МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ІНЖЕНЕРНИЙ НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ
ім. Ю.М. Потебні
ЗАПОРІЗЬКОГО НАЦІОНАЛЬНОГО УНІВЕРСИТЕТУ
КАФЕДРА ЕЛЕКТРОНІКИ, ІНФОРМАЦІЙНИХ СИСТЕМ ТА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Кваліфікаційна робота

перший (бакалаврський)
(рівень вищої освіти)

на тему **Розробка веб-застосунку для пошуку та прослуховування музики з функцією рекомендацій за настроєм користувача**

Виконав: студент 4 курсу, групи 6.1210-пзс
спеціальності 121 Інженерія програмного забезпечення
(код і назва спеціальності)
освітньої програми Програмне забезпечення систем
(код і назва освітньої програми)
О. Ю. Яндола
(ініціали та прізвище)
Керівник доцент кафедри ЕІС та ПЗ
А. І. Безверхий
(посада, вчене звання, науковий ступінь, підпис, ініціали та прізвище)
Рецензент директор ТОВ «Дискус»
Р. О. Лютий
(посада, вчене звання, науковий ступінь, підпис, ініціали та прізвище)

Запоріжжя

2024

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ІНЖЕНЕРНИЙ НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ
ім. Ю.М. Потебні
ЗАПОРІЗЬКОГО НАЦІОНАЛЬНОГО УНІВЕРСИТЕТУ

Кафедра Електроніки, інформаційних систем та програмного забезпечення
Рівень вищої освіти перший(бакалаврський)
Спеціальність 121 Інженерія програмного забезпечення
(код та назва)
Освітня програма Програмне забезпечення систем
(код та назва)

ЗАТВЕРДЖУЮ

Завідувач кафедри Т. В. Критська
“ 01 ” 2024 року

З А В Д А Н Н Я

НА КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТОВІ

Яндолі Олександру Юрійовичу
(прізвище, ім’я, по батькові)

1.Тема роботи Розробка веб-застосунку для пошуку та прослуховування музики з функцією рекомендацій за настроєм користувача
керівник роботи Безверхий Анатолій Ігорович, к.т.н., доцент
(прізвище ім’я, по батькові, науковий ступінь, вчене звання)
затверджені наказом ЗНУ від 29.12.2022 № 1893-с

2. Строк подання студентом кваліфікаційної роботи
3. Вихідні дані кваліфікаційної роботи бакалавра
- комплект нормативних документів ;
- технічне завдання до роботи.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)
- огляд та збір літератури стосовно теми кваліфікаційної роботи;
- огляд та аналіз існуючих рішень та аналогів;

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)
слайдів презентації

6. Консультанти розділів бакалаврської роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата Завдання прийняв

7. Дата видачі завдання 01.03.2024

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів бакалаврської роботи	Строк виконання етапів бакалаврської роботи	Примітка
1	Аналіз існуючих рішень	03.01 – 25.01.24	виконано
2	Постановка завдання	26.01 – 29.01.24	виконано
3	Огляд мов програмування	01.01 – 07.02.24	виконано
4	Аналіз сучасних фреймворків	08.02 – 22.02.24	виконано
5	Огляд архітектурних патернів	23.02 – 28.02.24	виконано
6	Огляд управлінь залежностями	01.03 – 07.03.24	виконано
7	Огляд СКБД	08.03 – 13.03.24	виконано
8	Огляд API	14.03 – 18.03.24	виконано
9	Розробка архітектури системи	19.03 – 26.03.24	виконано
10	Розробка фронтенд-частини	27.03 – 23.04.24	виконано
11	Розробка бекенд-частини	24.04 – 10.05.24	виконано
12	Тестування та оптимізація	11.05 – 29.05.24	виконано

Студент _____ Яндола О.Ю.
(підпис) (прізвище та ініціали)
Керівник роботи _____ Безверхий А.І.
(підпис) (прізвище та ініціали)

Нормоконтроль пройдено
Нормоконтролер _____ Скрипник І.А.
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Сторінок — 74

Рисунків — 20

Таблиць — 1

Джерел — 12

Яндола О. Ю. Розробка веб-застосунку для пошуку та прослуховування музики з функцією рекомендацій за настроєм користувача: кваліфікаційна робота бакалавра спеціальності 121 «Інженерія програмного забезпечення» / наук. керівник А. І. Безверхий. Запоріжжя : ЗНУ, 2024. 75 с.

У даній роботі здійснюється аналіз технологій JavaScript, React, а також можливостей Deezer API для розробки веб-застосунку для пошуку та прослуховування музики з функцією рекомендацій за настроєм користувача. Вивчаються основні функціональні можливості Deezer API для отримання інформації про музичні треки та їх метадані.

На основі здобутих знань і аналізу сучасних тенденцій у галузі веб-розробки розробляється архітектура веб-застосунку з використанням JavaScript та React. Програмна реалізація включає в себе функціонал для пошуку музичних треків, прослуховування музики та рекомендацій на основі настрою користувача, використовуючи алгоритми аналізу текстів пісень.

Проведено тестування працездатності створеного веб-застосунку. Виконано порівняння функціональності застосунку з наявними ринковими рішеннями.

Ключові слова: Deezer API, веб-застосунок, музичні рекомендації, аналіз настрою, JavaScript, React.

ABSTRACT

Pages — 74

Drawings — 20

Tables — 1

Source — 12

Yandola O.Yu. Development of a web application for searching and listening to music with the function of recommendations based on the user's mood: bachelor's thesis of the specialty 121 "Software engineering" / Science. manager A. I. Bezverkhy. Zaporizhzhia: ZNU, 2024. 75 p.

This work analyzes JavaScript, React technologies, as well as Deezer API capabilities for developing a web application for searching and listening to music with the function of recommendations based on the user's mood. Basic functionality of the Deezer API for obtaining information about music tracks and their metadata is studied.

Based on the acquired knowledge and analysis of modern trends in the field of web development, the architecture of a web application is developed using JavaScript and React. The software implementation includes functionality to search for music tracks, listen to music and make recommendations based on the user's mood using lyrics analysis algorithms.

Performance testing of the created web application was carried out. A comparison of the functionality of the application with existing market solutions was performed.

Keywords: Deezer API, web application, music recommendations, sentiment analysis, JavaScript, React.

ЗМІСТ

ВСТУП

Актуальність теми

У сучасному світі музика є невід'ємною частиною повсякденного життя мільйонів людей. З розвитком цифрових технологій і інтернету, доступ до музики став простішим та зручнішим. Стримінгові сервіси, такі як Spotify, Apple Music, Deezer та інші, дозволяють користувачам слухати музику будь-де і будь-коли, створювати власні плейлисти та отримувати персоналізовані рекомендації. Проте, зростаюча кількість музичного контенту вимагає ефективних методів його організації та рекомендацій.

Актуальність даного дослідження полягає в розробці веб-додатку, який дозволяє користувачам не лише шукати і слухати музику, але й отримувати рекомендації на основі настрою. Такий підхід до рекомендацій може значно покращити користувацький досвід, забезпечуючи більш персоналізований і емоційно резонуючий музичний контент.

Інтеграція з різними музичними API, такими як Deezer API для отримання музичного контенту та Genius API для отримання текстів пісень, дозволяє реалізувати розширені функціональні можливості веб-додатку. Використання бібліотеки для аналізу настрою текстів пісень (Sentiment) забезпечує можливість автоматичного визначення емоційного тону треків, що є важливим кроком для реалізації персоналізованих рекомендацій.

Окрім того, розробка даного веб-додатку здійснюється на основі мови програмування JavaScript, яка є однією з найпопулярніших мов для веб-розробки. JavaScript забезпечує можливість створення динамічних та інтерактивних користувацьких інтерфейсів, а також ефективну взаємодію з різними веб-сервісами та API. Використання сучасних бібліотек і фреймворків, таких як React для фронтенд-розробки, дозволяє створювати швидкі та зручні у користуванні веб-додатки.

Таким чином, тема дипломної роботи є актуальною з точки зору розвитку сучасних музичних веб-додатків та вдосконалення методів персоналізованих рекомендацій, що забезпечують більш глибокий і емоційно-задовольняючий досвід для користувачів. Вибір JavaScript як основної мови програмування підкреслює прагнення до використання передових технологій для досягнення високої продуктивності та зручності використання веб-додатку.

Мета дослідження

Метою дослідження є розробка веб-застосунку для пошуку та прослуховування музики, що надаватиме користувачам унікальний досвід завдяки функції рекомендацій за настроєм.

Завдання дослідження

У рамках дослідження для розробки веб-застосунку для пошуку та прослуховування музики з функцією рекомендацій за настроєм користувача визначено наступні завдання:

1. Аналіз існуючих музичних стримінгових сервісів та технологій:
   - Дослідити функціональні можливості популярних музичних стримінгових сервісів, таких як Spotify, Apple Music, Deezer тощо.
   - Оцінити існуючі підходи до персоналізації та рекомендацій музичного контенту.
2. Вибір та інтеграція API для отримання музичного контенту та текстів пісень:
   - Обрати відповідні музичні API для отримання даних про треки, альбоми та виконавців.
   - Інтегрувати API для отримання текстів пісень для подальшого аналізу настрою.
3. Розробка та реалізація алгоритму аналізу настрою текстів пісень:

- Використати бібліотеки для аналізу текстів пісень.
- Створити механізм визначення емоційного тону треків на основі текстів пісень.

4. Розробка фронтенд-частини веб-додатку:
    - Використати JavaScript та React для створення динамічного та інтерактивного інтерфейсу користувача.
    - Забезпечити зручний інтерфейс для пошуку музики, прослуховування треків та отримання рекомендацій на основі настрою.
5. Реалізація системи користувацької автентифікації та профілю:
    - Інтегрувати Firebase для автентифікації користувачів та зберігання їхніх даних.
    - Забезпечити можливість реєстрації, входу в систему, а також управління користувацькими профілями.
6. Тестування та оптимізація веб-додатку:
    - Провести тестування функціональних можливостей веб-додатку для забезпечення стабільної роботи.
    - Оптимізувати продуктивність додатку для швидкої та плавної взаємодії з користувачами..

Виконання зазначених завдань дозволить створити сучасний, функціональний та зручний у користуванні веб-додаток для прослуховування музики та отримання рекомендацій на основі настрою, що забезпечить унікальний користувацький досвід.

Об’єкт дослідження

Об’єктом дослідження є процес розробки та функціонування веб-застосунку для пошуку та прослуховування музики, який інтегровано з Deezer API та оснащено системою рекомендацій музики на основі настрою користувача.

Предмет дослідження

Предметом дослідження даного дипломного проекту є процес розробки веб-додатку для музичних сервісів, зокрема дослідження методів інтеграції сторонніх API, таких як Deezer API, для забезпечення функціональності пошуку та прослуховування музики, а також використання Firebase для автентифікації користувачів та зберігання даних. Особлива увага приділяється розробці механізмів рекомендації музики на основі настрою користувача, використовуючи аналіз текстів пісень.

У рамках дослідження розглядаються наступні аспекти:

- Вивчення існуючих методів інтеграції сторонніх API для музичних сервісів. Аналіз переваг та недоліків використання різних API, оцінка їх функціональних можливостей та обмежень.
- Аналіз способів забезпечення безпеки та збереження даних користувачів за допомогою Firebase. Оцінка ефективності різних методів автентифікації та збереження даних, розгляд аспектів захисту персональних даних користувачів.
- Дослідження алгоритмів аналізу текстів пісень для визначення настрою та рекомендації музики. Вивчення існуючих підходів до аналізу текстів, розгляд можливих методів класифікації музики за настроєм, оцінка точності та ефективності різних алгоритмів.

Основною метою даного дослідження є розробка ефективного та зручного веб-додатку, який дозволяє користувачам не лише шукати та прослуховувати музику, але й отримувати персоналізовані рекомендації на основі їх настрою.

Методи дослідження

Для досягнення мети дипломного проекту були використані наступні методи дослідження:.

1. Аналіз літератури та існуючих рішень:
    - Вивчення наукових статей, технічної документації та інших джерел, присвячених розробці музичних стримінгових сервісів, інтеграції сторонніх API та аналізу текстів пісень.
    - Аналіз існуючих веб-додатків для прослуховування музики, їх архітектури та функціональних можливостей.
2. Емпіричний аналіз:
    - Практичне тестування та порівняння різних API для визначення найбільш придатного для розробки додатку.
    - Емпіричне дослідження методів автентифікації користувачів та забезпечення безпеки даних з використанням Firebase.
3. Проектування та розробка:
    - Створення архітектури веб-додатку з використанням сучасних фреймворків та бібліотек, таких як React для фронтенд-розробки та Firebase для управління користувацькими даними.
    - Інтеграція Deezer API для отримання інформації про музику та Genius API для отримання текстів пісень.
4. Аналіз даних та алгоритмізація:
    - Використання бібліотеки Sentiment для аналізу текстів пісень та визначення їх настрою.
    - Розробка алгоритмів персоналізованих рекомендацій музики на основі настрою користувача.
5. Тестування та оптимізація:
    - Функціональне тестування розробленого веб-додатку для перевірки його стабільності та зручності використання.
    - Оптимізація продуктивності додатку для забезпечення швидкої та плавної роботи.
6. Документування:

- Створення технічної документації для опису процесу розробки, архітектури та функціональності веб-додатку.
- Підготовка користувацьких інструкцій для кінцевих користувачів додатку.

Ці методи дослідження забезпечили комплексний підхід до розробки веб-додатку, що дозволяє інтегрувати музичні сервіси, аналізувати тексти пісень для визначення настрою та надавати персоналізовані музичні рекомендації користувачам.

Практичне значення одержаних результатів

Практичне значення одержаних результатів дослідження полягає в створенні зручного та інтуїтивно зрозумілого веб-застосунку, який забезпечує користувачам унікальний досвід пошуку та прослуховування музики, адаптований під їхній настрій. Це сприяє підвищенню задоволеності від використання, дозволяє відкривати нову музику, що відповідає індивідуальним перевагам, та зміцнює емоційний зв'язок із музикою. Розробка такого застосунку не лише відповідає сучасним тенденціям персоналізації сервісів але й має значний потенціал для впливу на музичну індустрію, зокрема, забезпечуючи артистам нові можливості для просування їх творчості.

Глосарій

Веб-застосунок (Web Application) — програмне забезпечення, яке користувачі можуть використовувати за допомогою веб-браузера через Інтернет або інтранет.

JavaScript — мова програмування, що широко використовується для створення інтерактивних ефектів у веб-браузерах.

