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

Факультет прикладної математики

Кафедра програмного забезпечення комп'ютерних систем

«До захисту допущено»
Завідувач кафедри
_____ Євгенія СУЛЕМА
«___» _____ 2025 р.

Дипломний проєкт

на здобуття ступеня бакалавра

за освітньо-професійною програмою «Інженерія програмного забезпечення мультимедійних та інформаційно-пошукових систем»

спеціальності 121 Інженерія програмного забезпечення

на тему: «Вебзастосунок «SpotiStats» для відстежування даних про музичні вподобання»

Виконав:
студент IV курсу, групи КП-11
Бодаква Кирил Ігорович _____

Керівник:
доцент кафедри ПЗКС, к.т.н,
Рибачок Наталія Антонівна _____

Консультант з нормоконтролю:
доцент кафедри ПЗКС, к.т.н., доцент,
Онай Микола Володимирович _____

Рецензент:
доцент кафедри ЕІ ФЕЛ, к.т.н.,
Вутнтесмері Юрій Володимирович _____

Засвідчую, що у цьому дипломному проєкті немає запозичень з праць інших авторів без відповідних посилань.
Студент _____

Київ – 2025 року

Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»
Факультет прикладної математики
Кафедра програмного забезпечення комп'ютерних систем

Рівень вищої освіти – перший (бакалаврський)

Спеціальність – 121 «Інженерія програмного забезпечення»

Освітньо-професійна програма «Інженерія програмного забезпечення мультимедійних та інформаційно-пошукових систем»

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Євгенія СУЛЕМА
«___»_____2024 р.

ЗАВДАННЯ

на дипломний проєкт студенту

Бодакві Кирилу Ігоровичу

1. Тема проєкту «Вебзастосунок «SpotiStats» для відстежування даних про музичні вподобання», керівник проєкту Рибачок Наталія Антонівна, доцент кафедри ПЗКС, к.т.н., затверджені наказом по університету №1808-С від «29» травня 2025 р.

2. Термін подання студентом проєкту «13» червня 2025 р.

3. Вихідні дані до проєкту: див. Технічне завдання.

4. Зміст пояснювальної записки:
   – огляд та аналіз наявних рішень;
   – обґрунтування вибору засобів розроблення вебзастосунку;
   – структурно-алгоритмічна організація застосунку;
   – аналіз реалізованого вебзастосунку.

5. Перелік обов'язкового графічного матеріалу:
   – функціональність вебзастосунку (креслення);
   – архітектура вебзастосунку (креслення);

– схематичне представлення процесу OAuth 2.0 для автентифікації при взаємодії зі Spotify API (плакат);
– структура вебсайту (плакат).

6. Консультанти розділів проєкту

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Нормоконтроль	Онай М.В., доцент		

7. Дата видачі завдання «31» жовтня 2024 р.

Календарний план

№ з/п	Назва етапів виконання дипломного проєкту	Термін виконання етапів проєкту	Примітка
1.	Вивчення літератури за тематикою проєкту	12.11.2024	
2.	Розроблення та узгодження технічного завдання	22.11.2024	
3.	Розроблення структури вебзастосунку	12.12.2024	
4.	Підготовка матеріалів першого розділу дипломного проєкту	25.12.2024	
5.	Розроблення дизайну сторінок та графічних елементів	05.02.2025	
6.	Підготовка матеріалів другого розділу дипломного проєкту	12.02.2025	
7.	Програмна реалізація вебзастосунку	18.03.2025	
8.	Тестування вебзастосунку	25.03.2025	
9.	Підготовка матеріалів третього розділу дипломного проєкту	30.03.2025	
10.	Підготовка матеріалів четвертого розділу дипломного проєкту	22.04.2025	
11.	Підготовка графічної частини дипломного проєкту	26.04.2025	
12.	Оформлення документації дипломного проєкту	24.05.2025	

Студент Кирил БОДАКВА

Керівник проєкту Наталія РИБАЧОК

АНОТАЦІЯ

Даний дипломний проєкт присвячений створенню вебзастосунку «SpotiStats» для відстежування даних про музичні вподобання користувачів Spotify. Вебзастосунок являє собою вебсайт, який надає зареєстрованим користувачам інструменти візуалізації їхніх музичних звичок на основі даних, наданих Spotify API.

Функціональність вебзастосунку включає в себе автентифікацію користувачів через Spotify API, перегляд детальної статистики прослуховувань за різними проміжками часу (пісні, альбоми, виконавці, жанри), автоматичний імпорт та оновлення даних прослуховувань з Spotify. Інформаційна безпека вебзастосунку забезпечена за рахунок використання протоколу HTTPS, обмеження терміну дії сесій, захисту від CSRF та XSS атак, а також дотримання норм GDPR. Гостьовий доступ до вебзастосунку не передбачено.

У даному дипломному проєкті розроблено: архітектуру вебзастосунку, алгоритм автентифікації користувачів через Spotify API з використанням OAuth 2.0, механізм збереження та оновлення токенів доступу, структуру бази даних, клієнтський та серверний код вебзастосунку, а також графічні елементи та дизайн вебсторінок.

ABSTRACT

The diploma project dedicated to the creation of the SpotiStats web application for tracking data about the musical preferences of Spotify users. The web application is a website that provides registered users with tools for visualizing their musical habits based on data provided by the Spotify API.

The functionality of the web application includes user authentication via Spotify API, viewing detailed listening statistics for different time periods (songs, albums, artists, genres), automatic import and update of listening data from Spotify. The information security of the web application is ensured using the HTTPS protocol, session timeout restrictions, protection against CSRF and XSS attacks, and compliance with GDPR standards. Guest access to the web application is not provided.

This thesis project developed: web application architecture, user authentication algorithm via Spotify API using OAuth 2.0, mechanism for storing and updating access tokens, database structure, client and server code of the web application, as well as graphic elements and web page design.

ДП.045440-01-90 Вебзастосунок «SpotiStats» для відстежування даних про музичні вподобання. Відомість проєкту

Позначення	Найменування	Кіл-ть	Примітка
	Документація проєкту		
ДП.045440-02-91	Вебзастосунок	5	
	«SpotiStats» для		
	відстежування даних		
	про музичні вподобання.		
	Технічне завдання		
ДП.045440-03-81	Вебзастосунок	50	
	«SpotiStats» для		
	відстежування даних		
	про музичні вподобання.		
	Пояснювальна записка		
ДП.045440-04-51	Вебзастосунок	4	
	«SpotiStats» для		
	відстежування даних		
	про музичні вподобання.		
	Програма та методика		
	тестування		
ДП.045440-05-34	Вебзастосунок	15	
	«SpotiStats» для		
	відстежування даних		
	про музичні вподобання.		
	Керівництво користувача		

Позначення	Найменування	Кіл-ть	Примітка
ДП.045440-06-99	Вебзастосунок	1	
	«SpotiStats» для		
	відстежування даних		
	про музичні вподобання.		
	Функціональність		
	вебзастосунку. UML-		
	діаграма прецедентів		
ДП.045440-07-99	Вебзастосунок	1	
	«SpotiStats» для		
	відстежування даних		
	про музичні вподобання.		
	Архітектура		
	вебзастосунку. DFD-		
	діаграма		
ДП.045440-08-98	Вебзастосунок	1	
	«SpotiStats» для		
	відстежування даних		
	про музичні вподобання.		
	Компакт-диск		

Факультет прикладної математики

Кафедра програмного забезпечення комп'ютерних систем

«ЗАТВЕРДЖЕНО»

Завідувач кафедри

_____ Євгенія СУЛЕМА

«___» _____ 2024 р.

ВЕБЗАСТОСУНОК «SPOTISTATS» ДЛЯ ВІДСТЕЖУВАННЯ ДАНИХ ПРО МУЗИЧНІ ВПОДОБАННЯ

Технічне завдання

ДП.045440-02-91

«ПОГОДЖЕНО»

Керівник проєкту:

_____ Наталія РИБАЧОК

Нормоконтроль:

_____ Микола ОНАЙ

Виконавець:

_____ Кирил БОДАКВА

2024

ЗМІСТ

1. НАЙМЕНУВАННЯ ТА ГАЛУЗЬ ЗАСТОСУВАННЯ

Назва розробки: Вебзастосунок «SpotiStats» для відстежування даних про музичні вподобання.

Галузь застосування: інформаційні технології.

2. ПІДСТАВА ДЛЯ РОЗРОБЛЕННЯ

Підставою для розроблення є завдання на дипломне проєктування, затверджене кафедрою програмного забезпечення комп'ютерних систем Національного технічного університету України «Київський політехнічний інститут імені Ігоря Сікорського» (КПІ ім. Ігоря Сікорського).

3. ПРИЗНАЧЕННЯ РОЗРОБКИ

Розробка вебзастосунку «SpotiStats» призначена для створення зручного інструменту для аналізу особистих музичних вподобань на основі даних, що надаються Spotify API.

4. ВИМОГИ ДО ПРОГРАМНОГО ПРОДУКТУ

Вебзастосунок повинен забезпечувати такі основні вимоги:

1. Можливість динамічного оновлення вмісту вебсторінок.
2. Можливість автентифікації через обліковий запис Spotify.
3. Можливість перегляду статистики прослуховування музики за різні проміжки часу.
4. Можливість аналізувати свої музичні вподобання за піснями, альбомами, виконавцями та жанрами.
5. Можливість перегляду своїх улюблених виконавців, жанри пісень,

альбоми.

6. Можливість перегляду історії прослуховувань.
7. Забезпечення автоматичного імпорту та оновлення даних прослуховувань зі Spotify.

Вимоги до якості:

1. Відгук на дії користувача (фільтрація, сортування, оновлення даних) має відбуватися в межах 1-2 секунд.
2. Система повинна обробляти не менше 500 запитів одночасно без значної деградації швидкодії.
3. Система повинна автоматично відновлюватися після збоїв без втрати користувацьких даних.
4. Система повинна підтримувати збільшення кількості користувачів без суттєвого погіршення продуктивності.
5. Архітектура системи має забезпечувати можливість горизонтального масштабування при зростанні навантаження.
6. Система повинна мати модульну структуру для розширення можливостей та інтеграції нових модулів при подальшій підтримці продукту.

5. ВИМОГИ ДО ПРОЄКТНОЇ ДОКУМЕНТАЦІЇ

У процесі виконання проєкту повинна бути розроблена наступна документація:

1) пояснювальна записка;
2) програма та методика тестування;
3) керівництво користувача;
4) креслення:
   – «Функціональність вебзастосунку. UML-діаграма прецедентів»;
   – «Архітектура вебзастосунку. DFD-діаграма».

6. ЕТАПИ ПРОЄКТУВАННЯ

Вивчення літератури за тематикою роботи 12.11.2024
Розроблення та узгодження технічного завдання 22.11.2024
Розроблення структури вебзастосунку 12.12.2025
Розроблення дизайну сторінок та графічних елементів 05.02.2025
Програмна реалізація вебзастосунку 18.03.2025
Тестування вебзастосунку 25.03.2025
Підготовка матеріалів текстової частини проєкту 22.04.2025
Підготовка матеріалів графічної частини проєкту 26.04.2025
Оформлення технічної документації проєкту 26.05.2025

7. ПОРЯДОК ТЕСТУВАННЯ РОЗРОБКИ

Тестування розробленого програмного продукту виконується відповідно до «Програми та методики тестування».

Факультет прикладної математики

Кафедра програмного забезпечення комп'ютерних систем

«ЗАТВЕРДЖЕНО»

Завідувач катедри

_____ Євгенія СУЛЕМА

«___» _____ 2025 р.

ВЕБЗАСТОСУНОК «SPOTISTATS» ДЛЯ ВІДСТЕЖУВАННЯ ДАНИХ ПРО МУЗИЧНІ ВПОДОБАННЯ

Пояснювальна записка

ДП.045440-03-81

«ПОГОДЖЕНО»

Керівник проєкту:

_____ Наталія РИБАЧОК

Нормоконтроль:

_____ Микола ОНАЙ

Виконавець:

_____ Кирил БОДАКВА

2025

ЗМІСТ

СПИСОК ТЕРМІНІВ, СКОРОЧЕНЬ ТА ПОЗНАЧЕНЬ

Spotify – це стримінговий сервіс потокового аудіо, що дозволяє слухати музичні композиції та подкасти.

Функціональність – це низка можливостей (функцій), які надає інформаційна або інша система чи пристрій.

API (Application Programming Interface) – це спосіб взаємодії комп'ютерних програм між собою. Набір визначень підпрограм, протоколів взаємодії та засобів для створення програмного забезпечення.

IOS – це мобільна операційна система, розроблена компанією Apple Inc. для своїх смартфонів iPhone, планшетів iPad, медіаплеєрів iPod та розумних годинників Apple Watch.

Android – операційна система з відкритим кодом, розроблена компанією Google. Вона використовується на смартфонах, планшетах, телевізорах, розумних годинниках, автомобілях та інших пристроях.

Інтерфейс користувача – засіб зручної взаємодії користувача (людини) з інформаційною системою.

Баг – помилка, вада або дефект в комп'ютерній програмі або системі, що викликає в ній неправильний або неочікуваний результат чи неочікувану поведінку.

OAuth – це відкритий стандарт для делегації доступу, який зазвичай використовується як спосіб для користувачів Інтернету для надання вебсайтів або додатків доступу до їх інформації на інших вебсайтів, але не надаючи їм паролів.

HTTPS (HyperText Transfer Protocol Secure) – зашифрована версія протоколу HTTP, що використовує TLS для шифрування всього зв'язку між клієнтом та сервером.

Токен доступу – це рядок, отриманий під час аутентифікації (використовуючи додаток або сервер авторизації).

CSRF (Cross-Site Request Forgery) – це атака, при якій зловмисник змушує користувача виконати небажану дію на сайті, на якому той авторизований.

XSS (Cross-Site Scripting) – це атака, при якій зловмисник впроваджує шкідливий JavaScript-код на вебсторінку, яку переглядають інші користувачі.

SQL-ін'єкція – це тип атаки, коли зловмисник вводить шкідливий SQL-код у запити бази даних вебзастосунку.

GDPR (General Data Protection Regulation) – це закон Європейського Союзу про захист персональних даних, що набрав чинності 25 травня 2018 року. Він встановлює жорсткі вимоги до компаній щодо збору, обробки та зберігання даних громадян ЄС.

Кросбраузерність – це властивість вебсайту або вебзастосунку за можливістю однаково відображатися та функціонувати відповідно до поставленого завдання в усіх браузерах або ж поступово погіршуватися, коли функції браузера відсутні або недостатні.

RESTful API (Representational State Transfer Application Programming Interface) – це архітектурний стиль для розробки вебсервісів. Він базується на наборі принципів, які сервери і клієнти використовують для обміну даними через протокол HTTP.

ECMAScript – стандарт мови програмування, затверджений міжнародною організацією ECMA згідно зі специфікацією ECMA-262. Найвідомішими реалізаціями стандарту є мови JavaScript, JScript та ActionScript, які широко використовується у веброзробці.

SSR (Server-Side Rendering) – це підхід до рендерингу вебсторінок, при якому сервер генерує повністю готовий HTML-документ перед відправкою його клієнту. При використанні SSR користувач отримує готову до показу сторінку, що покращує швидкість початкового завантаження та SEO.

SSG (Static Site Generation) – це метод створення вебсайтів, при якому HTML-файли генеруються заздалегідь під час процесу збірки, а не при

кожному запиті. Статично згенеровані сторінки можуть бути розміщені на CDN, що забезпечує надзвичайно швидке завантаження.

JWT (JSON Web Token) – це формат токена, який використовується для передачі інформації між двома сторонами. Він зазвичай використовується для аутентифікації в вебзастосунках.

SPA (Single Page Application) – це вебдодаток або вебсайт, який взаємодіє з користувачем шляхом динамічного переписування поточної вебсторінки з новими даними з вебсервера замість методу за замовчуванням завантаження всіх нових сторінок

ВСТУП

У сучасному світі музика є важливою частиною життя багатьох людей, слугуючи джерелом натхнення, релаксації та самовираження. З розвитком цифрових платформ стрімінгові сервіси, такі як Spotify, стали невіддільною частиною повсякденного життя, дозволяючи мільйонам користувачів створювати персоналізовані плейлисти, відкривати нових виконавців і ділитися улюбленою музикою [1]. Однак, разом зі зростанням популярності музичних платформ, виникає потреба в інструментах для аналізу власних музичних уподобань, вивчення поведінки в прослуховуванні, а також обміну цією інформацією з іншими.

Попри те, що на ринку вже існують інструменти для аналізу музичних вподобань, більшість з них мають обмежену функціональність у своїх безплатних версіях [2]. Це створює бар’єр для повноцінного використання цих рішень, оскільки розширені можливості зазвичай доступні лише за підпискою або платними ліцензіями. Такий підхід не завжди відповідає очікуванням користувачів, які шукають зручний і доступний спосіб глибшого аналізу своєї музичної активності.

Метою даного дипломного проєкту є створення вебзастосунку для відстежування даних про музичні вподобання, який інтегрується зі Spotify API та забезпечує зручний доступ до розширеної аналітики музичних вподобань користувачів [3]. Вебзастосунок надаватиме користувачеві за детальну статистику прослуховувань, аналіз жанрів, виконавців та пісень, а також дозволятиме візуалізувати ці дані. Крім того, він сприятиме створенню музичних спільнот, де користувачі зможуть взаємодіяти, обмінюватися смаками та знаходити нову музику.

1. ОГЛЯД ІСНУЮЧИХ ПРОГРАМНИХ РІШЕНЬ

1.1. Огляд проблеми, що вирішується ПЗ

Не зважаючи на популярність стримінгового сервісу Spotify, досі існує потреба в якісних інструментах для аналізу власних музичних уподобань. Сучасні конкурентоспроможні сервіси надають інструменти для перегляду детальної статистики прослуховувань за визначені часові проміжки, відображення улюблених пісень, альбомів та виконавців користувача та візуалізації даних. Окрім цього, кожен продукт намагається бути оригінальним та прагне утримувати зацікавленість користувачів на собі. Така тенденція породжує багато проєктів, які прагнуть знайти групи своїх користувачів.

Через це наявні сервіси для аналізу музичних вподобань можна поділити на умовні групи:

- Візуальні дошки, що представляють окремий вищезазначений параметр з музичних вподобань.
- Застосунки, що націлені дати естетичне задоволення при користуванні ними та мінімальну кількість інформації зі статистики прослуховувань.
- Застосунки, що надають спектр можливостей для детального аналізу власних музичних вподобань.

Не дивлячись на те, що сервіси, які акцентують на візуальній або художній складовій, приваблюють певну частину користувачів, значна більшість все ж таки віддає перевагу платформам, що пропонують ширший спектр інструментів для глибинного аналізу їхніх музичних уподобань. Враховуючи це, розробка інтуїтивно зрозумілого та багатофункціонального інструменту, що відповідає потребам сучасної аудиторії, зацікавленої в отриманні детальної та всебічної інформації про власні музичні звички. Саме такий інструмент дозволить задовольнити зростаючий попит на персоналізовану статистику та аналітику у сфері цифрової музики.

1.2. Аналіз існуючих рішень

1.2.1. Застосунок stats.fm

stats.fm – це мобільний застосунок, доступний для IOS та Android, що також має вебзастосунок [4, 5]. Він допомагає користувачам аналізувати свої музичні вподобання, отримуючи детальну статистику прослуховувань зі Spotify. Застосунок має зручний і мінімалістичний інтерфейс користувача, що робить його популярним серед музичних ентузіастів, які прагнуть краще зрозуміти свої музичні смаки [6].

stats.fm надає глибокий аналіз музичних вподобань користувача через інтеграцію зі Spotify API та дозволяє користувачам переглядати статистику за різними критеріями, такими як кількість прослуховувань, найулюбленіші пісні, жанри та виконавці, а також відображати зміни в їхніх музичних смаках з часом.

До функціональних можливостей застосунку належать:

- інструменти отримання детальної статистики прослуховувань – користувачі можуть побачити свою музичну активність за різні періоди часу (день, тиждень, місяць, рік);
- аналіз музичних сподобань – відображення улюблених виконавців, жанрів та пісень;
- персоналізовані звіти – користувачі можуть переглядати звіти про свою музичну діяльність;
- візуалізація даних – графічне відображення статистики для легкого розуміння музичних вподобань; приєднання до місцевих ліг;
- інтерактивність – користувачі можуть порівнювати свою статистику з іншими та ділитися нею.

Одна з основних можливостей застосунку – надання інструментів для отримання детальної статистики музичних вподобань користувачів, зокрема детальна статистика прослуховувань.

Користувачі можуть переглядати наступну інформацію:

- найпопулярніші жанри пісень, які слухав користувач;
- найпопулярніші пісні, які слухав користувач;
- найпопулярніші виконавці, яких слухав користувач;
- останні пісні, що прослухав користувач.

Застосунок має фільтри для даних про музичні вподобання. Приклад результату відстежування найпопулярніших виконавців за останні чотири тижні можна переглянути на рис. 1.1.

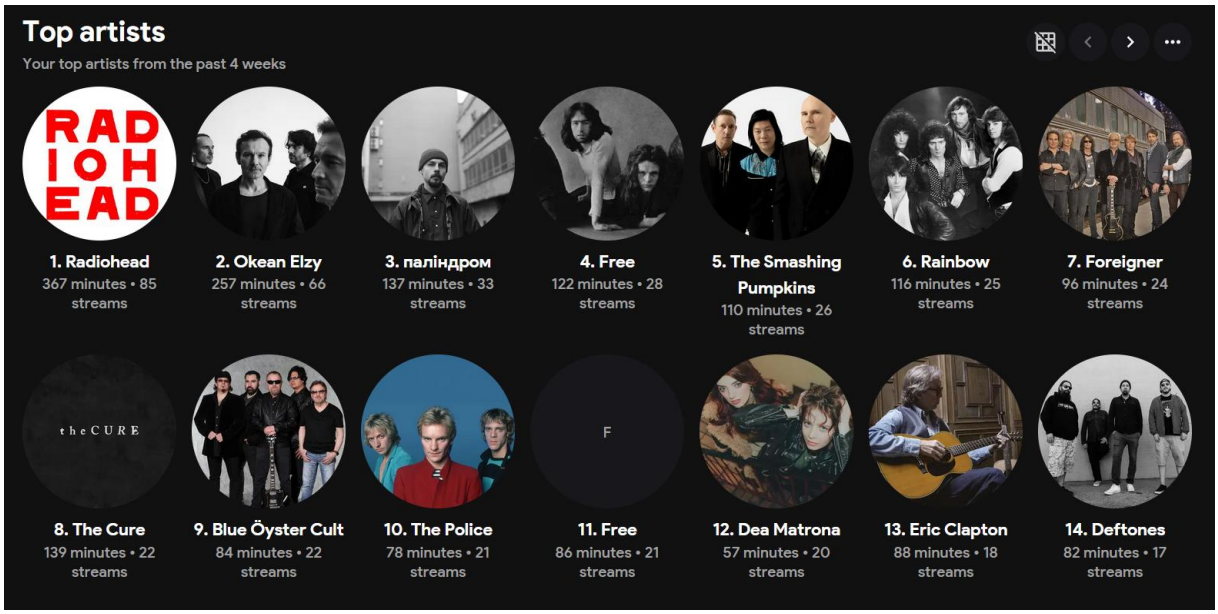


Рис. 1.1. Результат відстежування найпопулярніших виконавців за останні чотири тижні

У користувачів є можливість відстежувати останні пісні, які вони прослухали. Приклад результату відстежування останніх прослуханих пісень можна переглянути на рис. 1.2.

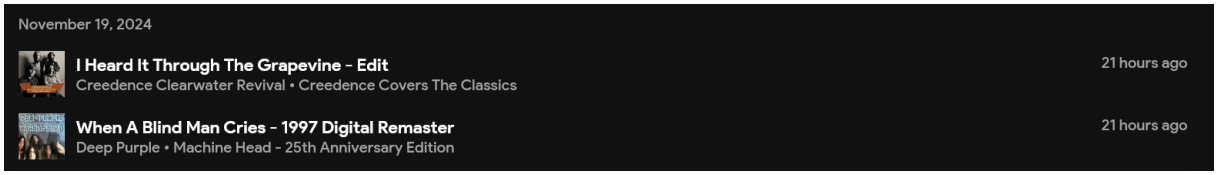


Рис. 1.2. Результат відстежування останніх прослуханих пісень

Окрім того, stats.fm має ще додаткові можливості, які вимагають Plus підписки. До таких можливостей можна віднести перегляд прослуховувань по годинах, приклад відстежування інформації наведено на рис. 1.3.

