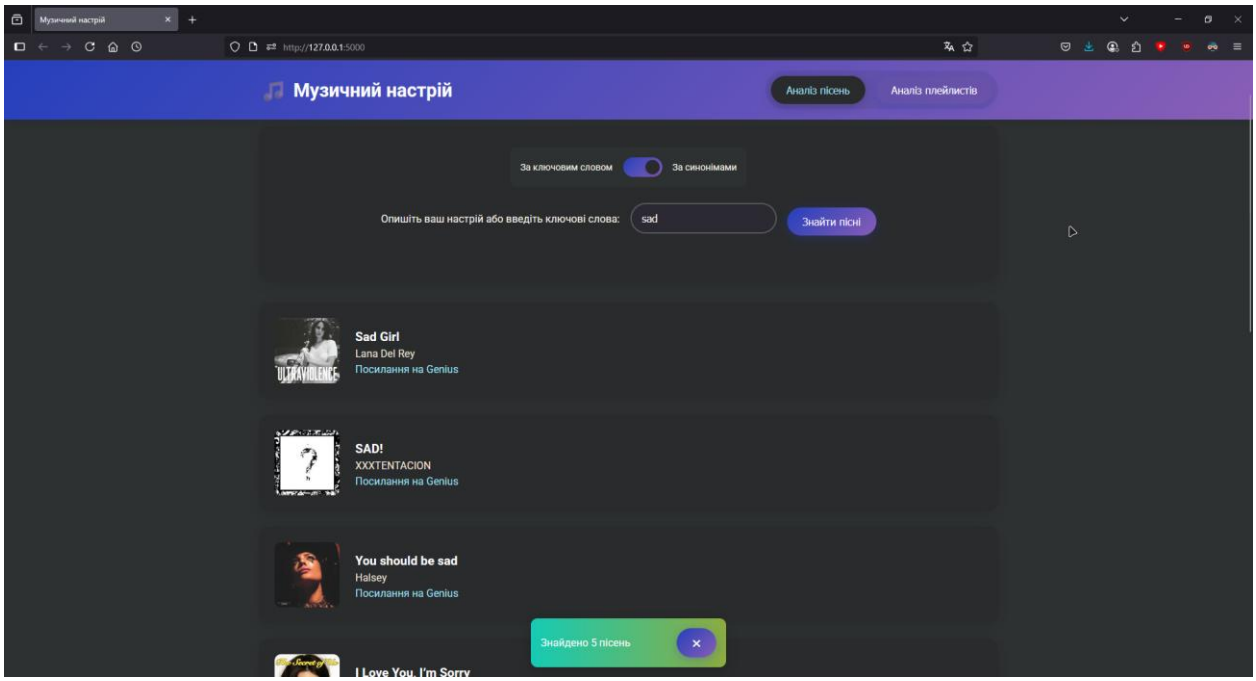

Рис. 3.1 – Результати пошуку за настроєм

3.2.6 Реалізація аналізу плейлистів

Обробка плейлистів реалізована за допомогою маршруту /analyze-playlist, який приймає HTTP-запити з методом **GET**. Користувач передає параметр url, що містить посилання на Spotify-плейлист. На початку функція перевіряє, чи передано **URL** у параметрах запиту за допомогою **request.args.get('url').** Якщо параметр відсутній, сервер повертає помилку зі статусом 400.

Далі система перевіряє, чи є відповідні дані в кеші, викликаючи функцію **get_playlist_from_db(playlist_url).** Якщо такий плейлист уже збережено в базі даних, його дані повертаються у форматі JSON. У протилежному випадку система переходить до обробки URL.

Щоб витягнути **playlist_id**, із посилання користувача послідовно перевіряються кілька регулярних виразів (через **re.search**) для підтримки різних форматів URL. Якщо жоден із шаблонів не спрацьовує, сервер повертає повідомлення про неправильний формат посилання.

Після отримання коректного **playlist_id** викликається метод **sp.playlist(playlist_id)** для завантаження основної інформації про плейлист із Spotify API. Отримані поля **name, owner['display_name'], description** та **images** зберігаються у словнику **playlist_data**.

Збір треків відбувається за допомогою **sp.playlist_tracks(playlist_id).** Система виконує ітеративну обробку результатів із підтримкою пагінації (через **sp.next(results)**), щоб отримати всі треки незалежно від їх кількості. Для кожного треку зберігаються такі поля: **track['name'],** імена виконавців (через об'єднання **artist['name']),** обкладинка альбому та пряме посилання на Spotify.

Після цього викликається функція **save_playlist_to_db(playlist_data, tracks_data, playlist_url),** яка зберігає плейлист і треки до бази даних для майбутніх звернень без повторного аналізу.

На завершення формується JSON-відповідь **response_data**, яка включає обкладинку, назву, автора, кількість треків, опис, URL плейлиста та детальний список треків. Цей об'єкт повертається користувачу у відповідь на запит.

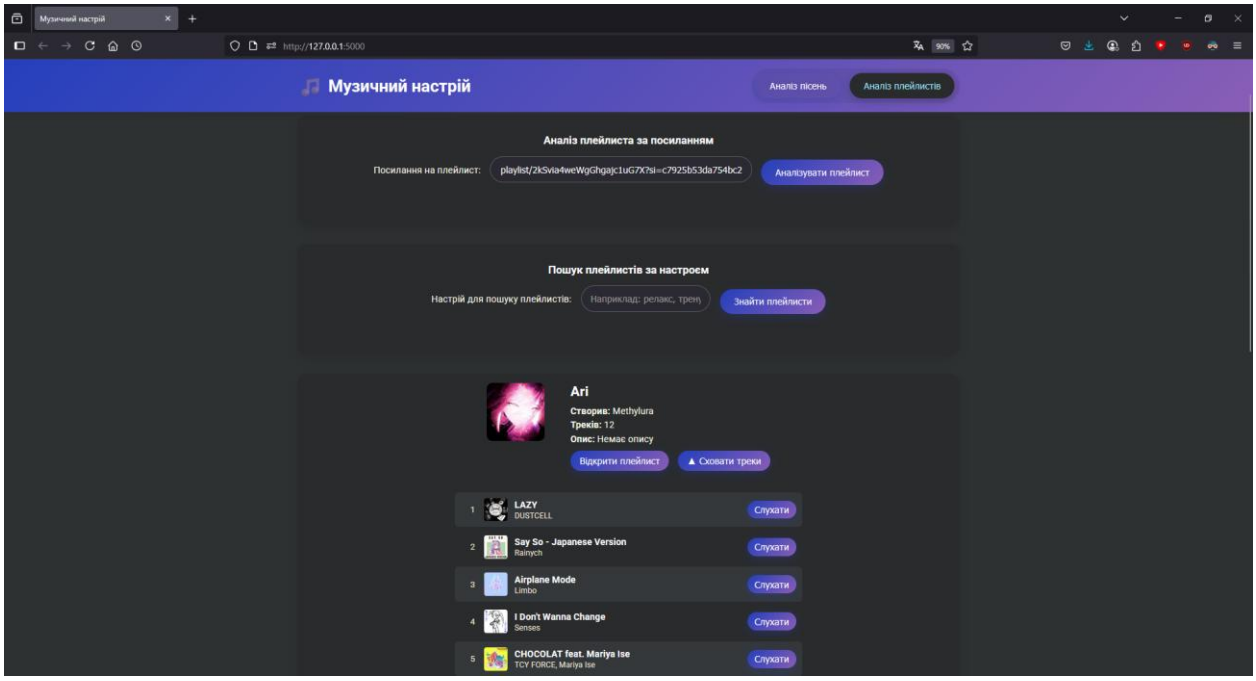

Рис. 3.2 – Відображення вмісту плейлисту на клієнті

3.2.7 Реалізація генерації хмари слів на основі тексту пісні

Основою для побудови хмари слів слугує очищений текст пісні, який попередньо проходить кілька обов’язкових етапів обробки: видалення службової інформації, фільтрація небажаних символів, нормалізація лексем, а також усунення стоп-слів (див. підрозділ 3.2.3). Після цього текст розбивається на окремі слова, і для кожного з них визначається частота вживання за допомогою класу **Counter** із бібліотеки **collections**.

На основі обчисленої частотності для кожного слова створюється словникова структура, яка містить два ключі: **text** — саме слово, та **size** — числовий показник, що відповідає кількості його вживань. У результаті формується список словників, який передається на клієнт у форматі JSON. Ці

дані далі використовуються у JavaScript-бібліотеці **WordCloud2.js** для побудови інтерактивної хмари слів безпосередньо у браузері користувача.

На серверній стороні реалізовано низку спеціалізованих ендпоінтів Flask, кожен з яких обслуговує різні сценарії генерації хмари слів. Ендпоінт **/generate-wordcloud** використовується для візуалізації окремої пісні. Користувач надсилає назву пісні та ім'я виконавця, після чого система перевіряє наявність очищеного тексту в кеші. Якщо дані знайдено — вони використовуються безпосередньо, інакше запускається процес пошуку та аналізу пісні через зовнішнє API.

Другий ендпоінт — **/generate-wordcloud-from-songs** — призначений для роботи зі списком пісень. Через POST-запит у форматі JSON передаються кілька пісень, очищені тексти яких об'єднуються в єдину текстову послідовність. Далі виконується агрегований підрахунок частотності слів, що дозволяє сформувати узагальнену картину лексичного наповнення вибраної добірки.

Окремий ендпоінт **/generate-playlist-wordcloud** реалізує аналогічну функціональність, але орієнтований на роботу з повноцінними Spotify-плейлистами. Інформація про композиції спочатку отримується з кешу або зовнішнього API, після чого формується спільна хмара слів на основі всіх треків у списку.

Загальна послідовність дій при генерації хмари слів включає кілька ключових етапів: завантаження очищених текстів, їх об'єднання в один рядок, токенізацію (розбиття на слова), обчислення частотності кожного слова та формування відповідної JSON-структури для подальшої візуалізації.

На клієнтській стороні реалізовано інтерактивне відображення хмари слів з підтримкою масштабування, динамічного налаштування кольорової гами та шрифтів. Слова, що мають вищу частотність, автоматично відображаються більшим шрифтом, що дозволяє швидко ідентифікувати ключові теми або мотиви пісні чи добірки композицій.

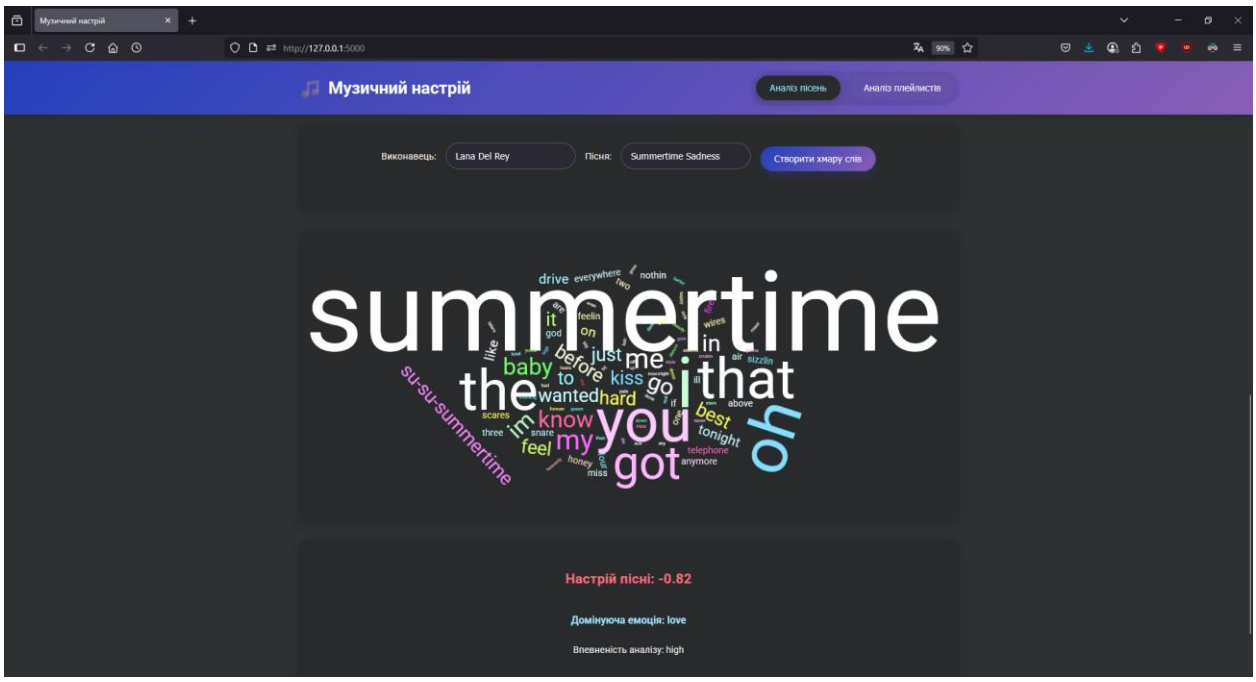

Рис. 3.3 – Відображення хмари слів та аналізу пісні

3.2.8 Реалізація пошуку плейлистів за емоціями

Запуск пошуку плейлистів за емоційним критерієм здійснюється через маршрут /find-playlists, який приймає HTTP-запити методом GET. У запиті користувач передає параметр mood, що визначає бажану емоцію або настрій, наприклад: "joy", "sadness", "relaxation". Після отримання запиту, система перевіряє наявність параметра mood через **request.args.get('mood').** Якщо параметр відсутній, сервер негайно повертає відповідь з кодом 400 і повідомленням про помилку.