React — відкрита JavaScript бібліотека для побудови користувацьких інтерфейсів, що дозволяє розробникам створювати великі веб-додатки, які можуть змінювати дані без перезавантаження сторінки.

Користувацький інтерфейс (UI) — Інтерфейс, через який користувач взаємодіє з додатком або системою, включаючи елементи управління, відображення інформації тощо.

Sentiment Analysis (Аналіз настрою) — Метод обробки природної мови, що використовується для визначення емоційного тону тексту, зокрема, чи є текст позитивним, негативним або нейтральним.

Deezer API — інтерфейс програмування додатків, наданий музичним сервісом Deezer, що дозволяє розробникам інтегрувати музичний контент і функції Deezer у власні додатки або веб-сайти.

Genius API — API, що дозволяє отримувати тексти пісень з бази даних Genius.

Персоналізація — процес налаштування досвіду користувача на веб-сайті або в додатку, з метою зробити його більш відповідним до індивідуальних потреб або інтересів користувача.

Рекомендаційна система — система, яка надає користувачам персоналізовані пропозиції (наприклад, музичні треки, фільми, книги тощо) на основі їхніх попередніх переваг, поведінки або взаємодій.

Інтерфейс користувача (User Interface - UI) — частина програмного забезпечення, через яку користувач взаємодіє з програмою або машиною, включаючи елементи дизайну, екрани, сторінки та візуальні елементи.

Firebase — Платформа для розробників від Google, що пропонує різні сервіси, такі як база даних у реальному часі, автентифікація, хостинг та інші інструменти для створення веб- та мобільних додатків.

API (Application Programming Interface) — набір правил, протоколів і інструментів для побудови програмного забезпечення та додатків, який дозволяє різним програмним продуктам або компонентам взаємодіяти між собою.

РОЗДІЛ 1 АНАЛІЗ ІСНУЮЧИХ РШЕНЬ

1.1 Аналіз програмних продуктів і аналогів

На ринку мобільних додатків існує значна кількість застосунків для пошуку та прослуховування музики, а також рекомендаційних сервісів. Деякі з найпопулярніших та функціональних додатків включають Spotify, Apple Music, Deezer, і Tidal. Розглянемо їх детальніше.

Spotify — один з найпопулярніших музичних сервісів у світі. Він надає користувачам доступ до мільйонів треків, подкастів та плейлистів.

Інтерфейс сервісу можна побачити на рисунку 1

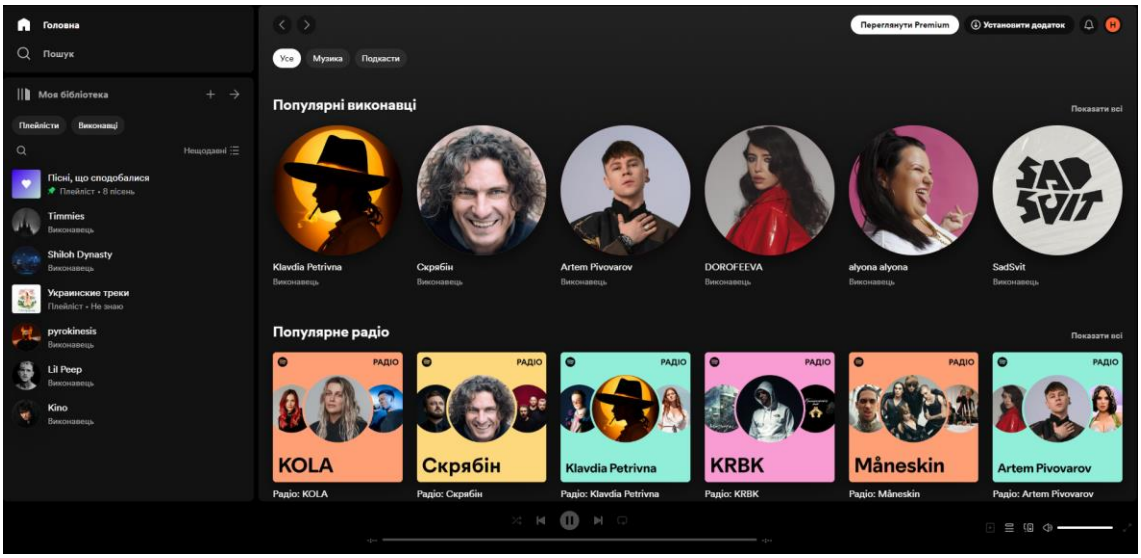

Рисунок 1 – Інтерфейс Spotify

Переваги:

- Велика музична бібліотека: Доступ до мільйонів треків.
- Рекомендації: Персоналізовані рекомендації на основі музичних уподобань користувача.

- Соціальні функції: Можливість ділитися плейлистами, переглядати музичні вподобання друзів та слідкувати за улюбленими виконавцями.
- Офлайн режим: Можливість завантажувати музику для прослуховування без Інтернету.

Недоліки:

- Платна підписка: Обмежений функціонал у безкоштовній версії, наявність реклами.
- Відсутність високоякісного звуку: Якість звуку нижча порівняно з деякими конкурентами, такими як Tidal.

Apple Music — музичний сервіс від Apple, який також пропонує доступ до мільйонів пісень та плейлистів.

Інтерфейс сервісу можна побачити на рисунку 2

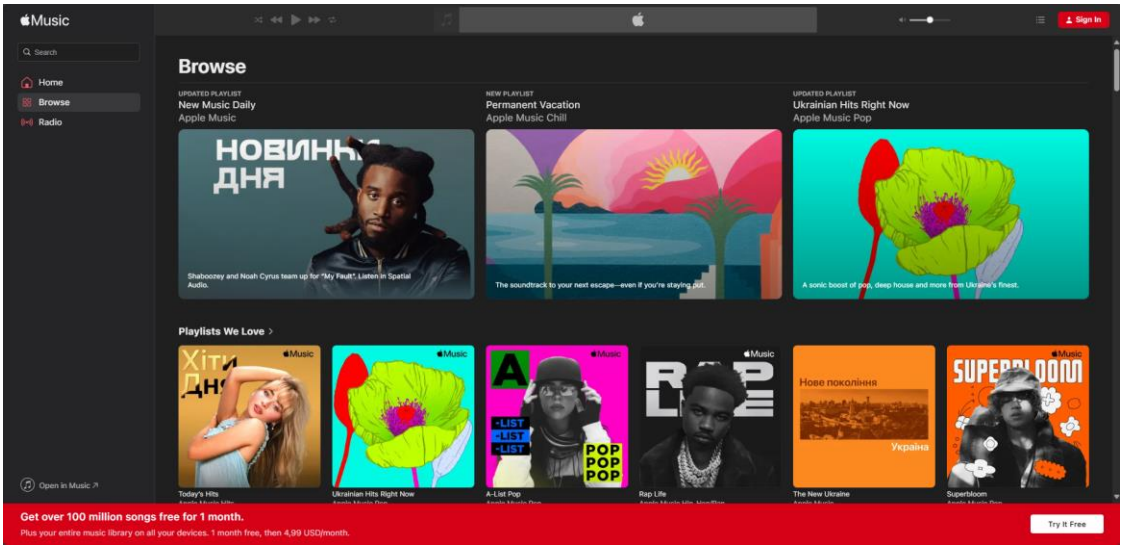

Рисунок 2 – Інтерфейс Apple Music

Переваги:

- Інтеграція з iTunes: Синхронізація музичної бібліотеки з iTunes.

- Ексклюзивний контент: Доступ до ексклюзивних релізів та треків від виконавців.
- Рекомендації: Персоналізовані рекомендації на основі музичних уподобань користувачів.

Недоліки:

- Платна підписка: Висока вартість підписки.
- Інтерфейс: Інтерфейс може бути менш інтуїтивним для нових користувачів, особливо тих, хто не користується іншими продуктами Apple.

Deezer — ще один популярний музичний сервіс, який пропонує широкий спектр музики та подкастів.

Інтерфейс сервісу можна побачити на рисунку 3

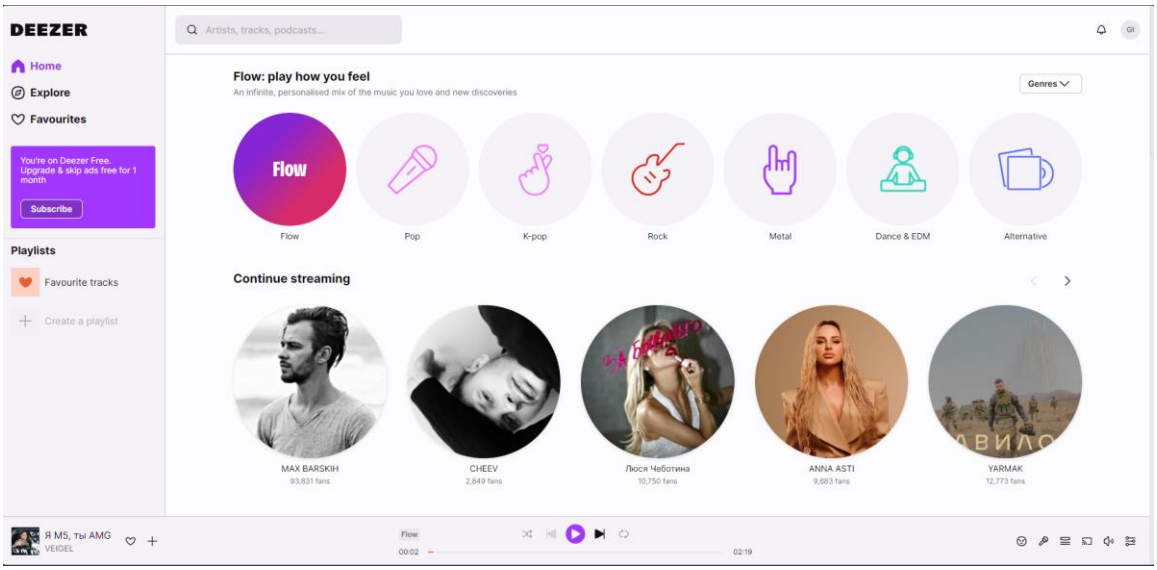

Рисунок 3 – Інтерфейс Deezer API

Переваги:

- Flow: Персоналізований плейлист, який оновлюється на основі музичних уподобань користувача.

- Широка музична бібліотека: Велика кількість треків та подкастів.
- Кастомізація плейлистів: Можливість створення та налаштування власних плейлистів.

Недоліки:

- Платна підписка: Обмежений функціонал у безкоштовній версії, наявність реклами.
- Інтерфейс: Інтерфейс може бути менш привабливим порівняно з іншими сервісами.

Таблиця 1 – *Порівняння характеристик Spotify, Apple Music та Deezer*

Властивість	**Spotify**	**Apple Music**	**Deezer**
Музична бібліотека	Велика (70+ мільйонів треків)	Велика (75+ мільйонів треків)	Велика (73+ мільйони треків)
Рекомендації	Персоналізовані	Персоналізовані	Персоналізовані (Flow)
Соціальні функції	Є (обмін плейлистами, друзі)	Обмежені (ділитися треками)	Є (обмін плейлистами, друзі)
Офлайн режим	Є	Є	Є
Якість звуку	Стандартна	Стандартна	Висока (Hi-Fi у платній версії)
Ексклюзивний контент	Мало	Багато	Мало
Інтеграція	Інтеграція з Facebook та ін.	Глибока інтеграція в екосистему Apple	Інтеграція з соціальними мережами
Платформа	iOS, Android, Windows, macOS, Web	iOS, macOS, Android, Windows	iOS, Android, Windows, macOS, Web
Безкоштовна версія	Є (з рекламою)	Немає	Є (з рекламою)
Платна підписка	$9.99/місяць	$9.99/місяць	$9.99/місяць

Функції платної підписки	Відсутність реклами, офлайн режим	Відсутність реклами, офлайн режим	Відсутність реклами, офлайн режим, Hi-Fi якість звуку
Інтерфейс	Інтуїтивний	Менш інтуїтивний для нових користувачів	Менш привабливий
Наявність подкастів	Є	Є	Є
Підтримка мов	Багато	Багато	Багато

Аналізуючи вищенаведені сервіси «див. Табл. 1», можна зробити висновок, що мій веб-застосунок є доволі унікальним із за можливості надавати музичні рекомендації за настроєм користувача. Основний акцент при розробці веб-застосунку буде зроблено на персоналізованих рекомендаціях та інтеграції з зовнішніми сервісами для аналізу настрою музичних треків.

1.2 Постановка завдання

Метою даної дипломної роботи є розробка веб-застосунку для прослуховування музики з унікальними можливостями, які відрізняють його від існуючих аналогів. Застосунок має забезпечити користувачам комфортний і персоналізований досвід прослуховування музики.

Основний функціонал застосунку:

1. Авторизація та аутентифікація:
    - Реєстрація користувачів через електронну пошту та пароль.
    - Вхід через соціальні мережі (Google).
    - Відновлення паролю через електронну пошту.
2. Профіль користувача:
    - Редагування особистих даних (ім'я).
    - Зміна аватару користувача.
3. Пошук музики:

- Пошук треків, альбомів, виконавців.

4. Прослуховування музики:
    - Вбудований аудіоплеєр з можливістю перемикання треків, зміни гучності, паузи та відтворення.
    - Підтримка режиму повтору.
5. Рекомендації:.
    - Можливість вибору музики за настроєм користувача. Користувач може обрати один із запропонованих настроїв, і застосунок запропонує відповідні треки.
6. Інтерактивний інтерфейс:
    - Зручний дизайн інтерфейсу користувача.

Унікальний функціонал:

1. Музичні рекомендації за настроєм:
    - Застосунок використовує алгоритми аналізу текстів пісень для визначення їх настрою. За допомогою API Genius отримуються тексти пісень, які аналізуються для визначення настрою (веселий, сумний, енергійний тощо). Це дозволяє пропонувати користувачам плейлисти, які відповідають їх поточному настрою.

Таким чином, застосунок надає користувачам новий досвід завдяки унікальному функціоналу, спрямованому на покращення взаємодії з музикою відповідно до їхнього настрою та вподобань.

РОЗДІЛ 2 ДОСЛІДЖЕННЯ ЗАСОБІВ ПРОГРАМНОЇ РЕАЛІЗАЦІЇ ВЕБ-ЗАСТОСУНКУ

2.1 Огляд мов програмування

У цьому розділі буде проведено огляд основних мов програмування, які використовуються для розробки сучасних веб-застосунків. Тут будуть розглянуті найпоширеніші мови програмування, їхні переваги, недоліки та сфери застосування.