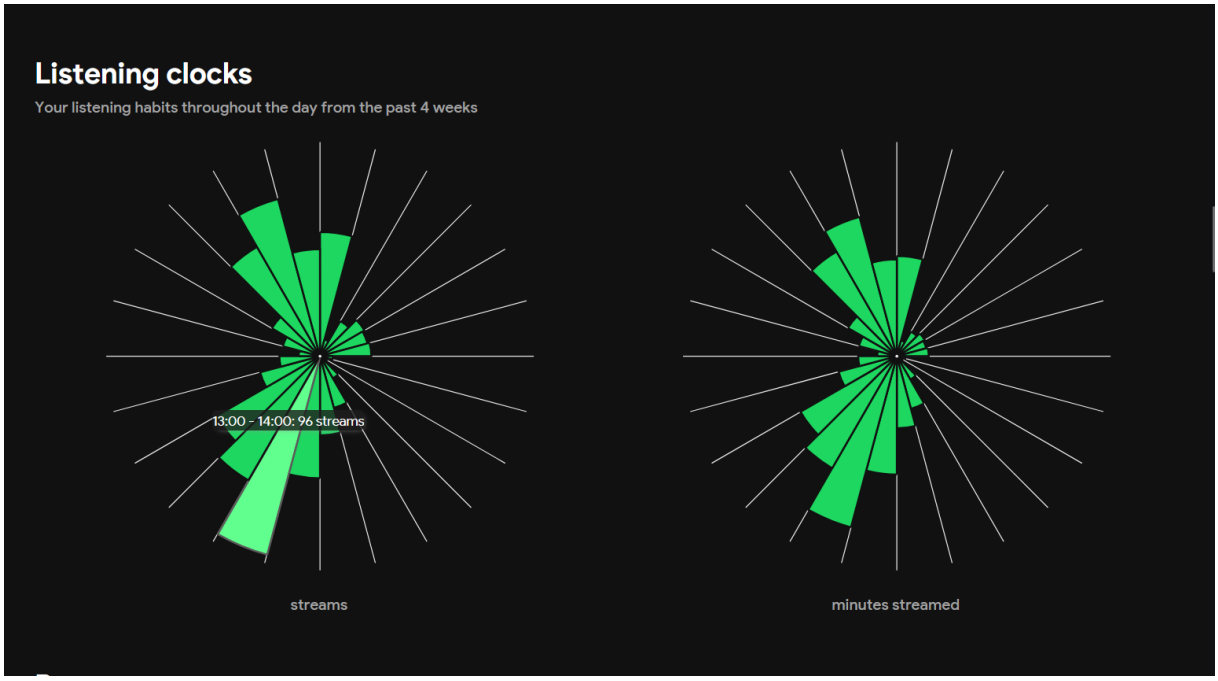


Рис. 1.3. Результат відстежування звички слухати музику упродовж 4 тижнів погодинно

Загалом, інтерфейс користувача stats.fm є зручним та має локалізації, проте відсутня світла тема та часто трапляються баги [7]. В мобільній версії без підписки присутня реклама та відсутній доступ до розширеної аналітики музичних вподобань користувачів.

1.2.2. Застосунок Obscutify Music

Obscurify Music – це вебзастосунок, який дозволяє користувачам аналізувати свої музичні вподобання, синхронізуючись зі Spotify. Його головною метою є показ унікальності музичного смаку користувача та надання персоналізованої статистики. Вебзастосунок має мінімалістичний інтерфейс та невелику авдиторію користувачів.

Функціональні можливості Obscurify Music:

- інструменти для аналізу музичних вподобань – застосунок аналізує прослуховування для визначення рівня «рідкісної» музики користувача порівняно з іншими користувачами Spotify;
- жанрова та емоційна аналітика – представлення жанрів музики користувача та розподіл емоційних характеристик;
- порівняння з іншими користувачами – обчислення унікальності музичного смаку у порівнянні з іншими користувачами;
- персоналізовані плейлисти – застосунок пропонує нові пісні чи виконавців, які можуть зацікавити користувача;
- історія музичного смаку – перегляд зміни музичних вподобань з часом.

Одна з основних можливостей вебзастосунку – надання статистики музичних вподобань, зокрема рівень унікальності музичного смаку. Користувачі можуть переглядати наступну інформацію:

- емоційні характеристики пісень;
- найпопулярніші пісні та жанри пісень, що слухав користувач;
- рейтинг унікальності музичного смаку користувача;
- унікальні пісні та виконавці, яких слухає користувач;
- діаграма прослуханих пісень в десятиліттях.

Інтерфейс перегляду статистики музичних вподобань в Obscurify Music відрізняється простотою, але водночас дещо поступається в різноманітності функціоналу та можливостей аналізу, порівняно з більш розширеними сервісами, такими як stats.fm. Зокрема, приклад візуалізації рейтингу жанрів користувача, як це представлено на рис. 1.4, демонструє обмежений набір даних та інструментів для детального вивчення музичних смаків, що робить процес аналізу менш глибоким та інформативним у порівнянні з альтернативними рішеннями.

Рис. 1.4. Перегляд рейтингу жанрів користувача

Привабливою особливістю Obscurify Music є функція аналізу оригінальності та непопулярності музики, яка приваблює користувачів, які прагнуть відрізнятися своїми музичними вподобаннями. Для оцінки цього аспекту вебзастосунок використовує власні алгоритми, які визначають, наскільки унікальним є вибір музики користувача. Зокрема, чим менш відомі або популярні артисти входять до його плейлистів та прослуховувань, тим вище відображається показник оригінальності на спеціальній діаграмі розподілу. У користувача є можливість відслідковувати два рейтинги: поточний, який відображає актуальну ситуацію, та загальний, що враховує весь період використання сервісу. Деталі відображення цієї діаграми розподілу, що показує співвідношення популярної та не дуже музики, наочно представлені на рис. 1.5.

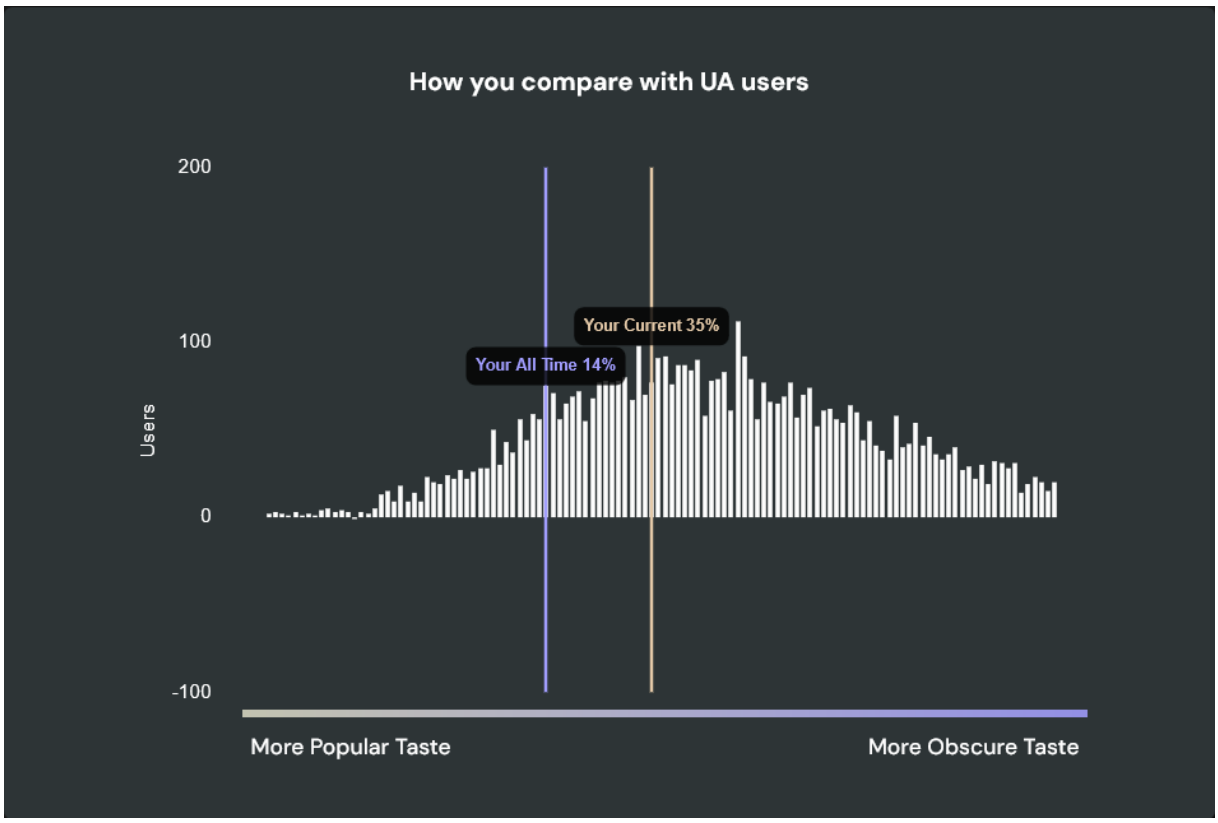


Рис. 1.5. Діаграма розподілу унікальності музичних вподобань

Вебзастосунок має схожий процес отримання статистики найпрослухованіших пісень та виконавців. Приклад візуалізації можна переглянути на рис. 1.6.

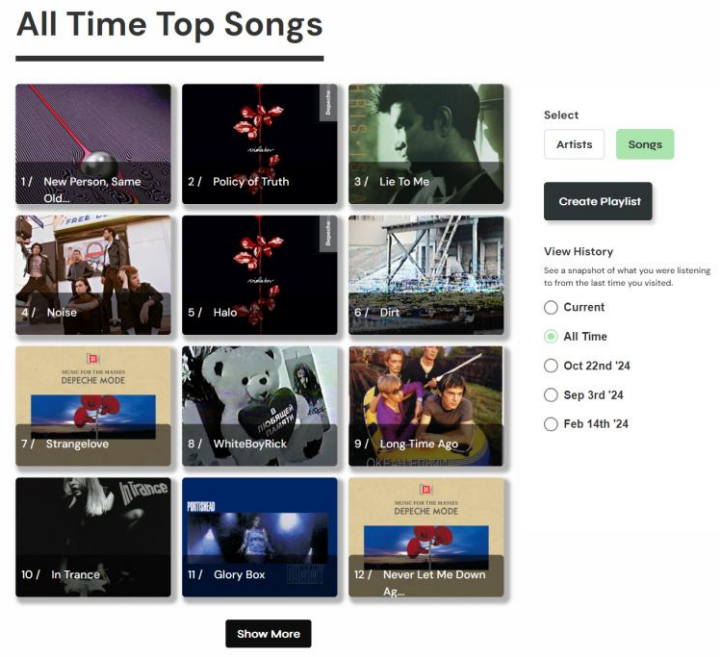


Рис. 1.6. Результат візуалізації найпопулярніших пісень за весь час

Попри оригінальність застосунку Obscurify Music, до його недоліків можна віднести наступне:

- недостатньо зручний інтерфейс з погляду на використання робочого простору;
- відсутність гнучкого перегляду статистики;
- відсутність доступу до розширеної аналітики музичних вподобань користувачів.

1.3. Результати аналізу

Провівши аналіз наявних рішень, можна зробити висновок, що ринок сервісів для аналізу музичних вподобань є відносно молодим, адже більшість з оглянутих рішень належать до категорії молодих стартапів. Проте для успішного виходу на цей ринок новий гравець має запропонувати конкурентні переваги, які дозволять досягти успіху.

Серед популярних сервісів, що надають інструменти для аналізу музичних вподобань, можна виділити stats.fm та Obscurify Music. Вебзастосунок Obscurify Music орієнтований на користувачів, які лише поверхнево цікавляться аналізом музичних уподобань, і тому має обмежений набір інструментів для глибокого аналізу. Натомість stats.fm пропонує ширший спектр можливостей для детального аналізу. Обидва сервіси мають свої унікальні переваги та недоліки.

Для того, щоб розроблюваний вебзастосунок зміг зайняти нішу на ринку інструментів для аналізу музичних вподобань, важливо зосередитися на унікальних функціях, які не пропонують конкуренти та інтеграції з іншими платформами, а також забезпечити інтуїтивно зрозумілий та простий користувацький інтерфейс. Окрім цього, надзвичайно важливо приділяти увагу стабільній та коректній роботі сервісу, оскільки це є одним із ключових факторів успіху будь-якого проєкту.

1.4. Перелік вимог до функціональності програмної системи

Після ретельного аналізу наявних сервісів, призначених для аналізу музичних вподобань, а також оцінки їхньої популярності серед користувачів, було визначено стратегічні цілі проєкту, які відображають очікувані результати та бажаний вплив на аудиторію. Ці цілі були деталізовані та структуровані у вигляді дерев цілей, які наочно демонструють взаємозв'язок між глобальними завданнями та конкретними досягненнями, що сприятимуть їх реалізації. Паралельно з цим, було сформовано чіткий перелік результатів, щоб забезпечити відповідність вебзастосунку потребам користувачів та вимогам ринку. Крім того, ключовим результатом проведеного аналізу стало формулювання вичерпних вимог до розроблюваної системи, що охоплюють функціональні, нефункціональні, технічні та організаційні аспекти. Ці вимоги слугуватимуть основою для подальшого проєктування, розробки та тестування вебзастосунку.

На рис. 1.7 зображено дерево цілей проєкту.

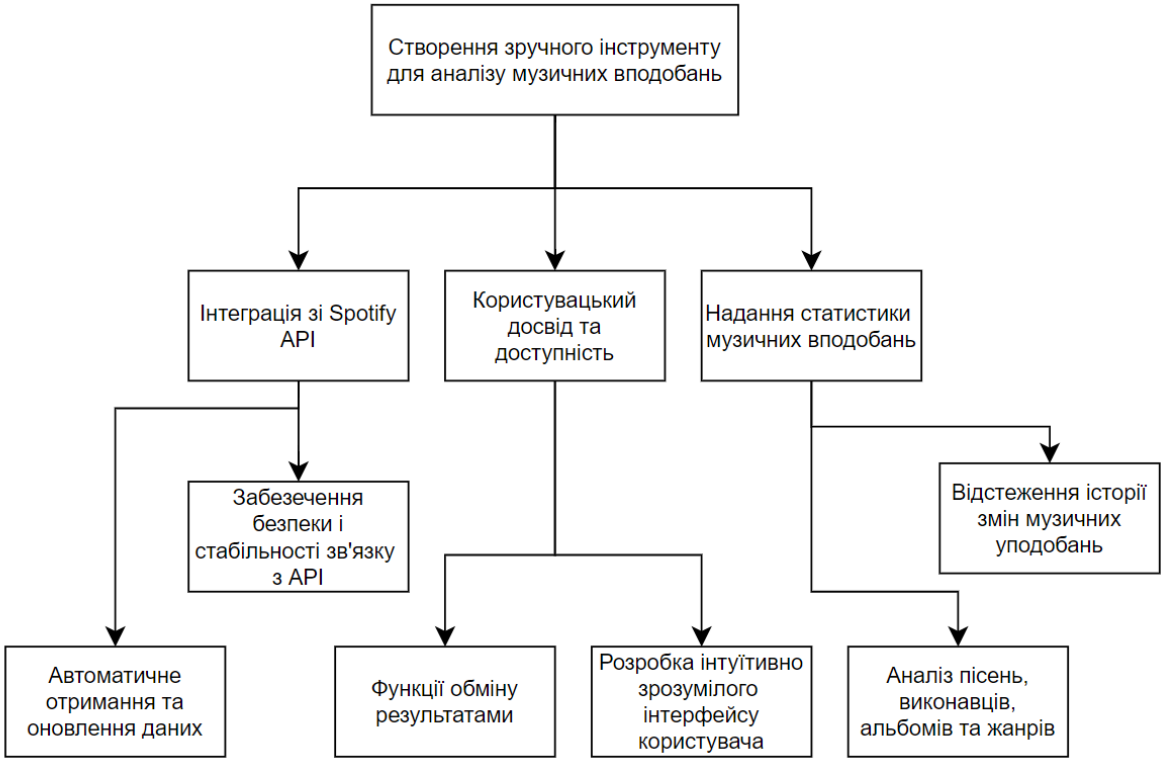


Рис. 1.7. Дерево цілей проєкту

На рис. 1.8 зображено дерево результатів проєкту.

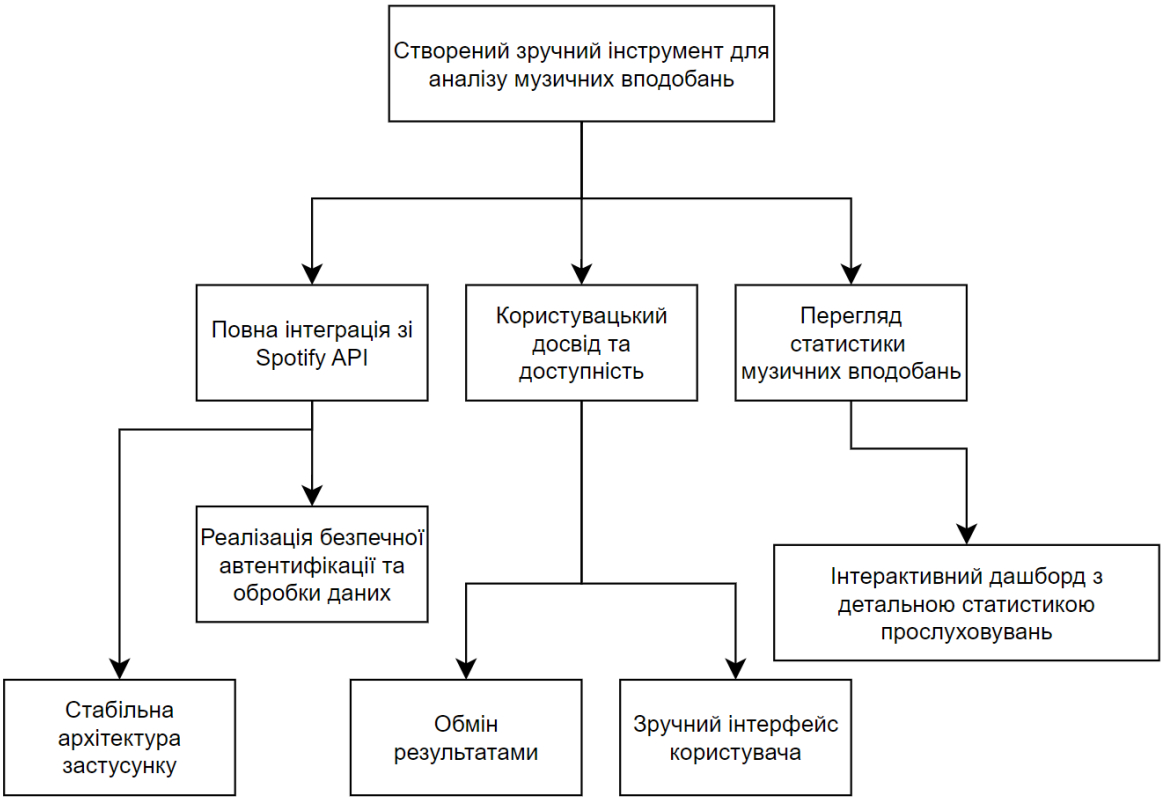


Рис. 1.8. Дерево результатів проєкту

З огляду на представлені діаграми, виділено наступні функціональні вимоги до розроблюваного сервісу:

- Реалізація автентифікації через обліковий запис Spotify з використанням OAuth 2.0.
- Забезпечення збереження та захисту даних користувачів.
- Реалізація автоматичного імпорту та оновлення даних користувача.
- Забезпечення можливості аналізу даних за різними періодами часу.
- Проведення аналізу даних з метою визначення основних музичних вподобань за різними параметрами (виконавець, альбом, жанр тощо).
- Реалізація можливості для обміну даними результатів користувацького аналізу.

- Використання механізмів обробки помилок при імпорті даних та взаємодії зі Spotify API.

На основі функціональних вимог можна скласти список основних вимог до розроблюваного застосунку:

- Можливість динамічного оновлення вмісту вебсторінок.
- Можливість автентифікації через обліковий запис Spotify.
- Можливість перегляду статистики прослуховування музики за різні проміжки часу.
- Можливість аналізувати свої музичні вподобання за піснями, альбомами, виконавцями та жанрами.
- Можливість перегляду своїх улюблених виконавців, жанри пісень, альбоми.
- Можливість перегляду історії прослуховувань.
- Забезпечення автоматичного імпорту та оновлення даних прослуховувань зі Spotify.

Вимоги до якості визначають технічні характеристики, що забезпечують належне функціонування системи. Ці вимоги для розроблюваного вебзастосунку є наступними:

- Відгук на дії користувача (фільтрація, сортування, оновлення даних) має відбуватися в межах 1-2 секунд.
- Система повинна обробляти не менше 500 запитів одночасно без значної деградації швидкодії.
- Система повинна автоматично відновлюватися після збоїв без втрати користувацьких даних.
- Система повинна підтримувати збільшення кількості користувачів без суттєвого погіршення продуктивності.
- Архітектура системи має забезпечувати можливість горизонтального масштабування при зростанні навантаження.

- Система повинна мати модульну структуру для розширення можливостей та інтеграції нових модулів при подальшій підтримці продукту.
- Інтерфейс має бути простим та зручним у використанні.
- Інтерфейс повинен коректно відображатися в браузерах Chrome, Firefox та Safari.

1.5. Перелік вимог до безпеки ПЗ

Вимоги до безпеки визначають заходи захисту даних та забезпечення конфіденційності користувачів. Для розроблюваного вебзастосунку, що відстежує дані про музичні вподобання, вони повинні бути наступними:

- Система повинна використовувати протокол HTTPS для захищеної передачі даних між клієнтом та сервером [9].
- Токени доступу повинні зберігатися у безпечному виді та оновлюватися [10].
- Обов'язкове використання протоколу OAuth 2.0 для безпечної аутентифікації через Spotify API, що забезпечує перевірку особистості користувача.
- Система повинна мати захист від CSRF та XSS, а також від SQL-ін'єкцій [11-13].
- Сесії мають бути обмежені по тривалості та автоматично завершуватися.
- Система повинна бути виконана відповідно до норм і стандартів щодо зберігання та обробки персональних даних згідно з GDPR [14].

1.6. Висновки до розділу

Проведений аналіз вимог до вебзастосунку «SpotiStats» дозволив сформувати комплексне бачення проєкту. Перелік вимог охоплює не лише функціональні можливості системи, але і якісні характеристики, які

забезпечуватимуть належне функціонування вебзастосунку та задоволення потреб користувачів.

При розробці вимог особливу увагу було приділено аспектам безпеки, що відповідають сучасним тенденціям захисту персональних даних. Запропоновані заходи з шифрування, безпечної автентифікації забезпечують відповідність розроблюваного програмного забезпечення міжнародним стандартам безпеки, що значно підвищить довіру користувачів та знизить ризики несанкціонованого доступу до чутливої інформації. Як показав аналіз, визначені функціональні вимоги повністю узгоджуються з деревом цілей та деревом результатів проєкту. Це свідчить про системний підхід до розробки вимог та забезпечує реалізацію всіх поставлених завдань без суттєвих відхилень від початкового бачення продукту.

Важливим результатом проведеної роботи є окреслення чітких меж проєкту. Такий підхід дозволяє ефективно розподілити доступні ресурси, сконцентрувавши їх на розробці ключових функцій продукту та уникаючи розпорошення на другорядні задачі, які не становлять основної цінності для користувачів.

Детальне опрацювання вимог до якості створює передумови для забезпечення високого рівня користувацького досвіду. Врахування аспектів продуктивності, надійності та зручності використання є критичним фактором для успішного позиціювання проєкту на ринку сервісів аналізу музичних вподобань, де конкуренція буде зростати з часом.

У процесі аналізу було виявлено, що орієнтація на модульну архітектуру та сучасні технології створює міцну основу для подальшого розвитку системи. Як відомо, такий підхід забезпечує гнучкість при інтеграції нових функціональних можливостей та адаптації до змін у вимогах користувачів чи технологічному середовищі. Врахування наявних часових та ресурсних обмежень дозволяє створити досяжний план реалізації.

2. ОБҐРУНТУВАННЯ ВИБОРУ ЗАСОБІВ РЕАЛІЗАЦІЇ

Розробка вебзастосунку для аналізу музичних вподобань на основі Spotify API вимагає ретельного підходу до вибору технологій та інструментів. Це фундаментальний етап, що визначає не лише швидкість розробки, але й довгострокову перспективу проєкту.

Правильно підібраний набір технологій суттєво впливатиме на темпи розробки продукту. Наприклад, використання фреймворків, які мають вбудовану підтримку для роботи з API (як-от Axios для HTTP-запитів до Spotify API) може зекономити десятки годин на написанні власного коду для обробки запитів. Для застосунку, що аналізує музичні вподобання, важливо обрати такі інструменти, які дозволять швидко опрацьовувати великі обсяги даних про прослуховування та преференції користувачів.

Надійність вебзастосунку сильно залежить від обраних технологій. Для проєкту з аналізу музичних даних критично важливо забезпечити:

- безперебійну роботу з API сервісу Spotify;
- швидке опрацювання та візуалізацію даних;
- здатність обробляти запити від багатьох користувачів одночасно;
- захист персональних даних користувачів.

Вебзастосунок повинен бути зрозумілим та привабливим для цільової аудиторії. Вибір технологій, які підтримують сучасні принципи UI/UX дизайну (як-от Material UI або Tailwind CSS), дозволить створити інтуїтивно зрозумілий інтерфейс для відображення складних даних про музичні вподобання. Важливо також враховувати кросбраузерність та адаптивність для різних пристроїв, оскільки користувачі можуть аналізувати свої музичні преференції як на мобільних пристоях, так і на настільних комп’ютерах [15].

Також необхідно враховувати інструменти для обробки обмежень API та можливості для автентифікації користувачів через OAuth. Обираючи технології для розробки, необхідно зважати на їхню перспективність та активність спільноти. Для проєкту з аналізу музичних вподобань важливо,

щоб обрані інструменти підтримувалися та регулярно оновлювалися, а також дозволяли легко впроваджувати нові функції в майбутньому. Новітні технології можуть надати конкурентні переваги, але потенційно нести ризики нестабільності або відсутності достатньої документації.

Таким чином, етап вибору технологічного стека для вебзастосунку аналізу музичних вподобань є стратегічно важливим і вимагає комплексного підходу з урахуванням специфіки проєкту, потреб користувачів та довгострокових перспектив розвитку.