У разі коректного запиту, система підключається до локальної бази даних SQLite, у якій зберігаються попередньо проаналізовані плейлисти. За допомогою SQL-запиту **SELECT * FROM playlists WHERE overall_sentiment IS NOT NULL** завантажуються всі плейлисти, у яких збережено результати емоційного аналізу.

Для кожного знайденого плейлиста дані **overall_sentiment** парсяться як JSON-об'єкт. Із нього витягується емоційний профіль (**emotion_profile**) —

розподіл інтенсивності емоцій, а також рівень упевненості (**confidence**) у результаті аналізу. Якщо дані некоректні або відсутні, система пропускає обробку відповідного плейлиста.

Далі виконується порівняння запитаного настрою з доступними емоціями в емоційному профілі. Якщо знаходиться точна або часткова відповідність (наприклад, "happy" для "happiness"), обчислюється бал відповідності (**mood_score**). Отриманий бал коригується коефіцієнтом на основі рівня впевненості: високий (high) збільшує бал на 50%, середній (medium) — на 20%, низький (low) не впливає на результат.

Після обробки всіх плейлистів, отримані результати сортуються за спаданням фінального балу відповідності. Обираються п’ять найрелевантніших плейлистів, формується JSON-відповідь, яка містить основну інформацію про ці плейлисти.

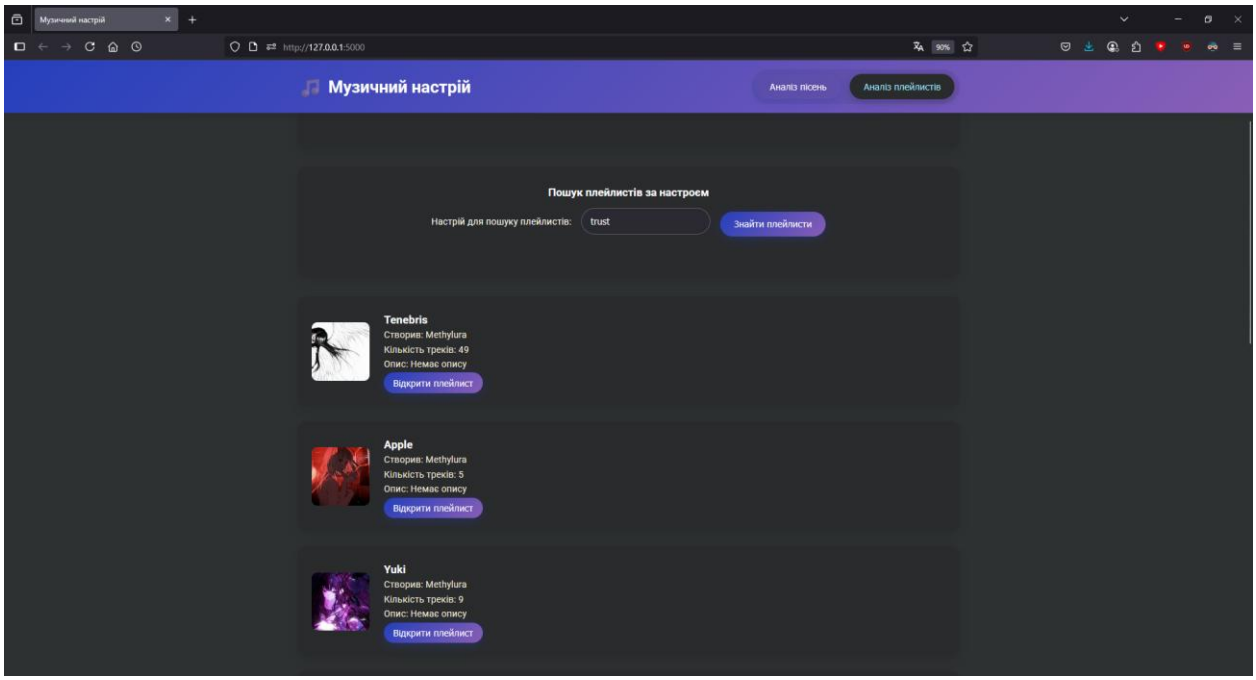

Рис. 3.4 – Результати пошуку за настроєм

3.3 Висновки до розділу 3

У даному розділі було детально описано процес реалізації інформаційної системи для аналізу текстів пісень, визначення їхньої емоційної спрямованості та візуалізації у вигляді хмар слів. Реалізація базувалася на використанні сучасних інструментів та технологій, таких як Python, Flask, SQLite, бібліотеки NLTK, TextBlob, Spotipy, а також інтеграції з зовнішніми API сервісів Genius і Spotify.

Було створено ефективну структуру бази даних, що забезпечує кешування пошукових запитів, текстів пісень і результатів емоційного аналізу, що значно знижує навантаження на зовнішні сервіси та підвищує швидкодію системи. Реалізовані модулі пошуку пісень за ключовими словами та синонімами дозволяють формувати гнучкі та релевантні добірки композицій відповідно до настрою користувача.

Особливу увагу приділено етапам попередньої обробки текстів пісень, очищенню від шумових елементів та подальшому лексиконному й статистичному аналізу емоційного забарвлення. Застосування поєднання алгоритмів VADER та TextBlob забезпечує глибоку оцінку настрою текстів, а агрегування результатів дозволяє будувати узагальнені емоційні профілі для цілих плейлистів.

У результаті виконаної роботи було повністю реалізовано функціональні можливості, передбачені технічним завданням, та закладено основу для подальшого розвитку системи — зокрема, розширення мовної підтримки, впровадження машинного навчання та вдосконалення рекомендаційного механізму на основі емоційного профілю користувача.

3.4 Перелік джерел посилань до розділу 3

1. ECMAScript® 2018 language specification. ECMAScript® 2024 Language Specification. URL: https://262.ecma-international.org/9.0/index.html (date of access: 17.06.2025).

2. JavaScript Guide - JavaScript | MDN. MDN Web Docs. URL: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide (date of access: 17.06.2025).

3. SQLite documentation. SQLite Home Page. URL: https://sqlite.org/docs.html (date of access: 17.06.2025).

4. The Python Language Reference – Python 3.10.17 documentation. 3.13.5 Documentation. URL: https://docs.python.org/3.10/reference/index.html (date of access: 17.06.2025).

5. Web API | spotify for developers. Home | Spotify for Developers. URL: https://developer.spotify.com/documentation/web-api (date of access: 17.06.2025).

ВИСНОВКИ

У кваліфікаційній роботі розроблено вебзастосунок для аналізу емоційного забарвлення текстів пісень із використанням методів обробки природної мови (NLP) та інтеграції з API Spotify і Genius. Сформовано структурно-функціональну модель системи, яка охоплює процеси збору текстових даних, їх попередньої обробки, емоційного аналізу та візуалізації результатів у формі хмари слів. Обґрунтовано вибір клієнт-серверної архітектури та розроблено програмну реалізацію з урахуванням вимог масштабованості, продуктивності й адаптивності інтерфейсу.

У межах роботи було реалізовано ключові програмні модулі:

- модуль автоматичного отримання текстів пісень через API-сервіси (Genius, Spotify) з підтримкою механізмів кешування;
- NLP-модуль для токенізації, очищення тексту та фільтрації лексики з урахуванням специфіки пісенних текстів;
- модуль емоційного аналізу на основі комбінації словникових і алгоритмічних методів (VADER, TextBlob, NRC Lexicon);
- функціонал візуалізації у вигляді інтерактивної хмари слів із можливістю пошуку плейлистів інших користувачів.

Система протестована на прикладах текстів пісень, забезпечує високу швидкість обробки запитів. Структура бази даних і застосовані алгоритми дозволяють надалі розширювати функціональність системи — зокрема реалізувати підтримку мультимовного аналізу, генерацію статистичних звітів або інтеграцію з іншими стримінговими платформами.

Розроблений застосунок наочно демонструє ефективність автоматизованого емоційного аналізу музичних текстів, дозволяє знизити часові витрати на дослідження великих корпусів лірики та створити інструмент для персоналізованої взаємодії з музичним контентом.

ДОДАТОК А
СПИСОК ТЕРМІНІВ, СКОРОЧЕНЬ ТА ПОЗНАЧЕНЬ

VADER	Valence Aware Dictionary and sEntiment Reasoner
NLP	Natural language processing
API	Application Programming Interface
JSON	JavaScript Object Notation
URL	Uniform Resource Locator
Плейлист	Упорядкований за тими чи іншими правилами, список аудіо, або відео файлів
Парсинг	Процес автоматизованого витягування та структурування даних з документів або інших джерел інформації