JavaScript — одна з найпоширеніших мов програмування для веб-розробки. Вона широко використовується як на стороні клієнта, так і на стороні сервера.

Сфера застосування:

- Основна мова для розробки інтерактивних веб-сторінок. Завдяки Node.js, JavaScript також використовується для серверного програмування.

Переваги:

- Широке розповсюдження та підтримка.
- Висока швидкість виконання завдяки V8 Engine.
- Велика кількість бібліотек і фреймворків (React, Angular, Vue.js).
- Підтримка асинхронного програмування.

Недоліки:

- Обмеженість у типах даних (динамічна типізація).
- Проблеми з безпекою через XSS (Cross-Site Scripting) атаки.

Python — це високоурівнева мова програмування, яка відома своєю читабельністю та простотою у використанні.

Сфера застосування:

- Розробка серверної частини веб-застосунків, наукові обчислення, машинне навчання.

Переваги:

- Простота синтаксису, що полегшує навчання та використання.
- Величезна стандартна бібліотека та великий вибір сторонніх бібліотек.
- Підтримка паралельного програмування.

Недоліки:

- Відносно повільне виконання у порівнянні з JavaScript та C++.
- Використання великої кількості пам'яті.

Java — це мова програмування загального призначення з сильним типізованим синтаксисом, яка широко використовується для створення корпоративних додатків.

Сфера застосування:

- Розробка великих корпоративних систем, мобільних додатків (Android), веб-сервісів.

Переваги:

- Платформонезалежність завдяки Java Virtual Machine (JVM).
- Висока продуктивність.
- Добре структурована та підтримує об'єктно-орієнтоване програмування.

Недоліки:

- Більш складний синтаксис у порівнянні з Python.
- Вимагає більше пам'яті.

PHP — це серверна мова програмування, яка спеціально створена для розробки веб-додатків.

Сфера застосування:

- Розробка серверної частини веб-додатків.

Переваги:

- Відмінна інтеграція з HTML.
- Широке використання у веб-розробці, велика кількість готових рішень.
- Простота у використанні та швидке навчання.

Недоліки:

- Відсутність підтримки багатопоточності.
- Існує думка про нижчий рівень безпеки у порівнянні з іншими мовами.

Вибір мови програмування залежить від проекту та вимог до нього. Для розробки інтерактивних веб-додатків на стороні клієнта найкраще підходить JavaScript, завдяки своїм фреймворкам та бібліотекам. На стороні сервера я вирішив використовувати Node.js тому, що використання JavaScript як на фронтенді так і на бекенді спрощує розробку та підтримку коду .

2.2 Аналіз сучасних фреймворків для створення веб-застосунків

Для розробки сучасного веб-застосунку для прослуховування музики важливо вибрати правильний фреймворк, який забезпечить високу продуктивність, зручність розробки та підтримку сучасних технологій. Нижче наведено аналіз основних фреймворків, які можуть бути використані для реалізації проекту.

React — це бібліотека JavaScript, розроблена компанією Facebook, яка використовується для створення користувацьких інтерфейсів. React забезпечує високий рівень продуктивності і зручності розробки завдяки компонентному підходу. Основні переваги React:

- Компонентний підхід: React дозволяє створювати повторно використовувані компоненти, що спрощує розробку і підтримку великомасштабних застосунків.
- Віртуальний DOM: Використання віртуального DOM підвищує продуктивність за рахунок мінімізації змін у реальному DOM.
- Велика спільнота і екосистема: React має велику спільноту розробників та широкий вибір бібліотек і інструментів, таких як Redux для управління станом та React Router для маршрутизації

Angular — це фреймворк для розробки веб-застосунків, створений компанією Google. Angular використовує TypeScript як основну мову програмування і надає потужні інструменти для розробки складних додатків. Основні переваги Angular:

- Всеосяжний фреймворк: Angular надає все необхідне для розробки веб-застосунків, включаючи двостороннє зв'язування даних, залежності і шаблони.
- TypeScript: Використання TypeScript забезпечує статичну типізацію, що покращує якість коду та полегшує його підтримку.
- Потужна CLI: Angular CLI (Command Line Interface) спрощує процес створення, тестування та розгортання додатків.

Vue.js — це прогресивний фреймворк JavaScript для створення користувацьких інтерфейсів. Vue поєднує переваги React і Angular, забезпечуючи простоту використання та високу продуктивність. Основні переваги Vue.js:

- Простота і гнучкість: Vue.js простий у вивченні та використанні, що робить його привабливим для новачків і досвідчених розробників.
- Компонентна структура: Як і React, Vue дозволяє створювати компоненти, що повторно використовуються, забезпечуючи модульність та зручність підтримки коду.
- Реактивність: Vue.js забезпечує автоматичне оновлення інтерфейсу користувача при зміні даних завдяки реактивній системі.

Svelte — це сучасний фреймворк для створення користувацьких інтерфейсів, який відрізняється від традиційних фреймворків тим, що більша частина роботи виконується під час компіляції. Основні переваги Svelte:

- Відсутність віртуального DOM: Svelte генерує високопродуктивний код, який безпосередньо маніпулює DOM, що підвищує продуктивність.
- Простота коду: Svelte дозволяє писати менше коду завдяки вбудованій реактивності та декларативному синтаксису.

- Малі розміри бандлів: Завдяки компіляції більша частина роботи виконується під час зборки, що зменшує розмір кінцевого бандлу.

Вибір фреймворку залежить від вимог проекту та особистих уподобань розробників. У своєму проекті я вирішив використовувати React у поєднанні з іншими сучасними технологіями для забезпечення високої продуктивності, зручності розробки і масштабованості застосунку.

2.3 Огляд архітектурних патернів

Архітектурні паттерни відіграють важливу роль у розробці програмного забезпечення, оскільки вони визначають структуру системи, спрощують її підтримку, масштабування та забезпечують гнучкість. У цьому розділі ми розглянемо найпоширеніші архітектурні паттерни, які використовуються в розробці веб-застосунків.

Модель-Вид-Контролер (MVC) — це архітектурний паттерн, який розділяє додаток на три взаємозалежні компоненти: Модель, Вид і Контролер.

- Модель: Відповідає за управління даними, бізнес-логікою і правилами додатка.
- Вид: Відповідає за відображення інформації користувачу.
- Контролер: Відповідає за прийом вхідних даних, обробку запитів і передачу даних між Моделлю та Видом.

Переваги:

- Чітке розділення обов'язків.
- Покращена модульність і тестування.

Недоліки:

- Може бути складним для розуміння новачками.
- Може призвести до дублювання коду при неправильному використанні.

Схему архітектури можна побачити на рисунку 4.

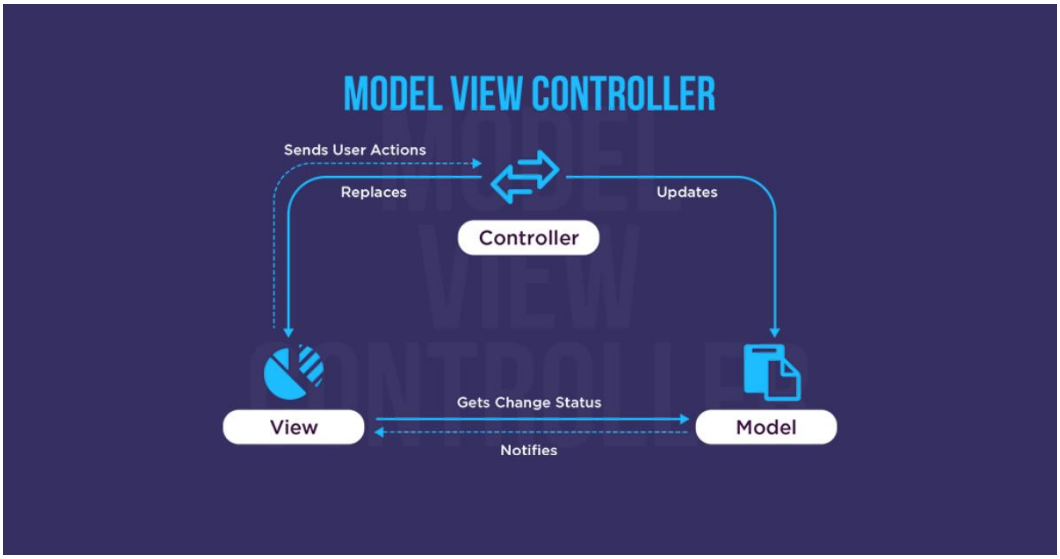

Рисунок 4 – Схема архітектури MVC

Модель-Вид-Відображення (MVP) — це варіація MVC, де Контролер замінюється Презентером.

- Модель: Відповідає за управління даними та бізнес-логікою.
- Вид: Відповідає за відображення даних користувачу.
- Презентер: Діє як посередник між Моделлю і Видом, обробляє логіку відображення.

Переваги:

- Легше тестувати логіку відображення.
- Чітке розділення обов'язків.

Недоліки:

- Може бути складним для розуміння і впровадження.
- Більше коду в порівнянні з іншими паттернами.

Схему архітектури можна побачити на рисунку 5:

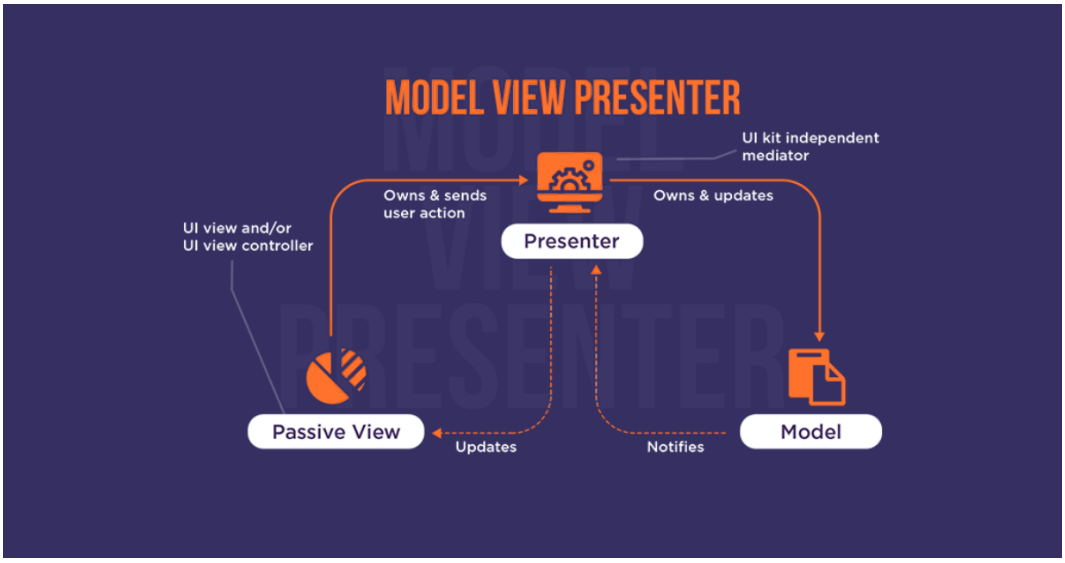

Рисунок 5 – Схема архітектури MVP

Модель-Вид-В'юМодель (MVVM) — це паттерн, який часто використовується в розробці односторінкових додатків.

- Модель: Відповідає за управління даними та бізнес-логікою.
- Вид: Відповідає за відображення даних користувачу.
- В'юМодель: Відповідає за перетворення даних Моделі у формат, придатний для Виду, і обробляє взаємодію користувача.

Переваги:

- Легке двостороннє зв'язування даних.
- Покращена модульність і тестування.

Недоліки:

- Може бути складним для реалізації.
- Вимагає більше ресурсів на підтримку двостороннього зв'язування.

Схему архітектури можна побачити на рисунку 6

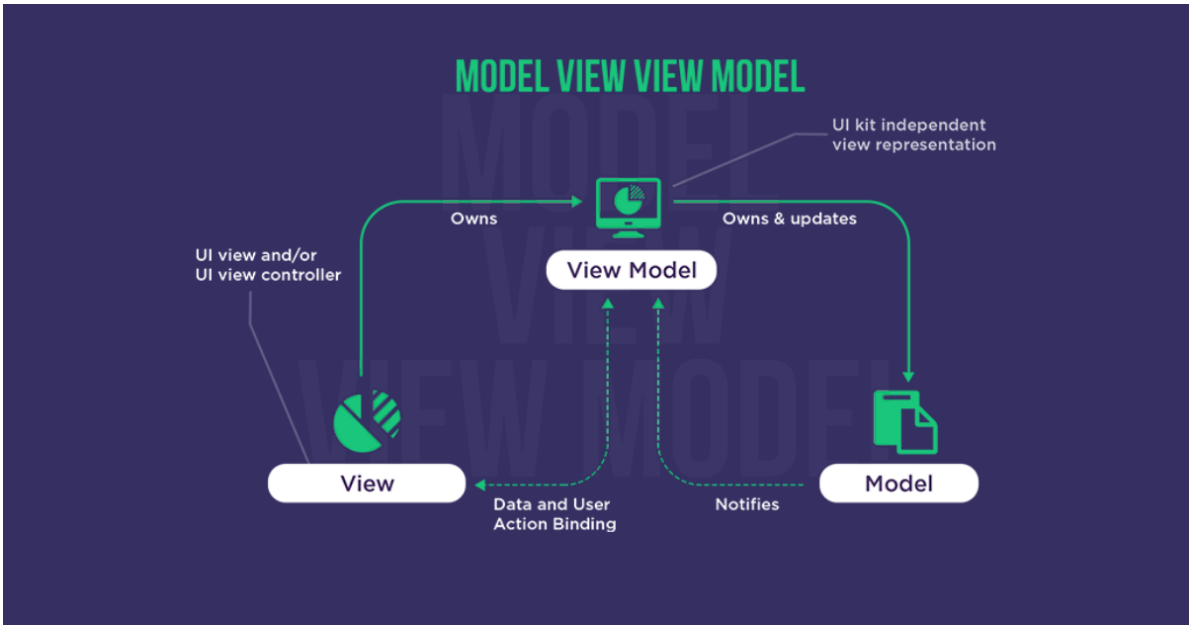

Рисунок 6 – Схема архітектури MVVM

Односторінковий додаток (SPA) — це архітектурний підхід, при якому додаток складається з однієї HTML-сторінки, що динамічно оновлюється в залежності від дій користувача.

Переваги:

- Швидше взаємодія з користувачем, оскільки не потрібно завантажувати нові сторінки.
- Менше навантаження на сервер.