2.1. Обґрунтування вибору мови програмування

Розробка вебсервісу вимагає ретельного підходу до вибору технологічного інструментарію. Архітектура такого програмного забезпечення традиційно поділяється на фронтенд та бекенд компоненти, кожен з яких потребує застосування спеціалізованих мов програмування. При виборі мов програмування необхідно враховувати не лише їхню популярність, але й сумісність зі специфічними вимогами проєкту, потенціал для ефективної реалізації функціональності та перспективи довгострокової підтримки.

2.2. Вибір технологій для розроблення серверної частини

2.2.1. Середовище Node.js

Node.js являє собою кросплатформне середовище виконання JavaScript, побудоване на JavaScript-рушії V8 від Google Chrome. Ця платформа дозволяє виконувати JavaScript-код поза межами браузера, що відкриває можливості для створення потужних серверних застосунків. До переваг Node.js можна віднести:

- завдяки рушію V8 та неблокуючій (asynchronous, non-blocking) моделі вводу/виводу, Node.js забезпечує високу швидкість обробки запитів, що особливо важливо для систем реального часу;

- розробники можуть використовувати JavaScript як на клієнтській, так і на серверній стороні, що спрощує процес розробки та знижує поріг входу;
- велика кількість готових модулів у менеджері пакетів npm значно пришвидшує розробку застосунків;
- Node.js працює на різних операційних системах (Windows, macOS, Linux), що полегшує розгортання та супровід застосунків;
- завдяки подієво-орієнтованій архітектурі, Node.js зручна у використанні застосунками, що активно взаємодіють із користувачем у реальному часі.

До недоліків платформи можна віднести наступні пункти:

- основний цикл подій Node.js є однопоточним, що може створювати проблеми при виконанні ресурсомістких операцій або блокуючого коду;
- обробка зображень або складні обчислення краще реалізовувати поза Node.js або за допомогою робітників (Worker Threads), що ускладнює архітектуру.

2.2.2. Express.js

Express.js є мінімалістичним, але водночас потужним вебфреймворком для платформи Node.js, який широко використовується для створення серверних застосунків та RESTful API [16]. Завдяки своїй простоті, гнучкості та широкій екосистемі, Express.js став стандартом де-факто в екосистемі Node.js для розробки серверної логіки. Він забезпечує розробників базовим інструментарієм для обробки HTTP-запитів, маршрутизації, взаємодії з базами даних, реалізації middleware-компонентів та керування конфігурацією застосунку. Express.js не нав’язує жорсткої структури, що дає можливість реалізовувати як прості, так і масштабовані високонавантажені рішення.

До переваг Express.js можна віднести:

- завдяки мінімалістичному підходу та чіткій документації, Express.js є зручним інструментом як для початківців, так і для досвідчених розробників;
- відсутність нав’язаних шаблонів чи архітектурних обмежень дозволяє адаптувати структуру застосунку відповідно до конкретних вимог проєкту;
- Express.js пропонує інтуїтивно зрозумілий API для налаштування маршрутизації HTTP-запитів, що спрощує створення масштабованих REST-інтерфейсів;
- фреймворк підтримує широкий спектр middleware для обробки запитів, авторизації, логування, обробки помилок тощо, що значно розширює його функціональність без необхідності створювати все з нуля.

Недоліками вебфреймворку Express.js є:

- на відміну від деяких фреймворків, Express.js не містить вбудованих засобів для автентифікації, авторизації, управління сесіями, обробки файлів чи валідації, що вимагає підключення сторонніх рішень;
- в умовах високих навантажень або складних обчислень, Express.js може поступатися в продуктивності іншим, більш оптимізованим фреймворкам.

2.2.3. TypeScript

TypeScript являє собою відкриту мову програмування, розроблену компанією Microsoft, яка є надмножиною JavaScript. Вона з’явилася як відповідь на обмеження JavaScript при розробці масштабних застосунків. TypeScript додає до JavaScript статичну типізацію та інші можливості об’єктноорієнтованого програмування, зберігаючи при цьому повну сумісність із JavaScript кодом.

Основною особливістю TypeScript є компіляція у стандартний JavaScript, що забезпечує виконання коду в будь-якому середовищі, яке підтримує JavaScript. Компілятор TypeScript перетворює написаний код у JavaScript відповідно до обраної цільової версії стандарту ECMAScript, що дозволяє розробникам використовувати сучасні можливості мови, не турбуючись про сумісність із різними браузерами [17].

Одним із найвагоміших аргументів на користь TypeScript для розробки є впровадження статичної типізації. На відміну від JavaScript, де типи визначаються під час виконання програми, TypeScript дозволяє визначати типи даних на етапі написання коду.

З особливостей TypeScript можна винести такі переваги:

- статична перевірка типів у TypeScript перевіряє відповідність на етапі компіляції, що допомагає виявити потенційні помилки до виконання коду;
- інтерфейси та типи TypeScript створюють чіткі контракти між різними частинами застосунку, що знижує ризик непорозумінь та помилок при взаємодії компонентів;
- TypeScript дозволяє впевнено змінювати структуру коду, оскільки компілятор виявить місця, які потребують оновлення після змін;
- типи та інтерфейси слугують документацією, що значно спрощує розуміння коду іншими розробниками або при поверненні до проєкту після перерви;
- мова програмування стимулює використання чітких абстракцій та інтерфейсів, що сприяє кращій організації коду;
- TypeScript швидко адаптується до нових можливостей JavaScript, дозволяючи використовувати новітні функції мови.

Недоліками TypeScript є:

- Деякі динамічні патерни JavaScript складно типізувати в TypeScript. У таких випадках можна використовувати гнучкіші підходи до

типізації або виділяти не типізовані частини коду у відокремлені модулі.

- TypeScript не може виявити деякі типи логічних помилок або помилок, пов’язаних із виконанням коду в середовищі браузера. У таких випадках доцільно використовувати юніт-тести та інтеграційні тести.
- Декларації типів не повністю відповідають фактичній поведінці деяких бібліотек.

Статична типізація, покращений досвід розробки, структурована архітектура та глибока інтеграція з екосистемою Next.js значно перевищують потенційні недоліки, пов’язані з додатковою складністю та кривою навчання.

2.3. Вибір технологій для розроблення клієнтської частини

У світі розробки вебзастосунків основними технологіями залишаються JavaScript, CSS та HTML. Ці технології вже давно стали стандартом для розробки функціонального інтерфейсу завдяки своїй гнучкості. Упродовж останніх років розвиток вебзастосунків супроводжується еволюцією використовуваних інструментів. Результати дослідження State of JS 2024 демонструють, що розробники активно обирають JavaScript-фреймворки для спрощення створення складних інтерфейсів. Найбільшу популярність серед JavaScript-фреймворків має React. Цей фреймворк залишається фаворитом завдяки своїй простоті використання, широкій екосистемі та зручності при розробці як невеликих, так і масштабних проєктів. Крім React, розробниками активно використовується Angular, який добре підходить для великих комерційних рішень, завдяки своїй структурованості та інтегрованим можливостям. Також значну увагу приділяють Vue.js, який поєднує простоту та ефективність, що особливо важливо для проєктів, де потрібно швидко отримати результат без надмірної складності.

Svelte, Solid чи Qwik поки що менш розповсюджені, але вони представляють інтерес завдяки інноваційному підходу до розробки та оптимізації продуктивності застосунку.

Ринок веброзробки свідчить про увагу до створення компонентного підходу, який дозволяє не лише розбивати інтерфейс на невеликі, окремі частини та спрощує подальше модифікування та тестування.

2.3.1. Next.js

Next.js – це сучасний фреймворк для розробки вебзастосунків з відкритим кодом, створений приватною компанією Vercel, який побудований на базі бібліотеки React. Він спрощує створення як традиційних, так і складних інтерфейсів завдяки вбудованим засобам для рендерингу на стороні сервера (SSR) та генерації статичних сторінок (SSG) [18-19]. Використання Next.js дозволяє розробникам оптимізувати продуктивність застосунку, покращити SEO-результати та зменшити час завантаження сторінок шляхом попереднього рендерингу контенту.

Основні можливості Next.js охоплюють наступні аспекти:

- серверний рендеринг та статична генерація: Next.js підтримує як динамічний серверний рендеринг, так і генерацію статичних сторінок під час збірки проєкту, що дозволяє оптимізувати час завантаження сторінки;
- компонентна архітектура на базі React: Next.js інтегровано з React, тому такі потужності, як декларативне описання інтерфейсу чи можливості повторного використання компонентів доступні у процесі розробки;
- фреймворк надає вбудовану систему маршрутизації, де кожен файл у каталозі «pages» автоматично стає маршрутом вебзастосунку. Крім цього, Next.js дозволяє створювати API-шлюзи в одному проєкті, що сприяє інтеграції бекенд-логіки без необхідності використання окремого серверного середовища;

- Next.js дозволяє значно покращити час завантаження сторінок завдяки вбудованим механізмам оптимізації;
- Next.js має можливість інтеграції з NextAuth.js;
- Next.js сумісний із сучасними хмарними сервісами для розгортання, що сприяє швидкому запуску проєкту з мінімальними зусиллями;
- підтримка TypeScript.

Основні переваги використання Next.js включають:

- наявність усіх необхідних інструментів для створення сучасного вебзастосунку без необхідності використання численних зовнішніх бібліотек чи додаткових налаштувань;
- завдяки серверному рендерингу та статичній генерації Next.js забезпечує швидкий час відгуку сторінки;
- інтеграція з NextAuth.js значно спрощує інтеграцію механізмів авторизації, що дозволяє зосередитися на основній функціональності застосунку, забезпечуючи при цьому високий рівень безпеки;
- проєкти на базі Next.js легко підтримувати та розширювати завдяки вбудованій маршрутизації та компонентній архітектурі.

Next.js є надійним, масштабованим та зручним рішенням для розробки вебзастосунку.

2.3.2. NextAuth.js

NextAuth.js являє собою бібліотеку з відкритим кодом, розроблену для спрощення та оптимізації процесу впровадження автентифікації в вебзастосунки, створені на базі фреймворку Next.js. Однією з ключових переваг NextAuth.js є її широка підтримка різноманітних провайдерів автентифікації, що дозволяє розробникам легко інтегрувати різні методи входу для користувачів.

До переваг відносяться:

- безпека – використання стандартизованого протоколу OAuth забезпечує високий рівень захищеності даних користувача, оскільки жодні конфіденційні дані (наприклад, пароль) не передаються безпосередньо між застосунком та користувачем;
- простота реалізації – NextAuth.js пропонує готову конфігурацію для інтеграції з Spotify, що значно зменшує час розробки та зусилля при впровадженні автентифікації;
- бібліотека дозволяє додавання та налаштовування провайдерів автентифікації, що дає можливість адаптувати систему під специфічні вимоги проєкту.

Завдяки можливості інтеграції з Spotify, NextAuth.js дає змогу створити зручний та безпечний механізм входу до вебзастосунку, що дозволяє ефективно управляти сесіями користувачів.

2.3.3. Material UI

Material UI є однією з найпопулярніших бібліотек для розробки вебінтерфейсів на базі React, що реалізує принципи дизайну Google Material Design. Вона пропонує широкий спектр готових візуальних компонентів, які відповідають сучасним стандартам, дозволяючи розробникам швидко створювати привабливі, адаптивні та зручні для користувача інтерфейси. Завдяки своїй архітектурі, Material UI забезпечує легкість інтеграції в проєкти, підвищує продуктивність розробки та сприяє єдиному стилю оформлення застосунку. В результаті використання цієї бібліотеки розробники отримують інструмент, який дозволяє не лише прискорити розробку, а й гарантувати високий рівень користувацького досвіду.

Бібліотека покликана створити інтуїтивно зрозумілі інтерфейси, що допомагають користувачам швидко орієнтуватися у застосунку. Завдяки Material UI компоненти вже попередньо стилізовані та враховують рекомендації Google, що робить адаптацію дизайну під індивідуальні

потреби проєкту максимально простою.

Матеріал UI підтримує різноманітні сценарії використання – від створення простих вебсторінок до комплексних односторінкових застосунків із підтримкою адаптивного дизайну. Додатковою перевагою є легкість кастомізації, що дозволяє змінювати кольорові схеми та стилі компонентів за допомогою інтегрованої системи тем. Це дозволяє розробникам не лише дотримуватися встановленого стандарту, а й втілювати власні креативні ідеї, пристосовуючи візуальне представлення до особливостей бренду або вимог проєкту.

Основні переваги використання Material UI можна узагальнити наступним чином:

- компоненти відповідають рекомендаціям Material Design Google;
- готові компоненти значно скорочують час створення інтерфейсу;
- повна сумісність та простота впровадження у проєкти, що використовують React;
- компоненти бібліотеки оптимізовані для різних пристроїв і екранів.

Material UI – ефективний інструмент для розробки вебзастосунків, що поєднує естетичність, функціональність та технологічну надійність.

2.3.4. Tailwind CSS

Tailwind CSS являє собою сучасний CSS-фреймворк, який відрізняється утилітарним підходом до написання стилів. Замість традиційного створення окремих CSS-класів для кожного елемента, Tailwind надає розгалужений набір класів, що дозволяють безпосередньо задавати стилі елементів у розмітці. Такий підхід сприяє створенню гнучких, адаптивних та високопродуктивних інтерфейсів, зменшуючи залежність від попередньо визначених шаблонів та готових компонентів.

Однією з ключових особливостей Tailwind CSS є високий ступінь кастомізації. Розробники можуть налаштовувати стандартні налаштування конфігураційного файлу, створюючи власні палітри кольорів, просторові

відступи, типографіку та інші візуальні параметри, що відповідають вимогам конкретного проєкту. Завдяки цьому Tailwind дозволяє досягти унікального стилю інтерфейсу, зберігаючи високий рівень гнучкості при роботі зі складними дизайнами.

При використанні Tailwind CSS разом із Material UI можливо об’єднати переваги обох підходів. Material UI забезпечує швидке створення базових компонентів згідно з принципами Material Design, а Tailwind дозволяє додатково тонко налаштувати їх зовнішній вигляд, виконуючи кастомізацію за допомогою утилітарних класів. Ця комбінація сприяє як збереженню єдиного стилю застосунку, так і зменшенню часових витрат на адаптацію готових компонентів до конкретних дизайнерських рішень.

Основні переваги використання Tailwind CSS в контексті сумісної роботи з Material UI можна підсумувати наступним чином:

- Tailwind дозволяє створювати індивідуальну палітру стилів для тонкої кастомізації компонентів Material UI;
- утилітарний підхід дозволяє швидко визначати стилі без необхідності написання окремих CSS-файлів;
- завдяки використанню лише необхідних класів, загальний обсяг стилів зменшується, що позитивно впливає на швидкість завантаження;
- Tailwind CSS легко інтегрується з React і Material UI, що забезпечує єдиний стиль та високий рівень підтримки сучасних стандартів веброзробки.

Material UI відкриває можливості для більш тонкого налаштування та кастомізації дизайну, що сприяє досягненню єдиного стилістичного рішення у вебзастосунках.

2.4. Висновки

Вибір засобів реалізації – це ключовий етап розроблення, що визначає ефективність, функціональність, масштабованість та підтримку застосунку.

Головним критерієм при виборі інструментів розроблення має бути їхня здатність вирішувати поставлені завдання. Також слід враховувати очікувані навантаження, з котрими має впоратись застосунок, та швидкість розроблення при використанні різних інструментів.

У цьому розділі розглянуто інструменти, котрі будуть використані для реалізації дипломного проєкту. Для створення серверної частини застосунку вирішено використовувати платформу Node.js із фреймворком Express.js та мовою програмування TypeScript. Ця комбінація забезпечить високу швидкість обробки запитів завдяки асинхронній моделі, статичну типізацію для зменшення помилок при розробці та гнучкість при створенні RESTful API для взаємодії зі Spotify API.

Основою для створення клієнтської частини обрано React із фреймворком Next.js, що надає переваги серверного рендерингу, статичної генерації сторінок та вбудованої маршрутизації. Для автентифікації користувачів буде використано бібліотеку NextAuth.js, яка забезпечить безпечну інтеграцію із сервісом Spotify через протокол OAuth. Для стилізації користувацького інтерфейсу вирішено комбінувати Material UI та Tailwind CSS, що дозволяє швидко створювати привабливі компоненти з можливістю тонкого налаштування дизайну. Ця комбінація інструментів забезпечить ефективну розробку інтуїтивно зрозумілого та адаптивного інтерфейсу для аналізу музичних вподобань користувачів.

Обрані технології відповідають сучасним тенденціям веброзробки, мають активну спільноту підтримки, добре документовані та оптимізовані для розробки масштабованих вебзастосунків, що гарантує їхню довгострокову підтримку, можливість оновлення та адаптацію до майбутніх вимог. Активна спільнота також забезпечить швидке вирішення будь-яких проблем, котрі можуть виникнути під час розробки та підтримки проєкту.

3. РОЗРОБЛЕННЯ ВЕБЗАСТОСУНКУ

3.1. Загальний опис програми

Програмне забезпечення реалізоване у вигляді вебзастосунку та може бути розділене на дві суттєві частини: серверну (back-end) та клієнтську (front-end). Узагальнена структурна організація вебзастосунку представлена на рис. 3.1.

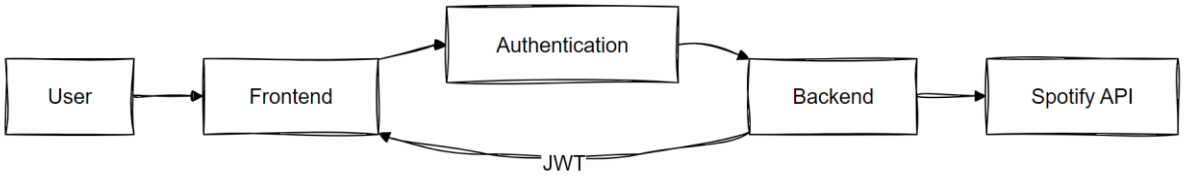


Рис. 3.1. Узагальнена структурна організація вебзастосунку

Завершивши загальний огляд функціональної структури вебзастосунку, логічним кроком є перехід до глибшого дослідження його архітектурних особливостей, які визначають ефективність, масштабованість та надійність системи. На рисунку 3.1 представлено узагальнену структуру організації вебзастосунку, яка ілюструє ключові компоненти та їх взаємозв'язки, слугуючи відправною точкою для подальшого аналізу. Проте для досягнення повного розуміння принципів роботи системи, необхідно ретельно вивчити специфіку кожного з цих компонентів, розглянути їхні внутрішні механізми та дослідити особливості взаємодії між ними. Для цього буде використано детальний опис архітектури, що представлений на рис. 3.2, який дозволить розібрати кожен елемент системи окремо, визначити його роль та функціональне призначення. Такий підхід дозволить сформувати цілісне уявлення про архітектуру системи та забезпечити основу для прийняття обґрунтованих рішень щодо її подальшої оптимізації та розвитку.

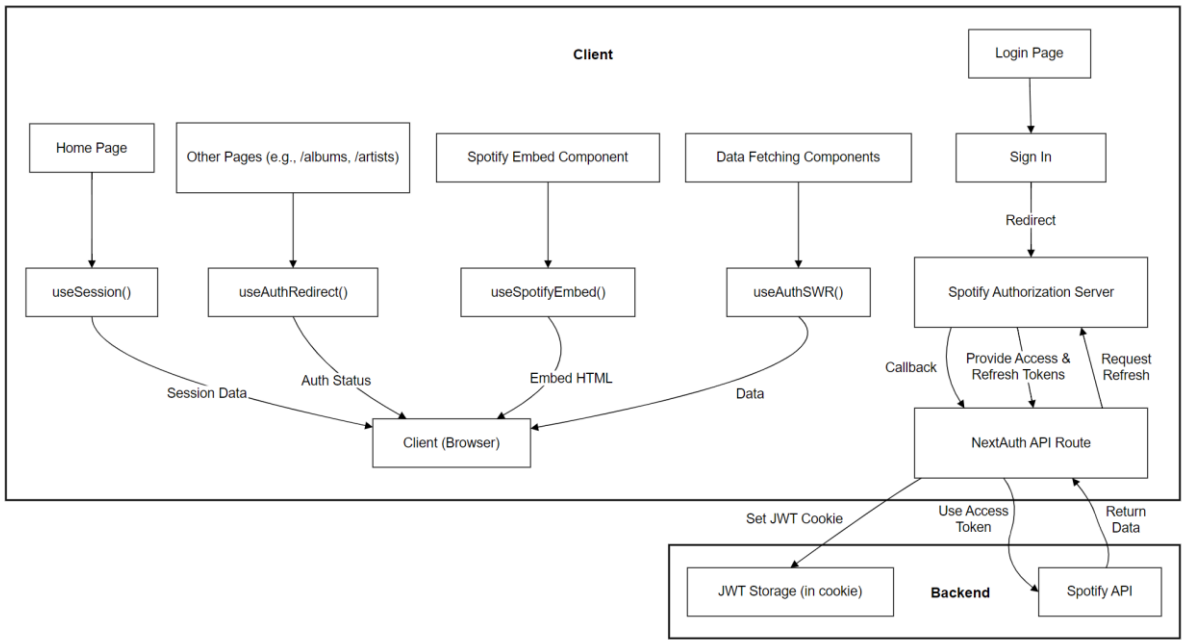


Рис. 3.2. Архітектура вебзастосунку

При першому переході на вебзастосунок «SpotiStats» користувач автоматично перенаправляється на сторінку авторизації. Це забезпечує безпеку доступу та персоналізацію сервісу для кожного користувача. Якщо користувач намагається перейти на будь-яку іншу сторінку без авторизації, система знову редиректить його на авторизацію. Авторизація здійснюється через Spotify OAuth 2.0, після чого користувач перенаправляється на головну сторінку застосунку, де він отримує початковий огляд своєї музичної статистики.

Головна сторінка надає користувачеві загальну статистику прослуховувань за певний проміжок часу, а також нещодавню історію прослуховувань. Це дозволяє швидко переглянути свою музичну активність та визначити тенденції у виборі пісень. Інтерфейс містить навігаційну панель у верхній частині, яка включає наступні посилання: Overall, Songs, Albums, Artists, Genres. Посилання Overall містить загальну статистику за вибраний період, що відображає зведену інформацію про активність користувача. Відповідно, Songs – детальна статистика прослуховуваних пісень, де можна переглядати найпопулярніші треки; Albums – статистика

альбомів, що дає можливість аналізувати слухацькі вподобання за альбомами; Artists – статистика виконавців, що допомагає зрозуміти, яких музикантів користувач слухає найчастіше; Genres – вподобання користувача за жанрами.

Ліворуч на панелі навігації розташоване лого застосунку, а праворуч є зображення профілю користувача, отримане з його облікового запису Spotify, а також кнопка виходу (Log out), яка дозволяє завершити сесію та вийти з системи.

Кожна сторінка, доступна через навігаційну панель, дозволяє користувачеві переглядати детальну статистику з можливістю застосування фільтрів за часом та кількістю відображуваних результатів. Наприклад, у розділі Songs можна переглянути список найпопулярніших пісень за вибраний період, а також обмежити кількість відображених результатів. Аналогічно в розділах Albums, Artists та Genres можна переглядати альбоми та виконавців, що були найчастіше прослуховувані в зазначений період часу.

Для розробки вебзастосунку «SpotiStats» було обрано клієнт-серверну архітектуру на основі Next.js. Серверна частина відповідає за автентифікацію та авторизацію користувачів через NextAuth.js, взаємодію зі Spotify API та обробку запитів від клієнтської частини. Клієнтська частина, реалізована як односторінковий застосунок, забезпечує відображення даних та взаємодію з користувачем. Таке архітектурне рішення дозволяє оптимально розподілити функціональність та забезпечити ефективну роботу застосунку.

Інтеграція з Spotify API через серверну частину надає додатковий рівень безпеки, оскільки конфіденційні дані, такі як токени доступу, зберігаються на сервері та не передаються безпосередньо клієнту. Реалізація механізму оновлення токенів доступу на серверній стороні забезпечує безперервність роботи застосунку та стабільний доступ до даних Spotify.

Клієнт-серверна архітектура спрощує розробку та розгортання,

дозволяючи швидше інтегрувати нові можливості та ефективно масштабувати застосунок за потреби. Завдяки використанню Next.js для обох частин застосунку, забезпечується єдиний підхід до розробки та узгодженість технологічного стека.

«SpotiStats» забезпечить зручний та інтуїтивно зрозумілий інтерфейс для перегляду та аналізу власних музичних вподобань, дозволяючи користувачеві отримати більше інформації про свої музичні звички. Простота у використанні та гнучкість налаштувань дають змогу кожному користувачу налаштувати відображення інформації відповідно до своїх потреб. Завдяки інтеграції з Spotify користувач може отримати максимальну користь від застосунку та зробити процес взаємодії з музикою ще більш захопливим та інформативним.

3.2. Аналіз функціональних вимог до вебзастосунку

Після аналізу наявних рішень було сформовано перелік функціональних вимог, необхідних для коректної роботи застосунку. Визначення цих вимог дозволило встановити їхню пріоритетність, що відображено у таблиці 3.1.