ДОДАТОК Б

```
1 from flask import Flask, request, jsonify, render_template
2 from flask_cors import CORS
3 import requests
4 from collections import Counter
5 import re
6 import os
7 from sklearn.feature_extraction.text import TfidfVectorizer
8 from sklearn.metrics.pairwise import cosine_similarity
9 from urllib.parse import quote
10 from bs4 import BeautifulSoup
11 import nltk
12 from nltk.corpus import wordnet
13 from nltk.sentiment.vader import SentimentIntensityAnalyzer
14 import spotipy
15 from spotipy.oauth2 import SpotifyClientCredentials
16 import json
17 from datetime import datetime
18 import string
19 from nltk.sentiment.vader import SentimentIntensityAnalyzer
20 from textblob import TextBlob
21 from collections import defaultdict
22 import json
23 from pathlib import Path
24 import sqlite3
25 import hashlib
26 import logging
27
28 app = Flask(__name__)
29
30 # Налаштування логування
31 logging.basicConfig(level=logging.INFO)
32 logger = logging.getLogger(__name__)
33
34 # Ініціалізація NLTK (для синонімів та VADER)
35 nltk.download('wordnet')
36 nltk.download('omw-1.4')
37 nltk.download('vader_lexicon')
38
39   GENIUS_API_TOKEN   =   "yIO204RXPGieVOdXfljLwT21Y2e-HfVvSvbJplZwg4strrPVxOW_heJ-
Vgs9WIC6"
40           SPOTIFY_CLIENT_ID           =           os.getenv('SPOTIFY_CLIENT_ID',
'de2f550be0bc4631be28d6992f9177da')
```

```
41         SPOTIFY_CLIENT_SECRET         =         os.getenv('SPOTIFY_CLIENT_SECRET',
'5cb1fddafa664b9aba3dd6e4d07dceb6')
42
43 # Ініціалізація Spotify клієнта
44 client_credentials_manager = SpotifyClientCredentials(
45     client_id=SPOTIFY_CLIENT_ID,
46     client_secret=SPOTIFY_CLIENT_SECRET
47 )
48 sp = spotipy.Spotify(client_credentials_manager=client_credentials_manager)
49
50 # Ініціалізація аналізатора настроїв
51 sia = SentimentIntensityAnalyzer()
52
53 # Ініціалізація бази даних
54 def init_database():
55     """Ініціалізація бази даних для кешування"""
56     conn = sqlite3.connect('songs_cache.db')
57     cursor = conn.cursor()
58
59     # Таблиця для кешування пісень
60     cursor.execute('''
61         CREATE TABLE IF NOT EXISTS songs_cache (
62             id INTEGER PRIMARY KEY AUTOINCREMENT,
63             song_id TEXT UNIQUE,
64             title TEXT,
65             artist TEXT,
66             lyrics TEXT,
67             preprocessed_lyrics TEXT,
68             cleaned_lyrics TEXT,
69             sentiment_data TEXT,
70             created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
71             updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
72         )
73     ''')
74
75     # Таблиця для кешування пошукових запитів
76     cursor.execute('''
77         CREATE TABLE IF NOT EXISTS search_cache (
78             id INTEGER PRIMARY KEY AUTOINCREMENT,
79             query_hash TEXT UNIQUE,
80             query TEXT,
81             results TEXT,
82             created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
83         )
84     ''')
85
```

```
86      # Таблиця для даних плейлистів
87      cursor.execute('''
88          CREATE TABLE IF NOT EXISTS playlists (
89              id INTEGER PRIMARY KEY AUTOINCREMENT,
90              url TEXT UNIQUE,
91              hash TEXT,
92              name TEXT,
93              owner TEXT,
94              description TEXT,
95              cover TEXT,
96              tracks_count INTEGER,
97              overall_sentiment TEXT,
98              created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
99          )
100     ''')
101
102     # Таблиця для треків з Spotify
103     cursor.execute('''
104         CREATE TABLE IF NOT EXISTS tracks (
105             id INTEGER PRIMARY KEY AUTOINCREMENT,
106             playlist_id INTEGER,
107             title TEXT,
108             artist TEXT,
109             cover TEXT,
110             url TEXT,
111             FOREIGN KEY (playlist_id) REFERENCES playlists (id)
112         )
113     ''')
114
115     # Індекси для швидкого пошуку
116              cursor.execute('CREATE   INDEX   IF   NOT   EXISTS   idx_song_id   ON
songs_cache(song_id)')
117           cursor.execute('CREATE   INDEX   IF   NOT   EXISTS   idx_title_artist   ON
songs_cache(title, artist)')
118             cursor.execute('CREATE   INDEX   IF   NOT   EXISTS   idx_query_hash   ON
search_cache(query_hash)')
119
120     conn.commit()
121     conn.close()
122     logger.info("Базу даних ініціалізовано")
123
124 def get_query_hash(query):
125     """Створює хеш для пошукового запиту"""
126     return hashlib.md5(query.lower().encode()).hexdigest()
127
128 def create_url_hash(url):
```

```
129     """Створює хеш для URL плейлиста"""
130     return hashlib.md5(url.encode()).hexdigest()
131
132 def get_song_from_cache(song_id=None, title=None, artist=None):
133     """Отримання пісні з кешу"""
134     conn = sqlite3.connect('songs_cache.db')
135     cursor = conn.cursor()
136
137     if song_id:
138         cursor.execute('SELECT * FROM songs_cache WHERE song_id = ?', (song_id,))
139     elif title and artist:
140          cursor.execute('SELECT * FROM songs_cache WHERE title = ? AND artist =
?', (title, artist))
141     else:
142         conn.close()
143         return None
144
145     result = cursor.fetchone()
146     conn.close()
147
148     if result:
149         logger.info(f"Пісню знайдено в кеші: {result[2]} - {result[3]}")
150         return {
151             'id': result[0],
152             'song_id': result[1],
153             'title': result[2],
154             'artist': result[3],
155             'lyrics': result[4],
156             'preprocessed_lyrics': result[5],
157             'cleaned_lyrics': result[6],
158             'sentiment_data': json.loads(result[7]) if result[7] else None,
159             'created_at': result[8],
160             'updated_at': result[9]
161         }
162     return None
163
164  def  save_song_to_cache(song_id,  title,  artist,  lyrics,  preprocessed_lyrics,
cleaned_lyrics, sentiment_data):
165     """Збереження пісні в кеш"""
166     conn = sqlite3.connect('songs_cache.db')
167     cursor = conn.cursor()
168
169     try:
170         cursor.execute('''
171             INSERT OR REPLACE INTO songs_cache
```

```
172             (song_id, title, artist, lyrics, preprocessed_lyrics, cleaned_lyrics,
sentiment_data, updated_at)
173             VALUES (?, ?, ?, ?, ?, ?, ?, CURRENT_TIMESTAMP)
174         ''', (song_id, title, artist, lyrics, preprocessed_lyrics, cleaned_lyrics,
json.dumps(sentiment_data)))
175
176         conn.commit()
177         logger.info(f"Пісню збережено в кеш: {title} - {artist}")
178     except Exception as e:
179         logger.error(f"Помилка збереження в кеш: {e}")
180     finally:
181         conn.close()
182
183 def get_search_from_cache(query):
184     """Отримання результатів пошуку з кешу"""
185     query_hash = get_query_hash(query)
186     conn = sqlite3.connect('songs_cache.db')
187     cursor = conn.cursor()
188
189     # Перевіряємо, чи не застарів кеш (наприклад, старше 24 годин)
190     cursor.execute('''
191         SELECT results FROM search_cache
192         WHERE query_hash = ? AND datetime(created_at) > datetime('now', '-1 day')
193     ''', (query_hash,))
194
195     result = cursor.fetchone()
196     conn.close()
197
198     if result:
199         logger.info(f"Результати пошуку знайдено в кеші для: {query}")
200         return json.loads(result[0])
201     return None
202
203 def save_search_to_cache(query, results):
204     """Збереження результатів пошуку в кеш"""
205     query_hash = get_query_hash(query)
206     conn = sqlite3.connect('songs_cache.db')
207     cursor = conn.cursor()
208
209     try:
210         cursor.execute('''
211             INSERT OR REPLACE INTO search_cache (query_hash, query, results)
212             VALUES (?, ?, ?)
213         ''', (query_hash, query, json.dumps(results)))
214
215         conn.commit()
```

```
216         logger.info(f"Результати пошуку збережено в кеш для: {query}")
217     except Exception as e:
218         logger.error(f"Помилка збереження пошуку в кеш: {e}")
219     finally:
220         conn.close()
221
222 def save_playlist_to_db(playlist_data, tracks_data, url):
223     """Зберігає плейлист і треки в базу даних"""
224     conn = sqlite3.connect('songs_cache.db')
225     cursor = conn.cursor()
226
227     playlist_hash = create_url_hash(url)
228
229     try:
230         # Перевіряємо, чи є вже такий плейлист
231        cursor.execute('SELECT id FROM playlists WHERE hash = ?', (playlist_hash,))
232         existing = cursor.fetchone()
233
234         if existing:
235             playlist_id = existing[0]
236             # Видаляємо старі треки
237                     cursor.execute('DELETE  FROM  tracks  WHERE  playlist_id  =  ?',
(playlist_id,))
238         else:
239             # Додаємо новий плейлист
240             cursor.execute('''
241                INSERT INTO playlists (url, hash, name, owner, description, cover,
tracks_count)
242                 VALUES (?, ?, ?, ?, ?, ?, ?)
243             ''', (
244                 url,
245                 playlist_hash,
246                 playlist_data['name'],
247                 playlist_data['owner'],
248                 playlist_data['description'],
249                 playlist_data['cover'],
250                 len(tracks_data))
251             )
252             playlist_id = cursor.lastrowid
253
254         # Додаємо треки
255         for track in tracks_data:
256             cursor.execute('''
257                 INSERT INTO tracks (playlist_id, title, artist, cover, url)
258                 VALUES (?, ?, ?, ?, ?)
259             ''', (
```

```
                playlist_id,
                track['title'],
                track['artist'],
                track['cover'],
                track['url']
            ))

        conn.commit()
        return playlist_id

    except Exception as e:
        conn.rollback()
        raise e
    finally:
        conn.close()

def get_playlist_from_db(url):
    """Отримує плейлист з бази даних за URL"""
    conn = sqlite3.connect('songs_cache.db')
    cursor = conn.cursor()

    playlist_hash = create_url_hash(url)

    try:
        # Отримуємо плейлист
        cursor.execute('''
            SELECT name, owner, description, cover, tracks_count
            FROM playlists WHERE hash = ?
        ''', (playlist_hash,))

        playlist = cursor.fetchone()
        if not playlist:
            return None

        # Отримуємо треки
        cursor.execute('''
            SELECT p.id FROM playlists p WHERE p.hash = ?
        ''', (playlist_hash,))
        playlist_id = cursor.fetchone()[0]

        cursor.execute('''
            SELECT title, artist, cover, url
            FROM tracks WHERE playlist_id = ?
        ''', (playlist_id,))

        tracks = cursor.fetchall()
```

```
306
307         return {
308             'name': playlist[0],
309             'owner': playlist[1],
310             'description': playlist[2],
311             'cover': playlist[3],
312             'tracks_count': playlist[4],
313             'tracks': [
314                 {
315                     'title': track[0],
316                     'artist': track[1],
317                     'cover': track[2],
318                     'url': track[3]
319                 }
320                 for track in tracks
321             ]
322         }
323
324     finally:
325         conn.close()
326
327 def get_cached_song_data(song_id, title, artist):
328     """Отримання повних даних пісні з кешу або API"""
329     # Спочатку перевіряємо кеш
330    cached_song = get_song_from_cache(song_id=song_id, title=title, artist=artist)
331
332     if cached_song:
333         return {
334             'lyrics': cached_song['lyrics'],
335             'preprocessed_lyrics': cached_song['preprocessed_lyrics'],
336             'cleaned_lyrics': cached_song['cleaned_lyrics'],
337             'sentiment': cached_song['sentiment_data']
338         }
339