Недоліки:

- Може вимагати більше часу для першого завантаження.
- Складність реалізації SEO (пошукової оптимізації).

Схему архітектури можна побачити на рисунку 7

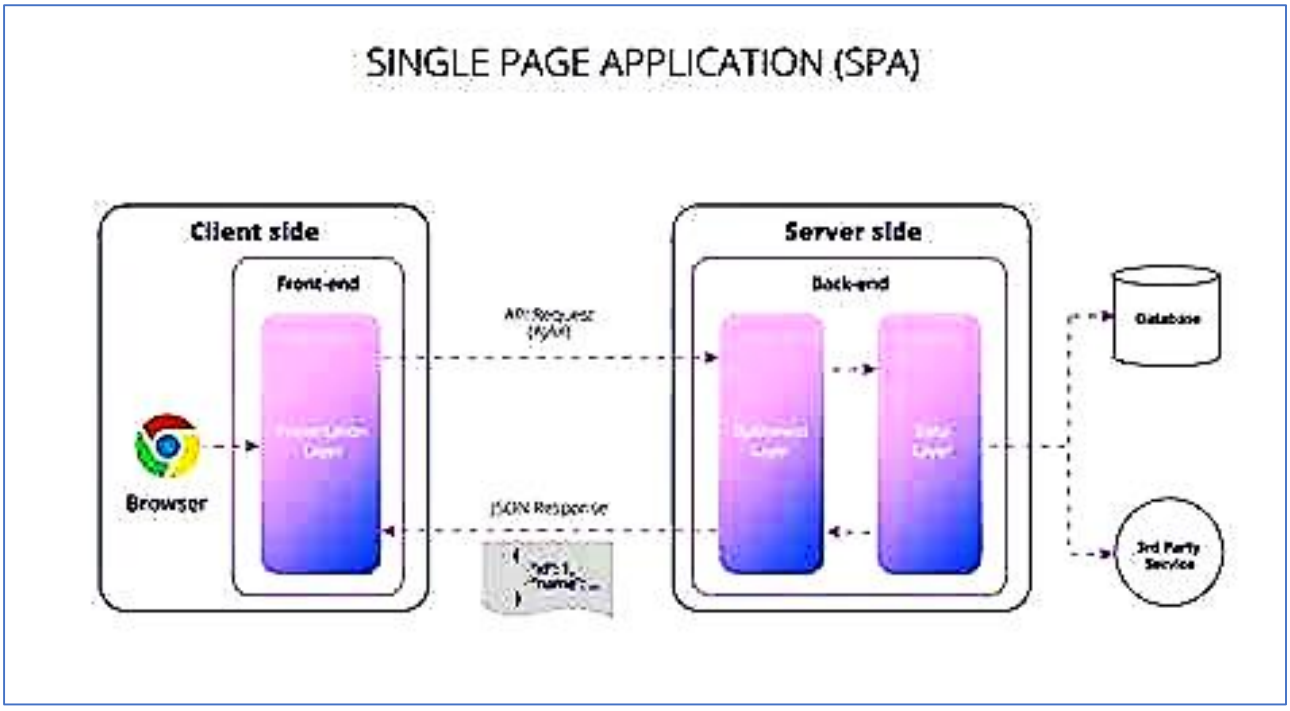

Рисунок 7 – Схема архітектури SPA

Компонентний підхід (Component-based Architecture) — це архітектурний паттерн, широко використовуваний в розробці сучасних веб-додатків. Він полягає в розділенні інтерфейсу користувача на незалежні, багаторазові компоненти, кожен з яких інкапсулює свою власну логіку і стан.

Переваги:

- Висока модульність.
- Полегшення тестування і підтримки коду.
- Можливість повторного використання компонентів у різних частинах програми.

Приклад:

- Використання компонентів в React для побудови інтерфейсу користувача.

Я використовую архітектурний патерн Модель-Вид-Контролер (MVC) у своєму веб-застосунку, який забезпечує чітке розділення обов'язків між різними частинами вашого додатка, що спрощує його підтримку та масштабування.

2.4 Огляд управлінь залежностями

Управління залежностями є важливою частиною розробки сучасних веб-застосунків. Це дозволяє розробникам легко підключати та керувати зовнішніми бібліотеками та фреймворками, які використовуються у проекті. У даному пункті буде розглянуто найбільш популярні засоби управління залежностями, їх переваги та недоліки, а також конкретні приклади використання.

npm (Node Package Manager) — найбільш популярний менеджер пакетів для JavaScript і використовується разом з Node.js. Він дозволяє розробникам встановлювати, оновлювати та видаляти бібліотеки та інші пакети, а також керувати їх версіями.

Переваги npm:

- Велика екосистема пакетів: npm має найбільший репозиторій пакетів, що дозволяє легко знаходити та використовувати необхідні бібліотеки.
- Простота використання: npm має простий у використанні інтерфейс командного рядка.
- Управління версіями: дозволяє точно вказувати версії пакетів, що запобігає конфліктам та збоїв.

Недоліки npm:

- Повільне встановлення: у деяких випадках встановлення пакетів може бути повільним.
- Великі розміри пакетів: деякі пакети можуть мати багато залежностей, що збільшує розмір проекту.

Приклад використання npm можна побачити на рисунку 8:

```
# Ініціалізація нового проекту з використанням npm
npm init -y

# Встановлення React та ReactDOM
npm install react react-dom
```

Рисунок 8 – Приклад використання npm

Yarn — альтернативний менеджер пакетів для JavaScript, створений компанією Facebook. Він був розроблений для вирішення деяких недоліків npm, таких як швидкість та безпека.

Переваги Yarn:

- Швидкість: Yarn використовує паралельне завантаження, що робить встановлення пакетів швидшим.
- Безпека: зберігає контрольні суми для перевірки цілісності пакетів.
- Детерміновані інсталяції: гарантує, що однакові версії пакетів встановлюються на всіх машинах.

Недоліки Yarn:

- Складність: може бути складнішим у використанні для новачків.
- Сумісність: деякі специфічні функції npm можуть не підтримуватися.

Приклад використання Yarn можна побачити на рисунку 9

```
# Ініціалізація нового проекту з використанням Yarn
yarn init -y

# Встановлення React та ReactDOM
yarn add react react-dom
```

Рисунок 9 – Приклад використання Yarn

Bower — один із перших менеджерів пакетів для фронтенд-розробки. Він орієнтований на керування компонентами, які працюють у веб-браузерах.

Переваги Bower:

- Простота: легкий у використанні для простих проектів.
- Фронтенд орієнтація: оптимізований для роботи з фронтенд-бібліотеками.

Недоліки Bower:

- Скорочення популярності: більшість сучасних проектів перейшли на npm або Yarn.
- Обмежений функціонал: не підтримує деякі функції, які доступні в npm або Yarn.

Приклад використання Bower можна побачити на рисунку 10

```
# Ініціалізація нового проекту з використанням Bower
bower init

# Встановлення jQuery
bower install jquery --save
```

Рисунок 10 – Приклад використання Bower

Управління залежностями є невід'ємною частиною розробки веб-застосунків. npm, Yarn та Bower є найбільш популярними інструментами для цієї мети, кожен з яких має свої переваги та недоліки. У своєму проекті я вирішив використовувати npm через його популярність, широку екосистему пакетів та інтеграцію з Node.js.

2.5 Огляд СКБД

Система керування базами даних (СКБД) є ключовим компонентом будь-якого веб-застосунку, який потребує зберігання, управління та доступу до великих обсягів даних. Вибір СКБД визначається вимогами проекту, включаючи масштабованість, продуктивність, безпеку та зручність використання. У цьому розділі розглянемо деякі з найбільш популярних СКБД, їхні особливості, переваги та недоліки.

Relational Database Management Systems (RDBMS) — використовують реляційну модель даних, де дані організовані у вигляді таблиць, що складаються з рядків і стовпців. RDBMS забезпечують потужні засоби для маніпуляції даними, включаючи SQL (Structured Query Language) для створення, читання, оновлення та видалення даних.

1. MySQL є одним із найпопулярніших RDBMS з відкритим кодом. Він широко використовується для розробки веб-застосунків завдяки своїй надійності, продуктивності та простоті використання.

   Переваги MySQL:

   - Простота використання: Легко встановлювати та налаштовувати.
   - Відмінна продуктивність: Підходить для обробки великих обсягів даних.
   - Підтримка спільноти: Велика кількість документації та активна спільнота користувачів.

   Недоліки MySQL:

   - Обмежена масштабованість: Може виникати потреба в додаткових рішеннях для масштабування на великих проектах.
   - Обмеженість функціоналу: Порівняно з іншими RDBMS деякі функції можуть бути обмежені.
2. PostgreSQL є потужною системою управління базами даних з відкритим кодом, відомою своєю надійністю та багатофункціональністю.

PostgreSQL підтримує розширені можливості, такі як зберігання великих обсягів даних та обробка складних запитів.

Переваги PostgreSQL:

- Розширені можливості: Підтримка складних запитів, транзакцій та зберігання великих обсягів даних.
- Масштабованість: Добре підходить для великих проектів.
- Безпека: Підтримка розширених механізмів безпеки.

Недоліки PostgreSQL:

- Складність налаштування: Може вимагати більше часу для налаштування та управління у порівнянні з іншими СКБД.
- Швидкість: В деяких випадках може бути повільнішою за MySQL при виконанні простих запитів.

NoSQL бази даних відрізняються від реляційних баз даних тим, що не використовують таблиці з фіксованою схемою. Вони забезпечують гнучкість у зберіганні та обробці великих обсягів різнорідних даних. NoSQL бази даних поділяються на кілька категорій: документні, графові, ключ-значення та колоночні.

1. MongoDB є однією з найпопулярніших документних NoSQL баз даних. Вона зберігає дані у вигляді JSON-подібних документів, що дозволяє легко масштабувати та обробляти великі обсяги даних.

   Переваги MongoDB:

   - Гнучкість: Не вимагає фіксованої схеми, що дозволяє зберігати різнорідні дані.
   - Масштабованість: Легко масштабується горизонтально.
   - Швидкість: Висока продуктивність при обробці великих обсягів даних.

   Недоліки MongoDB:

- Консистентність: Може виникати проблема з консистентністю даних у розподілених системах.
- Складність запитів: Обмежена підтримка складних запитів у порівнянні з SQL-базами даних.

2. Firebase — це платформа для розробки мобільних та веб-застосунків від Google, яка пропонує низку інструментів для розробників, включаючи хмарне сховище даних. Firebase використовує NoSQL базу даних для зберігання та синхронізації даних в реальному часі.

   Переваги Firebase:

   - Реальний час: Дані автоматично синхронізуються між клієнтами в реальному часі.
   - Інтеграція з іншими сервісами Google: Легко інтегрується з іншими сервісами Google, такими як Google Analytics та Google Cloud.
   - Безсерверна архітектура: Не вимагає управління серверами або інфраструктурою.

   Недоліки Firebase:

   - Вартість: Вартість може швидко зрости з ростом обсягу даних та кількості користувачів.
   - Обмежена гнучкість: Менша гнучкість порівняно з традиційними SQL або NoSQL базами даних.

Вибір системи керування базами даних залежить від проекту та вимог до зберігання та обробки даних. Реляційні бази даних, такі як MySQL та PostgreSQL, забезпечують потужні можливості для роботи з структурованими даними та складними запитами. NoSQL бази даних, такі як MongoDB та Firebase, забезпечують високу гнучкість та продуктивність для обробки великих обсягів різнорідних даних. Firebase пропонує зручність використання та синхронізацію даних у реальному часі, що робить його відмінним вибором для сучасних веб та мобільних застосунків. У своєму проекті я вирішив використовувати Firebase

через його можливості синхронізації даних у реальному часі та інтеграцію з іншими сервісами Google.

2.6 Огляд API

API (Application Programming Interface) — це набір правил і протоколів, що дозволяють одному програмному забезпеченню взаємодіяти з іншим. API забезпечує стандартизований спосіб виклику функцій та обміну даними між різними програмами, сервісами або компонентами. У контексті веб-застосунків, API часто використовується для зв'язку клієнтських додатків з серверними сервісами або зовнішніми платформами, такими як соціальні мережі, платіжні системи або музичні стримінгові сервіси.

API дозволяють розробникам використовувати функціональність інших додатків без необхідності глибокого розуміння їх внутрішньої реалізації. Це значно спрощує розробку, дозволяючи зосередитися на основних завданнях і забезпечити інтеграцію з різними сервісами. Вони можуть бути реалізовані у вигляді RESTful сервісів, GraphQL, SOAP та інших.

Deezer API надає доступ до великої бібліотеки музичних треків, альбомів та виконавців. Основні функції Deezer API включають:

- Пошук музики: дозволяє шукати треки, альбоми та виконавців за різними критеріями, такими як назва, жанр або виконавець.
- Отримання деталей треків: надає детальну інформацію про конкретні треки, включаючи назву, виконавця, альбом та інші метадані.
- Створення плейлистів: дозволяє створювати та керувати плейлистами користувача.
- Прослуховування треків: надає можливість інтеграції функцій для прослуховування треків безпосередньо в веб-застосунку.

- Аналіз популярності: дозволяє отримувати інформацію про популярні треки та виконавців.

Spotify API є потужним інструментом для доступу до великої музичної бази даних Spotify. Він пропонує схожі функції з Deezer API:

- Пошук музики: можливість шукати треки, альбоми та виконавців.
- Отримання деталей треків: доступ до детальної інформації про треки.
- Управління плейлистами: створення, оновлення та видалення плейлистів.
- Аналіз популярності: доступ до даних про популярні треки та виконавців.
- Прослуховування треків: інтеграція функцій для прослуховування музики в застосунку.

Apple Music API надає доступ до музичної бібліотеки Apple Music. Основні можливості включають:

- Пошук музики: пошук треків, альбомів та виконавців.
- Отримання деталей треків: доступ до інформації про треки.
- Створення та управління плейлистами: можливість створювати і керувати плейлистами користувача.
- Прослуховування треків: інтеграція функцій для прослуховування музики.
- Аналіз популярності: доступ до даних про популярні треки та виконавців.

Tidal API пропонує доступ до музичної бази даних Tidal з високою якістю звуку. Основні функції включають:

- Пошук музики: пошук треків, альбомів та виконавців.
- Отримання деталей треків: доступ до інформації про треки.

- Створення плейлистів: можливість створювати та керувати плейлистами.
- Прослуховування треків: інтеграція функцій для прослуховування музики.
- Аналіз популярності: доступ до даних про популярні треки та виконавців.