Таблиця 3.1

Функціональні вимоги до вебзастосунку

Код вимоги	Назва вимоги	Пріоритет вимоги
Q-1	Автентифікація через Spotify (OAuth 2.0)	1
Q-2	Збереження та захист даних користувачів	1
Q-3	Автоматичний імпорт даних користувача зі Spotify	1
Q-4	Автоматичне оновлення даних користувача при зміні історії прослуховувань	2

Продовження табл. 3.1

Q-5	Відображення історії прослуховувань за різні періоди	2
Q-6	Аналіз музичних уподобань за виконавцями	1
Q-7	Аналіз музичних уподобань за альбомами	1
Q-8	Аналіз музичних уподобань за жанрами	2
Q-9	Візуалізація результатів аналізу у вигляді графіків та діаграм	2
Q-10	Можливість поділитися результатами аналізу в соціальних мережах	3
Q-11	Механізми обробки помилок при імпорті даних	1
Q-12	Механізми обробки помилок при взаємодії зі Spotify API	1
Q-13	Автентифікація через Spotify (OAuth 2.0)	1

3.3. Особливості реалізації серверної частини застосунку

Серверна частина застосунку відповідає за опрацювання запитів від клієнтської частини, взаємодію з базами даних, реалізацію бізнес-логіки та забезпечення безпеки. Реалізація серверної частини проєкту зосереджена на забезпеченні механізмів автентифікації та авторизації, що є критичними компонентами серверної частини вебзастосунку. У розробленому застосунку використовується NextAuth.js – бібліотека з відкритим кодом, призначена для організації автентифікації в застосунках на базі Next.js. Дана бібліотека спрощує інтеграцію з різноманітними провайдерами автентифікації та забезпечує гнучкість у налаштуванні процесів автентифікації та сесій користувачів.

Реалізація автентифікації через NextAuth.js передбачає конфігурацію різних параметрів, враховуючи специфіку взаємодії з обраним провайдером, котрим для даного застосунку є Spotify. Однією з ключових переваг NextAuth.js є підтримка OAuth 2.0 – сучасного протоколу авторизації, що

дозволяє надавати обмежений доступ до ресурсів користувача без необхідності розкриття облікових даних. Для правильного функціонування застосунку визначено набір необхідних дозволів (scopes), які зазначаються під час автентифікації користувача. Серед них: «user-read-email» для доступу до електронної пошти користувача, «user-top-read» для отримання інформації про улюблених виконавців та пісні, а також «playlist-read-private», «playlist-modify-private» та «playlist-modify-public» для взаємодії з плейлистами користувача.

Важливим аспектом реалізації автентифікації є управління токенами доступу, які надаються провайдером автентифікації. NextAuth.js використовує JWT для зберігання інформації про сесію користувача [20]. Під час ініціювання сесії створюється JWT, який містить інформацію про користувача та токени доступу до Spotify API. Для забезпечення безперервності доступу до API реалізовано механізм оновлення токенів доступу на основі токенів оновлення (refresh tokens). Даний механізм особливо важливий, оскільки токени доступу мають обмежений термін дії, який визначається провайдером OAuth. Коли термін дії токена доступу закінчується, застосунок автоматично надсилає запит до Spotify API для отримання нового токена доступу, використовуючи збережений токен оновлення. Оновлений токен доступу зберігається в JWT і використовується для подальших запитів до API.

Для забезпечення доступу до інформації про сесію користувача на клієнтській стороні застосунку використовується функція session NextAuth.js. Дана функція дозволяє трансформувати дані з JWT у формат, доступний для використання в компонентах клієнтської частини. Це забезпечує безперешкодний доступ до необхідної інформації, такої як токен доступу та помилки, які можуть виникнути під час автентифікації. Важливо зазначити, що доступ до даних сесії на клієнтській стороні здійснюється через спеціальний хук useSession, який надається бібліотекою NextAuth.js. Даний хук дозволяє отримати актуальну інформацію про стан сесії

користувача, включаючи наявність активної сесії та дані користувача.

Реалізований механізм автентифікації також передбачає використання захищених маршрутів, які доступні лише для автентифікованих користувачів. Для цього використовується middleware NextAuth.js, який дозволяє перехоплювати запити до захищених маршрутів та перевіряти наявність активної сесії користувача. У випадку відсутності активної сесії, користувач перенаправляється на сторінку автентифікації. Даний підхід забезпечує надійний контроль доступу до захищених ресурсів застосунку та підвищує загальний рівень безпеки.

З метою забезпечення надійного захисту конфіденційної інформації, зокрема ідентифікатора клієнта (Client ID) та секретного ключа клієнта (Client Secret) Spotify, які є критично важливими для автентифікації та доступу до даних, у вебзастосунку використовується механізм змінних середовища. Дані змінні зберігаються у окремому файлі з назвою .env, який навмисно виключається з системи контролю версій (наприклад, Git). Це дозволяє запобігти випадковому розголошенню чутливих даних, які могли б потрапити у публічний доступ в разі завантаження репозиторію у відкритий доступ. Доступ до значень, що зберігаються у змінних середовища, здійснюється через вбудований об’єкт process.env, який надається платформою Next.js, що спрощує процес отримання необхідних даних. Такий підхід не лише підвищує загальний рівень безпеки вебзастосунку, захищаючи його від несанкціонованого доступу, але й значно спрощує процес конфігурації та розгортання застосунку в різних середовищах (розробки, тестування, продакшн), де можуть використовуватись різні значення змінних. Схематичне представлення процесу OAuth 2.0 автентифікації при взаємодії зі Spotify API, який є ключовим елементом для забезпечення безпечного доступу до даних, наочно проілюстровано на рисунку 3.2.

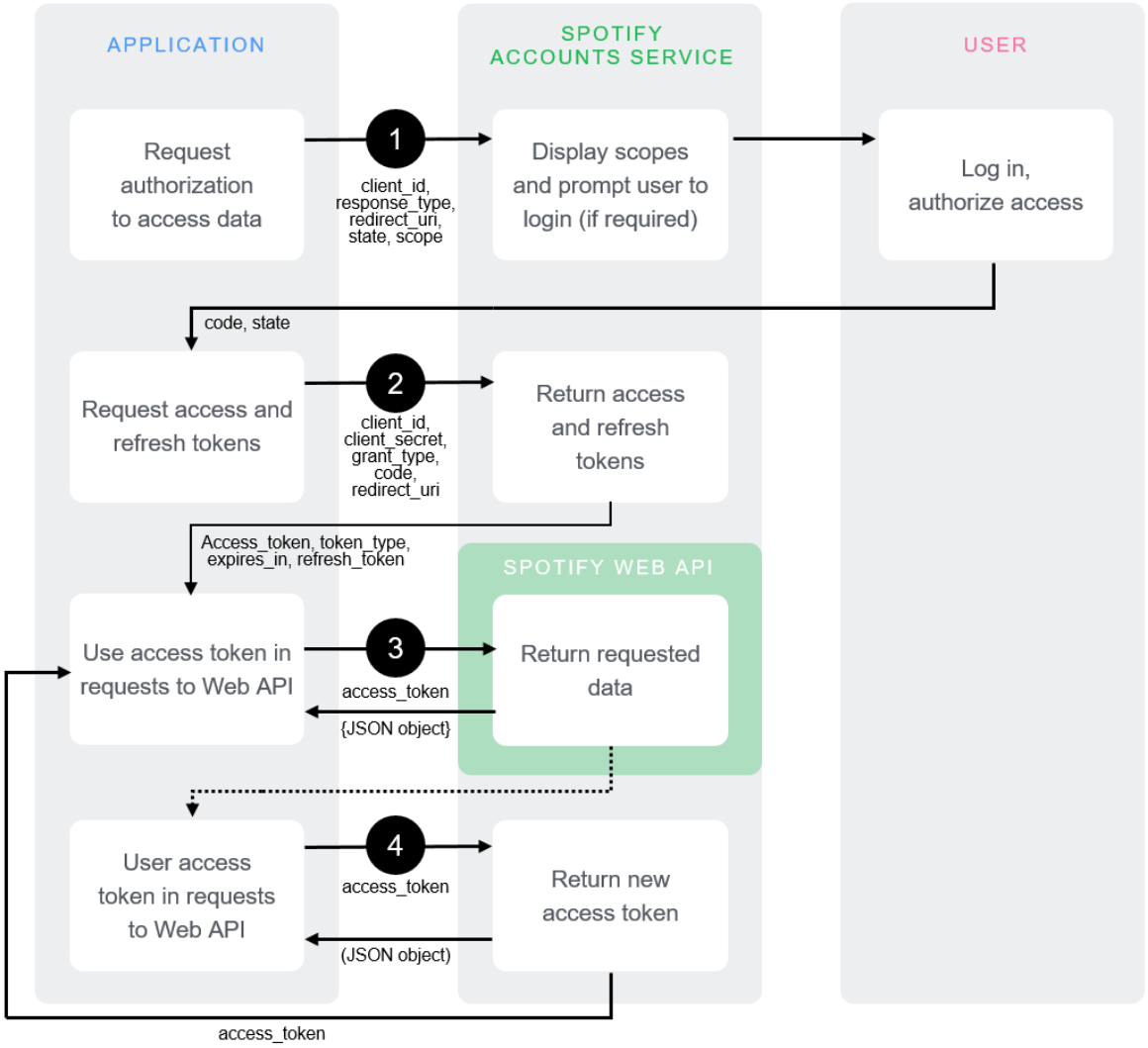


Рис. 3.2. Схематичне представлення процесу OAuth 2.0 автентифікації при взаємодії зі Spotify API

3.4. Особливості реалізації клієнтської частини застосунку

Клієнтська частина розроблюваного вебзастосунку реалізована як SPA з використанням фреймворку Next.js [21]. SPA підхід дозволяє забезпечити більш динамічний та швидкий досвід користувача, оскільки замість перезавантаження всієї сторінки при кожній взаємодії відбувається лише оновлення окремих компонентів. Використання Next.js сприяє покращенню продуктивності та зручності розробки, оскільки фреймворк надає готові рішення для маршрутизації, рендерингу на стороні сервера SSR та оптимізації коду.

Архітектура клієнтської частини базується на компонентному підході.

Інтерфейс користувача розділений на незалежні компоненти, кожен з яких відповідає за певну частину функціональності або відображення. Це дозволяє легко пере використовувати компоненти, спрощує процес розробки та підтримки коду.

Основні елементи архітектури клієнтської частини включають:

- Сторінки (Pages): маршрутизація в Next.js базується на файловій системі, де кожен файл у директорії app відповідає певному маршруту вебзастосунку.
- Компоненти (Components): інтерфейс користувача будується з багаторазово використаних компонентів, які розташовані у директорії components. Компоненти можуть бути як функціональними, так і класовими, і відповідають за відображення та обробку даних.
- Макет (Layout): макети використовуються для забезпечення консистентності інтерфейсу та визначають загальну структуру сторінок.
- Управління станом (State Manaement): для управління станом використовуються React Hooks.

Вебсайт має структуру, що зображена на рис. 3.2.

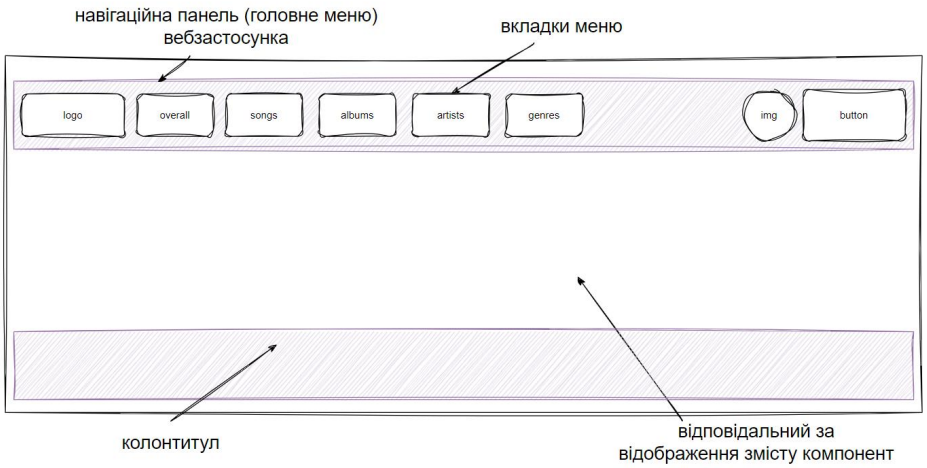


Рис. 3.2. Структура вебсайту

Згідно з вищезгаданою структурою, користувачеві завжди відображається головне меню застосунку. За допомогою вкладок навігаційної панелі та інших елементів інтерфейсу користувача можна змінювати маршрут вебзастосунку. Залежно від маршруту відображатимуться компоненти, відповідальні за відображення вмісту.

В застосунку реалізовано наступні основні маршрути:

- /login – сторінка авторизації користувача;
- /home – головна сторінка застосунку;
- /songs – сторінка зі статистикою про пісні користувача;
- /albums – сторінка зі статистикою про альбоми користувача;
- /artists – сторінка зі статистикою про виконавців користувача;
- /genres – сторінка зі статисткою про жанри користувача.

Клієнтська частина взаємодіє з сервером через API для отримання та оновлення даних. Для виконання HTTP запитів використовуються бібліотеки fetch та swr. SWR дозволяє спростити отримання даних і кешування, забезпечуючи швидке завантаження та оновлення інформації.

3.5. Висновки

Вебзастосунок реалізовано з використанням клієнт-серверної архітектури, що дозволило розподілити функціональність на дві основні частини: сервер та клієнт.

Серверна частина відповідає за обробку запитів від клієнта, автентифікацію та авторизацію користувачів, взаємодію з Spotify API та забезпечення загальної безпеки застосунку. Клієнтська частина, представлена у вигляді односторінкового застосунку, відповідає за відображення даних, взаємодію з користувачем та надсилання запитів до серверної частини.

Для серверної частини було обрано підхід, орієнтований на інтеграцію з існуючими сервісами автентифікації, зокрема Spotify, через бібліотеку

NextAuth.js. Це дозволило значно спростити процес автентифікації та авторизації, використовуючи протокол OAuth 2.0. Важливим аспектом є реалізація механізму оновлення токенів доступу, що забезпечує безперервність роботи застосунку та доступ до API Spotify.

З погляду клієнтської частини, використано фреймворк Next.js для створення односторінкового застосунку, що сприяє покращенню швидкодії та забезпеченню зручного користувацького досвіду. Архітектура клієнтської частини базується на компонентному підході, де інтерфейс користувача розділений на незалежні компоненти, що робить код більш модульним, зрозумілим та легким у підтримці.

Реалізовано основні маршрути для перегляду різноманітної статистики, включаючи головну сторінку, сторінки з інформацією про пісні, альбоми та виконавців. Інтеграція з Spotify API дозволяє отримувати інформацію про музичні вподобання користувача та відображати її у зручному та візуально привабливому форматі.

Використання клієнтської архітектури з виконанням операцій на стороні клієнта дозволило спростити розробку та розгортання, уникнувши складних серверних налаштувань. Застосунок забезпечує інтуїтивно зрозумілий інтерфейс для перегляду та аналізу музичних вподобань, дозволяючи користувачеві отримати цінну інформацію про свої музичні звички.

Успішне поєднання сучасних технологій, таких як Next.js та NextAuth.js, дозволило створити ефективний та зручний у використанні вебзастосунок «SpotiStats», що надає користувачам можливість досліджувати власні музичні вподобання та дізнаватися більше про свою музичну активність.

4. АНАЛІЗ РОЗРОБЛЕНОГО ВЕБЗАСТОСУНКУ

4.1. Аналіз реалізованого вебзастосунку

У результаті успішної розробки програмного забезпечення було створено вебзастосунок «SpotiStats». Програмне забезпечення є цінним інструментом для кожного шанувальника музики, адже воно пропонує глибокий аналіз індивідуальних музичних вподобань на основі даних, наданих Spotify API. Застосунок дозволяє користувачам зануритися у світ їхніх музичних звичок, переглядаючи загальну статистику прослуховувань, ретельно вивчаючи улюблені жанри, харизматичних виконавців, знакові альбоми та пісні, що найбільше резонують з їхніми смаками. Він також надає можливість аналізувати динаміку змін музичних вподобань за різні періоди часу, дозволяючи користувачам відстежувати еволюцію свого музичного смаку.

Реалізовано кожну з функціональних вимог, що були чітко визначені у пункті 1.4, гарантуючи повноцінне та зручне використання застосунку. Користувачі мають можливість проходити автентифікацію через Spotify, безпечно переглядаючи статистику прослуховувань за різні проміжки часу, детально аналізувати свої музичні вподобання, класифікуючи їх за жанрами, виконавцями, улюбленими альбомами та піснями, які відзначаються найбільшою популярністю в їхньому плейлисті. Вебзастосунок забезпечує автоматичний імпорт та регулярне оновлення даних безпосередньо з Spotify, гарантуючи, що інформація завжди залишається актуальною та точною.

Основною перевагою розробленого продукту є безплатний аналіз музичних вподобань. Взаємодія користувача з інтерфейсом, що має сучасний дизайн, зручна та інтуїтивно зрозуміла.

4.2. Тестування реалізованого застосунку

Тестування вебзастосунку є невід’ємною частиною розроблення, забезпечуючи високу якість та стабільну роботу, а також відповідність

очікування користувачів.

Тестування проводилось безперервно упродовж усього періоду розроблення, що дозволило швидко виявляти та виправляти більшість помилок та недоліків на самих ранніх етапах розробки. Це дає змогу заощадити значний час та ресурси, а також забезпечити вищий рівень якості кінцевого продукту. Для тестування застосовувався метод ручного тестування, що дозволило відтворити реальний досвід кінцевого користувача. Під час тестування вебзастосунку було залучено його потенційних користувачів. Це дозволило ретельно перевірити функціональність, зручність використання та загальну поведінку вебзастосунку.

Залучення реальних потенційних користувачів до процесу тестування виявилось надзвичайно цінним, оскільки це дозволило отримати цінні відгуки та виявити деякі недоліки, які не були помічені під час розробки.

На додаток до функціонального тестування, окремо проводилось ретельне тестування на відповідність вимогам безпеки. Тестування підтвердило, що вебзастосунок успішно використовує HTTPS для захищеної передачі даних між клієнтом та сервером, забезпечуючи конфіденційність та цілісність даних користувачів. Було виявлено, що «SpotiStats» ефективно захищений від CSRF та XSS, запобігаючи несанкціонованому доступу та шкідливим атакам. Також було підтверджено, що вебзастосунок повністю дотримується норм GDPR щодо зберігання та обробки персональних даних, гарантуючи захист приватності користувачів та дотримання міжнародних вимог.

У результаті ретельного тестування всі зібрані зауваження та пропозиції були уважно опрацьовані, а відповідні зміни були внесені у вебзастосунок. Це дозволило значно покращити якість та зручність використання «SpotiStats». За результатами фінальне тестування, проведене після внесення змін, не виявило критичних помилок у розробленому програмному забезпеченні, що підтвердило його готовність.

4.3. Рекомендації щодо подальшого вдосконалення

Розроблений вебзастосунок «SpotiStats» вже зараз пропонує потужний набір інструментів для аналізу музичних вподобань на основі даних Spotify, задовольняючи потреби багатьох користувачів. Але задля забезпечення постійного розвитку, покращення конкурентоспроможності та відповідям мінливим потребам користувачів, можна впроваджувати покращення.

На основі аналізу поточних тенденцій ринку, було розроблено ряд рекомендацій щодо подальшого вдосконалення вебзастосунку:

- Розширення функціональності аналізу музичних вподобань, додавши аналіз за різними параметрами, такими як настрій, темп, інструментарій, енергія, валентність та інші.
- Додання можливості порівнювати свої музичні вподобання з іншими користувачами. Це може бути реалізовано через рейтинги та порівняльні графіки.
- Інтеграція вебзастосунку з іншими популярними музичними сервісами, такими як Apple Music, YouTube Music, Deezer та інші.
- Створення мобільного застосунку для Android та IOS.
- Покращення візуалізації статистики створенням різних типів графіків.
- Додавання можливості експортувати результати аналізу у форматах pdf та png.
- Створення плейлистів для користувачів, що базуються на їх музчиних вподобаннях.
- Покращення SEO-оптимізації для пошукових систем, щоб зробити його більш помітним для користувачів, які шукають інструменти для аналізу Spotify.
- Додання підтримки інших мов, щоб зробити вебзастосунок доступним для більшої кількості користувачів у всьому світі.

Розроблений вебзастосунок «SpotiStats» має всі передумови для еволюції в соціальну платформу, де користувачі зможуть не тільки досліджувати свої музичні смаки, але й знаходити однодумців, ділитися музичними відкриттями та організовувати спільне дозвілля.

4.4. Висновки

У даному розділі представлено всебічний огляд розробленого вебзастосунку «SpotiStats», який демонструє успішне виконання кожної функціональної вимоги, сформульованої на початкових етапах проєкту. Проведене тестування підтверджує, що вебзастосунок повністю відповідає заданим критеріям безпеки та функціональності, забезпечуючи користувачам надійний та інтуїтивно-зрозумілий інструмент для аналізу музичних уподобань.

З огляду на сучасні тенденції ринку та результати аналізу відгуків, отриманих від потенційних користувачів, сформовано вичерпний перелік рекомендацій для подальшого розвитку та вдосконалення вебзастосунку. Ці рекомендації охоплюють широкий спектр можливостей, від покращення аналітичних функцій до інтеграції з іншими сервісами та створення мобільного застосунку.

Порівняння розробленого програмного забезпечення з наявними аналогами підкреслило не лише наявність ключових функціональних можливостей, необхідних для ефективного аналізу музичних вподобань, але й унікальність додаткових функцій, які вигідно виділяють «SpotiStats» серед конкурентів. Безплатний доступ до аналізу, зручний та інтуїтивно зрозумілий інтерфейс та сучасний дизайн забезпечують високу конкурентоспроможність програмного продукту.

Враховуючи всі ці фактори, можна зробити висновок, що вебзастосунок «SpotiStats» має значний потенціал для подальшого розвитку, розширення функціональності та перетворення на повноцінну соціальну платформу для любителів музики.

ВИСНОВКИ

Дипломний проєкт було присвячено створенню вебзастосунку «SpotiStats», що має на меті надати користувачам інструмент для поглибленого аналізу їхніх музичних вподобань на основі детальних даних, які надаються Spotify API. Основна ідея полягала в тому, щоб забезпечити користувачів інтуїтивно зрозумілим та ефективним інструментом, який дозволить їм досліджувати свої музичні звички, відкривати нових виконавців та жанри, а також ділитися своїми музичними відкриттями з друзями.

У ході виконання проєкту було проведено дослідження наявних рішень для аналізу музичних вподобань, яке виявило як їхні сильні сторони, так і ключові недоліки. Результати цього дослідження стали основою для формування концепції «SpotiStats», дозволивши визначити унікальні переваги та функціональні можливості, які забезпечать його конкурентоспроможність на ринку.

Ключовим етапом розроблення проєкту став вибір технологій, які повинні були забезпечити високу продуктивність, масштабованість, безпеку та зручність розробки. З огляду на ці вимоги, для реалізації клієнтської частини було обрано React з сучасним фреймворком Next.js, що дозволяє створювати швидкі та інтерактивні вебінтерфейси. Серверна частина проєкту була реалізована на платформі Node.js з використанням фреймворку Express.js, що забезпечило гнучкість та ефективність при обробці запитів. Для забезпечення безпечної автентифікації та авторизації користувачів було використано бібліотеку NextAuth.js, яка дозволяє легко інтегруватися з Spotify API та підтримує сучасні протоколи безпеки.

Процес розроблення супроводжувався ретельним тестуванням, яке проводилося на кожному етапі створення вебзастосунку. Систематичне тестування дозволило виявити та усунути більшість помилок та недоліків, забезпечивши високу якість та стабільність роботи «SpotiStats». Фінальне

тестування, проведене після завершення розробки, підтвердило відсутність критичних недоліків та готовність вебзастосунку до використання користувачами.

Аналіз розробленого вебзастосунку дозволив виявити його основні переваги у порівнянні з наявними аналогами. Серед них можна виділити наступні переваги:

- інтуїтивно зрозумілий інтерфейс, орієнтований на користувача з будь-яким рівнем досвіду;
- безплатний доступ до всіх функціональних можливостей, що робить його привабливим для широкої авдиторії;
- достатній набір інструментів для аналізу музичних вподобань, що дозволяє отримати глибоке розуміння своїх музичних звичок.

Окрім того, було визначено перспективні напрямки подальшого вдосконалення вебзастосунку, які спрямовані на розширення його функціональності, покращення інтеграції з іншими сервісами та перетворення на соціальну платформу для любителів музики.

Розроблений вебзастосунок «SpotiStats» надає користувачам зручний та функціональний інструмент для аналізу їхніх музичних вподобань на базі даних Spotify. Він дозволяє глибше зрозуміти свої музичні звички та відкрити нову музику. Маючи потенціал для подальшого розвитку та вдосконалення, «SpotiStats» може стати корисним помічником для всіх, хто цікавиться музикою.