340     # Якщо в кеші немає, отримуємо через API
341      logger.info(f"Пісню не знайдено в кеші, завантажуємо через API: {title} -
{artist}")
342
343     lyrics = get_song_lyrics(song_id)
344     if lyrics == "Текст пісні недоступний.":
345         return None
346
347     # Обробляємо текст
348     preprocessed_lyrics = preprocess_lyrics(lyrics)
349     cleaned_lyrics, _ = clean_text(preprocessed_lyrics)
350     sentiment = advanced_english_sentiment_analysis(preprocessed_lyrics)
```

```
351
352     # Зберігаємо в кеш
353         save_song_to_cache(song_id, title, artist, lyrics, preprocessed_lyrics,
cleaned_lyrics, sentiment)
354
355     return {
356         'lyrics': lyrics,
357         'preprocessed_lyrics': preprocessed_lyrics,
358         'cleaned_lyrics': cleaned_lyrics,
359         'sentiment': sentiment
360     }
361
362 # Завантаження словників з JSON
363 def load_emotion_lexicons():
364     try:
365          with open(Path(__file__).parent / 'lexicons' / 'emotion_lexicons.json',
'r', encoding='utf-8') as f:
366             return json.load(f)
367     except FileNotFoundError:
368         logger.warning("Файл emotion_lexicons.json не знайдено, використовуються
базові емоції")
369         return {
370             "joy": ["happy", "glad", "joyful", "cheerful", "delighted"],
371            "sadness": ["sad", "unhappy", "depressed", "melancholy", "sorrowful"],
372             "anger": ["angry", "mad", "furious", "rage", "irritated"],
373             "fear": ["afraid", "scared", "terrified", "anxious", "worried"],
374             "love": ["love", "adore", "cherish", "affection", "romance"],
375          "surprise": ["surprised", "amazed", "astonished", "shocked", "startled"]
376         }
377
378 def is_english(text):
379     """Перевіряє, що текст містить переважно англійські символи"""
380     english_chars = sum(1 for c in text if ord(c) < 128)
381     return english_chars / len(text) > 0.9 if text else False
382
383 def advanced_english_sentiment_analysis(text):
384     """
385     Просунутий аналіз настрою ТІЛЬКИ для англійських текстів
386     Повертає деталізовані метрики та емоційний профіль
387     """
388     if not is_english(text):
389         return {
390             "error": "Text is not English",
391             "is_english": False
392         }
393
```

```
394     # Ініціалізація аналізаторів
395     sia = SentimentIntensityAnalyzer()
396
397     #Розширений словник емоцій
398     emotion_lexicons = load_emotion_lexicons()
399
400     #Базовий аналіз через VADER
401     vader_scores = sia.polarity_scores(text)
402
403     # Підготовка тексту
404     translator = str.maketrans('', '', string.punctuation)
405     words = text.lower().translate(translator).split()
406
407     # Емоційний аналіз
408     emotion_counts = defaultdict(int)
409     total_emotional_words = 0
410
411     for word in words:
412         for emotion, lexicon in emotion_lexicons.items():
413             if word in lexicon:
414                 emotion_counts[emotion] += 1
415                 total_emotional_words += 1
416
417     # Нормалізація та розрахунок відсотків
418     emotion_profile = {}
419     if total_emotional_words > 0:
420         for emotion in emotion_lexicons:
421               emotion_profile[emotion] = round(emotion_counts.get(emotion, 0) /
total_emotional_words, 3)
422     else:
423         emotion_profile = {emotion: 0.0 for emotion in emotion_lexicons}
424
425     # Визначення домінуючої емоції
426       dominant_emotion = max(emotion_profile.items(), key=lambda x: x[1])[0] if
total_emotional_words > 0 else "neutral"
427
428     # Аналіз через TextBlob
429     blob = TextBlob(text)
430     subjectivity = round(blob.sentiment.subjectivity, 3)
431     polarity = round(blob.sentiment.polarity, 3)
432
433     # Деталізовані метрики
434     metrics = {
435         "word_count": len(words),
436         "emotional_words": total_emotional_words,
```

```
437             "emotional_density": round(total_emotional_words / len(words), 3) if
len(words) > 0 else 0,
438         "positive_words": emotion_counts.get("joy", 0) + emotion_counts.get("love",
0) +  emotion_counts.get("trust", 0),
439                           "negative_words":   emotion_counts.get("sadness",   0)   +
emotion_counts.get("anger",     0)     +     emotion_counts.get("fear",     0)     +
emotion_counts.get("disgust", 0),
440         "surprise_words": emotion_counts.get("surprise", 0)
441     }
442
443     # Узгодження результатів
444      if vader_scores['compound'] > 0.1 and dominant_emotion in ["anger", "fear",
"sadness"]:
445         dominant_emotion = "joy"  # Корекція суперечливих результатів
446
447     return {
448         "is_english": True,
449         "vader_scores": {
450             "compound": round(vader_scores['compound'], 3),
451             "positive": round(vader_scores['pos'], 3),
452             "neutral": round(vader_scores['neu'], 3),
453             "negative": round(vader_scores['neg'], 3)
454         },
455         "textblob_scores": {
456             "polarity": polarity,
457             "subjectivity": subjectivity
458         },
459         "emotion_profile": emotion_profile,
460         "dominant_emotion": dominant_emotion,
461         "advanced_metrics": metrics,
462         "emotional_words_ratio": f"{total_emotional_words}/{len(words)}",
463            "confidence": "high" if total_emotional_words >= 5 else "medium" if
total_emotional_words >= 2 else "low"
464     }
465
466 def save_intermediate_text(text, stage_name, output_dir="intermediate_texts"):
467     """Зберігає проміжний текст на різних етапах обробки"""
468     if not os.path.exists(output_dir):
469         os.makedirs(output_dir)
470
471     timestamp = datetime.now().strftime("%Y%m%d_%H%M%S")
472     filename = f"{stage_name}_{timestamp}.txt"
473     filepath = os.path.join(output_dir, filename)
474
475     with open(filepath, 'w', encoding='utf-8') as f:
476         f.write(text)
```

```
477
478     return filepath
479
480 def preprocess_lyrics(text):
481     """
482     Видаляє все до першої квадратної дужки [ (включаючи блоки перекладів),
483     потім очищує текст від технічних помiток.
484     """
485     # 1. Знаходимо позицію першої квадратної дужки
486     first_bracket_pos = text.find('[')
487
488     if first_bracket_pos == -1:
489         lyrics_part = text  # Якщо дужок немає, беремо весь текст
490     else:
491         lyrics_part = text[first_bracket_pos:]
492
493     # 3. Видаляємо всі технічні помітки в квадратних дужках
494     cleaned_text = re.sub(r'\[.*?\]', '', lyrics_part)
495
496     # 4. Видаляємо порожні рядки та зайві пробіли
497     lines = [line.strip() for line in cleaned_text.split('\n') if line.strip()]
498
499     return '\n'.join(lines)
500
501 def get_synonyms(word):
502     """Отримує 2 синоніми для англійського слова"""
503     logger.info(f"Пошук синонімів для англійського слова: '{word}'")
504
505     synonyms = set()
506     synsets = wordnet.synsets(word)
507     logger.info(f"WordNet знайшов {len(synsets)} synsets для '{word}'")
508
509     for syn in synsets:
510         for lemma in syn.lemmas():
511             lemma_name = lemma.name().lower().replace('_', ' ')
512             if lemma_name != word.lower():
513                 synonyms.add(lemma_name)
514
515     result = list(synonyms)[:2]
516     logger.info(f"Знайдено синоніми через WordNet: {result}")
517     return result
518
519 def search_songs_by_keywords(keywords):
520     """Пошук пісень з кешуванням"""
521     # Спочатку перевіряємо кеш
522     cached_results = get_search_from_cache(keywords)
```

```
523     if cached_results:
524         return cached_results
525 
526     # Якщо в кеші немає, робимо запит до API
527     headers = {"Authorization": f"Bearer {GENIUS_API_TOKEN}"}
528     url = f"https://api.genius.com/search?q={keywords}"
529     response = requests.get(url, headers=headers)
530 
531     if response.status_code == 200:
532         results = response.json()
533         # Зберігаємо в кеш
534         save_search_to_cache(keywords, results)
535         return results
536     else:
537         logger.error(f"Помилка API пошуку: {response.status_code}")
538         return {"response": {"hits": []}}
539 
540 def get_song_lyrics(song_id):
541     headers = {"Authorization": f"Bearer {GENIUS_API_TOKEN}"}
542     url = f"https://api.genius.com/songs/{song_id}"
543     response = requests.get(url, headers=headers)
544 
545     if response.status_code != 200:
546         logger.error(f"Помилка API: {response.status_code}")
547         return "Текст пісні недоступний."
548 
549     song_url = response.json()['response']['song']['url']
550     logger.info(f"URL пісні: {song_url}")
551 
552     page = requests.get(song_url)
553     if page.status_code != 200:
554         logger.error(f"Помилка доступу до сторінки: {page.status_code}")
555         return "Текст пісні недоступний."
556 
557     soup = BeautifulSoup(page.text, 'html.parser')
558 
559     # Спочатку пробуємо знайти в форматі старого сайту
560     lyrics_div = soup.find('div', class_='lyrics')
561     if lyrics_div:
562         logger.info("Текст знайдено в старому форматі")
563         lyrics = lyrics_div.get_text(separator="\n")
564     else:
565         # Пробуємо знайти в новому форматі
566                                     lyrics_container    =    soup.find_all('div',
class_=re.compile("^Lyrics__Container"))
567         if lyrics_container:
```

```
568            logger.info(f"Текст знайдено в новому форматі: {len(lyrics_container)}
контейнерів")
569             lyrics = "\n".join([container.get_text(separator="\n") for container
in lyrics_container])
570         else:
571             # Перевіряємо структуру HTML і шукаємо текст за іншими ознаками
572            logger.info("Не знайдено стандартні контейнери, пробуємо альтернативні
методи")
573           possible_lyrics_divs = [div for div in soup.find_all('div') if "lyrics"
in str(div).lower()]
574             if possible_lyrics_divs:
575                           logger.info(f"Знайдено можливі контейнери з текстом:
{len(possible_lyrics_divs)}")
576                largest_div = max(possible_lyrics_divs, key=lambda x: len(str(x)))
577                 lyrics = largest_div.get_text(separator="\n")
578             else:
579                 logger.error("Текст пісні не знайдено в жодному з форматів")
580                 return "Текст пісні недоступний."
581
582     return lyrics
583
584 def clean_text(text):
585     # Список стоп-слів для російської мови
586     stop_words = set([
587         "a", "b", "c", "d", "e", "f", "g", "h", "j", "k", "l", "m", "n", "o", "p",
"q", "r", "s", "t", "u", "v",
588         "с", "у", "к", "о", "а", "ж", "б", "г", "д", "е", "з", "и", "й", "л", "м",
"н", "п", "р", "т",
589          "ф", "ц", "ч", "ш", "щ", "ъ", "ы", "ь", "э", "ю"
590     ])
591
592     # Видаляємо стоп-слова
593     words = text.split()
594     filtered_words = [word for word in words if word.lower() not in stop_words]
595     filtered_text = " ".join(filtered_words)
596
597     # Видаляємо спеціальні символи та зайві пробіли
598     filtered_text = re.sub(r'\[.*?\]', '', filtered_text).strip()
599     filtered_text = filtered_text.replace('\n', ' ')
600     filtered_text = filtered_text.lower()
601     filtered_text = re.sub(r'[^a-яА-ЯёЁa-zA-Z\s-]', '', filtered_text)
602
603     filtered_path = save_intermediate_text(filtered_text, "wordcloud_lyrics")
604
605     return filtered_text, filtered_path
606
```

```
607 def has_words(text):
608     return bool(re.search(r'\w+', text))
609 
610 def find_similar_songs(user_keywords, songs_lyrics):
611     vectorizer = TfidfVectorizer()
612     texts = [user_keywords] + songs_lyrics
613     tfidf_matrix = vectorizer.fit_transform(texts)
614     similarity = cosine_similarity(tfidf_matrix[0:1], tfidf_matrix[1:])
615     return similarity[0]
616 
617 
618 @app.route('/')
619 def home():
620     return render_template('index.html')