В своєму проекті я обрав Deezer API завдяки своїй широкій музичній бібліотеці, що пропонує великий вибір треків, альбомів та виконавців, що дозволяє забезпечити користувачам доступ до різноманітної музики. Легкість інтеграції та зрозуміла документація роблять його зручним у розробці музчиного веб-застосунка. Deezer API надає всі необхідні функції для пошуку музики, прослуховування музики та отримання деталей треків. Крім того, Deezer є популярною музичною платформою з великою базою користувачів, що робить його надійним вибором.

2.7 Висновок з розділу 2

У цьому розділі було розглянуто різні засоби програмної реалізації веб-застосунків, включаючи мови програмування, фреймворки, архітектурні патерни, системи керування базами даних та API. Кожен з цих компонентів відіграє важливу роль у створенні сучасних, продуктивних та масштабованих веб-застосунків.

Було встановлено, що JavaScript разом з бібліотекою React є оптимальним вибором для розробки інтерактивних веб-застосунків завдяки своїй швидкості, великій спільноті та широкому вибору бібліотек.

Серед фреймворків, таких як React, Angular, Vue.js та Svelte, я обрав React через його компонентний підхід, високу продуктивність та велику екосистему

інструментів і бібліотек. React забезпечує зручність розробки та підтримки великих веб-застосунків.

Архітектурні патерни, такі як MVC, MVP, MVVM та SPA, забезпечують структуру та організацію коду, що спрощує його підтримку та масштабування. Використання архітектурного патерну MVC дозволяє чітко розділити обов'язки між компонентами застосунку, що підвищує його гнучкість та тестованість.

Для управління залежностями було розглянуто інструменти npm, Yarn та Bower. Обрано npm через його популярність, широку екосистему пакетів та інтеграцію з Node.js, що забезпечує зручність використання та управління залежностями у проекті.

Система керування базами даних (СКБД) є ключовим компонентом для зберігання та обробки даних. Реляційні бази даних, такі як MySQL та PostgreSQL, забезпечують потужні можливості для роботи зі структурованими даними. NoSQL бази даних, такі як MongoDB та Firebase, забезпечують гнучкість та продуктивність для обробки великих обсягів різнорідних даних. Firebase був обраний через його можливості синхронізації даних у реальному часі та інтеграцію з іншими сервісами Google.

Огляд API, включаючи Deezer API, Spotify API, Apple Music API та Tidal API, показав, що Deezer API є оптимальним вибором завдяки своїй широкій музичній бібліотеці, легкості інтеграції та зрозумілій документації. Використання Deezer API дозволяє створити функціональний та зручний музичний веб-застосунок, який відповідає потребам користувачів і забезпечує високу якість сервісу.

Отже, обрані засоби програмної реалізації забезпечують високу продуктивність, гнучкість та зручність розробки веб-застосунку, що дозволяє створити зручний та доволі унікальний продукт для пошуку та прослуховування музики з персоналізованими рекомендаціями.

РОЗДІЛ 3 ПРОЄКТУВАННЯ ТА РОЗРОБКА ВЕБ-ЗАСТОСУНКУ

3.1 Опис предметної області

У сучасному світі музика є невід'ємною частиною повсякденного життя багатьох людей. Музика може впливати на наш настрій, допомагати у відпочинку, зосередженні, або ж просто приносити задоволення. В епоху цифрових технологій користувачі очікують зручних та інтуїтивно зрозумілих сервісів для доступу до музики, які б забезпечували високий рівень персоналізації та відповідали їхнім вподобанням та емоційному стану.

На ринку існує багато додатків для прослуховування музики, однак не всі вони пропонують функції, які б задовольняли всі потреби користувачів. Серед основних проблем — складність у використанні, обмежений безкоштовний функціонал, а також недостатня персоналізація рекомендацій.

Метою цього проекту є розробка веб-застосунку, який дозволить користувачам легко знаходити та прослуховувати музику, отримувати персоналізовані рекомендації відповідно до їхнього настрою, а також мати доступ до сучасного і зручного інтерфейсу. Застосунок буде корисним як для звичайних користувачів, так і для музичних ентузіастів, які прагнуть відкривати нові треки, що відповідають їх емоційному стану.

Розробка веб-застосунку передбачає використання сучасних технологій, таких як React для фронтенду, Node.js та Express для бекенду, Firebase для аутентифікації та зберігання даних користувачів, а також інтеграцію з зовнішніми музичними API, такими як Deezer API для отримання музичного контенту та Genius API для аналізу текстів пісень.

Основною метою застосунку є надання користувачам зручного та інтуїтивно зрозумілого інструменту для доступу до музики. Застосунок дозволяє

здійснювати пошук музичних треків, відтворювати їх безпосередньо у браузері, отримувати рекомендації на основі настрою, а також керувати своїм профілем. Інтеграція з надійними джерелами даних забезпечує точність і актуальність інформації про музику.

Технічні аспекти:

1. Використання API:
    - Deezer API: Надає доступ до великої музичної бібліотеки для пошуку та прослуховування треків.
    - Genius API: Використовується для отримання текстів пісень та аналізу настрою музичних треків.
2. Технології фронтенду:
    - JavaScript та React: Основні технології для розробки інтерфейсу користувача, що забезпечують високу продуктивність та зручність використання.
    - React Context API: Використовується для управління глобальним станом застосунку, наприклад, станом аудіоплеєра та пошукових запитів.
3. Технології бекенду:
    - Node.js та Express: Використовуються для створення серверної частини застосунку, яка забезпечує обробку запитів та взаємодію з зовнішніми сервісами.
    - Контролери та маршрути: Контролери обробляють логіку запитів, а маршрути визначають, які запити приймаються та до яких контролерів направляються.
4. Використання Firebase
    - Аутентифікація: Firebase забезпечує аутентифікацію користувачів, включаючи реєстрацію, вхід через соціальні мережі та відновлення паролю.

- Зберігання даних профілю: Firebase використовується для зберігання та управління даними профілю користувача.

Таким чином, веб-застосунок поєднує в собі сучасні технології та сервіси для забезпечення зручного та інтуїтивно зрозумілого інтерфейсу, що надає користувачам можливість легко знаходити, відтворювати та отримувати рекомендації музичних треків відповідно до їх настрою.

3.2 Архітектура системи

Архітектура веб-застосунку для пошуку та прослуховування музики з функцією рекомендацій за настроєм користувача спирається на сучасні технології та передові архітектурні шаблони, що забезпечують високу продуктивність, зручність використання та масштабованість. Основні елементи цієї архітектури включають мову програмування JavaScript, фреймворк React для фронтенду, Node.js та Express для бекенду, Firebase для аутентифікації та зберігання даних користувачів, а також інтеграцію з Deezer API та Genius API для доступу до музичних та текстових даних відповідно.

Основні компоненти архітектури:

1. Фронтенд:
   - Використовує React для створення інтерактивного інтерфейсу користувача.
   - React Context API для управління станом додатку (наприклад, станом аудіоплеєра та пошукових запитів).
   - Firebase інтегровано для аутентифікації користувачів та зберігання даних профілю.
2. Бекенд:
   - Використовує Node.js та Express для обробки запитів від клієнта та взаємодії з зовнішніми API.

- Обробляє маршрутизацію запитів та управління даними.

3. Зовнішні сервіси:
   - Deezer API: Надає доступ до музичної бібліотеки для пошуку та прослуховування треків.
   - Genius API: Використовується для отримання текстів пісень для аналізу настрою музичних треків.

На наступній діаграмі розгортання показано фізичне розташування компонентів застосунку та їх взаємодію з іншими системами (див. Рис 11).

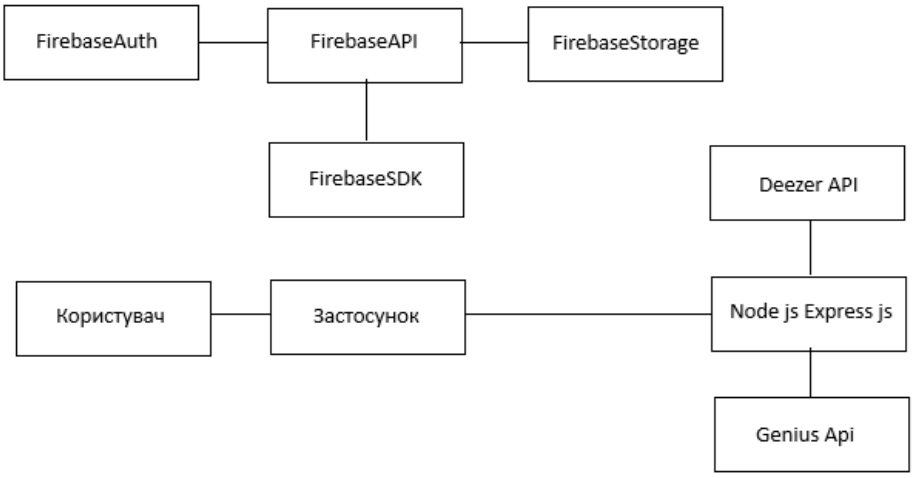

Рисунок 11 — Архітектура системи

Опис компонентів:

1. FirebaseAuth
   - Дозволяє користувачам реєструватися, входити в систему та виходити з неї. Підтримує різні методи аутентифікації, такі як електронна пошта/пароль, OAuth-провайдери (Google)
2. FirebaseAPI

- Забезпечує доступ до сервісів Firebase, таких як аутентифікація, зберігання даних та інші функціональні можливості.

3. FirebaseStorage
   - Використовується для зберігання та отримання даних, таких як зображення.
4. FirebaseSDK
   - Забезпечує інструменти та функції для взаємодії клієнтського додатку з Firebase.
5. Користувач
   - Взаємодіє з додатком, використовуючи його функції, такі як пошук музики, прослуховування треків, перегляд альбомів та отримання рекомендацій за настроем.
6. Застосунок
   - Забезпечує інтерфейс для користувача, де він може шукати музику, слухати треки, дивитись рекомендаціїї за настроєм та інше. Взаємодіє з Firebase для аутентифікації та зберігання даних, а також з Node.js сервером для отримання даних від сторонніх API.
7. Deezer API
   - Забезпечує інформацію про треки та альбоми. Використовується для пошуку музики та отримання її даних.
8. Node.js Express.js
   - Виконує запити до сторонніх API (таких як Deezer API і Genius API), обробляє отримані дані та передає їх клієнтському додатку.
9. Genius API
   - Призначення: Сторонній API для доступу до текстів пісень.
   - Функціональність: Забезпечує тексти пісень для треків. Використовується для отримання текстів пісень, які використовуються для аналізу за настроєм.

Архітектура веб-застосунку для пошуку та прослуховування музики забезпечує надійність, масштабованість та зручність використання. Інтеграція з Deezer API та Genius API надає користувачам доступ до точної та актуальної інформації про музику та тексти пісень. Використання Firebase для аутентифікації гарантує безпеку персональної інформації. Такий підхід дозволяє створювати високопродуктивний та зручний веб-застосунок.

3.3 Функціональні вимоги системи

У цьому розділі описані функціональні вимоги до системи для всіх користувачів. Наведені вимоги детально описують функції та можливості системи.

1. Реєстрація користувачів:
    - Користувач повинен мати можливість зареєструватися за допомогою електронної пошти та пароля.
    - Користувач повинен мати можливість увійти в систему за допомогою облікового запису Google.
    - Користувач повинен мати можливість відновити пароль через електронну пошту.
2. Редагування профілю:
    - Користувач повинен мати можливість редагувати особисті дані (ім'я).
    - Користувач повинен мати можливість змінювати аватар.
3. Пошук треків:
    - Користувач повинен мати можливість шукати треки та альбоми за допомогою пошуку.
4. Відтворення треків:
    - Користувач повинен мати можливість прослуховувати треки за допомогою аудіоплеєра.

- Користувач повинен мати можливість перемикати треки, змінювати гучність, ставити на паузу та відтворювати треки, перемотувати треки.
- Користувач повинен мати можливість використовувати режим повтору.

5. Персоналізовані рекомендації:
   - Користувач повинен мати можливість отримувати музичні рекомендації на основі свого настрою.
   - Користувач повинен мати можливість вибирати настрій (веселий, сумний, нейтральний) для отримання відповідних рекомендацій.
   - Система повинна аналізувати тексти пісень для визначення настрою за допомогою API Genius та sentiment.
6. Зручний дизайн:
   - Інтерфейс користувача повинен бути інтуїтивно зрозумілим та зручним для використання.

UseCase діаграму користувача можна побачити на рисунку 12.

Ці функціональні вимоги визначають основні завдання та обов'язки користувачів у системі, що дозволяє забезпечити ефективну роботу веб-застосунку та задоволення потреб користувачів.

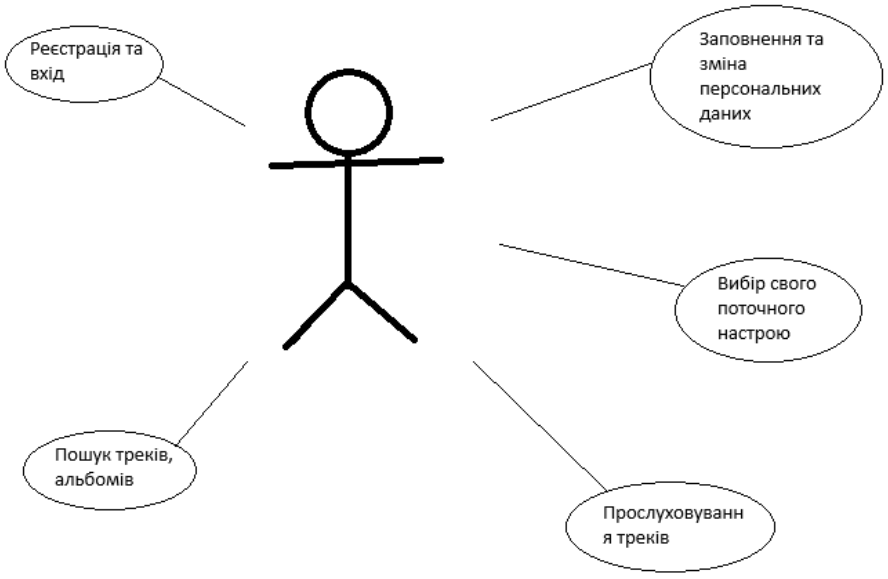

Рисунок 12 — UseCase діаграма можливостей взаємодії користувача з додатком

3.4 Вимоги до апаратного та програмного забезпечення

Для ефективної роботи веб-додатку необхідно забезпечити наступні вимоги до апаратного та програмного забезпечення:

Мінімальні вимоги:

- Процесор: 2 ядра, 2.4 GHz
- Оперативна пам'ять: 2 GB
- Дисковий простір: 20 GB
- Операційна система: Windows 7, macOS 10.12, Ubuntu 18.04 або новіші версії
- Браузер: Google Chrome 80+, Mozilla Firefox 75+, Microsoft Edge 80+, Safari 12+

Оптимальні вимоги:

- Процесор: 4 ядра, 3.0 GHz
- Оперативна пам'ять: 4 GB
- Дисковий простір: 50 GB SSD
- Операційна система: Windows 10, macOS 10.15, Ubuntu 20.04 або новіші версії
- Браузер: Останні стабільні версії Google Chrome, Mozilla Firefox, Microsoft Edge, Safari

Ці вимоги забезпечують стабільну та ефективну роботу веб-додатку для пошуку та прослуховування музики.