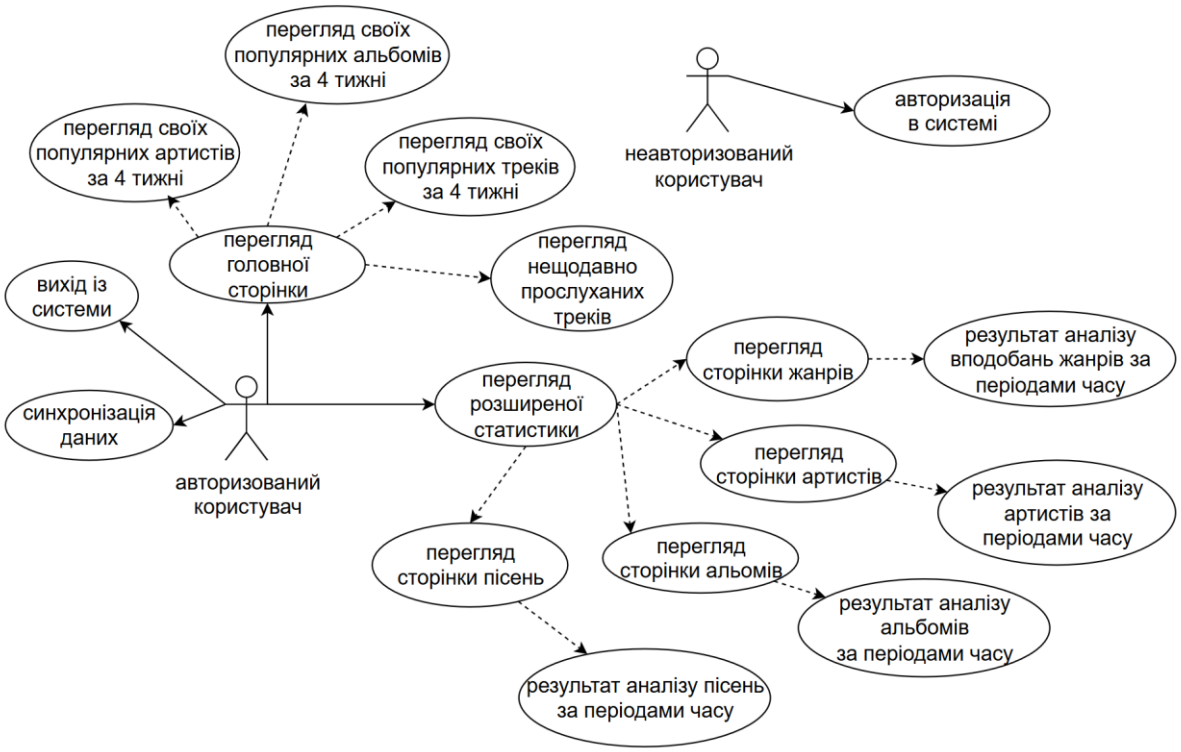
СПИСОК ВИКОРИСТАНИХ ЛІТЕРАТУРНИХ ДЖЕРЕЛ

1. stats.fm: official website of the stats.fm application [Електронний ресурс]. – Режим доступу: https://stats.fm/ – (03.02.2025).
2. Obscurify music: official website of the stats.fm application [Електронний ресурс]. Режим доступу: https://www.obscurifymusic.com/ – (03.02.2025).
3. Node.js: official documentation [Електронний ресурс]. Режим доступу: https://nodejs.org/docs/latest/api/ – (06.02.2025).
4. TypeScript: TypeScript documentation [Електронний ресурс]. Режим доступу: https://www.typescriptlang.org/docs/ – (06.02.2025).
5. Express.js: Production best practices – perfomance and reliability [Електронний ресурс]. Режим доступу: https://expressjs.com/en/advanced/best-practice-performance.html – (06.02.2025).
6. Next.js: official documentation [Електронний ресурс]. Режим доступу: https://nextjs.org/docs – (12.02.2025).
7. Material UI: Ready to use Material Design components [Електронний ресурс]. Режим доступу: https://mui.com/material-ui/ – (13.02.2025).
8. NextAuth.js: official documentation [Електронний ресурс]. Режим доступу: https://next-auth.js.org/getting-started/introduction – (13.02.2025).
9. TailwindCSS: Styling with utility classes [Електронний ресурс]. Режим доступу: https://tailwindcss.com/docs/styling-with-utility-classes – (13.02.2025).
10. State of JavaScript 2024: About [Електронний ресурс]. Режим доступу: https://2024.stateofjs.com/en-US/about/ – (27.02.2025).
11. Angular vs React vs Vue: The Best Framework for 2025 [Електронний ресурс]. Режим доступу: https://zerotomastery.io/blog/angular-vs-react-vs-vue/ – (27.02.2025).

12. Spotify for Developers: Getting started with Web API [Електронний ресурс]. Режим доступу: https://developer.spotify.com/documentation/web-api/tutorials/getting-started – (28.02.2025).
13. Mdn Web Docs: Web APIs [Електронний ресурс]. Режим доступу: https://developer.mozilla.org/en-US/docs/Web/API – (02.03.2025).
14. React: Managing State [Електронний ресурс]. Режим доступу: https://react.dev/learn/managing-state – (05.03.2025).
15. SWR: React Hooks for Data Fetching [Електронний ресурс]. Режим доступу: https://swr.vercel.app/ – (05.03.2025).
16. TestDevLab: What is Manual Testing [Електронний ресурс]. Режим доступу: https://www.testdevlab.com/blog/manual-testing – (06.03.2025).
17. Refactoring Guru: Design Patterns [Електронний ресурс]. Режим доступу: https://refactoring.guru/design-patterns – (10.04.2025).
18. Technovy.com: Top 10 Software Architecture & Design Patterns of 2025 [Електронний ресурс]. Режим доступу: https://tecnovy.com/en/top-10-software-architecture-patterns – (10.04.2025).
19. It jet: What is TypeScript? Benefits of TypeScript and Disadvantages [Електронний доступ]. Режим доступу: https://itjet.io/blog/what-is-typescript – (11.04.2025).
20. GeeksForGeeks: Complete Guide to Clean Architecture [Електронний доступ]. Режим доступу: https://www.geeksforgeeks.org/complete-guide-to-clean-architecture/ – (12.04.2025).
21. Dou.ua: Порівнюємо способи генерації сторінок: CSR, SSR, SSG, ISR. Гайд на основі стека React. [Електронний доступ]. Режим доступу: https://dou.ua/forums/topic/41585/ – (05.05.2025).

ДОДАТКИ

Додаток 1
Копії графічних матеріалів

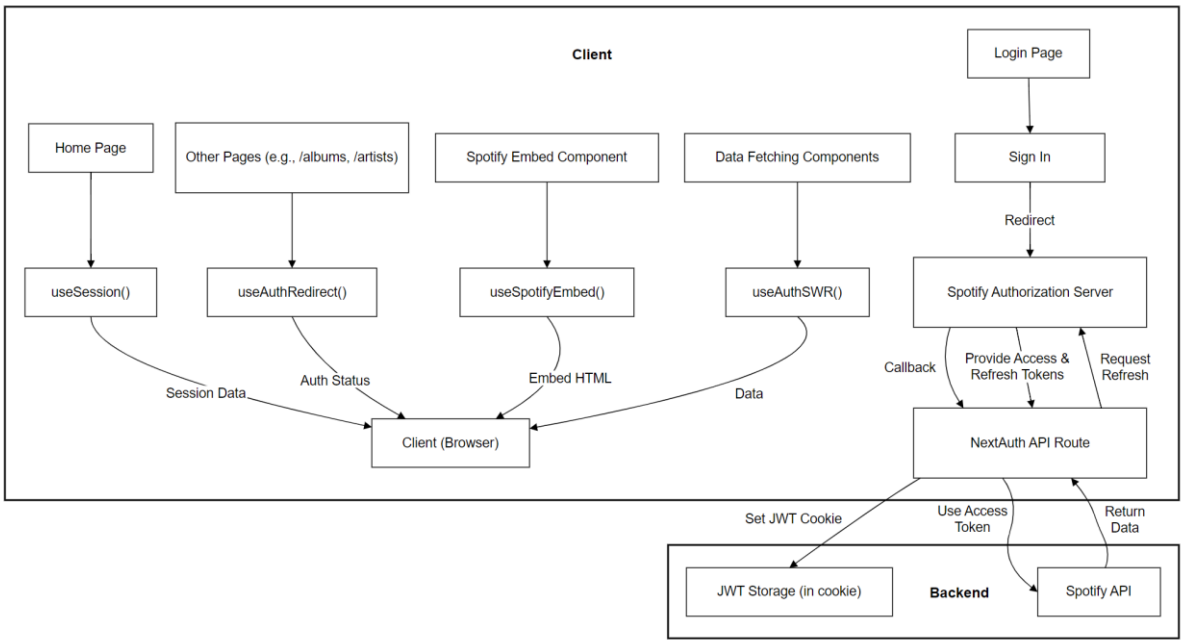


ДП.045440-06-99

Вебзастосунок «SpotiStats» для відстежування даних про музичні вподобання вебзастосунку.

Функціональність вебзастосунку.

UML-діаграма прецедентів

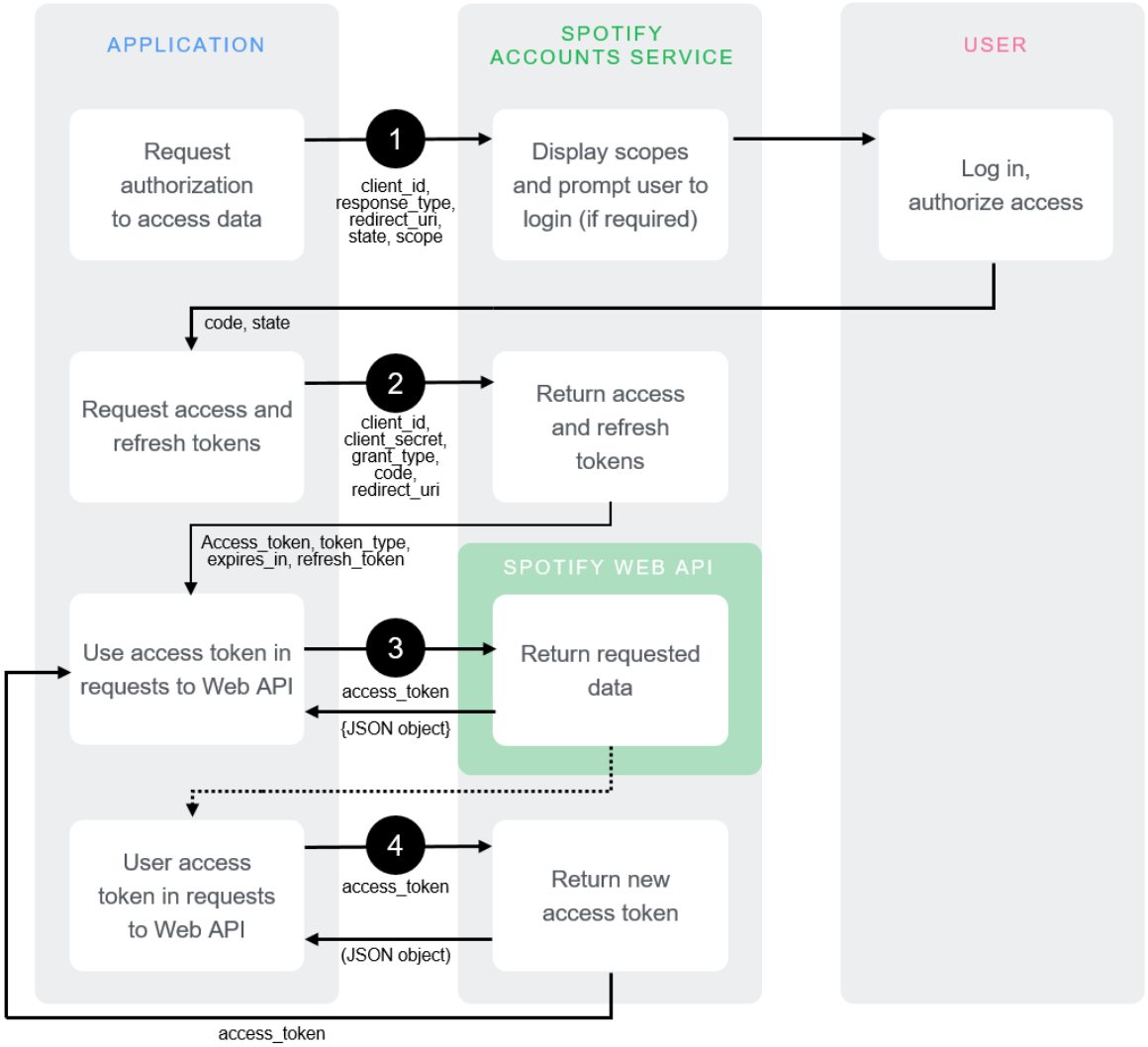


ДП.045440-07-99

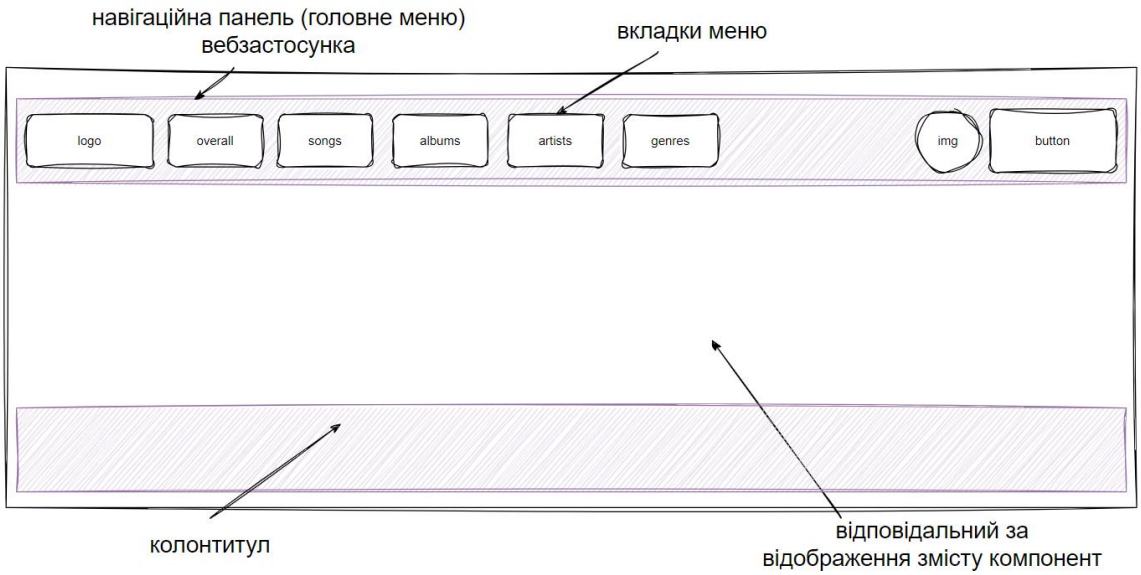
Вебзастосунок «SpotiStats» для відстежування даних про музичні вподобання вебзастосунку.

Архітектура вебзастосунку.

DFD-діаграма



Схематичне представлення процесу OAuth 2.0 автентифікації при взаємодії зі Spotify API
Бодаква К. І., група КП-11



Структура вебсайту
Бодаква К. І., група КП-11

Додаток 2
Лістинг програми

Фрагмент конфігурації NextAuth для роботи зі Spotify API

```
const handler = NextAuth({
  providers: [
    SpotifyProvider({
      clientId: environment.clientId,
      clientSecret: environment.clientSecret,
      authorization: {
        params: {
          scope: environment.scope,
        },
      },
    }),
  ],
  callbacks: {
    async jwt({ token, account }) {
      if (account) {
        return {
          ...token,
          accessToken: account.access_token,
          refreshToken: account.refresh_token,
          accessTokenExpires: Number(account.expires_at) * 1000,
        };
      }
      if (token.accessTokenExpires && Date.now() <
(token.accessTokenExpires as number)) {
        return token;
      }
      try {
        const response = await
fetch("https://accounts.spotify.com/api/token", {
          method: "POST",
          headers: {
            "Content-Type": "application/x-www-form-urlencoded",
            Authorization: `Basic
${Buffer.from(`${environment.clientId}:${environment.clientSecret}`).toSt
ring(
              "base64"
            )}`,
          },
          body: new URLSearchParams({
            grant_type: "refresh_token",
            refresh_token: token.refreshToken as string,
          }),
        });

        const data = await response.json();

        return {
          ...token,
          accessToken: data.access_token,
          accessTokenExpires: Date.now() + data.expires_in * 1000,
        };
      } catch (error) {
        console.error("Error refreshing access token", error);
        return { ...token, error: "RefreshAccessTokenError" };
      }
    },
    async session({ session, token }) {
      return {
        ...session,
        accessToken: token.accessToken,
```

```
        error: token.error,
      };
    },
  },
});
```

Фрагмент коду створення вбудованого плеєра

```
"use client";

import { SpotifyService } from "@/services";
import { useMemo, useState } from "react";
import useSWR from "swr";

export const useSpotifyEmbed = (url: string, isOpen?: boolean) => {
  const [isFirstFetch, setIsFirstFetch] = useState(true);

  const shouldFetch = useMemo(() => {
    if (!isFirstFetch) {
      return true;
    }
    return isOpen;
  }, [isFirstFetch, isOpen]);

  console.log("shouldFetch", shouldFetch, isFirstFetch);

  const { data: { html: embedHtml } = { html: "" }, ...rest } = useSWR(
    shouldFetch ? url : null,
    (songUrl: string) => {
      console.log("fetching embedHtml", songUrl);

      return SpotifyService.getOEmbed(songUrl);
    },
    {
      revalidateOnFocus: false,
      onSuccess: () => setIsFirstFetch(false),
      onError: (error) => {
        console.error("Error fetching Spotify embed:", error);
      },
    }
  );

  return { embedHtml, ...rest };
};
```

Фрагмент коду захисту маршрутів маршрутизації

```
"use client";

import { useSession } from "next-auth/react";
import { usePathname, useRouter } from "next/navigation";
import { useEffect, useState } from "react";

const PROTECTED_ROUTES = ["/home", "/songs", "/albums", "/artists",
"/genres"];

export function useAuthRedirect() {
  const { data: session, status } = useSession();
  const router = useRouter();
  const pathname = usePathname();
  const [isRedirecting, setIsRedirecting] = useState(false);

  useEffect(() => {
    if (status === "loading" || isRedirecting) return;

    const isAuthenticated = !!session?.accessToken;
    const isProtectedRoute = PROTECTED_ROUTES.some((route) =>
pathname?.startsWith(route));

    if (!isAuthenticated && isProtectedRoute) {
      setIsRedirecting(true);
      router.push("/login");
    }
  }, [session, status, router, pathname, isRedirecting]);

  return {
    isAuthenticated: !!session?.accessToken,
    isLoading: status === "loading",
    user: session?.user,
  };
}
```

Фрагмент коду сервісу для доступу до Spotify API

```
import {
  Limit,
  RecentlyPlayedResponse,
  SpotifyAlbum,
  SpotifyArtist,
  SpotifyPlaybackState,
  SpotifySong,
  TimeRange,
} from "@/types";

interface OEmbedResponse {
  html: string;
  width: number;
  height: number;
  version: string;
  provider_name: string;
  provider_url: string;
  type: string;
  title: string;
  thumbnail_url: string | null;
  thumbnail_width: number | null;
  thumbnail_height: number | null;
}

const buildUrl = (path: string, params: URLSearchParams) => {
  return new URL(`/v1${path}?${params}`,
"https://api.spotify.com").toString();
};

const fetchData = async <T>(
  url: string,
  // eslint-disable-next-line @typescript-eslint/no-explicit-any
  { params, accessToken }: { params?: Record<string, any>; accessToken?:
string }
): Promise<T> => {
  const { timeRange, ...restParams } = params || {};
  const timeRangeParam = timeRange ? { time_range: timeRange } : {};
  const allParams = { ...restParams, ...timeRangeParam };

  const urlWithParams = buildUrl(url, new URLSearchParams(allParams));
  const response = await fetch(urlWithParams, {
    method: "GET",
    headers: {
      "Content-Type": "application/json",
      Authorization: `Bearer ${accessToken}`,
    },
  });

  if (!response.ok) {
    throw new Error(`Failed to fetch: ${response.status}`);
  }

  return response.json();
};

export const SpotifyService = {
  getTopUserSongs: async (accessToken: string, params: { timeRange:
TimeRange; limit: Limit }) => {
    const url = "/me/top/tracks";
    return fetchData<{ items: SpotifySong[] }>(url, {
```

```
      accessToken,
      params: { ...params, scope: "user-top-read" },
    });
  },

  getTopUserAlbums: async (accessToken: string, params: { timeRange:
TimeRange; limit: Limit }) => {
    const tracksResponse = await
SpotifyService.getTopUserSongs(accessToken, params);

    const albumIds = Array.from(new Set(tracksResponse.items.map((track)
=> track.album.id)));

    const chunkedAlbumIds = albumIds.reduce<string[][]>((acc, id, i) => {
      const chunkIndex = Math.floor(i / 20);
      if (!acc[chunkIndex]) acc[chunkIndex] = [];
      acc[chunkIndex].push(id);
      return acc;
    }, []);

    const albumChunks = await Promise.all(
      chunkedAlbumIds.map((chunk) =>
SpotifyService.getAlbums(accessToken, { ids: chunk }))
    );

    const albums = albumChunks.flatMap((chunk) => chunk.albums);

    return { items: albums };
  },

  getAlbums: async (accessToken: string, params: { ids: string[] }) => {
    const url = "/albums";
    return fetchData<{ albums: SpotifyAlbum[] }>(url, {
      accessToken,
      params: { ids: params.ids.join(",") },
    });
  },

  getTopUserArtists: async (accessToken: string, params: { timeRange:
TimeRange; limit: Limit }) => {
    const url = "/me/top/artists";
    return fetchData<{ items: SpotifyArtist[] }>(url, { accessToken,
params });
  },

  getRecentlyPlayed: async (accessToken: string, params: { limit: Limit
}) => {
    const url = "/me/player/recently-played";
    return fetchData<RecentlyPlayedResponse>(url, { accessToken, params
});
  },

  getUserPlaylists: async (accessToken: string, params: { limit: Limit })
=> {
    const url = "/me/playlists";
    return fetchData(url, { accessToken, params });
  },

  getUserProfile: async (accessToken: string) => {
    const url = "/me";
    return fetchData(url, { accessToken });
  },
```

```
  getCurrentlyPlaying: async (accessToken: string):
Promise<SpotifyPlaybackState | null> => {
    const url = "/me/player/currently-playing";
    try {
      const urlWithParams = buildUrl(url, new URLSearchParams());
      const response = await fetch(urlWithParams, {
        method: "GET",
        headers: {
          "Content-Type": "application/json",
          Authorization: `Bearer ${accessToken}`,
        },
      });

      if (response.status === 204) {
        return null;
      }

      if (!response.ok) {
        if (response.status === 429) {
          const retryAfter = response.headers.get("Retry-After");
          throw new Error(`Rate limited. Retry after ${retryAfter}
seconds`);
        }
        throw new Error(`Failed to fetch currently playing:
${response.status}`);
      }

      const data = await response.json();
      return data as SpotifyPlaybackState;
    } catch (error) {
      throw error;
    }
  },

  getPlaylistTracks: async (accessToken: string, params: { playlistId:
string; limit: Limit }) => {
    const { playlistId, ...restParams } = params;
    const url = `/playlists/${playlistId}/tracks`;
    return fetchData(url, { accessToken, params: restParams });
  },

  getPlaylist: async (accessToken: string, params: { playlistId: string
}) => {
    const { playlistId } = params;
    const url = `/playlists/${playlistId}`;
    return fetchData(url, { accessToken });
  },

  getAlbum: async (accessToken: string, params: { albumId: string }) => {
    const { albumId } = params;
    const url = `/albums/${albumId}`;
    return fetchData(url, { accessToken });
  },

  getAlbumTracks: async (accessToken: string, params: { albumId: string
}) => {
    const { albumId } = params;
    const url = `/albums/${albumId}/tracks`;
    return fetchData<{ items: SpotifySong[] }>(url, { accessToken });
  },

  getTrack: async (accessToken: string, params: { trackId: string }) => {
    const { trackId } = params;
    const url = `/tracks/${trackId}`;
```

```
    return fetchData(url, { accessToken });
  },

  getArtist: async (accessToken: string, params: { artistId: string }) =>
{
    const { artistId } = params;
    const url = `/artists/${artistId}`;
    return fetchData(url, { accessToken });
  },

  getArtists: async (accessToken: string, params: { ids: string[] }) => {
    const url = "/artists";
    return fetchData(url, {
      accessToken,
      params: { ids: params.ids.join(",") },
    });
  },

  getArtistTopTracks: async (accessToken: string, params: { artistId:
string; market: string }) => {
    const { artistId, ...restParams } = params;
    const url = `/artists/${artistId}/top-tracks`;
    return fetchData(url, { accessToken, params: restParams });
  },

  getArtistAlbums: async (accessToken: string, params: { artistId:
string; limit: Limit }) => {
    const { artistId, ...restParams } = params;
    const url = `/artists/${artistId}/albums`;
    return fetchData(url, { accessToken, params: restParams });
  },

  getArtistRelatedArtists: async (accessToken: string, params: {
artistId: string }) => {
    const { artistId } = params;
    const url = `/artists/${artistId}/related-artists`;
    return fetchData(url, { accessToken });
  },

  getSearchResults: async (accessToken: string, params: { query: string;
type: string; limit: Limit }) => {
    const url = "/search";
    return fetchData(url, { accessToken, params });
  },

  getAudioFeatures: async (accessToken: string, params: { trackId: string
}) => {
    const { trackId } = params;
    const url = `/audio-features/${trackId}`;
    return fetchData(url, { accessToken });
  },

  getAudioAnalysis: async (accessToken: string, params: { trackId: string
}) => {
    const { trackId } = params;
    const url = `/audio-analysis/${trackId}`;
    return fetchData(url, { accessToken });
  },

  getOEmbed: async (url: string) => {
    const encodedUrl = encodeURIComponent(url);
    const response = await
fetch(`https://open.spotify.com/oembed?url=${encodedUrl}`);
```

```
    if (!response.ok) {
      throw new Error(`Failed to fetch oEmbed data: ${response.status}`);
    }

    return response.json() as Promise<OEmbedResponse>;
  },
};
```

Фрагмент коду SWR хука з NextAuth

```
"use client";

import { useSession } from "next-auth/react";
import { useMemo } from "react";
import useSWR, { SWRConfiguration } from "swr";

export const useAuthSWR = <T extends (...args: any[]) => unknown, ParamT
= Parameters<T>[1]>(
  fetcher: T | null,
  params: ParamT | null,
  options?: SWRConfiguration
) => {
  const { data: session } = useSession();

  const key = useMemo(() => {
    if (!session?.accessToken || !fetcher || !params) return null;
    return [fetcher.name, session.accessToken, params];
  }, [session?.accessToken, fetcher, params]);

  return useSWR<Awaited<ReturnType<T>>>(
    key,
    () => (fetcher ? (fetcher(session?.accessToken, params) as
Awaited<ReturnType<T>>) : null),
    {
      revalidateOnFocus: false,
      ...options,
    }
  );
};
```

Додаток 3
Копія презентації

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ІМЕНІ ІГОРЯ СІКОРСЬКОГО”

ФАКУЛЬТЕТ ПРИКЛАДНОЇ МАТЕМАТИКИ

КАФЕДРА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ КОМП’ЮТЕРНИХ СИСТЕМ

ВЕБЗАСТОСУНОК «SPOTISTATS» ДЛЯ ВІДСТЕЖУВАННЯ ДАННИХ ПРО МУЗИЧНІ ВПОДОБАННЯ

Виконав: студент групи КП-11 Бодаква Кирил Ігорович

Керівник: доцент кафедри ПЗКС, к.т.н. Рибачок Наталія Антонівна

Київ — 2025

ПОСТАНОВКА ЗАДАЧІ

Мета проєкту: створення зручного інструменту для відстежування особистих музичних вподобань на основі даних, що надаються Spotify API.