621 
622 @app.route('/find-songs', methods=['GET'])
623 def find_songs():
624     user_keywords = request.args.get('mood')
625     use_synonyms = request.args.get('synonyms', '').lower() == 'true'
626 
627     search_terms = [user_keywords]
628 
629     if use_synonyms:
630         # Отримуємо синоніми для основного слова
631         synonyms = get_synonyms(user_keywords)
632         if synonyms:
633             search_terms.extend(synonyms)
634 
635         logger.info(f"Пошук за словами: {search_terms}")
636 
637     all_songs = []
638 
639     for term in search_terms:
640         search_results = search_songs_by_keywords(term)
641 
642         if not search_results['response']['hits']:
643             continue
644 
645     songs_lyrics = []
646     songs_info = []
647      for song in search_results['response']['hits'][:10]:  # Беремо топ-10 для
кожного слова
648         song_id = song['result']['id']
649         title = song['result']['title']
650         artist = song['result']['primary_artist_names']
651 
```

```
652         # Отримуємо дані пісні (з кешу або API)
653         song_data = get_cached_song_data(song_id, title, artist)
654 
655         if not song_data:
656             continue
657 
658         songs_lyrics.append(song_data['preprocessed_lyrics'])
659 
660         song_info = {
661             "title": title,
662             "artist": artist,
663             "cover": song['result']['header_image_url'],
664             "url": song['result']['url'],
665             "lyrics": song_data['lyrics'],
666             "search_term": term,
667             "sentiment": song_data['sentiment']
668         }
669         songs_info.append(song_info)
670 
671     if songs_lyrics:
672         similarity_scores = find_similar_songs(user_keywords, songs_lyrics)
673         for song, score in zip(songs_info, similarity_scores):
674             song['similarity'] = score
675         all_songs.extend(songs_info)
676 
677     if not all_songs:
678         return jsonify({"error": "Пісні не знайдено"}), 404
679 
680     # Сортуємо всі пісні за схожістю та прибираємо дублікати
681     unique_songs = {}
682     for song in all_songs:
683         key = (song['title'], song['artist'])
684                     if key not in unique_songs or song['similarity'] >
unique_songs[key]['similarity']:
685             unique_songs[key] = song
686 
687      sorted_songs = sorted(unique_songs.values(), key=lambda x: x['similarity'],
reverse=True)
688 
689     result = []
690     for song in sorted_songs[:5]:  # Беремо топ-5 пісень
691         result.append({
692             "title": song['title'],
693             "artist": song['artist'],
694             "cover": song['cover'],
695             "url": song['url'],
```

```
            "lyrics": song['lyrics'],
            "found_by": song['search_term'] if use_synonyms else None,
            "sentiment": song['sentiment']
        })

    return jsonify(result)

@app.route('/generate-wordcloud', methods=['GET'])
def generate_wordcloud():
    artist_name = request.args.get('artist')
    song_title = request.args.get('song')

    logger.info(f"Пошук пісні для хмари слів: {artist_name} - {song_title}")

    # Спочатку перевіряємо кеш за назвою та виконавцем
    cached_song = get_song_from_cache(title=song_title, artist=artist_name)

    if cached_song:
        # Використовуємо дані з кешу
        words = cached_song['cleaned_lyrics'].split()
        word_freq = Counter(words)
            wordcloud_data = [{"text": word, "size": freq} for word, freq in
word_freq.items()]

        return jsonify({
            "wordcloud": wordcloud_data,
            "sentiment": cached_song['sentiment_data'],
        })

    # Якщо в кеші немає, шукаємо через API
    query = f"{quote(artist_name)} {quote(song_title)}"
    search_results = search_songs_by_keywords(query)

    if not search_results['response']['hits']:
        logger.info("Пісню не знайдено в API")
        return jsonify({"error": "Пісню не знайдено"}), 404

    song_id = search_results['response']['hits'][0]['result']['id']
    title = search_results['response']['hits'][0]['result']['title']
                                                            artist            =
search_results['response']['hits'][0]['result']['primary_artist_names']

    # Отримуємо повні дані пісні
    song_data = get_cached_song_data(song_id, title, artist)

    if not song_data:
```

```
740         return jsonify({"error": "Текст пісні недоступний"}), 404
741
742     if not has_words(song_data['cleaned_lyrics']):
743         return jsonify({"error": "Текст не містить слів"}), 404
744
745     words = song_data['cleaned_lyrics'].split()
746     word_freq = Counter(words)
747         wordcloud_data = [{"text": word, "size": freq} for word, freq in
word_freq.items()]
748
749     return jsonify({
750         "wordcloud": wordcloud_data,
751         "sentiment": song_data['sentiment'],
752     })
753
754 @app.route('/generate-wordcloud-from-songs', methods=['POST'])
755 def generate_wordcloud_from_songs():
756     data = request.json
757     logger.info("Отримано запит на створення хмари слів з кількох пісень")
758
759     if not data or 'songs' not in data:
760         logger.error("Помилка: не передано список пісень")
761         return jsonify({"error": "Не передано список пісень"}), 400
762
763     all_cleaned_texts = []
764     sentiment_scores = []
765
766     for song in data['songs']:
767         artist = song.get('artist')
768         title = song.get('title')
769
770         if not artist or not title:
771         logger.warning("Пропущено пісню без імені або виконавця")
772         continue
773
774         logger.info(f"Обробка пісні: {artist} - {title}")
775
776         # Перевірка кешу
777         cached = get_song_from_cache(title=title, artist=artist)
778
779         if cached:
780             logger.info("Дані знайдено в кеші")
781             all_cleaned_texts.append(cached['cleaned_lyrics'])
782             sentiment_scores.append(cached['sentiment_data'])
783             continue
784
```

```
785         # Пошук через API
786         query = f"{quote(artist)} {quote(title)}"
787         search_results = search_songs_by_keywords(query)
788 
789         if not search_results['response']['hits']:
790             logger.warning(f"Пісню не знайдено: {artist} - {title}")
791             continue
792 
793         song_info = search_results['response']['hits'][0]['result']
794         song_id = song_info['id']
795         real_title = song_info['title']
796         real_artist = song_info['primary_artist']['name']
797 
798         song_data = get_cached_song_data(song_id, real_title, real_artist)
799 
800         if not song_data or not has_words(song_data['cleaned_lyrics']):
801                 logger.warning(f"Немає доступного тексту для пісні: {artist} -
{title}")
802             continue
803 
804         all_cleaned_texts.append(song_data['cleaned_lyrics'])
805         sentiment_scores.append(song_data['sentiment'])  # або інший ключ
806 
807     if not all_cleaned_texts:
808         logger.error("Не вдалося отримати тексти жодної пісні")
809         return jsonify({"error": "Немає доступних текстів пісень"}), 404
810 
811     # Об'єднання текстів
812     combined_lyrics = " ".join(all_cleaned_texts)
813                     combined_path    =    save_intermediate_text(combined_lyrics,
"combined_wordcloud_lyrics")
814 
815     # Хмара слів
816     words = combined_lyrics.split()
817     word_freq = Counter(words)
818         wordcloud_data  =  [{"text":  word,  "size":  freq}  for  word,  freq  in
word_freq.items()]
819 
820 
821     logger.info(f"Об'єднаний текст містить {len(words)} слів")
822 
823     return jsonify({
824         "wordcloud": wordcloud_data,
825         'sentiment': {
826         'detailed_data': sentiment_scores,
827         'songs_count': len(data)}
```

```
828     })
829
830 @app.route('/analyze-playlist', methods=['GET'])
831 def analyze_playlist():
832     """Аналізує плейлист Spotify"""
833     logger.info("Отримано запит на отримання даних про плейлист")
834     try:
835         # Отримуємо URL з параметрів запиту
836         playlist_url_mess = request.args.get('url')
837         playlist_url = playlist_url_mess.split('?')[0]
838         if not playlist_url:
839             return jsonify({'error': 'URL плейлиста не вказано'}), 400
840
841         # Перевіряємо кеш в базі даних
842         cached_data = get_playlist_from_db(playlist_url)
843         if cached_data:
844             logger.info(f"Плейлист знайдено в кеші")
845             return jsonify({
846                 'cover': cached_data['cover'],
847                 'name': cached_data['name'],
848                 'owner': cached_data['owner'],
849                 'tracks_count': cached_data['tracks_count'],
850                 'description': cached_data['description'],
851                 'playlistUrl': playlist_url,
852                 'tracks': cached_data['tracks']
853             })
854
855         # Витягуємо ID плейлиста з URL
856         patterns = [
857             r'spotify:playlist:([a-zA-Z0-9]+)',
858             r'open\.spotify\.com/playlist/([a-zA-Z0-9]+)',
859             r'spotify\.com/playlist/([a-zA-Z0-9]+)'
860         ]
861
862         playlist_id = None
863         for pattern in patterns:
864             match = re.search(pattern, playlist_url)
865             if match:
866                 playlist_id = match.group(1)
867                 break
868
869         if not playlist_id:
870             return jsonify({'error': 'Невірний формат URL плейлиста'}), 400
871
872         # Отримуємо інформацію про плейлист через Spotify API
873         playlist = sp.playlist(playlist_id)
```

```
874
875         # Збираємо інформацію про плейлист
876         playlist_data = {
877             'name': playlist['name'],
878             'owner': playlist['owner']['display_name'],
879             'description': playlist['description'] or '',
880             'cover': playlist['images'][0]['url'] if playlist['images'] else None
881         }
882
883         # Отримуємо всі треки плейлиста
884         tracks_data = []
885         results = sp.playlist_tracks(playlist_id)
886
887         while results:
888             for item in results['items']:
889                 if item['track'] and item['track']['type'] == 'track':
890                     track = item['track']
891
892                     # Збираємо інформацію про трек
893                     track_info = {
894                         'title': track['name'],
895                               'artist': ', '.join([artist['name'] for artist in
track['artists']]),
896                                  'cover': track['album']['images'][0]['url'] if
track['album']['images'] else None,
897                         'url': track['external_urls']['spotify']
898                     }
899                     tracks_data.append(track_info)
900
901             # Отримуємо наступну сторінку треків, якщо є
902             results = sp.next(results) if results['next'] else None
903
904         # Зберігаємо в базу даних
905         save_playlist_to_db(playlist_data, tracks_data, playlist_url)
906
907         # Формуємо відповідь
908         response_data = {
909             'cover': playlist_data['cover'],
910             'name': playlist_data['name'],
911             'owner': playlist_data['owner'],
912             'tracks_count': len(tracks_data),
913             'description': playlist_data['description'],
914             'playlistUrl': playlist_url,
915             'tracks': tracks_data
916         }
917
```

```
        return jsonify(response_data)

    except spotipy.exceptions.SpotifyException as e:
        if e.http_status == 404:
            return jsonify({'error': 'Плейлист не знайдено'}), 404
        elif e.http_status == 401:
            return jsonify({'error': 'Помилка авторизації Spotify API'}), 401
        else:
            return jsonify({'error': f'Помилка Spotify API: {str(e)}'}), 500

    except Exception as e:
        return jsonify({'error': f'Внутрішня помилка сервера: {str(e)}'}), 500


@app.route('/generate-playlist-wordcloud', methods=['POST'])
def generate_playlist_wordcloud():
    data = request.get_json()
    playlist_url = data.get('playlist_url')

    if not playlist_url:
        return jsonify({"error": "Відсутнє посилання на плейлист"}), 400

    playlist = get_playlist_from_db(playlist_url)
    if not playlist or 'tracks' not in playlist:
       logger.error(f"Не вдалося отримати плейлист за посиланням: {playlist_url}")
        return jsonify({"error": "Плейлист не знайдено"}), 404

    all_cleaned_texts = []
    sentiment_scores = []

    for track in playlist['tracks']:
        title = track['title']
        artist = track['artist']

        # Спочатку намагаємося взяти з кешу
        cached_song = get_song_from_cache(title=title, artist=artist)
        if cached_song:
            song_data = cached_song
            logger.info("Дані знайдено в кеші")
            all_cleaned_texts.append(cached_song['cleaned_lyrics'])
            sentiment_scores.append(cached_song['sentiment_data'])
            continue

        # Якщо немає в кеші, шукаємо через API
        query = f"{quote(artist)} {quote(title)}"
        search_results = search_songs_by_keywords(query)
```

```
964         if not search_results.get('response', {}).get('hits'):
965             logger.info(f"Пісню не знайдено в API: {artist} - {title}")
966             continue  # пропускаємо цю пісню
967 
968         first_hit = search_results['response']['hits'][0]['result']
969         song_id = first_hit['id']
970         real_title = first_hit['title']
971         real_artist = first_hit.get('primary_artist_names', artist)
972 
973         # Отримуємо повні дані пісні за song_id
974         song_data = get_cached_song_data(song_id, real_title, real_artist)
975         if not song_data:
976             logger.warning(f"Текст пісні недоступний: {artist} - {title}")
977             continue
978 
979         # Перевіряємо, що є слова в тексті
980         if not has_words(song_data['cleaned_lyrics']):
981             logger.warning(f"Текст не містить слів: {artist} - {title}")
982             continue
983 
984         all_cleaned_texts.append(song_data['cleaned_lyrics'])
985         sentiment_scores.append(song_data['sentiment'])
986 
987     if not all_cleaned_texts:
988         logger.error("Не вдалося отримати тексти жодної пісні")
989         return jsonify({"error": "Немає доступних текстів пісень"}), 404
990 
991     # Розрахунок загального настрою плейлиста
992     if sentiment_scores:
993         songs_with_sentiment = len(sentiment_scores)
994 
995         # Ініціалізація сум для емоцій
996         emotion_totals = {
997             'joy': 0, 'sadness': 0, 'anger': 0, 'fear': 0,
998             'love': 0, 'surprise': 0, 'disgust': 0, 'trust': 0
999         }
1000 
1001          dominant_emotions = []
1002          confidence_levels = []
1003 
1004          # Підсумовування тільки потрібних метрик
1005          for score in sentiment_scores:
1006              # Emotion profile
1007              if 'emotion_profile' in score:
1008                  for key in emotion_totals:
1009                      emotion_totals[key] += score['emotion_profile'].get(key, 0)
```

```
1010
1011            # Домінуючі емоції та рівні впевненості
1012            if 'dominant_emotion' in score:
1013                dominant_emotions.append(score['dominant_emotion'])
1014            if 'confidence' in score:
1015                confidence_levels.append(score['confidence'])
1016
1017        # Розрахунок середніх значень тільки для емоцій
1018        overall_sentiment = {
1019            'emotion_profile': {
1020                key: round(total / songs_with_sentiment, 3)
1021                for key, total in emotion_totals.items()
1022            },
1023            'songs_analyzed': songs_with_sentiment
1024        }
1025
1026        # Визначення домінуючої емоції плейлиста
1027        if dominant_emotions:
1028            emotion_counts = Counter(dominant_emotions)
1029            most_common_emotion = emotion_counts.most_common(1)[0][0]
1030            overall_sentiment['dominant_emotion'] = most_common_emotion
1031            overall_sentiment['emotion_distribution'] = dict(emotion_counts)
1032        else:
1033            overall_sentiment['dominant_emotion'] = 'unknown'
1034            overall_sentiment['emotion_distribution'] = {}
1035
1036        # Визначення загального рівня впевненості
1037        if confidence_levels:
1038            confidence_counts = Counter(confidence_levels)
1039            most_common_confidence = confidence_counts.most_common(1)[0][0]
1040            overall_sentiment['confidence'] = most_common_confidence
1041        else:
1042            overall_sentiment['confidence'] = 'low'
1043
1044               logger.info(f"Загальний настрій плейлиста: домінуюча емоція -
{overall_sentiment.get('dominant_emotion', 'unknown')}")
1045    else:
1046        overall_sentiment = {
1047             'emotion_profile': {'joy': 0, 'sadness': 0, 'anger': 0, 'fear': 0,
'love': 0, 'surprise': 0, 'disgust': 0, 'trust': 0},
1048            'songs_analyzed': 0,
1049            'dominant_emotion': 'unknown',
1050            'emotion_distribution': {},
1051            'confidence': 'low'
1052        }
1053
```

```
1054     # Збереження загального настрою в базу даних
1055     try:
1056         conn = sqlite3.connect('songs_cache.db')
1057         cursor = conn.cursor()
1058 
1059         # Перевіряємо, чи існує запис плейлиста
1060         cursor.execute('''
1061             SELECT id FROM playlists WHERE url = ?
1062         ''', (playlist_url,))
1063 
1064         playlist_record = cursor.fetchone()
1065 
1066         if playlist_record:
1067             # Оновлюємо існуючий запис
1068             cursor.execute('''
1069                 UPDATE playlists
1070                 SET overall_sentiment = ?, updated_at = CURRENT_TIMESTAMP
1071                 WHERE url = ?
1072             ''', (json.dumps(overall_sentiment), playlist_url))
1073          logger.info(f"Оновлено загальний настрій для плейлиста: {playlist_url}")
1074         else:
1075             # Створюємо новий запис
1076             cursor.execute('''
1077                     INSERT INTO playlists (url, overall_sentiment, created_at,
updated_at)
1078                 VALUES (?, ?, CURRENT_TIMESTAMP, CURRENT_TIMESTAMP)
1079             ''', (playlist_url, json.dumps(overall_sentiment)))
1080              logger.info(f"Створено новий запис плейлиста із загальним настроєм:
{playlist_url}")
1081 
1082         conn.commit()
1083         conn.close()
1084 
1085     except sqlite3.Error as e:
1086         logger.error(f"Помилка при збереженні загального настрою плейлиста в БД:
{e}")
1087 
1088     # Об'єднуємо тексти та створюємо хмару слів
1089     combined_lyrics = " ".join(all_cleaned_texts)
1090                     combined_path    =    save_intermediate_text(combined_lyrics,
"combined_playlist_wordcloud_lyrics")
1091     words = combined_lyrics.split()
1092     word_freq = Counter(words)
1093         wordcloud_data  =  [{"text":  word,  "size":  freq}  for  word,  freq  in
word_freq.items()]
1094 
```

```
1095     logger.info(f"Об'єднаний текст містить {len(words)} слів")
1096
1097     return jsonify({
1098         "wordcloud": wordcloud_data,
1099         'sentiment': {
1100             'detailed_data': sentiment_scores,
1101             'overall_sentiment': overall_sentiment,
1102             'songs_count': len(playlist['tracks'])
1103         }
1104     })
1105
1106 @app.route('/find-playlists', methods=['GET'])
1107 def find_playlists():
1108     mood = request.args.get('mood')
1109
1110     if not mood:
1111         return jsonify({"error": "Відсутній параметр mood"}), 400
1112
1113     logger.info(f"Пошук плейлистів для настрою: {mood}")
1114
1115     # Підключаємося до бази даних
1116     conn = sqlite3.connect('songs_cache.db')
1117     cursor = conn.cursor()
1118
1119     try:
1120         # Отримуємо всі плейлисти з емоціями
1121          cursor.execute("SELECT * FROM playlists WHERE overall_sentiment IS NOT
NULL")
1122         playlists = cursor.fetchall()
1123
1124         # Отримуємо назви колонок
1125         column_names = [description[0] for description in cursor.description]
1126
1127         if not playlists:
1128             logger.info("У базі даних немає плейлистів з даними про настрій")
1129             return jsonify([]), 200
1130
1131         playlist_scores = []
1132
1133         for playlist_row in playlists:
1134             playlist = dict(zip(column_names, playlist_row))
1135             sentiment_data = playlist.get('overall_sentiment', '')
1136
1137             if not sentiment_data:
1138                 continue
1139
```

```
1140            # Парсимо JSON з даними про настрій
1141            try:
1142                sentiment = json.loads(sentiment_data)
1143                emotion_profile = sentiment.get('emotion_profile', {})
1144                confidence = sentiment.get('confidence', 'low')
1145
1146                if not emotion_profile:
1147                    continue
1148
1149            except json.JSONDecodeError:
1150                   logger.warning(f"Не вдалося розпізнати overall_sentiment для
плейлиста {playlist.get('name', 'unknown')}")
1151                continue
1152
1153            # Шукаємо максимальне значення для нашого настрою
1154            mood_score = 0
1155            mood_lower = mood.lower()
1156
1157            # Пряме збігання з емоціями з emotion_profile
1158            if mood_lower in emotion_profile:
1159                mood_score = emotion_profile[mood_lower]
1160            else:
1161                # Пошук за частковим збіганням
1162                for emotion, score in emotion_profile.items():
1163                        if mood_lower in emotion.lower() or emotion.lower() in
mood_lower:
1164                        mood_score = max(mood_score, score)
1165
1166            if mood_score > 0:
1167                # Застосовуємо множник впевненості
1168                confidence_multiplier = 1.0
1169                if confidence == 'high':
1170                    confidence_multiplier = 1.5
1171                elif confidence == 'medium':
1172                    confidence_multiplier = 1.2
1173                elif confidence == 'low':
1174                    confidence_multiplier = 1.0
1175
1176                final_score = mood_score * confidence_multiplier
1177
1178                playlist_scores.append({
1179                    'playlist': playlist,
1180                    'score': final_score
1181                })
1182
1183        # Сортуємо за спаданням відповідності та беремо топ-5
```

```
        playlist_scores.sort(key=lambda x: x['score'], reverse=True)
        top_playlists = playlist_scores[:5]

        logger.info(f"Знайдено {len(top_playlists)} відповідних плейлистів")

        # Формуємо результат
        result = []
        for item in top_playlists:
            playlist = item['playlist']
            result.append({
                'name': playlist.get('name', 'Без назви'),
                'owner': playlist.get('owner', 'Невідомий'),
                'description': playlist.get('description', ''),
                'tracks_count': playlist.get('tracks_count'),
                'cover': playlist.get('cover_url') or playlist.get('cover'),
                'url': playlist.get('url') or playlist.get('external_url')
            })

        return jsonify(result)

    except Exception as e:
        logger.error(f"Помилка при пошуку плейлистів: {str(e)}")
        return jsonify({"error": "Помилка при пошуку плейлистів"}), 500
    finally:
        conn.close()


if __name__ == '__main__':
    # Ініціалізуємо базу даних при запуску
    init_database()
    app.run(debug=True)
```