3.5 Модулі та алгоритми

Опис модулів:

Бекенд частина проекту відповідальна за обробку запитів від клієнтів, взаємодію з зовнішніми API та обробкою даних. Основні компоненти включають:

1. Сервер (server.js):
   - Відповідає за налаштування серверу, використання необхідних middleware та маршрутизацію запитів.
2. Контролери:
   - albumController.js: Обробляє запити, пов'язані з альбомами.
   - moodController.js: Обробляє запити, пов'язані з настроєм
   - searchController.js: Відповідає за обробку пошукових запитів.
   - sentimentController.js: Обробляє аналіз настрою.
3. Маршрути:
   - album.js, mood.js, search.js: Визначають маршрути для відповідних контролерів.

4. Сервіси:
    - deezerService.js: Взаємодія з Deezer API для отримання музичних даних.
    - geniusService.js: Взаємодія з Genius API для отримання текстів пісень.

Фронтенд частина проекту відповідає за взаємодію з користувачем, відображення даних та інтерфейс користувача. Основні компоненти включають:

1. Головні файли:
    - App.js: Основний компонент додатка.
    - index.js: Точка входу додатка.
2. Контексти:
    - AuthContext.js: Контекст для управління авторизацією користувачів.
    - MusicPlayerContext.js: Контекст для управління музичним плеєром.
    - SearchContext.js: Контекст для управління пошуковими запитами.
3. Компоненти:
    - Albums.js, Tracks.js: Компоненти для відображення альбомів та треків.
    - AudioPlayer.js: Компонент плеєру.
    - AuthForm: Компоненти для авторизації (LoginForm.js, SignupForm.js, ResetPassword.js).
    - Search.js: Компонент для пошуку музики, альбомів та їх відображення.
4. Сторінки:
    - AlbumPage.js: Сторінка для відображення деталей альбому.
    - LoginPage.js, SignupPage.js, ResetPasswordPage.js: Сторінки для авторизації.
    - MoodSelectorPage.js: Сторінка для вибору настрою.
    - ProfilePage.js: Сторінка профілю користувача.

Опис алгоритмів:

1. Алгоритм пошуку музики:
   - Опис: Алгоритм обробки пошукових запитів користувачів для отримання музичних треків і альбомів з API Deezer.
   - Кроки:
     1. Фронтенд надсилає HTTP-запит до бекенду на маршрут /search
     2. Бекенд отримує запит і передає його відповідному контролеру
     3. Контролер викликає функції сервісу для отримання даних з Deezer API
     4. Сервіс виконує запити до Deezer API для отримання треків та альбомів
     5. Контролер повертає результати на фронтенд.
     6. Фронтенд відображає результати пошуку у вигляді списку треків та альбомів.
2. Алгоритм аналізу настрою:
   - Опис: Алгоритм аналізу текстів пісень для визначення настрою користувача.
   - Кроки:
     1. Сервер отримує текст пісні за допомогою Genius API
     2. Текст пісні передається до модуля аналізу настрою
     3. Модуль аналізу використовує алгоритми обробки природної мови для визначення емоційного стану тексту
     4. Аналізований текст категоризується за визначеними настроями (, радісний, сумний, нейтральний)
3. Алгоритм рекомендацій на основі настрою:
   - Опис: Алгоритм надання музичних рекомендацій користувачам на основі їхнього настрою.
   - Кроки:

1. Користувач вибирає настрій у інтерфейсі додатка
2. Сервер отримує випадкові треки з Deezer API
3. Для кожного треку отримується текст пісні за допомогою Genius API та аналізується настрій тексту
4. Треки, настрій яких відповідає вибраному користувачем настрою, відбираються для рекомендацій
5. Відібрані треки формують список рекомендацій
6. Список рекомендацій відправляється користувачеві для відображення в інтерфейсі

Опис основних модулів:

1. Backend: server.js:

Цей файл є основним файлом конфігурації та запуску сервера. Він підключає необхідні модулі та маршрути, ініціалізує сервер (див. Лістинг 1).

Основні функції:

- Імпортує необхідні модулі для роботи додатка.
- Ініціалізує додаток Express
- Підключає мідлвар для парсингу JSON.
- Підключає маршрути для обробки запитів до різних частин додатка.
- Запускає сервер на вказаному порту.

Цей файл є центральною точкою налаштування та запуску сервера, забезпечуючи обробку HTTP-запитів і маршрутизацію до відповідних контролерів.

Літсинг 1 *server.js:*

```
const express = require('express');
const bodyParser = require('body-parser');
```

```
require('dotenv').config();
const app = express();
app.use(bodyParser.json());
app.use('/search', require('./routes/search'));
app.use('/album', require('./routes/album'));
app.use('/mood', require('./routes/mood'));
const PORT = process.env.PORT || 5000;
app.listen(PORT, () => {
  console.log(`Server running on port ${PORT}`);
});
```

2. Backend: deezerService.js:

Файл deezerService.js містить кілька функцій для виконання запитів до API Deezer. Основні функції цього сервісу включають пошук треків, альбомів і виконавців, а також отримання популярних треків та треків за жанрами.

Основні функціїї:

- searchDeezer — Загальна функція для виконання запитів до різних кінцевих точок API Deezer (див. Лістинг 2).

Лістинг 2 *Фунуція searchDeeze:*

```
const searchDeezer = async (endpoint, params) => {
  try {
    const response = await axios.get(`${DEEZER_API_URL}/${endpoint}`, { params });
    return response.data.data || [];
  } catch (error) {
    console.error(`Error fetching data from Deezer: ${endpoint}`, error.message);
    throw error;
  }
```

```
};
```

- getTracksFromSearch, getAlbumsFromSearch, getArtistsFromSearch, getTracksFromChart — повертає треки, альбоми, виконавців за пошуком, та популярні треки з чарту Deezer (див. Лістинг 3).

Лістинг 3 *Приклад функції getTracksFromSearch:*

```
const getTracksFromSearch = async (query, limit) => {
  return searchDeezer('search/track', { q: query, limit });
};
```

- getTracksFromGenre — повертає треки за заданим жанром (див. Лістинг 4)

Лістинг 4 *Функція getTracksFromGenre:*

```
const getTracksFromGenre = async (genreId, limit) => {
  const artists = await searchDeezer(`genre/${genreId}/artists`, {});
  const artistIds = artists.map(artist => artist.id);
  let allTracks = [];
  const trackPromises = artistIds.map(async (artistId) => {
    const tracks = await searchDeezer(`artist/${artistId}/top`, { limit });
    allTracks = allTracks.concat(tracks);
  });
  await Promise.all(trackPromises);
  return allTracks;
};
```

- getRandomDeezerTracks — Повертає випадкові треки з різних джерел (чарт, жанри, пошукові запити) (див. Лістинг 5).

Лістинг 5 *Функція getRandomDeezerTracks:*

```
const getRandomDeezerTracks = async (limit) => {
  const chartTracks = await getTracksFromChart();
  const genreTracks = await getTracksFromGenre(132, limit);
  const searchQueries = ['love', 'happy', 'sad', 'party',
'chill'];
  let searchTracks = [];
  for (const query of searchQueries) {
    const tracks = await getTracksFromSearch(query, limit);
    searchTracks = searchTracks.concat(tracks);
  }
  const allTracks = [...chartTracks, ...genreTracks,
...searchTracks];
  const uniqueTracks = Array.from(new Set(allTracks.map(track
=> track.id)))
    .map(id => allTracks.find(track => track.id === id));
  return uniqueTracks.sort(() => 0.5 - Math.random()).slice(0,
limit);
};
```

- getAlbumTracks, getAlbumInfo — Повертає треки з заданого альбому та інформацію про заданий альбом (див. Лістинг 6).

Лістинг 6 Функції getAlbumTracks та getAlbumInfo:

```
const getAlbumTracks = async (albumId) => {
  try {
    const { data } = await
axios.get(`${DEEZER_API_URL}/album/${albumId}/tracks`);
    return data.data;
  } catch (error) {
```

```
    console.error('Error fetching album tracks:',
error.message);
    throw error;
  }
};
const getAlbumInfo = async (albumId) => {
  try {
    const { data } = await
axios.get(`${DEEZER_API_URL}/album/${albumId}`);
    return data;
  } catch (error) {
    console.error('Error fetching album info:',
error.message);
    throw error;
  }
};
```

3. Backend: geniusService.js

Файл geniusService.js містить функції для взаємодії з API Genius та отримання текстів пісень. Основна мета цього сервісу - забезпечити можливість отримувати тексти пісень за назвою пісні та ім'ям виконавця, використовуючи API Genius.

Основні функції:

- getLyrics — Функція для отримання тексту пісні за назвою пісні та ім'ям виконавця (див. Лістинг 7).

Лістинг 7 *Функція getLyrics:*

```
const getLyrics = async (songTitle, artistName) => {
  try {
```

```
    console.log(`Fetching lyrics for: ${songTitle} by
${artistName}`);
    const response = await
axios.get(`${GENIUS_API_URL}/search`, {
      params: {
        q: `${songTitle} ${artistName}`
      },
      headers: {
        Authorization: `Bearer ${GENIUS_API_KEY}`
      }
    });

    if (response.data.response.hits.length === 0) {
      console.warn(`No hits found for: ${songTitle} by
${artistName}`);
      throw new Error('No hits found for the song');
    }
    const songPath =
response.data.response.hits[0].result.path;
    const songPage = await
axios.get(`https://genius.com${songPath}`);
    const lyrics = extractLyricsFromPage(songPage.data);
    return lyrics;
  } catch (error) {
    console.error('Error fetching lyrics:', error.message);
    throw new Error('Failed to fetch lyrics');
  }
};
```

- extractLyricsFromPage — Функція для вилучення тексту пісні зі сторінки пісні на сайті Genius (див. Лістинг 8).

Лістинг 8 *Функція extractLyricsFromPage:*

```
const extractLyricsFromPage = (htmlContent) => {
  const $ = cheerio.load(htmlContent);
  let lyrics = '';
  // Спробуємо отримати текст із використанням різних
селекторів
  $('.lyrics, .Lyrics__Container-sc-1ynbvzw-6,
.Lyrics__Container-sc-1ynbvzw-2').each((i, elem) => {
    $(elem).find('br').replaceWith('\n');
    lyrics += $(elem).text() + '\n';
  });
  // Якщо все ще немає тексту, пробуємо інші методи
  if (!lyrics) {
    $('.Lyrics__Root-sc-1ynbvzw-0').each((i, elem) => {
      $(elem).find('br').replaceWith('\n');
      lyrics += $(elem).text() + '\n';
    });
  }
  // Якщо все ще немає тексту, виводимо помилку
  if (!lyrics) {
    throw new Error('Lyrics not found on the page');
  }
  return lyrics.trim();
};
```

4. Backend: moodController.js

Файл moodController.js містить функції для обробки запитів, пов'язаних з рекомендаціями на основі настрою. Він отримує треки з Deezer, аналізує їх тексти за допомогою Genius і визначає їхній настрій.

Основні функції:

- getCachedLyrics — Використовується для збереження текстів пісень у кеші, щоб уникнути повторних запитів до API Genius. (див. Лістинг 9).

Лістинг 9 *Функція getCachedLyrics:*

```
const cache = new Map();
const getCachedLyrics = async (title, artist) => {
  const cacheKey = `${title}-${artist}`;
  if (cache.has(cacheKey)) {
    return cache.get(cacheKey);
  }
  const lyrics = await getLyrics(title, artist);
  cache.set(cacheKey, lyrics);
  return lyrics;
};
```

- filterTracksByMood — Аналізує тексти треків та фільтрує їх за настроєм користувача. (див. Лістинг 10).

Лістинг 10 *Функція filterTracksByMood:*

```
const filterTracksByMood = async (tracks, mood) => {
  const moodTrackPromises = tracks.map(async (track) => {
    try {
      console.log(`Processing track: ${track.title} by
${track.artist.name}`);
      const lyrics = await getCachedLyrics(track.title,
track.artist.name);
      console.log(`Fetched lyrics for track: ${track.title}`);
      const trackMood = analyzeMood(lyrics);
```

```
      console.log(`Analyzed mood for track: ${track.title} -
Mood: ${trackMood}`);
      if (trackMood === mood) {
        return {
          id: track.id,
          title: track.title,
          artist: track.artist,
          album: track.album,
          cover: track.album.cover,
          cover_medium: track.album.cover_medium,
          cover_small: track.album.cover_small
        };
      }
    } catch (error) {
      console.error(`Error processing track: ${track.title} by
${track.artist.name}`, error.message);
    }
    return null;
  });
  const moodTracks = (await
Promise.all(moodTrackPromises)).filter(track => track !==
null);
  return moodTracks;
};
```

- getMoodRecommendations — Координує процес отримання треків з Deezer, їх фільтрацію за настроєм і повертає результати користувачу (див. Лістинг 11).

Лістинг 11 *Функція filterTracksByMood:*

```
exports.getMoodRecommendations = async (req, res) => {
  const { mood, page = 1, limit = 20 } = req.query;
  const requestedAmount = 6;
  const offset = (page - 1) * limit;
  try {
    let tracks = [];
    let attempts = 0;
    const maxAttempts = 3;
    while (tracks.length < requestedAmount && attempts < 
maxAttempts) {
      console.log(`Fetching ${limit} random Deezer tracks`);
      let fetchedTracks = await getRandomDeezerTracks(limit, 
offset);
      console.log(`Fetched ${fetchedTracks.length} tracks from 
Deezer`);
      const moodTracks = await 
filterTracksByMood(fetchedTracks, mood);
      tracks = tracks.concat(moodTracks).slice(0, 
requestedAmount);
      attempts++;
    }
    if (tracks.length < requestedAmount) {
      console.warn(`Only ${tracks.length} tracks found 
matching mood: ${mood}`);
    }
    console.log(`Returning ${tracks.length} tracks matching 
mood: ${mood}`);
    res.json(tracks);
  } catch (error) {
```

```
    console.error('Error getting mood recommendations:',
error);
    res.status(500).send('Server error');
  }
};
```

5. Backend: searchController.js

Контролер для обробки запитів, пов'язаних з пошуком треків, альбомів та виконавців. Він використовує функції з сервісу Deezer для виконання пошукових запитів та повертає результати клієнту (див. Лістинг 12).