Завдання:

1. Проаналізувати аналогічні програмні рішення присутні на ринку.
2. Визначити вимоги до розроблюваного ПЗ.
3. Обрати та обґрунтувати засоби реалізації.
4. Розробити вебзастосунок для відстежування даних про музичні вподобання, що враховує недоліки існуючих рішень.
5. Провести тестування розробленого програмного забезпечення.
6. Визначити напрямки подальшого розвитку проєкту.

АКТУАЛЬНІСТЬ

1. Відсутність зручних інструментів аналізу даних про слухацькі звички користувачів Spotify.
2. Задоволення потреб у персоналізованому аналізі музичних вподобань зацікавлених користувачів.
3. Бажання глибшого розуміння музичних вподобань музичних спільнот.
4. Виявлення потенціалу для створення платформи для обміну музичними вподобаннями та тенденціями між користувачами.

ІСНУЮЧІ АНАЛОГИ

Вебзастосунок
«stats.fm»

Вебзастосунок
«Obscurify Music»

ФУНКЦІОНАЛЬНІ ВИМОГИ

Вебзастосунок повинен забезпечувати такі основні вимоги:

1. Можливість динамічного оновлення вмісту вебсторінок.
2. Можливість автентифікації через обліковий запис Spotify.
3. Можливість перегляду статистики прослуховування музики за різні проміжки часу.
4. Можливість аналізувати свої музичні вподобання за піснями, альбомами, виконавцями та жанрами.
5. Можливість перегляду своїх улюблених виконавців, жанри пісень, альбоми.
6. Можливість перегляду історії прослуховувань.
7. Забезпечення автоматичного імпорту та оновлення даних прослуховувань зі Spotify.

НЕФУНКЦІОНАЛЬНІ ВИМОГИ

Вебзастосунок повинен забезпечувати такі вимоги:

1. Відгук на дії користувача (фільтрація, сортування, оновлення даних) має відбуватися в межах 1-2 секунд.
2. Система повинна обробляти не менше 500 запитів одночасно без значної деградації швидкодії.
3. Система повинна автоматично відновлюватися після збоїв без втрати користувацьких даних.
4. Система повинна підтримувати збільшення кількості користувачів без суттєвого погіршення продуктивності.
5. Архітектура системи має забезпечувати можливість горизонтального масштабування при зростанні навантаження.

ОБГРУНТУВАННЯ ВИБОРУ ЗАСОБІВ РОЗРОБЛЕННЯ

Мова програмування TypeScript

Середовище виконання Node.js

Фреймворк Next.js

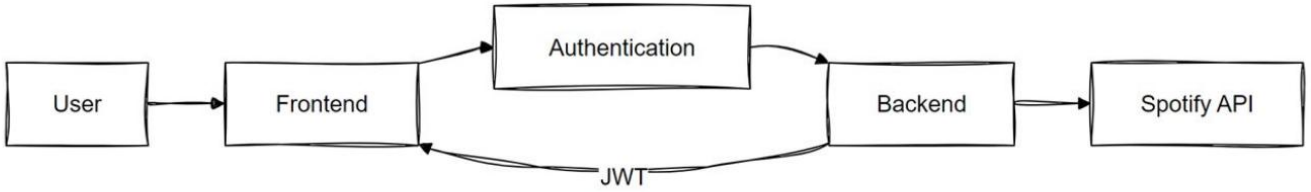
Бібліотека MaterialUI

Бібліотека NextAuth.js

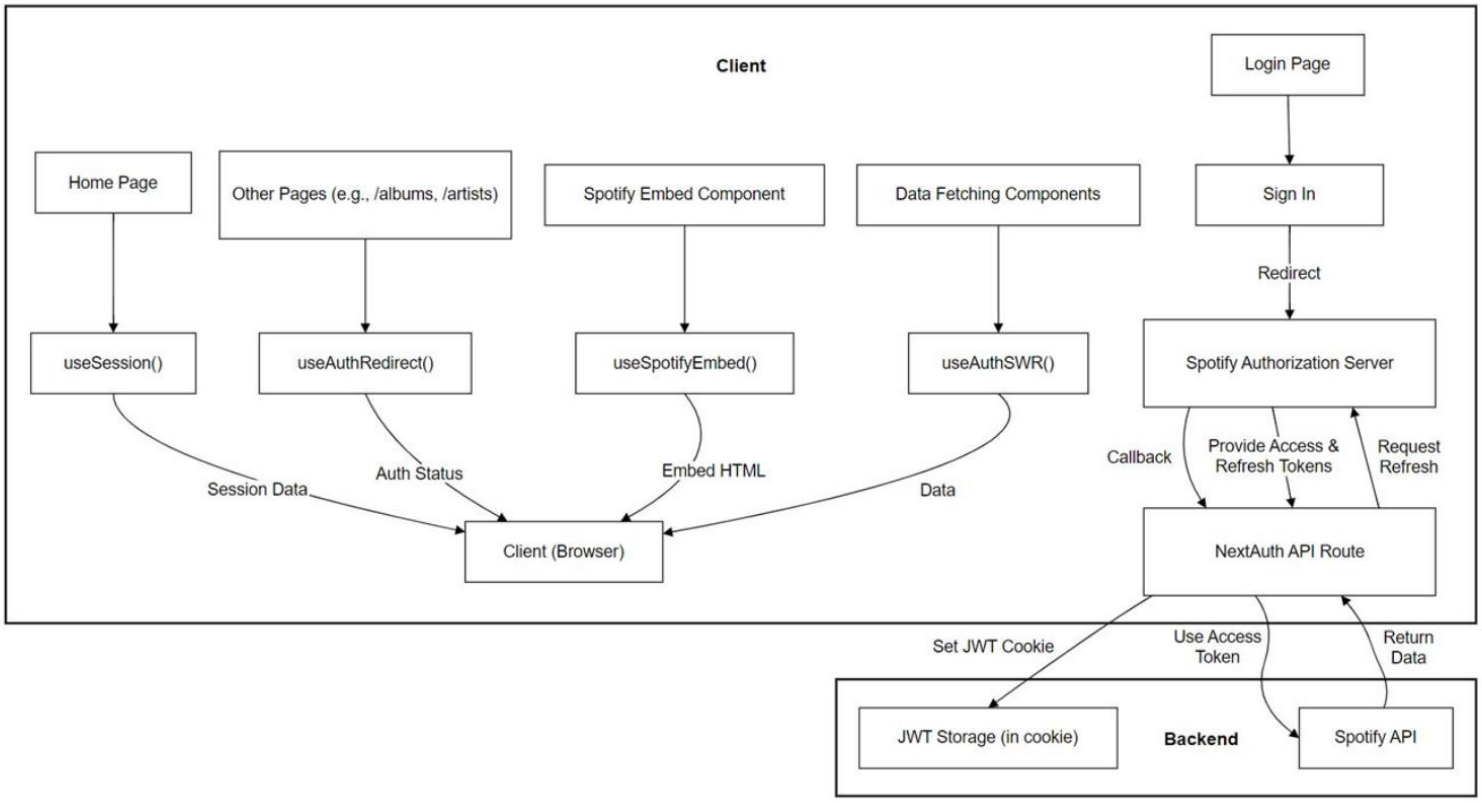
CSS-фреймворк TailwindCSS

Фреймворк для вебзастосунків ExpressJS

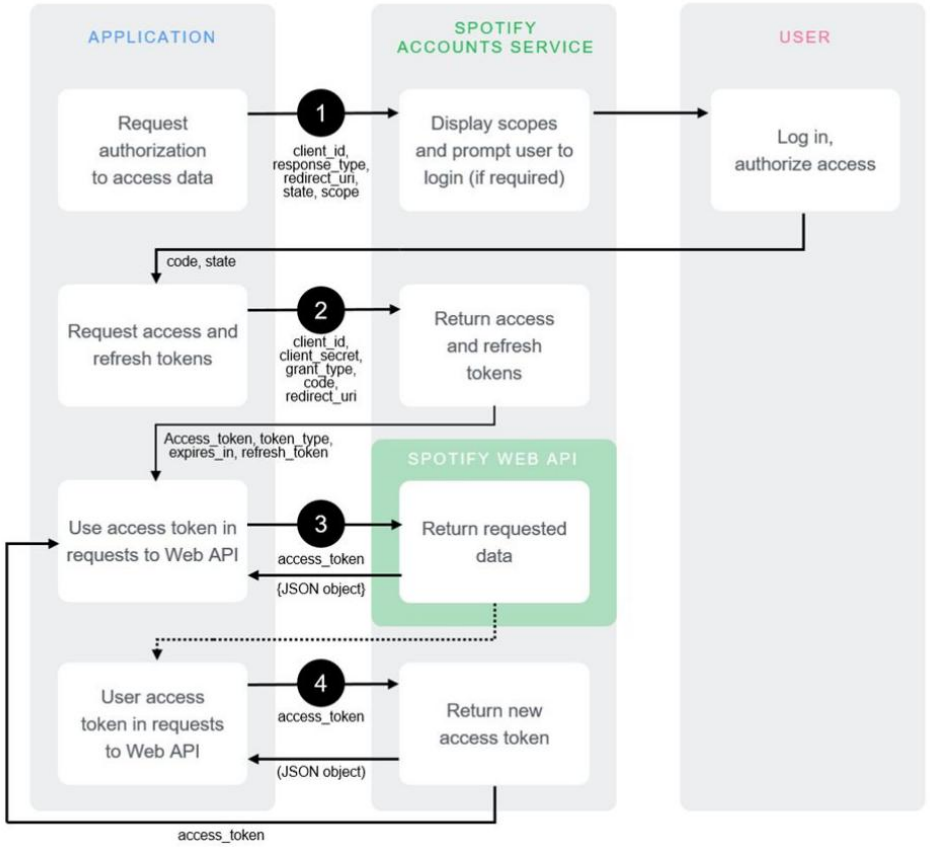
АРХІТЕКТУРА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ



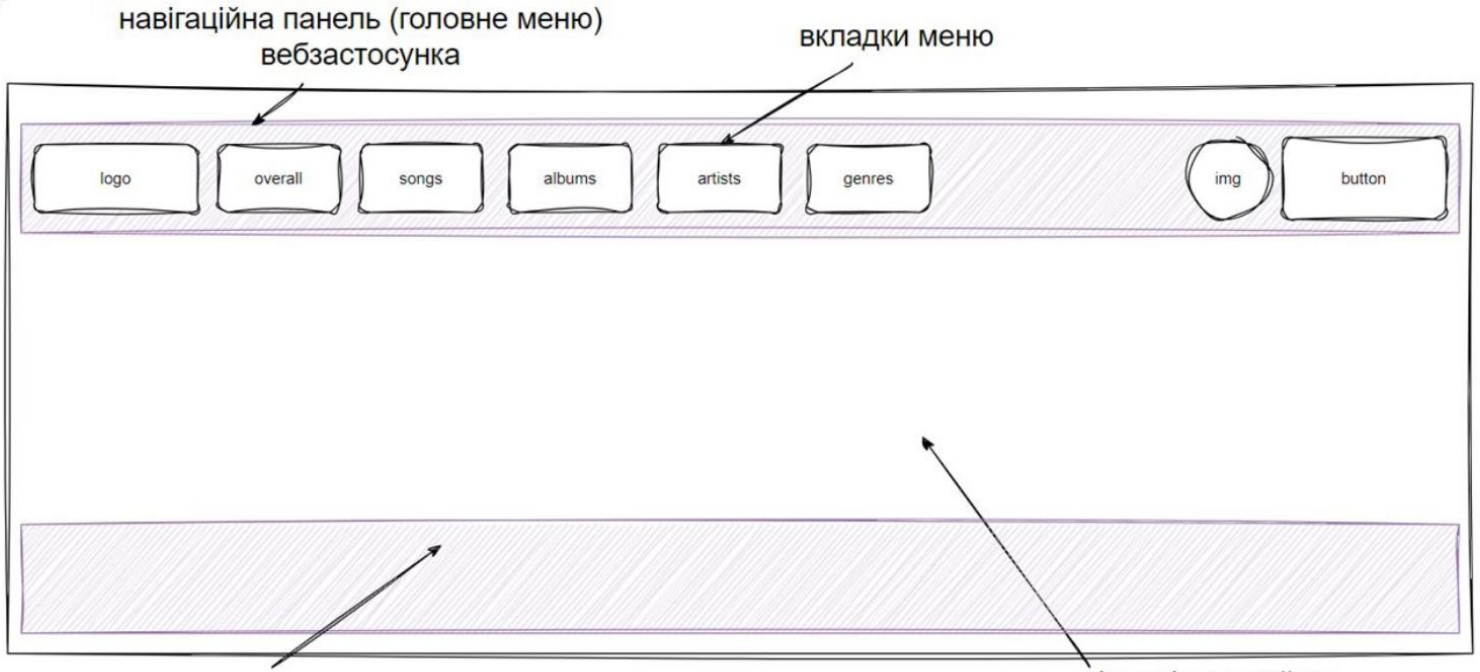
АРХІТЕКТУРА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ



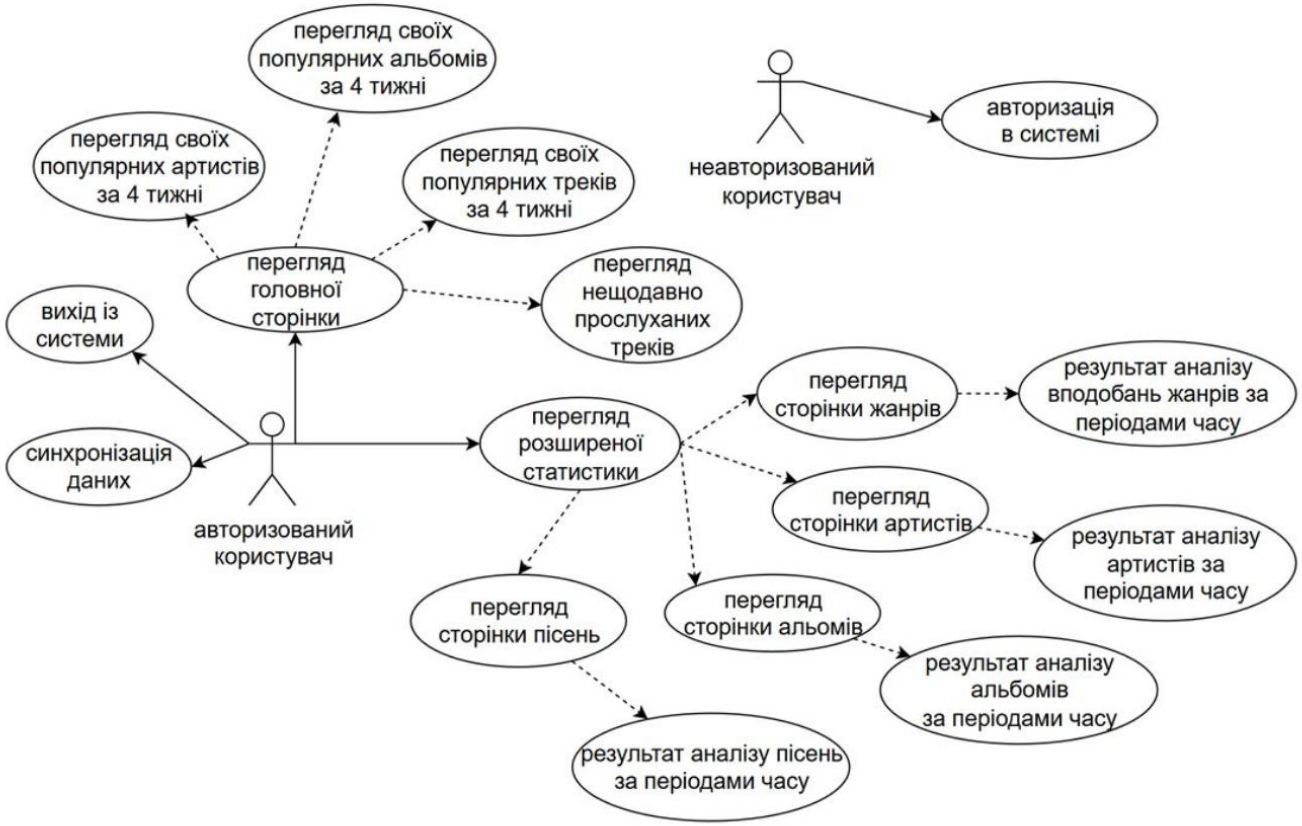
СХЕМАТИЧНЕ ПРЕДСТАВЛЕННЯ ПРОЦЕСУ АВТЕНТИФІКАЦІЇ



СТРУКТУРА ВЕБСАЙТУ



ДІАГРАМА ВАРІАНТІВ ВИКОРИСТАННЯ ЗАСТОСУНКУ



ТЕСТУВАННЯ ВЕБЗАСТОСУНКУ

Тестування вебзастосунку здійснювалося методом ручної перевірки, що включала безпосередню оцінку роботи інтерфейсу, взаємодії користувача з елементами управління та коректності відображення інформації.

Ключовим підходом стало відтворення типових сценаріїв використання, що дозволило імітувати досвід реального користувача. Додатково залучення потенційних користувачів забезпечило отримання цінних відгуків та ідентифікацію неочевидних недоліків. Проведене фінальне тестування підтвердило відсутність критичних помилок і готовність продукту до використання.

РОЗГОРТАННЯ ВЕБЗАСТОСУНКУ

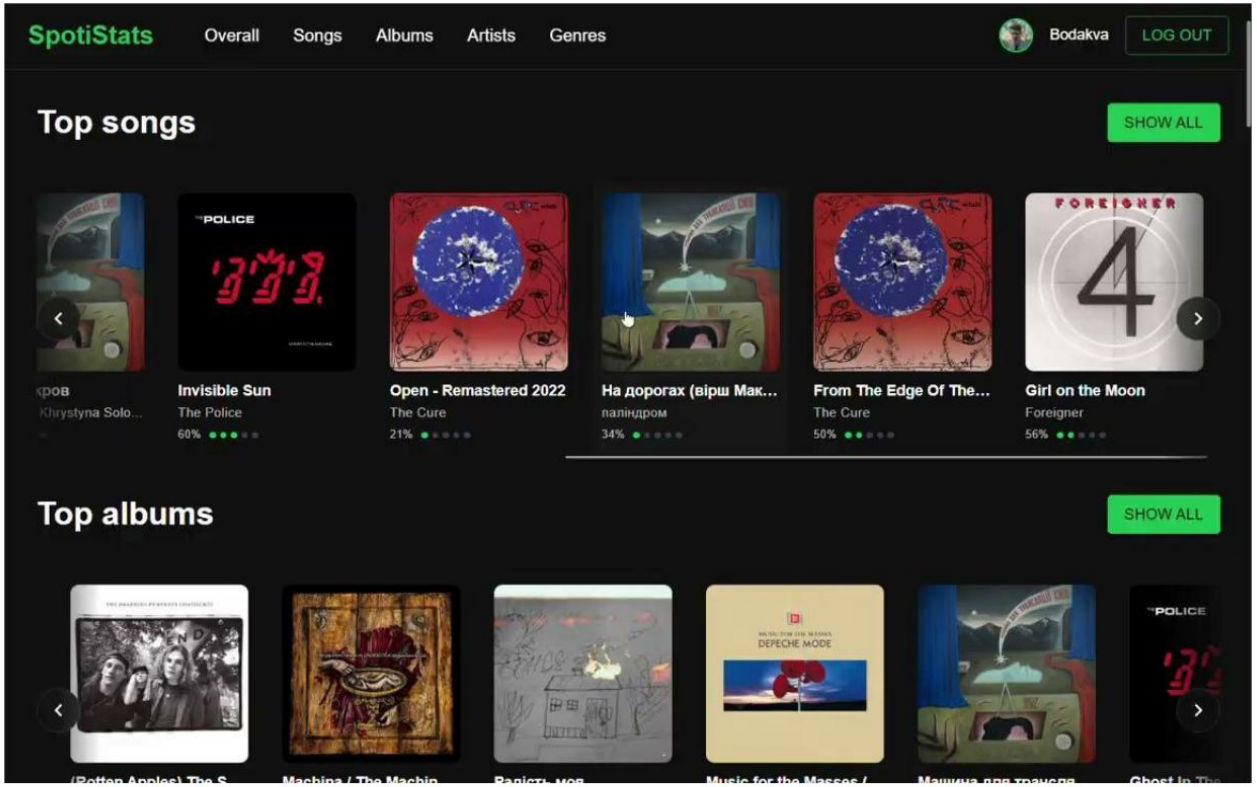
Платформа: Vercel

Протокол HTTPS, перегляд у реальному часі

Vercel

14/19

ВІДЕОДЕМОНСТРАЦІЯ



ШЛЯХИ ПОДАЛЬШОГО РОЗВИТКУ

1. Покращення візуалізації статистики створенням різних типів графіків.
2. Додавання можливості експортувати результати аналізу у форматах pdf та png.
3. Створення плейлистів для користувачів, що базуються на їх музичних вподобаннях.
4. Додання підтримки інших мов, щоб зробити вебзастосунок доступним для більшої кількості користувачів у всьому світі.

ВИСНОВКИ

1. Проаналізовано існуючі рішення: stats.fm та Obscurify Music
2. Визначено вимоги до розроблюваного програмного забезпечення.
3. Обрано програмні засоби для розробки та обгрунтовано їх вибір.
4. Спроєктовано архітектуру вебзастосунку.
5. Розроблено клієнтську та серверну частини ПЗ.
6. Реалізовано поставлені до ПЗ вимоги.
7. Протестовано розроблене ПЗ, виправлено виявлені помилки.
8. Запропоновано напрямки подальшого розвитку продукту.

Розробка виконана у повному обсязі відповідно до Технічного завдання, тестування проведено згідно з затвердженою Програмою та методикою тестування

УНІКАЛЬНІСТЬ

Назва організації

National Technical University of Ukraine Igor Sikorskyi Kyiv Politech Institute

Заголовок

Бакалавр КП-11 Бодаква Кирил Ігорович

Автор	Науковий керівник / Експерт
Бодаква Кирил Ігорович	**Рибачок Наталя Антонівна**

підрозділ

ФПМ, К-ра програмного забезпечення комп'ютерних систем

Обсяг знайдених подібностей

Коефіцієнт подібності визначає, який відсоток тексту по відношенню до загального обсягу тексту було знайдено в різних джерелах. Зверніть увагу, що високі значення коефіцієнта не автоматично означають плагіат. Звіт має аналізувати компетентна / уповноважена особа.

10	9590	79907
Довжина фрази для коефіцієнта подібності 2	Кількість слів	Кількість символів

ДЯКУЮ ЗА УВАГУ!

Факультет прикладної математики

Кафедра програмного забезпечення комп'ютерних систем

«ЗАТВЕРДЖЕНО»

Завідувач кафедри

_____ Євгенія СУЛЕМА

«___» _____ 2024 р.

ВЕБЗАСТОСУНОК «SPOTISTATS» ДЛЯ ВІДСТЕЖУВАННЯ ДАНИХ ПРО МУЗИЧНІ ВПОДОБАННЯ

Програма та методика тестування

ДП.045440-04-51

«ПОГОДЖЕНО»

Керівник проєкту:

_____ Наталія РИБАЧОК

Нормоконтроль:

_____ Микола ОНАЙ

Виконавець:

_____ Кирил БОДАКВА

2024

ЗМІСТ

1. ОБ’ЄКТ ВИПРОБУВАНЬ

Вебзастосунок «SpotiStats» для відстежування даних про музичні вподобання, розроблений з використанням таких технологій як: Node.js, TypeScript, React, Next.js, Material UI, Tailwind CSS, NextAuth.js.

2. МЕТА ТЕСТУВАННЯ

У процесі тестування має бути перевірено наступне:

1) працездатність компонентів вебзастосунку та взаємодія між компонентами;
2) маршрутизація користувацької частини вебзастосунку;
3) зручність користувацького інтерфейсу;
4) доступ до необхідних даних;
5) взаємодія клієнтської та серверної частини вебзастосунку;
6) відповідність вимогам Технічного завдання.

3. МЕТОДИ ТЕСТУВАННЯ

Упродовж усього процесу розробки вебзастосунку «SpotiStats» здійснювалося систематичне тестування нових функцій. Такий підхід дозволив ідентифікувати та виправити потенційні недоліки на ранніх етапах, що значно заощадило час і ресурси команди, а також позитивно вплинуло на якість кінцевого продукту. Застосовувався метод ручного тестування, коли тестувальник відтворював сценарії використання, типові для звичайного користувача, що забезпечило перевірку функціональності, зручності та загальної поведінки вебзастосунку в реальних умовах.