Рис. Б.1 – Код файлу app.py

```
<!DOCTYPE html>
<html lang="uk">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Музичний настрій</title>
    <link href="https://fonts.googleapis.com/css2?family=Roboto&display=swap"
rel="stylesheet">
    <script src="https://code.jquery.com/jquery-3.6.4.min.js"></script>
```

```
9       <script src="https://cdn.jsdelivr.net/npm/toastify-js"></script>
10      <script
src="https://cdnjs.cloudflare.com/ajax/libs/wordcloud2.js/1.1.1/wordcloud2.min.js"><
/script>
11      <link rel="stylesheet" href="/static/styles.css">
12  </head>
13  <body>
14      <!-- Шапка з навігацією -->
15      <header class="header">
16          <div class="nav-container">
17              <div class="logo">
18                  Музичний настрій
19              </div>
20              <nav class="nav-tabs">
21                  <button class="nav-tab active" data-page="songs">Аналіз
пісень</button>
22                  <button class="nav-tab" data-page="playlists">Аналіз
плейлистів</button>
23              </nav>
24          </div>
25      </header>
26
27      <!-- Основний контент -->
28      <main class="main-content">
29          <!-- Сторінка аналізу пісень -->
30          <div id="songs-page" class="page active">
31              <h1>Аналіз пісень за настроєм</h1>
32
33              <div class="search-container">
34                  <div class="search-mode">
35                      <span class="mode-label">За ключовим словом</span>
36                      <label class="switch">
37                          <input type="checkbox" id="searchModeToggle">
38                          <span class="slider"></span>
39                      </label>
40                      <span class="mode-label">За синонімами</span>
41                  </div>
42
43                  <form id="moodForm">
44                      <label for="mood">Опишіть ваш настрій або введіть ключові
слова:</label>
45                      <input type="text" id="mood" name="mood" required
placeholder="Наприклад: сумний, веселий, енергійний">
46                      <button type="submit">Знайти пісні</button>
47                  </form>
48              </div>
```

```
49
50              <div id="results"></div>
51
52              <!-- Кнопка для створення хмари слів із знайдених пісень -->
53              <div class="wordcloud-button-container">
54                  <button id="generateWordcloudFromDisplayedSongs" type="button"
hidden>Створити хмару слів із знайдених пісень</button>
55              </div>
56
57              <h2>Створити хмару слів для конкретної пісні</h2>
58              <div class="search-container">
59                  <form id="wordcloudForm">
60                      <label for="artist">Виконавець:</label>
61                      <input type="text" id="artist" name="artist" required
placeholder="Назва виконавця">
62
63                      <label for="song">Пісня:</label>
64                      <input type="text" id="song" name="song" required
placeholder="Назва пісні">
65
66                      <button type="submit">Створити хмару слів</button>
67                  </form>
68              </div>
69
70              <div id="wordcloudResult">
71                  <div id="wordcloud"></div>
72                  <div id="sentimentDisplay" class="sentiment-display"
hidden></div>
73              </div>
74          </div>
75
76          <!-- Сторінка аналізу плейлистів -->
77          <div id="playlists-page" class="page">
78              <h1>Аналіз плейлистів</h1>
79
80              <div class="playlist-input-container">
81                  <h3>Аналіз плейлиста за посиланням</h3>
82                  <form id="playlistUrlForm">
83                      <label for="playlistUrl">Посилання на плейлист:</label>
84                      <input type="url" id="playlistUrl" name="playlistUrl"
class="wide-input" required placeholder="https://open.spotify.com/playlist/...">
85                      <button type="submit">Аналізувати плейлист</button>
86                  </form>
87              </div>
88
89              <div class="playlist-input-container">
```

```
90                 <h3>Пошук плейлистів за настроєм</h3>
91                 <form id="playlistMoodForm">
92                     <label for="playlistMood">Настрій для пошуку
плейлистів:</label>
93                     <input type="text" id="playlistMood" name="playlistMood"
required placeholder="Наприклад: релакс, тренування, романтика">
94                     <button type="submit">Знайти плейлисти</button>
95                 </form>
96             </div>
97
98             <div id="playlistResults"></div>
99
100            <!-- Кнопка для створення хмари слів із плейлиста -->
101            <div class="wordcloud-button-container">
102                <button id="generatePlaylistWordcloud" type="button"
hidden>Створити хмару слів із плейлиста</button>
103            </div>
104
105            <div id="playlistWordcloudResult">
106                <div id="playlistWordcloud"></div>
107                <div id="playlistSentimentDisplay" class="sentiment-display"
hidden></div>
108            </div>
109        </div>
110        <div style="height: 100px;"></div>
111    </main>
112
113    <script>
114        // Глобальні змінні
115        let currentSongs = [];
116        let currentPlaylistData = null;
117
118        // Функція перемикання між сторінками
119        function switchPage(pageId) {
120            // Приховуємо всі сторінки
121            document.querySelectorAll('.page').forEach(page => {
122                page.classList.remove('active');
123            });
124
125            // Прибираємо активний клас з усіх вкладок
126            document.querySelectorAll('.nav-tab').forEach(tab => {
127                tab.classList.remove('active');
128            });
129
130            // Показуємо вибрану сторінку
131            document.getElementById(pageId + '-page').classList.add('active');
```

```
132
133             // Додаємо активний клас до вибраної вкладки
134             document.querySelector(`[data-
page="${pageId}"]`).classList.add('active');
135         }
136
137         // Обробники подій для навігації
138         document.querySelectorAll('.nav-tab').forEach(tab => {
139             tab.addEventListener('click', () => {
140                 const pageId = tab.getAttribute('data-page');
141                 switchPage(pageId);
142             });
143         });
144
145         // Функція для відображення помилок у консолі та користувачеві
146         function showToastWithFade(message, background) {
147     const toast = Toastify({
148         text: message,
149         duration: -1, // вимикаємо автовидалення
150         close: true,
151         gravity: "bottom",
152         position: "center",
153         backgroundColor: background,
154         className: "toastify-fade",
155         stopOnFocus: true,
156     });
157
158     toast.showToast();
159
160     // Чекаємо, поки елемент з'явиться в DOM (у наступному "тіку" подій)
161     requestAnimationFrame(() => {
162         const toastEl = toast.toastElement;
163
164         if (!toastEl) return;
165
166         // Обробник закриття по хрестику
167         const closeBtn = toastEl.querySelector(".toast-close");
168         if (closeBtn) {
169             closeBtn.addEventListener("click", (e) => {
170                 e.stopPropagation();
171                 toastEl.classList.add("toastify-hide");
172                 setTimeout(() => toastEl.remove(), 500);
173             });
174         }
175
176         // Автоматичне закриття через 3 секунди
```

```
177        setTimeout(() => {
178            toastEl.classList.add("toastify-hide");
179            setTimeout(() => toastEl.remove(), 500);
180        }, 3000);
181    });
182}
183
184function showSuccess(message) {
185    showToastWithFade(message, "linear-gradient(to right, #00b09b, #96c93d)");
186}
187
188function showError(message) {
189    showToastWithFade(message, "linear-gradient(to right, #ff5f6d, #ffc371)");
190}
191
192
193function showSuccess(message) {
194    showToastWithFade(message, "linear-gradient(to right, #00b09b, #96c93d)");
195}
196
197function showError(message) {
198    showToastWithFade(message, "linear-gradient(to right, #ff5f6d, #ffc371)");
199}
200
201
202        // ОБРОБНИКИ ДЛЯ СТОРІНКИ ПЛЕЙЛИСТІВ
203
204        // Обробка форми для аналізу плейлиста за посиланням
205        document.getElementById('playlistUrlForm').addEventListener('submit',
async (e) => {
206    e.preventDefault();
207    const playlistUrl = document.getElementById('playlistUrl').value;
208
209    try {
210        const response = await fetch(`/analyze-
playlist?url=${encodeURIComponent(playlistUrl)}`);
211        if (!response.ok) throw new Error(`Помилка HTTP: ${response.status}`);
212
213        const data = await response.json();
214        currentPlaylistData = data;
215
216        const resultsDiv = document.getElementById('playlistResults');
217
218        // Генерація блоку з плейлистом
219        resultsDiv.innerHTML = `
220<div class="playlist-item">
```

```
221    <div style="display: flex; gap: 20px; align-items: flex-start;">
222        <img src="${data.cover || '/default-playlist.png'}" alt="Обкладинка
плейлиста">
223        <div class="playlist-info">
224            <h3>${data.name}</h3>
225            <p><strong>Створив:</strong> ${data.owner}</p>
226            <p><strong>Треків:</strong> ${data.tracks ? data.tracks.length :
0}</p>
227            <p><strong>Опис:</strong> ${data.description || 'Немає опису'}</p>
228            <div style="display: flex; justify-content: center; gap: 15px;
margin-top: 10px;">
229                <button onclick="window.open('${playlistUrl}',
'_blank')">Відкрити плейлист</button>
230                <button id="toggleTracks">▼ Показати треки</button>
231            </div>
232        </div>
233    </div>
234    <div id="tracksContainer" style="max-height: 0; overflow: hidden; transition:
max-height 0.3s ease-out; margin-top: 0;">
235        <div id="tracksList" style="padding-top: 16px;"></div>
236    </div>
237</div>
238
239`;
240
241        // Подія розкриття треків
242        const toggleBtn = document.getElementById('toggleTracks');
243        const tracksContainer = document.getElementById('tracksContainer');
244
245        toggleBtn.addEventListener('click', () => {
246            if (tracksContainer.style.maxHeight === '0px') {
247                // Обчислюємо висоту вмісту та встановлюємо її
248                const contentHeight =
document.getElementById('tracksList').scrollHeight;
249                tracksContainer.style.maxHeight = `${contentHeight}px`;
250                toggleBtn.textContent = '▲ Сховати треки';
251            } else {
252                tracksContainer.style.maxHeight = '0';
253                toggleBtn.textContent = '▼ Показати треки';
254            }
255        });
256
257        // Генерація треків у вигляді списку
258        if (data.tracks && data.tracks.length > 0) {
259            const tracksHtml = data.tracks.map((track, index) => `
```

```
260                <div class="track-item" style="display: flex; align-items:
center; padding: 10px; border-radius: 6px; background: ${index % 2 === 0 ? '#f0f0f0'
: '#ffffff'}; margin-bottom: 4px;">
261                    <div style="width: 40px; text-align: center; color: #666;
font-size: 0.9em;">${index + 1}</div>
262                    <img src="${track.cover || '/default-song.png'}"
alt="Обкладинка треку" style="width: 40px; height: 40px; object-fit: cover; border-
radius: 4px; margin-right: 12px;">
263                    <div style="flex: 1; min-width: 400px;">
264                        <div style="font-weight: bold; white-space: nowrap;
overflow: hidden; text-overflow: ellipsis;">${track.title}</div>
265                        <div style="font-size: 0.85em; color: #666; white-space:
nowrap; overflow: hidden; text-overflow: ellipsis;">${track.artist}</div>
266                    </div>
267                    ${track.url ? `<button onclick="window.open('${track.url}',
'_blank')" style="padding: 6px 12px;">Слухати</a>` : ''}
268                </div>
269            `).join('');
270
271            document.getElementById('tracksList').innerHTML = tracksHtml;
272            document.getElementById('generatePlaylistWordcloud').hidden = false;
273            showSuccess(`Плейлист завантажено успішно. Знайдено
${data.tracks.length} треків`);
274        } else {
275            document.getElementById('generatePlaylistWordcloud').hidden = true;
276            showError("Не вдалося завантажити треки з плейлиста");
277        }
278
279    } catch (error) {
280        showError("Помилка при аналізі плейлиста", error);
281    }
282});
283
284
285        // Обробка форми для пошуку плейлистів за настроєм
286        document.getElementById('playlistMoodForm').addEventListener('submit',
async (e) => {
287            e.preventDefault();
288            const mood = document.getElementById('playlistMood').value;
289
290            try {
291                const response = await fetch(`/find-
playlists?mood=${encodeURIComponent(mood)}`);
292                console.log("API відповідь статус (пошук плейлистів):",
response.status);
293
```

```
294                 if (!response.ok) {
295                     throw new Error(`Помилка HTTP: ${response.status}`);
296                 }
297 
298                 const playlists = await response.json();
299                 console.log("Знайдені плейлисти:", playlists);
300 
301                 const resultsDiv = document.getElementById('playlistResults');
302                 resultsDiv.innerHTML = playlists.map(playlist => `
303                     <div class="playlist-item">
304                         <img src="${playlist.cover || '/default-playlist.png'}"
alt="Обкладинка плейлиста">
305                         <div class="playlist-info">
306                             <h3>${playlist.name}</h3>
307                             <p>Створив: ${playlist.owner}</p>
308                             <p>Кількість треків: ${playlist.tracks_count ||
'Невідомо'}</p>
309                             <p>Опис: ${playlist.description || 'Немає опису'}</p>
310                             <button onclick="window.open('${playlistUrl}',
'_blank')" style="display: inline-block; padding: 8px 16px; ">Відкрити плейлист</a>
311                         </div>
312                     </div>
313                 `).join('');
314 
315                 if (playlists.length > 0) {
316                     showSuccess(`Знайдено ${playlists.length} плейлистів`);
317                 } else {
318                     showError("Плейлисти не знайдені");
319                 }
320 
321             } catch (error) {
322                 showError("Помилка при пошуку плейлистів", error);
323             }
324         });
325 
326         // Обробка кнопки для створення хмари слів із плейлиста
327 
document.getElementById('generatePlaylistWordcloud').addEventListener('click', async
(e) => {
328             e.preventDefault();
329             e.stopPropagation();
330 
331             try {
332                 if (!currentPlaylistData || !currentPlaylistData.tracks ||
currentPlaylistData.tracks.length === 0) {
333                     throw new Error("Дані плейлиста не знайдені");
```

```
334                 }
335
336                 console.log("Відправка плейлиста для генерації хмари слів:",
currentPlaylistData.playlistUrl);
337
338                 const response = await fetch('/generate-playlist-wordcloud', {
339                     method: 'POST',
340                     headers: {
341                         'Content-Type': 'application/json',
342                     },
343                     body: JSON.stringify({
344                         playlist_url: currentPlaylistData.playlistUrl
345                     }),
346                 });
347
348                 console.log("API відповідь статус (хмара з плейлиста):",
response.status);
349
350                 if (!response.ok) {
351                     const errorData = await response.json();
352                     throw new Error(errorData.error || `Помилка HTTP:
${response.status}`);
353                 }
354
355                 const data = await response.json();
356                 console.log("Отримані дані хмари слів з плейлиста:", data);
357
358                 const wordcloudElement =
document.getElementById('playlistWordcloud');
359                 wordcloudElement.innerHTML = '';
360
361                 document.getElementById('playlistSentimentDisplay').hidden =
true;
362
363                 if (!data.wordcloud && Array.isArray(data)) {
364                     generateWordCloud(wordcloudElement, data);
365                 } else if (data.wordcloud) {
366                     generateWordCloud(wordcloudElement, data.wordcloud);
367
368                     const sentiment = data.sentiment;
369                     if (sentiment?.detailed_data &&
sentiment.detailed_data.length > 0) {
370                         // Збираємо статистику за емоціями
371                         const emotions = {};
372                         let totalCompound = 0;
373                         let validScores = 0;
```

```
374
375                         sentiment.detailed_data.forEach(songData => {
376                             if (songData.dominant_emotion) {
377                                 const emotion = songData.dominant_emotion;
378                                 emotions[emotion] = (emotions[emotion] || 0) + 1;
379                             }