Лістинг 12 *searchController.js:*

```
const deezerService =
require('../services/deezerService');
    exports.search = async (req, res) => {
      const { q } = req.query;
      try {
        const [tracks, albums, artists] = await Promise.all([
          deezerService.getTracksFromSearch(q, 30),
          deezerService.getAlbumsFromSearch(q, 30),
          deezerService.getArtistsFromSearch(q, 30)
        ]);
        res.json({ tracks, albums, artists });
      } catch (error) {
        console.error('Error in search API:', error);
        res.status(500).send("Server error in search API");
      }
    };
```

6. Backend: albumController.js

Контролер для обробки запитів, пов'язаних з альбомами. Він використовує функції з сервісу Deezer для отримання треків альбому та інформації про альбом (див. Лістинг 13).

Лістинг 13 *albumController.js:*

```
const deezerService = require('../services/deezerService');
exports.getAlbumTracks = async (req, res) => {
  const { albumId } = req.params;
  try {
    const tracks = await
deezerService.getAlbumTracks(albumId);
    res.json({ tracks });
  } catch (error) {
    console.error('Error fetching album tracks: ', error);
    res.status(500).send("Server error");
  }
};
exports.getAlbumInfo = async (req, res) => {
  const { albumId } = req.params;
  try {
    const albumInfo = await
deezerService.getAlbumInfo(albumId);
    res.json(albumInfo);
  } catch (error) {
    console.error('Error fetching album info:', error);
    res.status(500).send("Server error");
  }
};
```

7. Backend: sentimentController.js

Контролер для аналізу настрою текстів пісень. Він використовує бібліотеку Sentiment для аналізу текстів і визначення настрою на основі їхнього змісту.

Лістинг 14 *sentimentController.js:*

```
const Sentiment = require('sentiment');
const sentiment = new Sentiment();
const analyzeMood = (lyrics) => {
  const result = sentiment.analyze(lyrics);
  const score = result.score;
  if (score > 0) {
    return 'happy';
  } else if (score < 0) {
    return 'sad';
  } else {
    return 'neutral';
  }
};
module.exports = { analyzeMood };
```

8. Frontend: MoodSelector.js

Компонент для вибору настрою та завантаження треків з сервера на основі обраного настрою. Він використовує хуки для керування станом і виконує запити до сервера для отримання треків.

Основні функції:

- fetchTracks — Виконує HTTP-запит до сервера для отримання треків та використовує AbortController для відміни попередніх запитів (див. Лістинг 15).

Лістинг 15 *Функція fetchTracks:*

```
const fetchTracks = useCallback(async (selectedMood, page = 1)
=> {
    if (controller) {
      controller.abort();
    }
    const newController = new AbortController();
    setController(newController);
    try {
      const response = await
axios.get(`http://localhost:5000/mood/recommendations`, {
        params: { mood: selectedMood, page },
        signal: newController.signal,
      });
      return response.data;
    } catch (error) {
      if (axios.isCancel(error)) {
        console.log('Previous request canceled');
      } else {
        onErrorChange('Failed to fetch tracks. Please try
again.');
      }
    }
  }, [controller, onErrorChange]);
```

- handleMoodSelect — Викликається при виборі настрою та очищує попередні треки, встановлює стан завантаження і виконує новий запит для отримання треків (див. Лістинг 16).

Лістинг 16 *Функція handleMoodSelect:*

```
const handleMoodSelect = useCallback(async (selectedMood) => {
    onTracksUpdate([]);
    onLoadingChange(true);
    onErrorChange(null);
    setMood(selectedMood);
    setPage(1);
    const newTracks = await fetchTracks(selectedMood);
    onTracksUpdate(newTracks);
    onLoadingChange(false);
  }, [fetchTracks, onTracksUpdate, onLoadingChange,
onErrorChange]);
```

- handleLoadMore — Викликається при натисканні кнопки "Load More" та завантажує додаткові треки і додає їх до існуючого списку (див. Лістинг 17).

Лістинг 17 *Функція handleLoadMore:*

```
const handleLoadMore = useCallback(async () => {
    onLoadingChange(true);
    onErrorChange(null);
    const newPage = page + 1;
    const newTracks = await fetchTracks(mood, newPage);
    onTracksUpdate(prevTracks => [...prevTracks,
...newTracks]);
    setPage(newPage);
    onLoadingChange(false);
  }, [fetchTracks, mood, page, onTracksUpdate, onLoadingChange,
onErrorChange]);
```

9. Frontend: Search.js

Компонент Search відповідає за пошук музики на основі введеного користувачем запиту. Він взаємодіє з бекендом для отримання результатів пошуку і відображає їх у вигляді списку треків та альбомів.

Основні функції:

- searchMusic — Виконує HTTP-запит до бекенду для отримання треків та альбомів на основі пошукового запиту. Оновлює стан треків, альбомів, оригінального списку треків і встановлює стан пошуку. У випадку помилки, очищує результати і встановлює стан пошуку в false (див. Лістинг 18).

Лістинг 18 *Функція searchMusic:*

```
const searchMusic = useCallback(async (query) => {
    if (query.trim() === '') {
      setTracks([]);
      setAlbums([]);
      setSearched(false);
      return;
    }
    try {
      const response = await axios.get(`http://localhost:5000/search`, { params: { q: query } });
      setTracks(response.data.tracks || []);
      setAlbums(response.data.albums || []);
      setOriginalTrackList(response.data.tracks || []);
      setMainTrackList(response.data.tracks || []);
      setSearched(true);
```

```
    } catch (error) {
      console.error('Error fetching data: ', error);
      setTracks([]);
      setAlbums([]);
      setSearched(false);
    }
  }, [setAlbums, setMainTrackList, setOriginalTrackList,
setSearched, setTracks]);
```

3.7 Проект інтерфейсу

Сучасний музичний додаток вимагає ретельно продуманого та зручного інтерфейсу, який дозволяє користувачам насолоджуватися музикою без зайвих зусиль. Наш музичний додаток створений з метою забезпечити користувачів інтуїтивно зрозумілим інтерфейсом для пошуку музики, отримання рекомендацій на основі настрою та керування плейлістами.

Процес розробки інтерфейсу розпочався з детального аналізу потреб користувачів та їх очікувань. Було зібрано вимоги, розроблено початкові прототипи та проведено тестування з реальними користувачами. На основі отриманих відгуків інтерфейс постійно вдосконалювався, щоб максимально відповідати потребам користувачів.

Використання сучасних технологій, таких як JavaScript та React, дозволило створити швидкодіючі та ефективні елементи інтерфейсу. Завдяки архітектурному підходу з використанням контекстів (Context API) додаток є легким у підтримці та масштабуванні. Це особливо важливо для забезпечення постійного розширення функціональності.

Інтеграція з API Deezer та Genius надає користувачам доступ до великої кількості музичного контенту та текстів пісень. Це значно полегшує процес

пошуку музики та забезпечує додаткові можливості, такі як аналіз настрою текстів пісень для отримання персоналізованих рекомендацій.

Результатом комплексного підходу до розробки став інтерфейс, який дозволяє користувачам легко знаходити та прослуховувати музику, отримувати рекомендації на основі їхнього настрою, переглядати та керувати даними профіля. Завдяки зручності та інтуїтивній зрозумілості інтерфейсу, користувачі можуть зосередитися на насолодженні музикою без зайвих труднощів.