Залучення потенційних користувачів до тестування стало джерелом цінних відгуків та дозволило виявити деякі недоліки, які не були помічені під час внутрішньої розробки.

Всі зауваження та пропозиції, отримані в результаті тестування, були ретельно проаналізовані та враховані при внесенні змін у вебзастосунок, що позитивно вплинуло на його якість та зручність. Фінальне тестування не виявило критичних помилок, що підтвердило готовність «SpotiStats» до використання користувачами.

4. ЗАСОБИ ТА ПОРЯДОК ТЕСТУВАННЯ

Працездатність вебзастосунку перевіряється шляхом:

1) динамічного ручного тестування інтерфейсу користувача – перевірка реакції вебсторінок на різні типи вхідних даних, включаючи граничні та недопустимі значення, з метою виявлення помилок валідації та стабільності роботи елементів інтерфейсу;
2) динамічного ручного тестування на відповідність функціональним вимогам – перевірка відповідності роботи вебзастосунку заявленим функціональним вимогам, таким як авторизація через Spotify, перегляд статистики прослуховування, аналіз музичних вподобань, імпорт даних та відображення результатів;
3) статичного тестування коду;
4) тестування вебзастосунку у різних браузерах – перевірка коректності відображення та функціонування вебзастосунку в різних браузерах (Chrome, Firefox, Safari) для забезпечення однакового користувацького досвіду;
5) тестування стабільності роботи при різних умовах – перевірка стабільності роботи вебзастосунку при різних умовах, включаючи інтенсивне використання API Spotify;
6) тестування зручності використання – оцінка зручності навігації, інтуїтивності інтерфейсу, швидкості завантаження сторінок та інших аспектів, що впливають на користувацький досвід.

Факультет прикладної математики

Кафедра програмного забезпечення комп’ютерних систем

«ЗАТВЕРДЖЕНО»

Завідувач катедри

_____ Євгенія СУЛЕМА

«___» _____ 2025 р.

ВЕБЗАСТОСУНОК «SPOTISTATS» ДЛЯ ВІДСТЕЖУВАННЯ ДАНИХ ПРО МУЗИЧНІ ВПОДОБАННЯ

Керівництво користувача

ДП.045440-05-34

«ПОГОДЖЕНО»

Керівник проєкту:

_____ Наталія РИБАЧОК

Нормоконтроль:

_____ Микола ОНАЙ

Виконавець:

_____ Кирил БОДАКВА

2025

ЗМІСТ

1. Опис структури вебзастосунку

Вебзастосунок «SpotiStats» для відстежування даних про музичні вподобання складається з вебсторінок, котрі відображаються динамічно в результаті взаємодії користувача з вебзастосунком.

Вебзастосунок складається із таких сторінок:

1) сторінка авторизації;
2) головна сторінка;
3) сторінка з візуалізацією вподобань користувача за піснями;
4) сторінка з візуалізацією вподобань користувача за альбомами;
5) сторінка з візуалізацією вподобань користувача за артистами.

2. Процедура авторизації

Щоб користуватись основними функціональними можливостями застосунку необхідно авторизуватись в системі. Користувача зустрічає сторінка авторизації, що представлена на рис. 1.

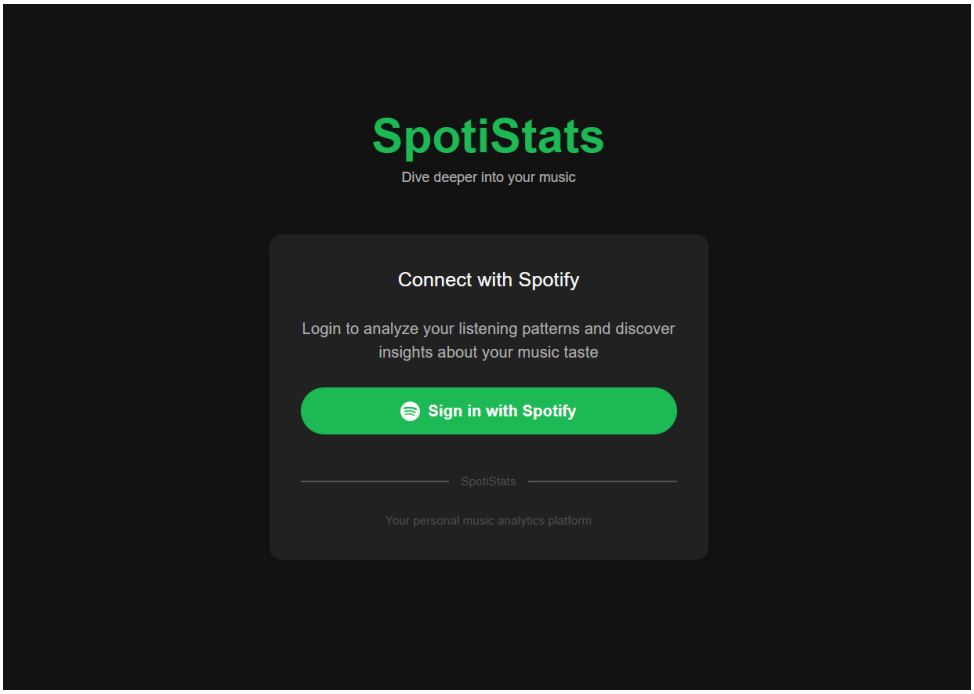


Рис. 1. Сторінка авторизації

Для успішної авторизації в системі, користувачу необхідно ввести свій логін (або адресу електронної пошти) та відповідний пароль у спеціально передбаченій формі авторизації (рис 2). Альтернативним варіантом є використання спрощеного процесу авторизації за допомогою облікових записів Google, Facebook або Apple. Для цього користувачу достатньо натиснути на відповідну кнопку «Continue», після чого система автоматично перенаправить його на сторінку авторизації вибраного провайдера, після успішного проходження якої користувач буде автоматично авторизований у системі, що робить процес швидким та зручним.

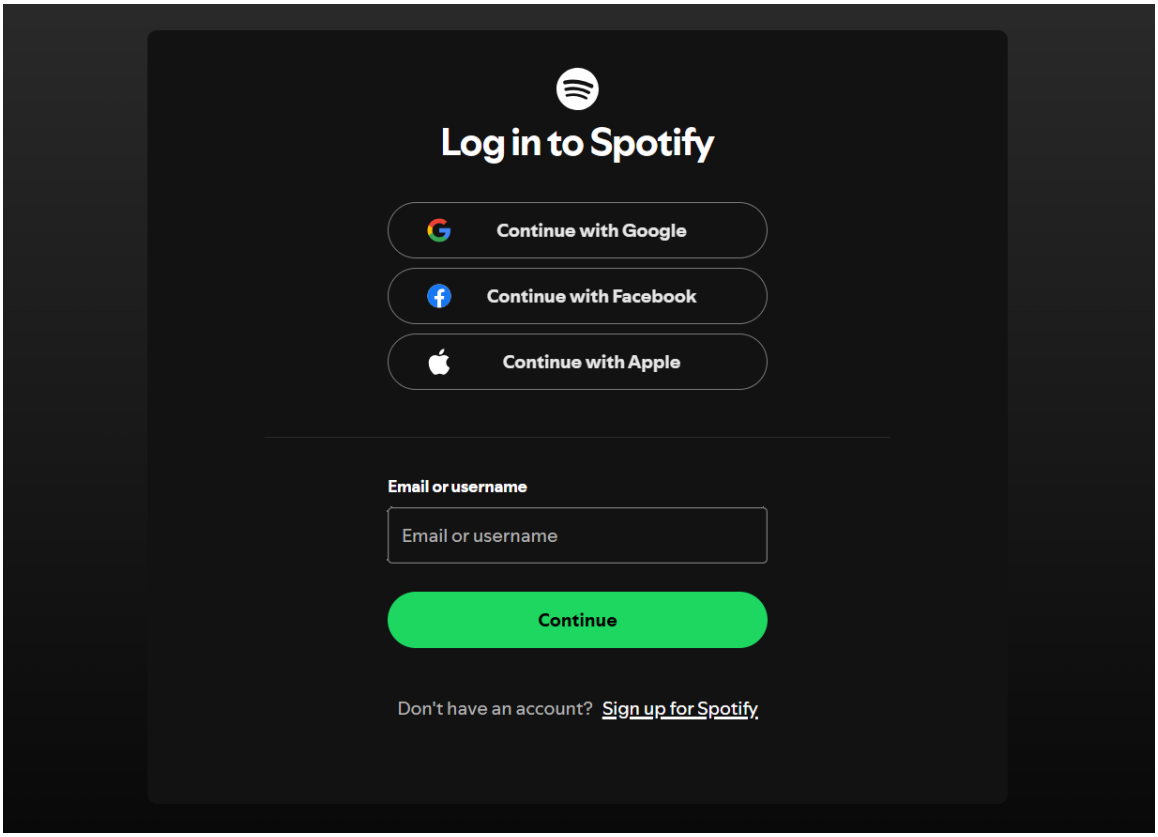


Рис. 2. Форма авторизації користувача

3. Головна сторінка вебзастосунку

Після авторизації користувач потрапляє на головну сторінку (рис. 3). Авторизувавшись в системі користувачу завжди буде відображатись

головне меню в верхній частині сторінки. Головне меню містить наступні елементи:

- посилання на головну сторінку вебзастосунку з обмеженим показом музичних вподобань «Overall»;
- посилання на сторінку вебзастосунку «Songs» з візуалізацією вподобань користувача за піснями;
- посилання на сторінку вебзастосунку «Albums» з візуалізацією вподобань користувача за альбомами;
- посилання на сторінку вебзастосунку «Artists» з візуалізацією вподобань користувача за артистами;
- посилання на сторінку вебзастосунку «Albums» з візуалізацією вподобань користувача за альбомами;
- посилання на сторінку вебзастосунку «Genres» з візуалізацією вподобань користувача за жанрами;
- аватар користувача та кнопка «Log Out»;

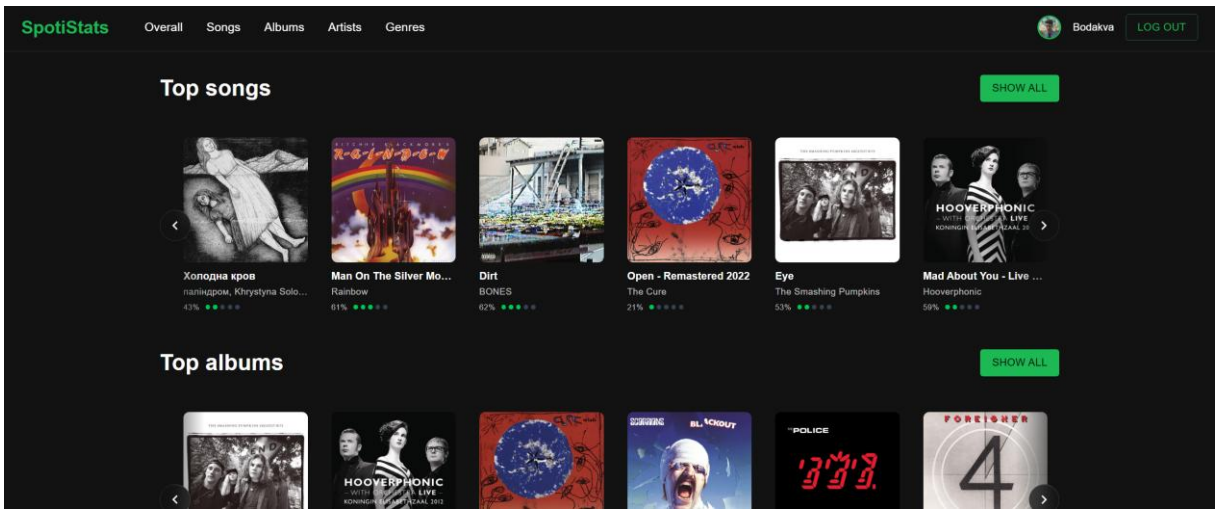


Рис. 3. Головна сторінка вебзастосунку

Слід зазначити, що для забезпечення зручності використання на мобільних пристроях, головне меню трансформується у спадаючий список (рис. 4) з усіма зазначеними вище посиланнями на сторінки вебзастосунку.

Це дозволяє ефективно використовувати обмежений простір екрану, зберігаючи при цьому доступ до всього функціоналу вебзастосунку.

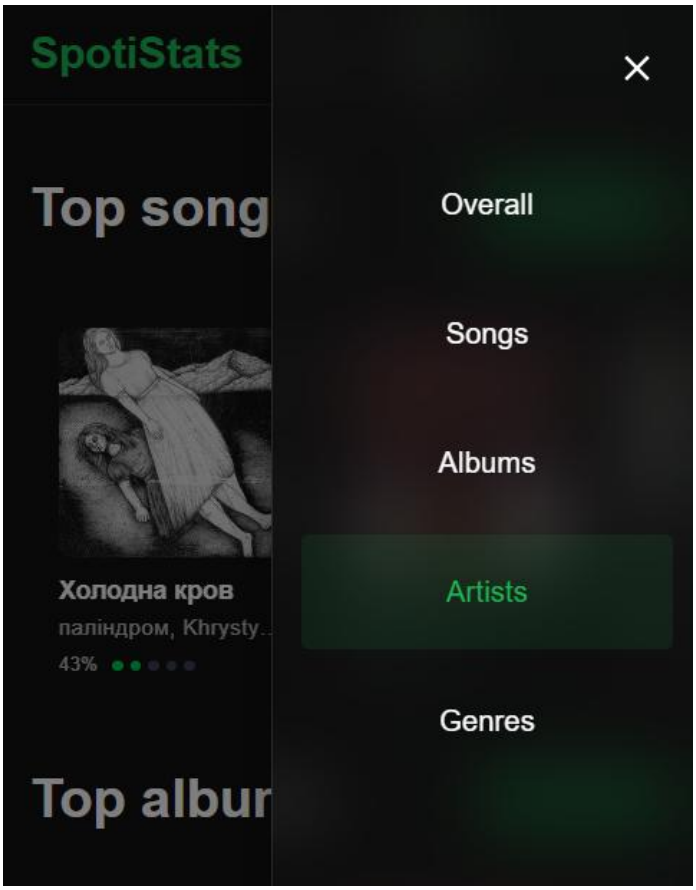


Рис. 4. Меню вебзастосунку в форматі для мобільних пристроїв

Головна сторінка вебзастосунку також містить інформацію про пісню, яку зараз слухає користувач (рис. 5), а також відображає історію нещодавніх прослуховувань (рис. 6).

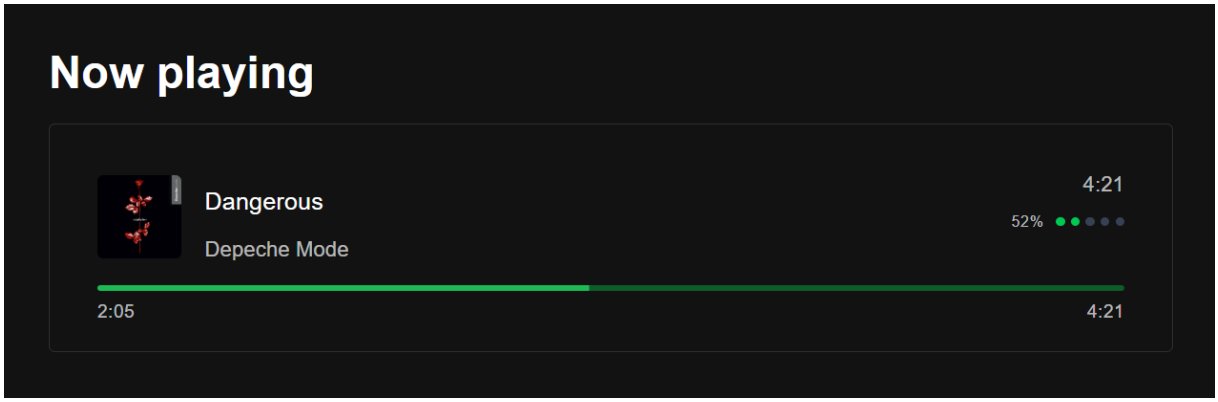


Рис. 5. Інформація про пісню, що зараз слухає користувач

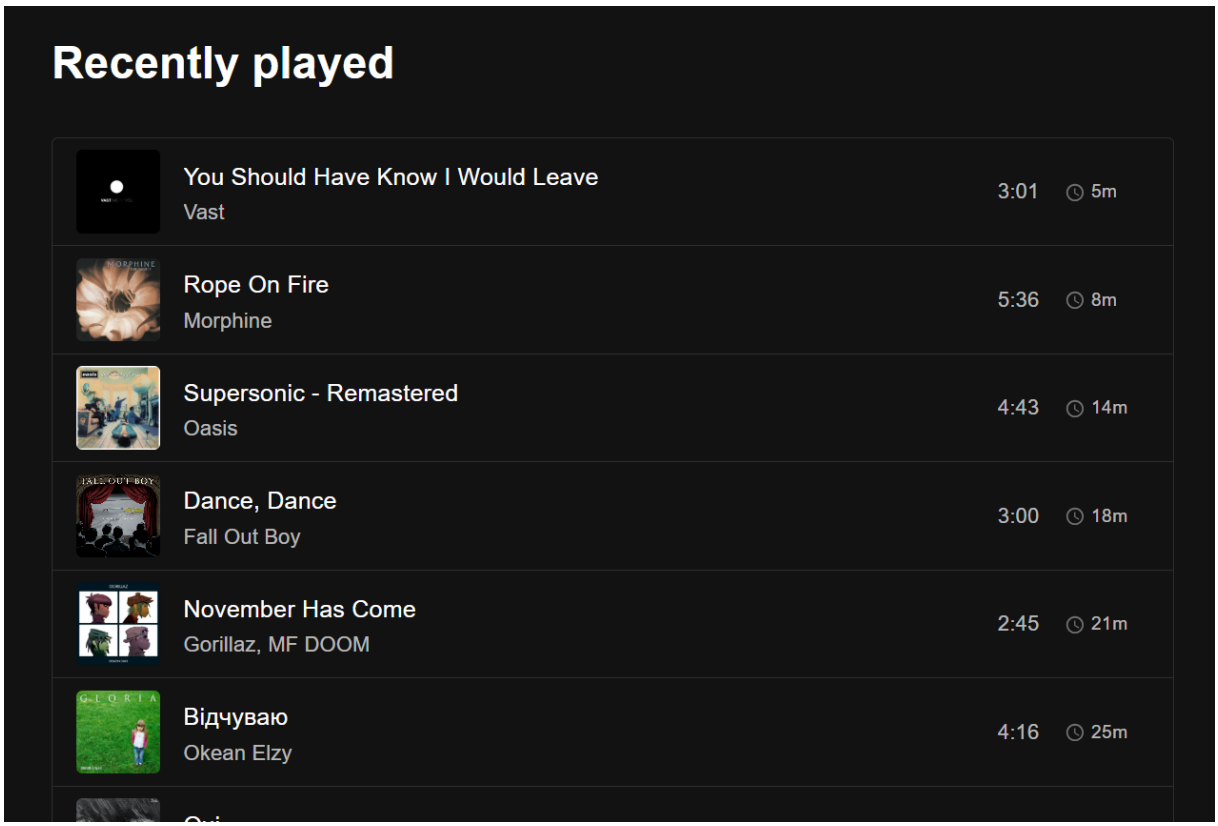


Рис. 6. Інформація нещодавно прослухані пісні користувачем

З головної сторінки вебзастосунку, представленої вище, користувач отримує швидкий доступ до каруселей (рис. 7) зі своїми найулюбленішими піснями, альбомами та виконавцями, які відображають його музичні вподобання за останні чотири тижні. Перегляд каруселей підтримує інтуїтивне сенсорне керування гортанням, що особливо зручно для користувачів мобільних пристроїв, а також передбачено зручні кнопки для навігації, що забезпечують комфортний перегляд на будь-яких пристроях. Карусель має приємну та плавну анімацію при гортанні, що робить процес перегляду більш привабливим та зручним. Для більш детального ознайомлення з музичними вподобаннями, користувачеві пропонується перейти на відповідні сторінки пісень, альбомів чи виконавців, де він зможе знайти розширену інформацію та статистику.

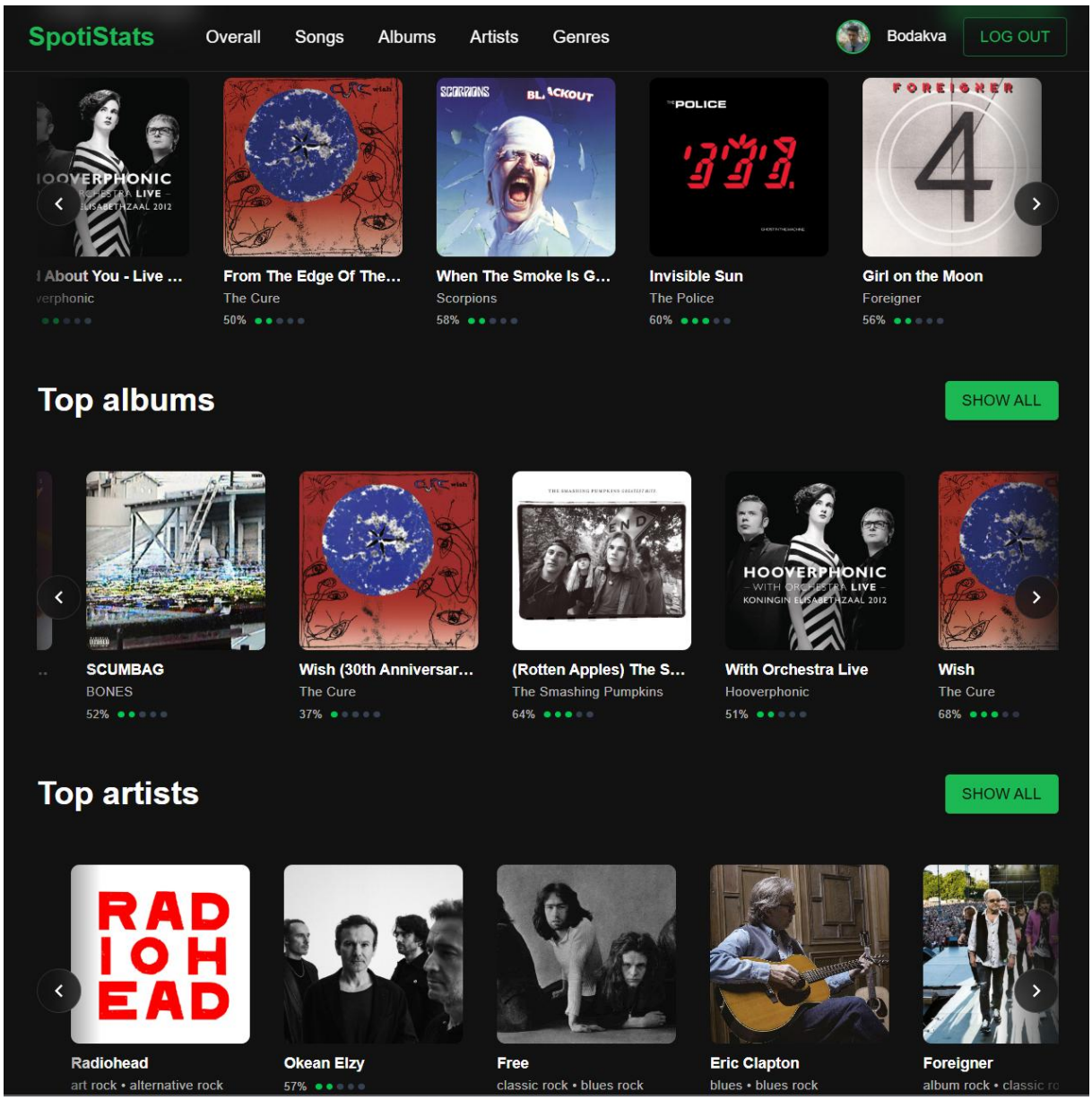


Рис. 7. Карусель музичних вподобань на головній сторінці

4. Сторінка «Songs»

На сторінці «Songs», візуальне представлення якої можна побачити на рис. 8, користувачеві надається можливість переглянути детальний аналіз своїх музичних вподобань, організований у розрізі окремих пісень. Для кожної пісні, представленої у списку, відображається показник популярності, що визначає її місце серед усього прослуханого контенту користувача, а також лічильник, який вказує на порядковий номер пісні у рейтингу музичних вподобань. Крім того, користувачеві надаються зручні

та інтуїтивно зрозумілі фільтри, які дозволяють налаштувати відображення пісень у відповідності до його потреб. Наприклад, він може переглянути лише ті пісні, які були прослухані за певний часовий проміжок, такий як останні чотири тижні, шість місяців або за весь час використання вебзастосунку. Також доступна опція фільтрування за кількістю відображених елементів, що дозволяє користувачеві налаштувати кількість пісень, які одночасно відображаються на сторінці, оптимізуючи таким чином перегляд відповідно до розміру екрану його пристрою та особистих вподобань.