380
381                             if (songData.vader_scores?.compound !== undefined) {
382                                 totalCompound += songData.vader_scores.compound;
383                                 validScores++;
384                             }
385                         });
386
387                         const averageCompound = validScores > 0 ? totalCompound /
validScores : 0;
388                         const mostCommonEmotion = Object.keys(emotions).length >
0 ?
389                             Object.keys(emotions).reduce((a, b) => emotions[a] >
emotions[b] ? a : b) : 'невідомо';
390
391                         const sentimentDisplay =
document.getElementById('playlistSentimentDisplay');
392                         sentimentDisplay.hidden = false;
393
394                         // Створюємо список емоцій з кількістю
395                         const emotionStats = Object.entries(emotions)
396                             .sort(([,a], [,b]) => b - a)
397                             .map(([emotion, count]) =>
398                                 `<span class="emotion-item">${emotion}:
${count}</span>`
399                             ).join('');
400
401                         sentimentDisplay.innerHTML = `
402                             <div class="sentiment-item">
403                                 <div class="sentiment-score" style="color:
${averageCompound >= 0 ? '#28a745' : '#dc3545'}">
404                                     Середній настрій плейлиста:
${averageCompound.toFixed(2)}
405                                 </div>
406                             </div>
407                             <div class="sentiment-item">
408                                 <div class="dominant-emotion">Домінуюча емоція:
${mostCommonEmotion}</div>
409                             </div>
410                             <div class="sentiment-item">
```

```
411                                 <div>Кількість треків:
${sentiment.detailed_data.length}</div>
412                             </div>
413                             ${emotionStats ? `<div class="emotion-
stats">${emotionStats}</div>` : ''}
414                         `;
415                     } else if (sentiment?.average_compound !== undefined) {
416                         const score = sentiment.average_compound;
417                         const sentimentDisplay =
document.getElementById('playlistSentimentDisplay');
418                         sentimentDisplay.hidden = false;
419                         sentimentDisplay.innerHTML = `
420                             <div class="sentiment-score" style="color: ${score >=
0 ? '#28a745' : '#dc3545'}">
421                                 Середній настрій плейлиста: ${score.toFixed(2)}
422                             </div>
423                         `;
424                     }
425                 } else {
426                     throw new Error("Невідомий формат даних хмари слів");
427                 }
428
429                 showSuccess("Хмара слів із плейлиста створена успішно");
430             } catch (error) {
431                 showError("Помилка при створенні хмари слів із плейлиста",
error);
432             }
433         });
434
435         // Функція для генерації хмари слів
436         function generateWordCloud(element, data) {
437             console.log("Генерація хмари слів, к-ть слів:", data.length);
438
439             if (data.length === 0) {
440                 element.innerHTML = '<p class="error">Недостатньо даних для
створення хмари слів</p>';
441                 return;
442             }
443
444             try {
445                 // Перевірка формату даних
446                 const wordList = Array.isArray(data[0]) ?
447                     data :
448                     data.map(item => [item.text, item.size]);
449
```

```
450                 console.log("Підготовлені дані для хмари:", wordList.slice(0, 5),
"...");
451
452                 // Перевірка, завантажена бібліотека WordCloud
453                 if (typeof WordCloud !== 'function') {
454                     throw new Error("Бібліотека WordCloud не завантажена");
455                 }
456
457                 WordCloud(element, {
458                     list: wordList,
459                     gridSize: 10,
460                     weightFactor: 8,
461                     fontFamily: 'Roboto, sans-serif',
462                     color: 'random-dark',
463                     backgroundColor: '#ffffff',
464                     rotateRatio: 0.5,
465                     minSize: 4,
466                     padding: 1,
467                     wait: 75,
468                     click: function(item) {
469                         const words = element.querySelectorAll('span');
470                         words.forEach(word =>
word.classList.remove('highlight'));
471                         item.element.classList.add('highlight');
472                         showSuccess(`Ви вибрали слово: ${item[0]}`);
473                     }
474                 });
475                 console.log("Хмара слів успішно згенерована");
476             } catch (error) {
477                 console.error("Помилка при відображенні хмари слів:", error);
478                 element.innerHTML = `<p class="error">Помилка створення хмари
слів: ${error.message}</p>`;
479             }
480         }
481
482         // Перевірка завантаження бібліотеки WordCloud при завантаженні сторінки
483         window.addEventListener('load', function() {
484             console.log("Сторінка завантажена");
485             if (typeof WordCloud !== 'function') {
486                 console.error("Помилка: бібліотека WordCloud не завантажена");
487                 showError("Бібліотека WordCloud не завантажена, хмари слів можуть
не працювати");
488             } else {
489                 console.log("Бібліотека WordCloud успішно завантажена");
490             }
491         });
```

```
492
493         // ОБРОБНИКИ ДЛЯ СТОРІНКИ ПІСЕНЬ
494
495         // Обробка форми для пошуку пісень
496         document.getElementById('moodForm').addEventListener('submit', async (e)
=> {
497             e.preventDefault();
498             const mood = document.getElementById('mood').value;
499             const searchBySynonyms =
document.getElementById('searchModeToggle').checked;
500
501             let endpoint = `/find-songs?mood=${encodeURIComponent(mood)}`;
502             if (searchBySynonyms) {
503                 endpoint += `&synonyms=true`;
504             }
505
506             try {
507                 const response = await fetch(endpoint);
508                 console.log("API відповідь статус:", response.status);
509
510                 if (!response.ok) {
511                     throw new Error(`Помилка HTTP: ${response.status}`);
512                 }
513
514                 const songs = await response.json();
515                 console.log("Отримані дані:", songs);
516
517                 // Зберігаємо інформацію про пісні глобально
518                 currentSongs = songs.map(song => ({
519                     artist: song.artist,
520                     title: song.title
521                 }));
522
523                 const resultsDiv = document.getElementById('results');
524                 resultsDiv.innerHTML = songs.map(song => `
525                     <div class="song">
526                         <img src="${song.cover}" alt="Обкладинка пісні">
527                         <div class="song-info">
528                             <h3>${song.title}</h3>
529                             <p>${song.artist}</p>
530                             <a href="${song.url}" target="_blank">Посилання на
Genius</a>
531                         </div>
532                     </div>
533                 `).join('');
534
```

```
535                 // Показуємо кнопку, якщо є пісні
536                 if (currentSongs.length > 0) {
537 
document.getElementById('generateWordcloudFromDisplayedSongs').hidden = false;
538                     showSuccess(`Знайдено ${currentSongs.length} пісень`);
539                 } else {
540 
document.getElementById('generateWordcloudFromDisplayedSongs').hidden = true;
541                     showError("Пісні не знайдені");
542                 }
543             } catch (error) {
544                 showError("Помилка при пошуку пісень", error);
545             }
546         });
547 
548         // Обробка форми для створення хмари слів з однієї пісні
549         document.getElementById('wordcloudForm').addEventListener('submit', async
(e) => {
550             e.preventDefault();
551             const artist = document.getElementById('artist').value;
552             const song = document.getElementById('song').value;
553 
554             try {
555                 const response = await fetch(`/generate-
wordcloud?artist=${encodeURIComponent(artist)}&song=${encodeURIComponent(song)}`);
556                 console.log("API відповідь статус (хмара слів):",
response.status);
557 
558                 if (!response.ok) {
559                     throw new Error(`Помилка HTTP: ${response.status}`);
560                 }
561 
562                 const data = await response.json();
563                 console.log("Отримані дані хмари слів:", data);
564 
565                 const wordcloudElement = document.getElementById('wordcloud');
566                 wordcloudElement.innerHTML = '';
567 
568                 // Очищаємо відображення настрою
569                 document.getElementById('sentimentDisplay').hidden = true;
570 
571                 // Перевіряємо структуру даних
572                 if (!data.wordcloud && Array.isArray(data)) {
573                     console.log("Отримані дані у старому форматі, перетворюємо");
574                     generateWordCloud(wordcloudElement, data);
575                 } else if (data.wordcloud) {
```

```
576                     generateWordCloud(wordcloudElement, data.wordcloud);
577
578                     // Отримуємо та відображаємо дані про настрій
579                     const sentiment = data.sentiment;
580                     console.log("Результат аналізу настрою:", sentiment);
581
582                     if (sentiment.vader_scores) {
583                         const sentimentDisplay =
document.getElementById('sentimentDisplay');
584                         sentimentDisplay.hidden = false;
585
586                         const compound = sentiment.vader_scores.compound;
587                         const dominantEmotion = sentiment.dominant_emotion ||
'невідомо';
588                         const confidence = sentiment.confidence || 'невідомо';
589
590                         sentimentDisplay.innerHTML = `
591                             <div class="sentiment-item">
592                                 <div class="sentiment-score" style="color:
${compound >= 0 ? '#28a745' : '#dc3545'}">
593                                     Настрій пісні: ${compound.toFixed(2)}
594                                 </div>
595                             </div>
596                             <div class="sentiment-item">
597                                 <div class="dominant-emotion">Домінуюча емоція:
${dominantEmotion}</div>
598                             </div>
599                             <div class="sentiment-item">
600                                 <div>Впевненість аналізу: ${confidence}</div>
601                             </div>
602                         `;
603                     }
604                 } else {
605                     throw new Error("Невідомий формат даних хмари слів");
606                 }
607
608                 showSuccess("Хмара слів створена успішно");
609             } catch (error) {
610                 showError("Помилка при створенні хмари слів", error);
611             }
612         });
613
614         // Обробка кнопки для створення хмари слів із відображених пісень
615
document.getElementById('generateWordcloudFromDisplayedSongs').addEventListener('cli
ck', async (e) => {
```

```
616             e.preventDefault();
617             e.stopPropagation();
618
619             try {
620                 if (!currentSongs || currentSongs.length === 0) {
621                     throw new Error("Пісні не знайдені");
622                 }
623
624                 console.log("Відправка пісень для генерації хмари слів:",
currentSongs);
625
626                 const response = await fetch('/generate-wordcloud-from-songs', {
627                     method: 'POST',
628                     headers: {
629                         'Content-Type': 'application/json',
630                     },
631                     body: JSON.stringify({ songs: currentSongs }),
632                 });
633
634                 console.log("API відповідь статус (хмара з пісень):",
response.status);
635
636                 if (!response.ok) {
637                     const errorData = await response.json();
638                     throw new Error(errorData.error || `Помилка HTTP:
${response.status}`);
639                 }
640
641                 const data = await response.json();
642                 console.log("Отримані дані хмари слів з пісень:", data);
643
644                 const wordcloudElement = document.getElementById('wordcloud');
645                 wordcloudElement.innerHTML = '';
646
647                 document.getElementById('sentimentDisplay').hidden = true;
648
649                 if (!data.wordcloud && Array.isArray(data)) {
650                     generateWordCloud(wordcloudElement, data);
651                 } else if (data.wordcloud) {
652                     generateWordCloud(wordcloudElement, data.wordcloud);
653
654                     const sentiment = data.sentiment;
655                     if (sentiment?.detailed_data &&
sentiment.detailed_data.length > 0) {
656                         // Збираємо статистику за емоціями
657                         const emotions = {};
```

```
658                         let totalCompound = 0;
659                         let validScores = 0;
660
661                         sentiment.detailed_data.forEach(songData => {
662                             if (songData.dominant_emotion) {
663                                 const emotion = songData.dominant_emotion;
664                                 emotions[emotion] = (emotions[emotion] || 0) + 1;
665                             }
666
667                             if (songData.vader_scores?.compound !== undefined) {
668                                 totalCompound += songData.vader_scores.compound;
669                                 validScores++;
670                             }
671                         });
672
673                         const averageCompound = validScores > 0 ? totalCompound /
validScores : 0;
674                         const mostCommonEmotion = Object.keys(emotions).length >
0 ?
675                             Object.keys(emotions).reduce((a, b) => emotions[a] >
emotions[b] ? a : b) : 'невідомо';
676
677                         const sentimentDisplay =
document.getElementById('sentimentDisplay');
678                         sentimentDisplay.hidden = false;
679
680                         // Створюємо список емоцій з кількістю
681                         const emotionStats = Object.entries(emotions)
682                             .sort(([,a], [,b]) => b - a)
683                             .map(([emotion, count]) =>
684                                 `<span class="emotion-item">${emotion}:
${count}</span>`
685                             ).join('');
686
687                         sentimentDisplay.innerHTML = `
688                             <div class="sentiment-item">
689                                 <div class="sentiment-score" style="color:
${averageCompound >= 0 ? '#28a745' : '#dc3545'}">
690                                     Середній настрій:
${averageCompound.toFixed(2)}
691                                 </div>
692                             </div>
693                             <div class="sentiment-item">
694                                 <div class="dominant-emotion">Домінуюча емоція:
${mostCommonEmotion}</div>
695                             </div>
```

```
                            <div class="sentiment-item">
                                <div>Кількість пісень:
${sentiment.detailed_data.length}</div>
                            </div>
                            ${emotionStats ? `<div class="emotion-
stats">${emotionStats}</div>` : ''}
                        `;
                    } else if (sentiment?.average_compound !== undefined) {
                        const score = sentiment.average_compound;
                        const sentimentDisplay =
document.getElementById('sentimentDisplay');
                        sentimentDisplay.hidden = false;
                        sentimentDisplay.innerHTML = `
                            <div class="sentiment-score" style="color: ${score >=
0 ? '#28a745' : '#dc3545'}">
                                Середній настрій пісень: ${score.toFixed(2)}
                            </div>
                        `;
                    }
                } else {
                    throw new Error("Невідомий формат даних хмари слів");
                }

                showSuccess("Хмара слів із пісень створена успішно");
            } catch (error) {
                showError("Помилка при створенні хмари слів із пісень", error);
            }
        });
    </script>
</body>
</html>
```

Рис. Б.2 – Код файлу index.html