Детальніше інтерфейс та його ключові елементи представлені на рисунках 13 — 19

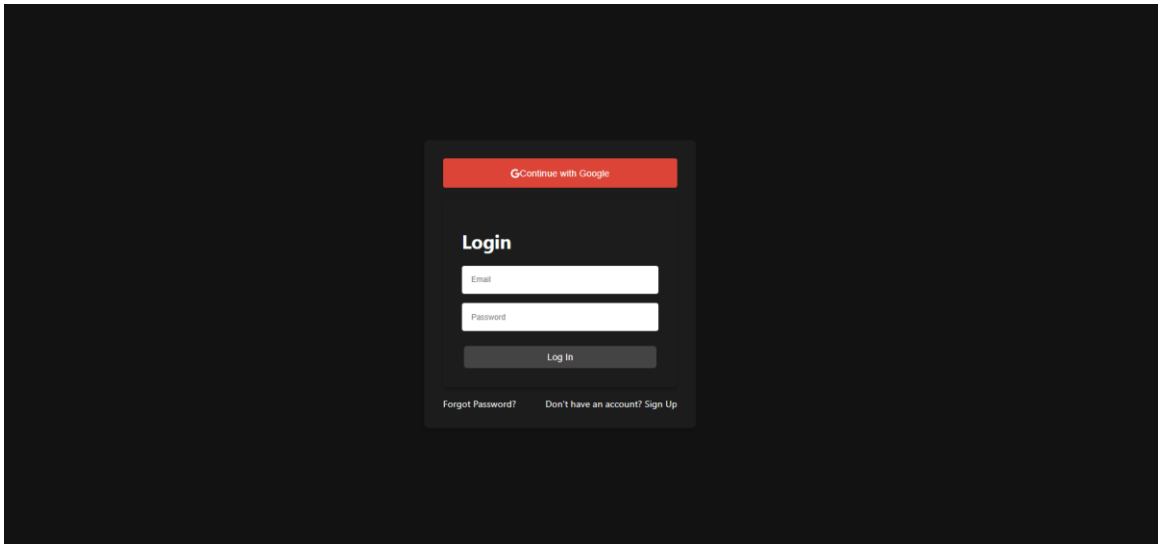

Рисунок13 — Сторінка Login

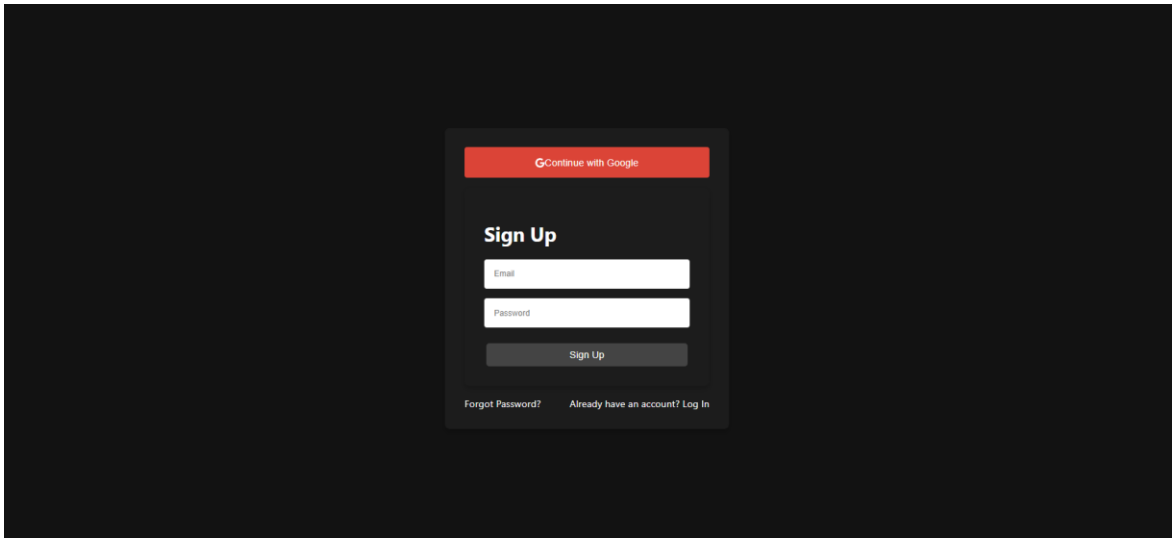

Рисунок 14 — Сторінка SignUp

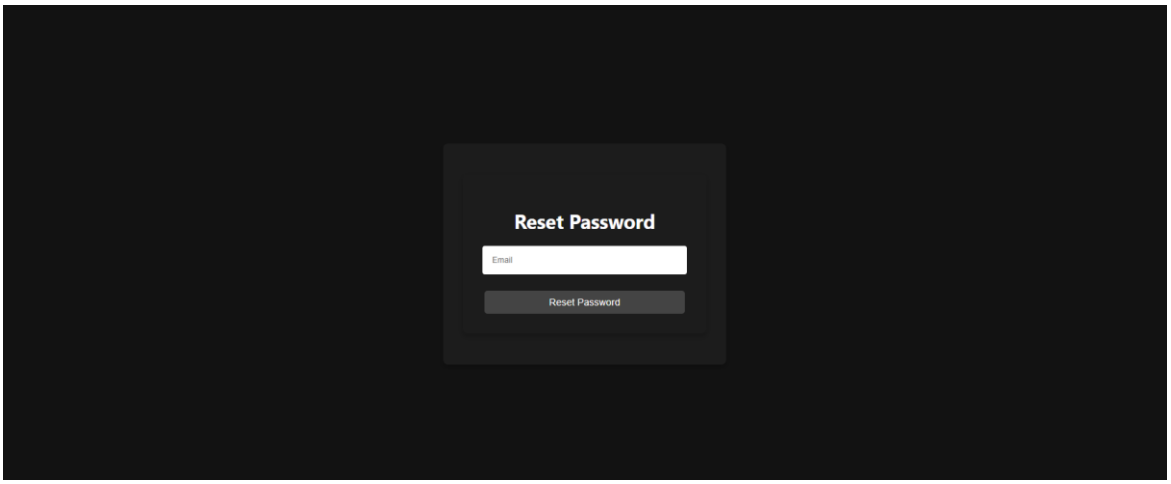

Рисунок 15 — Сторінка скидання паролю

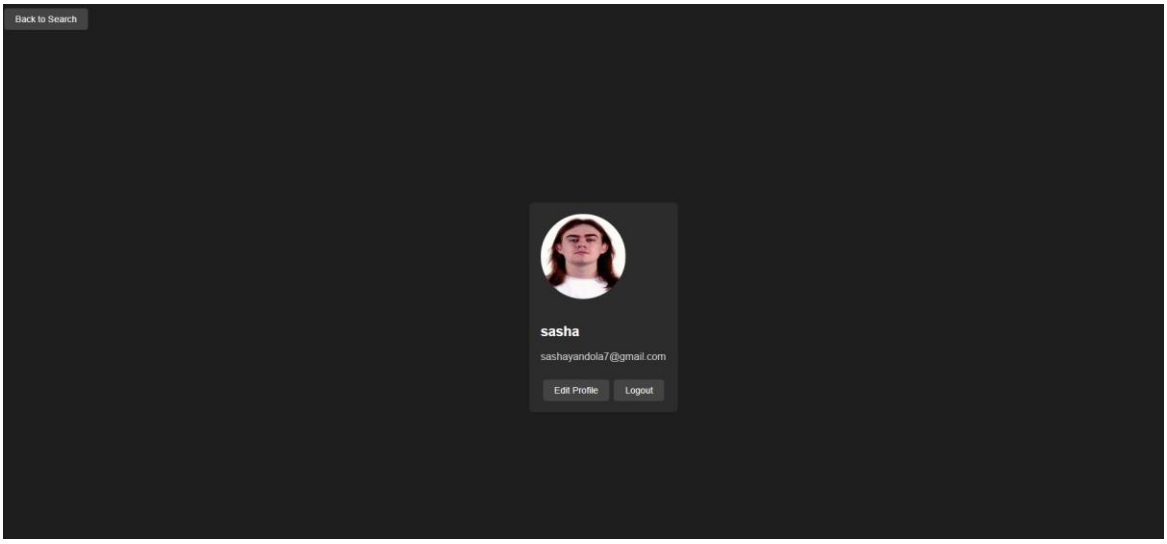

Рисунок 16 — Сторінка профілю

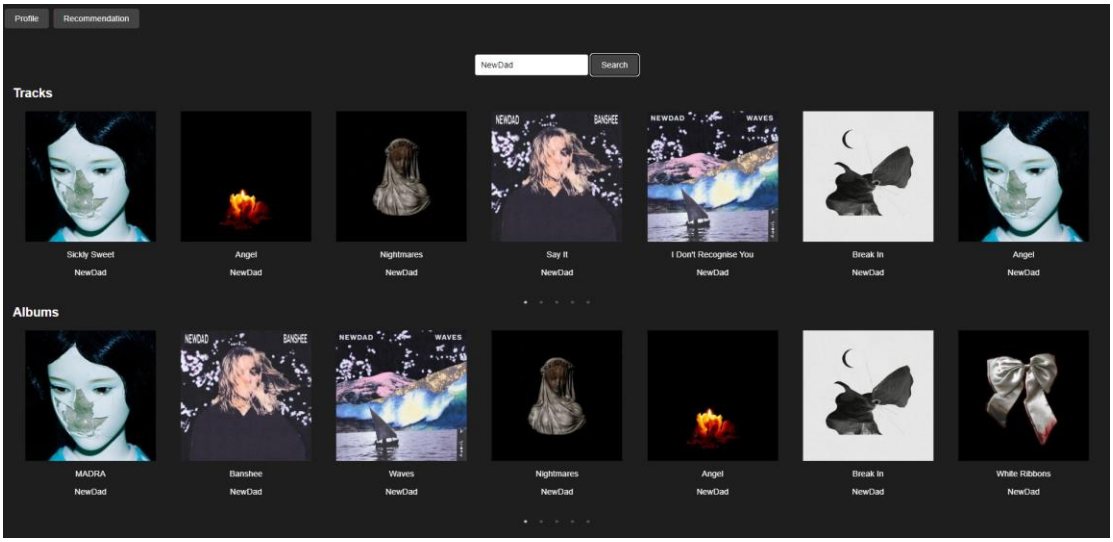

Рисунок 17 — Головна сторінка пошуку треків та альбомів

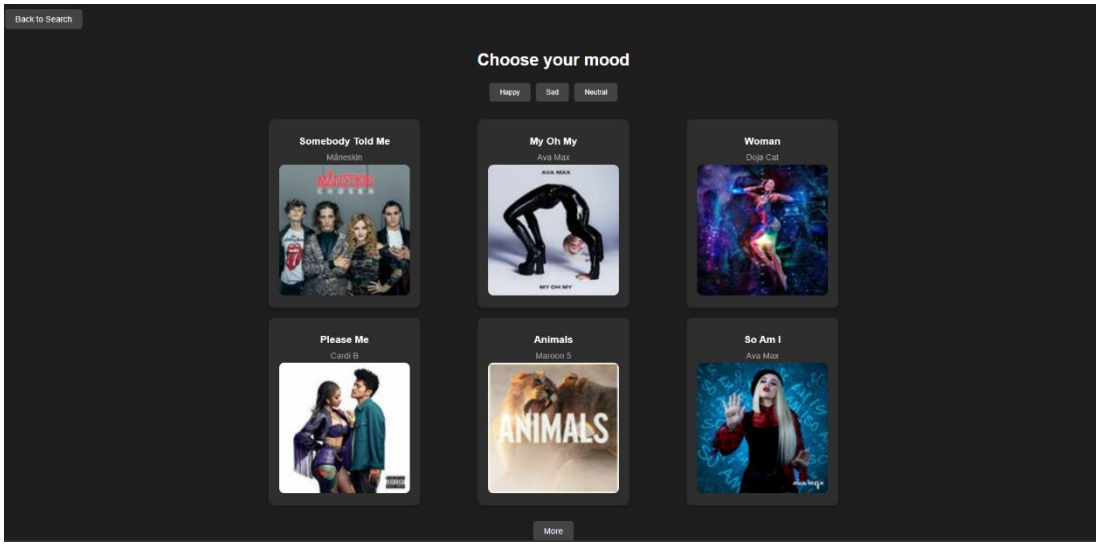

Рисунок 18 — Сторінка рекомендацій за настроєм користувача

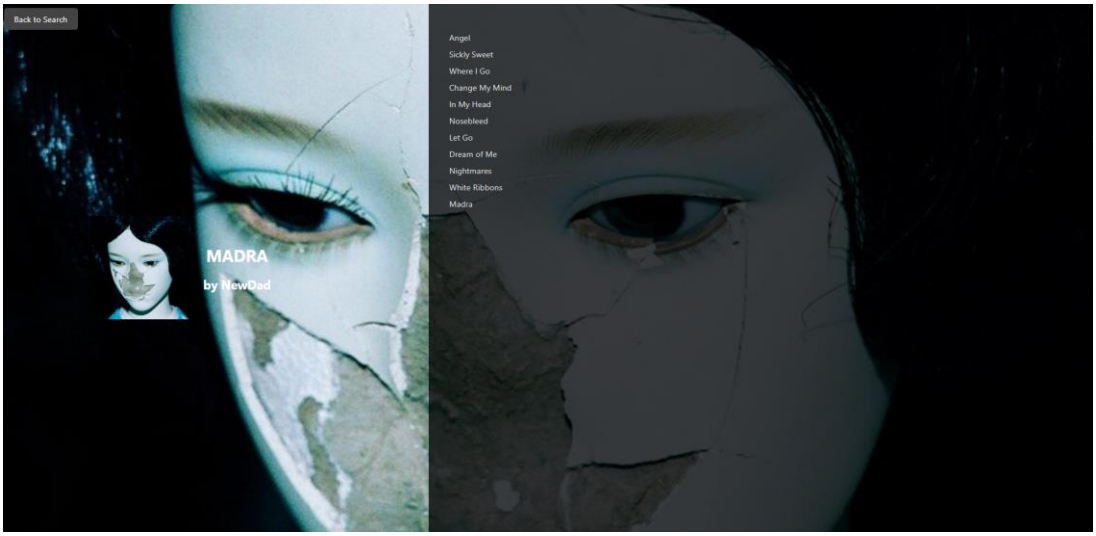

Рисунок 19 —Сторінка альбому

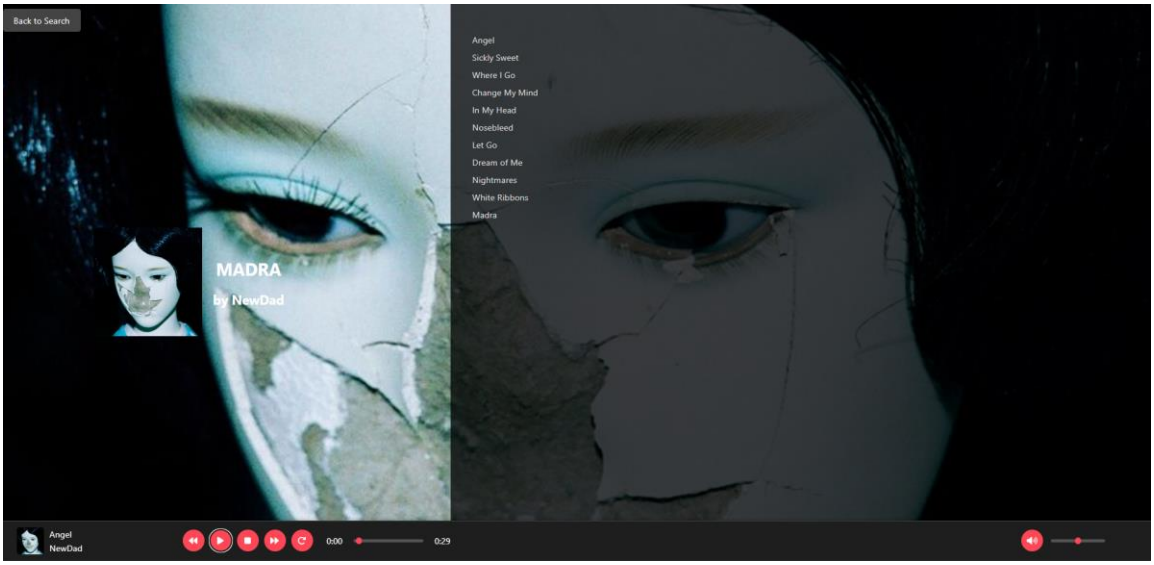

Рисунок 20— Сторінка альбому з вбудованим плеєром

3.8 Тестування

Для забезпечення якості та стабільності додатку було проведено тестування в різних браузерах, таких як Microsoft Edge, Google Chrome, Mozilla Firefox та Opera. Тестування також включало перевірку основних функцій долатку, таких як:

1. Реєстрація та авторизація.
2. Налаштування персональних даних в профілі.
3. Пошук музики та альбомів.
4. Прослуховування музики.
5. Отримання рекомендацій музики за вибраним настроєм користувача.

Додаток пройшов ручне тестування для перевірки всіх функціональних можливостей та інтерфейсу, що підтвердило його високу стабільність та продуктивність.

ВИСНОВКИ

1. В результаті аналізу було встановлено, що існуючі музичні сервіси не повністю задовольняють потреби користувачів у персоналізації та рекомендаціях музики відповідно до їхнього настрою.
2. Основною метою дипломної роботи є розробка веб-застосунку, що забезпечує користувачам унікальний досвід пошуку та прослуховування музики з функцією рекомендацій за настроєм.
3. JavaScript був обраний як основна мова для фронтенд- та бекенд-розробки завдяки своїй популярності, швидкості та великій екосистемі.
4. Для фронтенд-розробки було обрано React завдяки його компонентному підходу, високій продуктивності та великій спільноті розробників.
5. Deezer API був обраний для доступу до музичної бібліотеки, а Genius API – для отримання текстів пісень, що дозволяє реалізувати функцію рекомендацій за настроєм.
6. Створений веб-застосунок для пошуку та прослуховування музики з функцією рекомендацій за настроєм користувача.
7. Застосунок був протестований на предмет функціональності та продуктивності, проведено оптимізацію для забезпечення швидкої та плавної роботи.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. JavaScript. Mozilla Developer Network (MDN) site. URL: https://developer.mozilla.org/en-US/docs/Web/JavaScript (дата звернення 14.01.2024).
2. React – A JavaScript library for building user interfaces. React site. URL: https://reactjs.org/ (дата звернення 14.01.2024).
3. Deezer API. Deezer Developers site. URL: https://developers.deezer.com/api (дата звернення 14.01.2024).
4. Firebase Documentation. Firebase site. URL: https://firebase.google.com/docs/ (дата звернення 14.01.2024).
5. Genius API Documentation. Genius site. URL: https://docs.genius.com/ (дата звернення 14.01.2024).
6. NPM Documentation. NPM site. URL: https://docs.npmjs.com/ (дата звернення 14.01.2024).
7. Axios Documentation. Axios site. URL: https://axios-http.com/docs/intro (дата звернення 14.01.2024).
8. Express Documentation. Express site. URL: https://expressjs.com/en/starter/installing.html (дата звернення 14.01.2024).
9. Node.js Documentation. Node.js site. URL: https://nodejs.org/docs/latest/api/ (дата звернення 14.01.2024).
10. Sentiment Analysis API Documentation. Sentiment Analysis site. URL: https://www.npmjs.com/package/sentiment (дата звернення 14.01.2024).
11. Spotify. URL: https://open.spotify.com/ (дата звернення 14.01.2024).
12. Яндола О. Ю., Безверхий А. І. Розробка веб-застосунку для пошуку та прослуховування музики з функцією рекомендацій за настроєм користувача. Збірник наукових праць студентів, аспірантів, докторантів і молодих вчених «Молода наука-2024» / Запорізький національний університет. Запоріжжя: ЗНУ, 2024. Т.5. С. 217-218.

Декларація
академічної доброчесності
здобувача ступеня вищої освіти ЗНУ

Я, Яндола Олександр Юрійович, студент _4_ курсу, форми навчання денної, Інженерного навчально-наукового інституту, спеціальність 121 Інженерія програмного забезпечення, адреса електронної пошти ipz20bd-217@stu.zsea.edu.ua, — підтверджую, що написана мною кваліфікаційна робота на тему «Розробка веб-застосунку для пошуку та прослуховування музики з функцією рекомендацій за настроєм користувача» відповідає вимогам академічної доброчесності та не містить порушень, що визначені у ст.42 Закону України «Про освіту», зі змістом яких ознайомлений.

- заявляю, що надана мною для перевірки електронна версія роботи є ідентичною її друкованій версії;

- згоден на перевірку моєї роботи на відповідність критеріям академічної доброчесності у будь-який спосіб, у тому числі за допомогою інтернет-систем, а також на архівування моєї роботи в базі даних цієї системи.

Дата 14.06.2024 Підпис _____ Яндола Олександр Юрійович
(студент)

Дата 14.06.2024 Підпис _____ Безверхий Анатолій Ігорович
(науковий керівник)