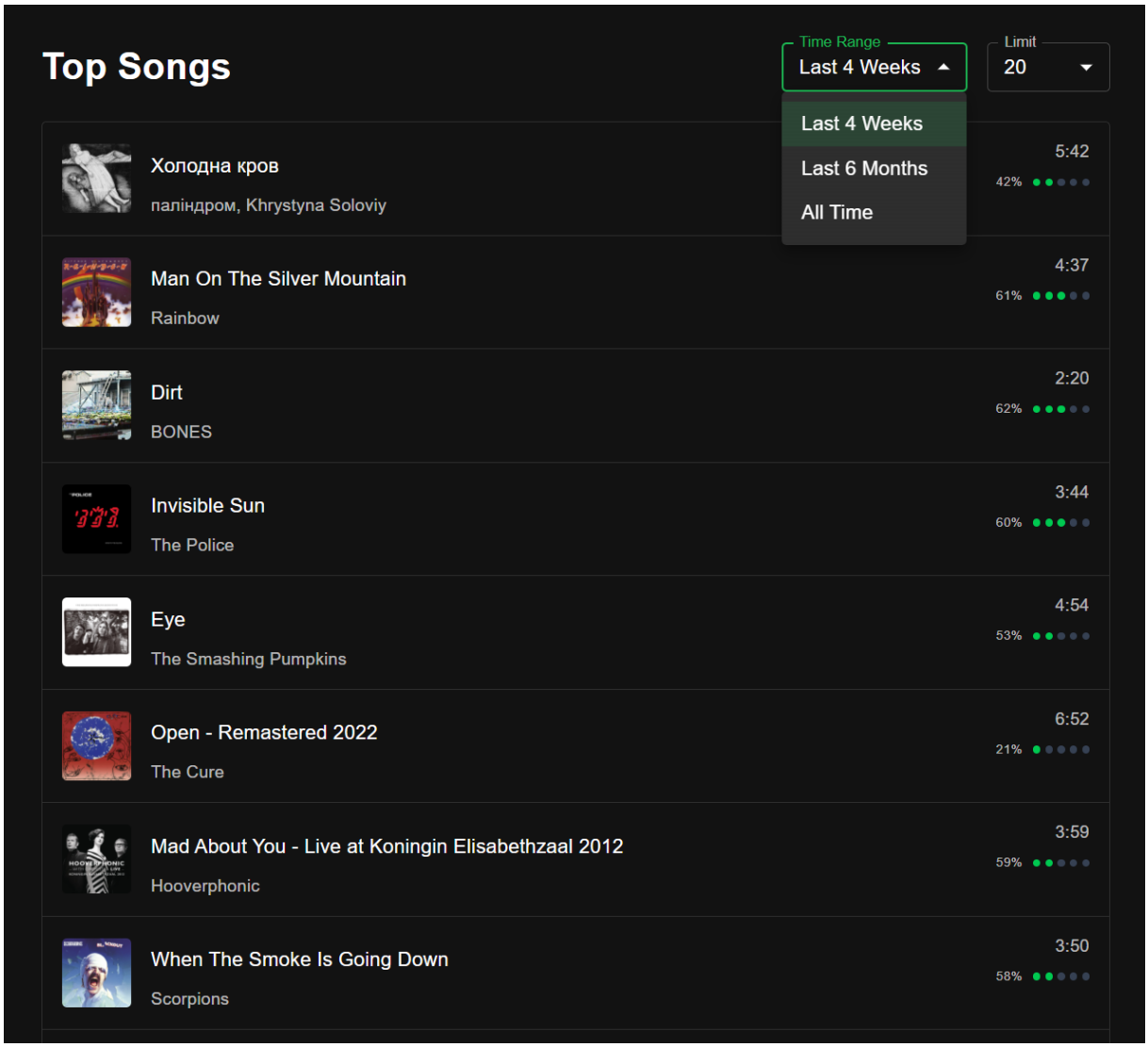


Рис. 8. Сторінка «Songs»

Більше того, якщо користувач натискає на конкретну пісню у списку, система автоматично відображає спливаюче вікно (рис. 9), яке містить розширену інформацію про обраний трек. Серед цієї інформації користувач знайде назву альбому, до якого належить пісня, що дозволяє йому зрозуміти контекст її створення, дату релізу пісні або альбому, що надає інформацію про її актуальність та історичне значення, а також позицію пісні в альбомі, що може бути важливим для розуміння концепції альбому як цілісного музичного твору. На додаток до цього, у спливаючому вікні користувачеві надається можливість прослухати обрану пісню безпосередньо у вебзастосунку завдяки вбудованому програвачу, а також передбачена зручна кнопка для швидкого переходу до прослуховування цієї пісні на платформі Spotify, що забезпечує інтеграцію з основним сервісом прослуховування музики.

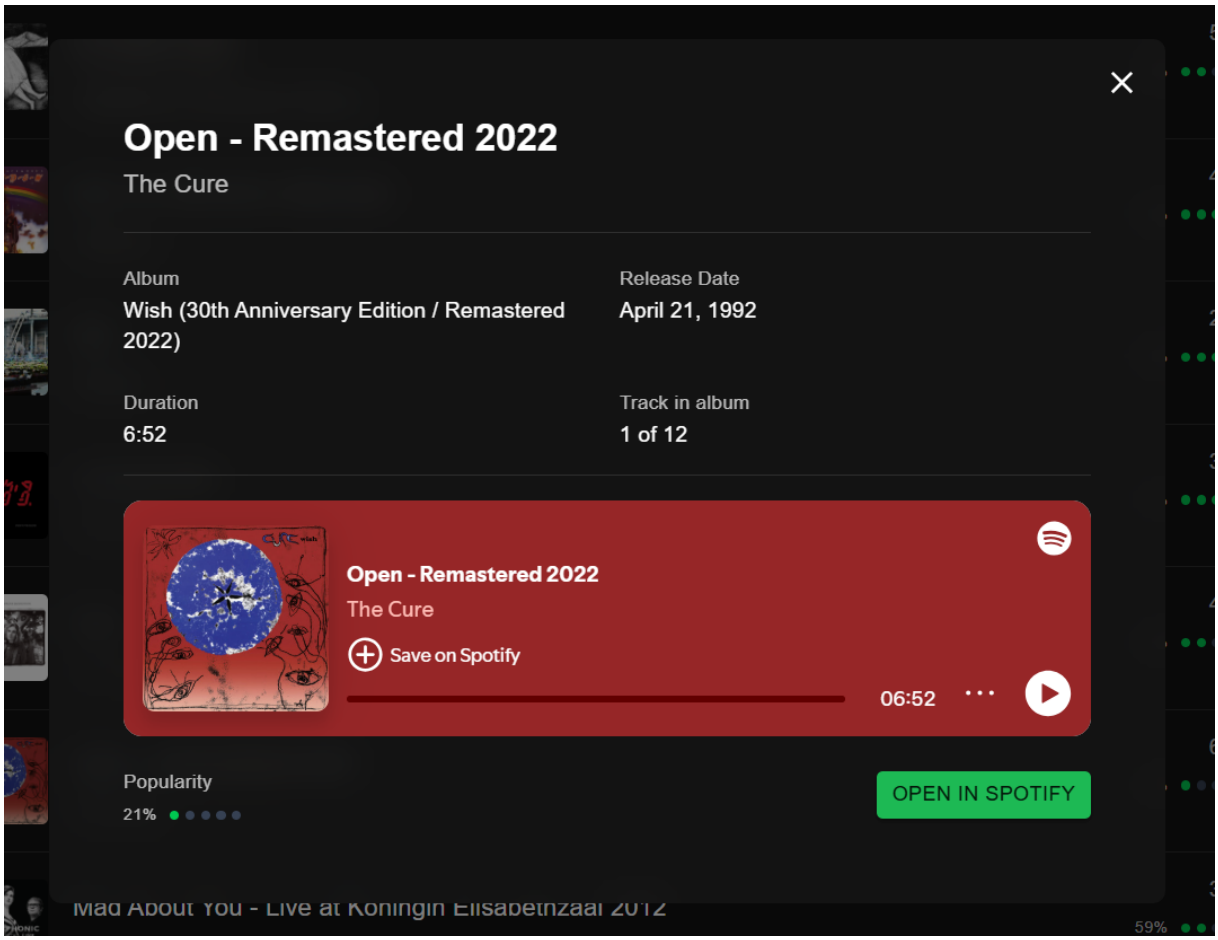


Рис. 9. Спливаюче вікно з інформацією про пісню

5. Сторінка «Albums»

На сторінці «Albums» (рис. 10) користувач може переглянути візуалізацію своїх музичних вподобань, організовану за альбомами. Для кожного альбому у списку відображається показник популярності та лічильник, що відображає його місце у музичних вподобаннях користувача. Крім того, користувачу доступні зручні фільтри, що дозволяють відображати його улюблені альбоми за певний часовий проміжок (наприклад, за останній чотири тижні, шість місяців або за весь час) та за кількістю відображених елементів.

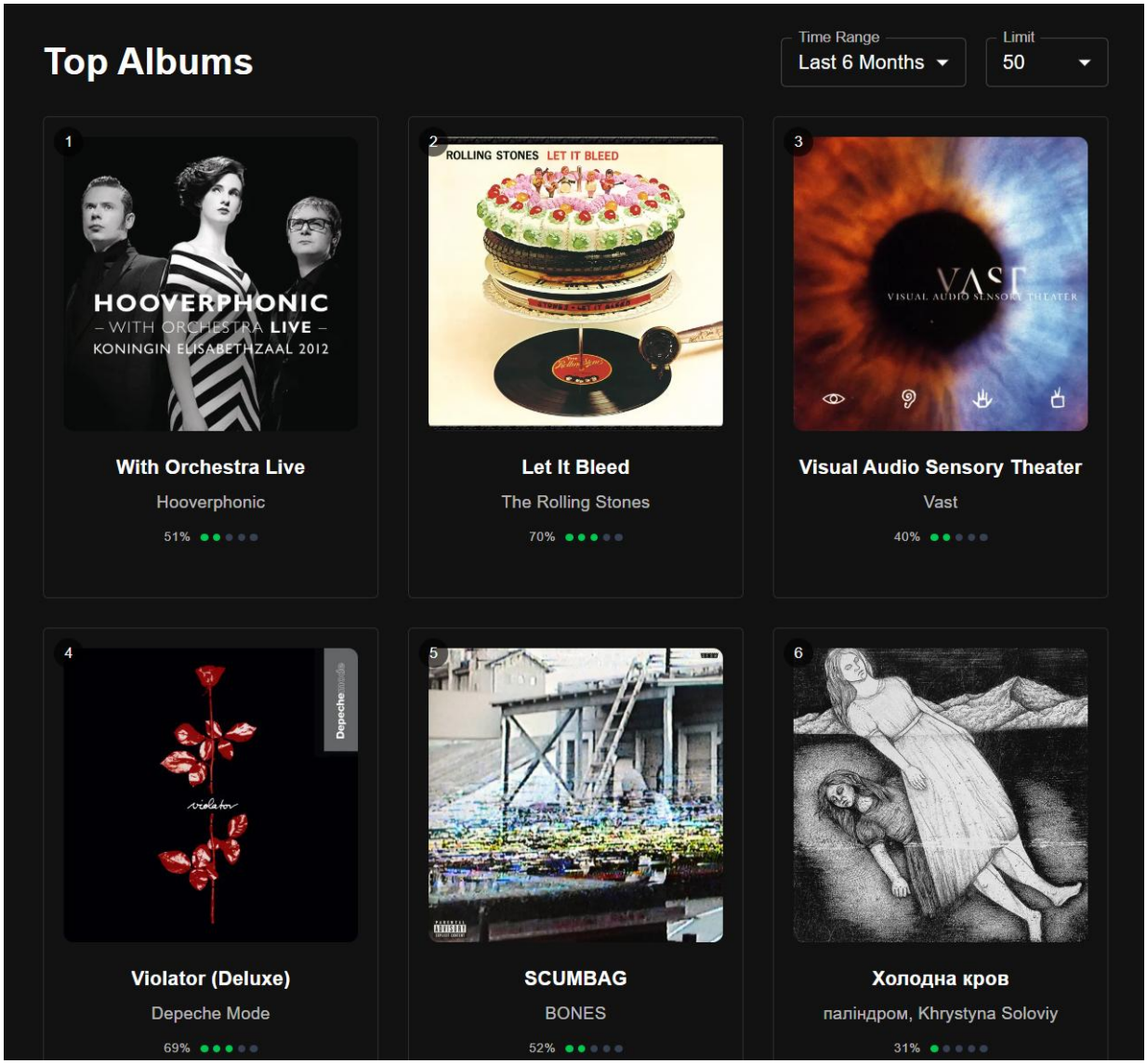


Рис. 10. Сторінка «Albums»

Крім того, при натисканні на зображення окремого альбому у списку, система відкриває спливаюче вікно (рис. 11), в якому користувачеві надається доступ до більш детальної інформації про вибраний музичний реліз. Зокрема, у вікні відображається назва альбому (або синглу, якщо це окремий трек), дата його випуску, що дозволяє оцінити його актуальність та історичний контекст, а також перелік жанрів, до яких він відноситься, що допомагає користувачеві зрозуміти музичну спрямованість альбому. Додатково, відображається загальна кількість пісень, що входять до складу альбому. Додатково користувачеві надається вбудований програвач цього альбому та кнопка для прослуховування на платформі Spotify.

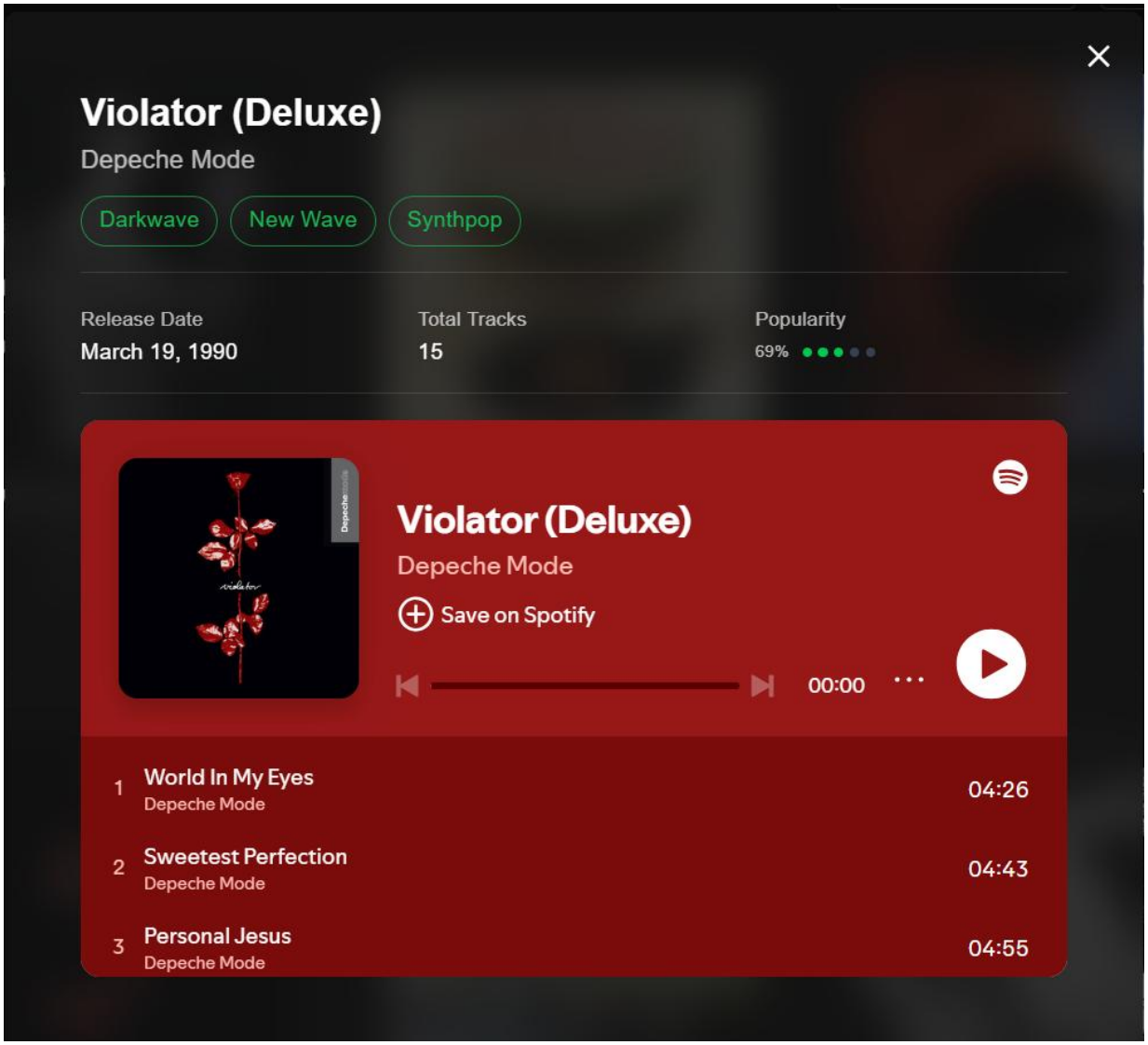


Рис. 11. Спливаюче вікно з інформацією про альбом

6. Сторінка «Artists»

На сторінці «Artists» (рис. 12) користувач може переглянути візуалізацію своїх музичних вподобань, організовану за виконавцями. Для кожного артиста у списку відображається показник популярності та лічильник, що відображає його місце у музичних вподобаннях користувача. Крім того, користувачу доступні зручні фільтри, що дозволяють відображати його улюблених артистів за певний часовий проміжок (наприклад, за останній чотири тижні, шість місяців або за весь час) та за кількістю відображених елементів.

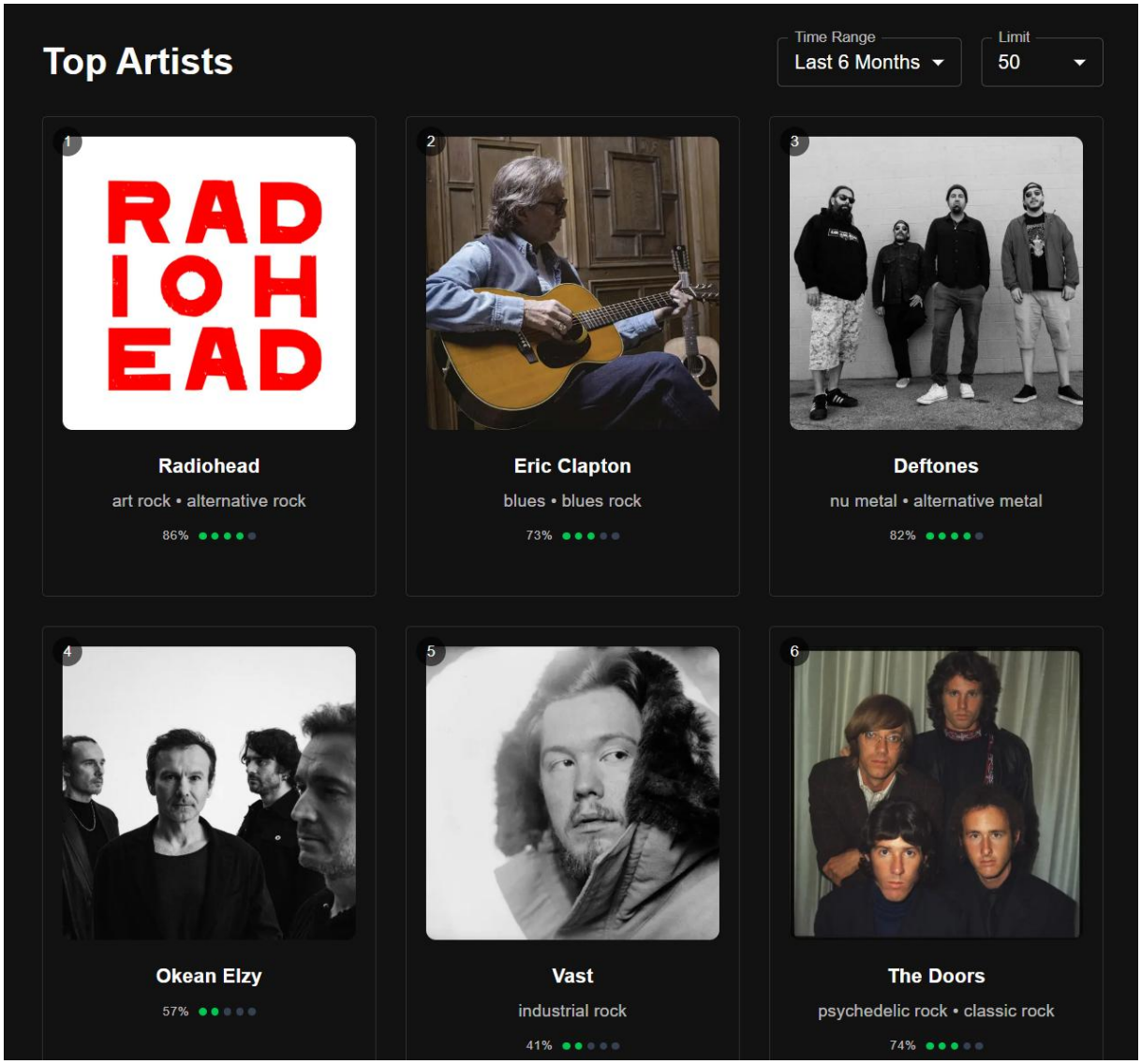


Рис. 12. Сторінка «Artists»

Також при натисканні на окремого артиста користувачеві відображається спливаюче вікно (рис. 13) з жанрами, в яких працює артист. Додатково користувачеві надається вбудований програвач з найпопулярнішими піснями артиста та кнопка для прослуховування на платформі Spotify.

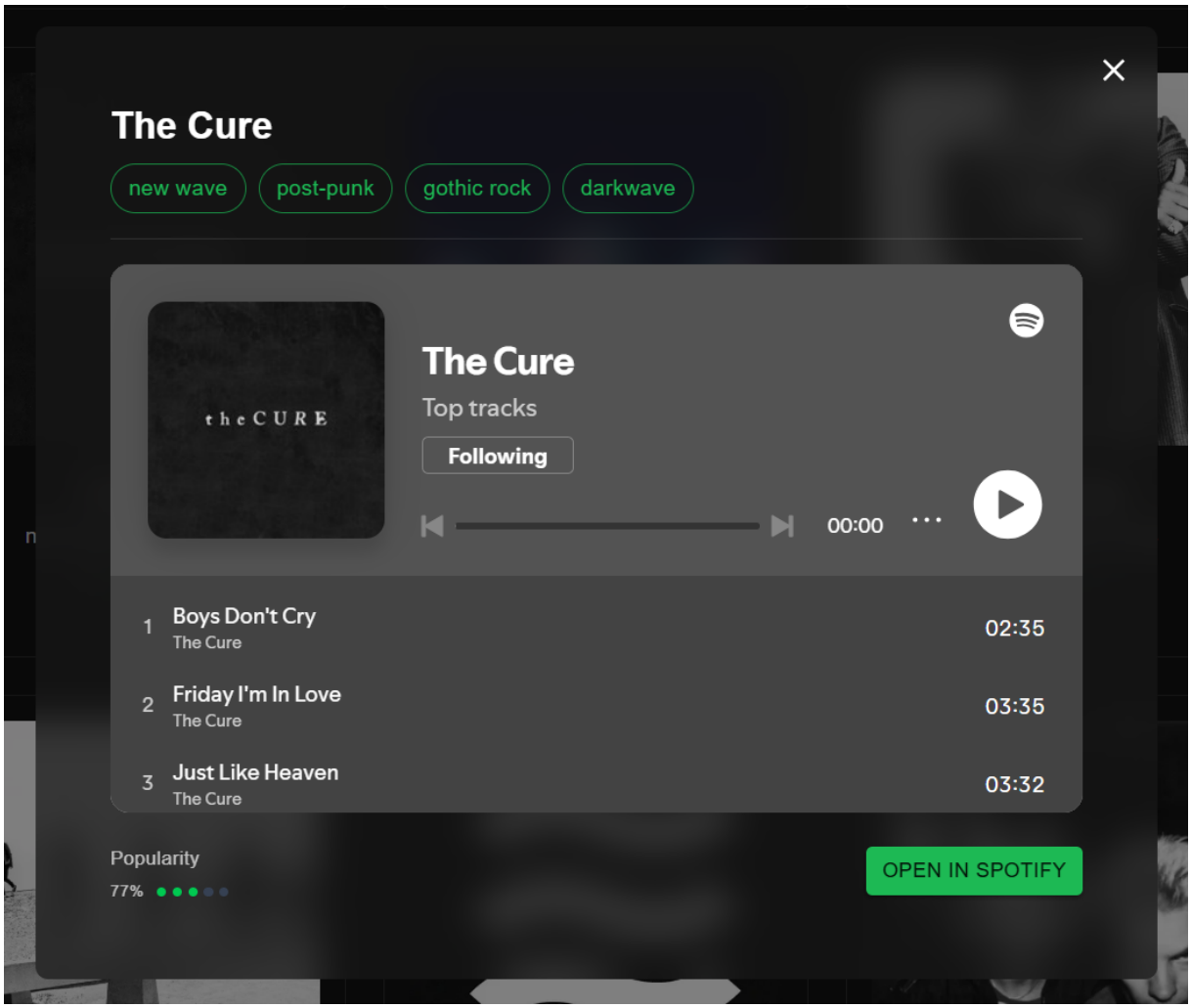


Рис. 13. Спливаюче вікно з інформацією про артиста

7. Сторінка «Genres»

На сторінці «Genres» (рис. 14) користувачу надається можливість візуалізувати свої музичні смаки у розрізі жанрів. Для кожного жанру, представленого у списку, відображається кількість артистів, чия музика відноситься до цього жанру, а також безпосередньо перелік цих артистів, що

дозволяє користувачеві швидко оцінити свою жанрову орієнтацію та улюблених виконавців у кожному жанрі. Користувачеві доступні фільтри для відображення його улюблених жанрів за часовим проміжком та за кількістю.

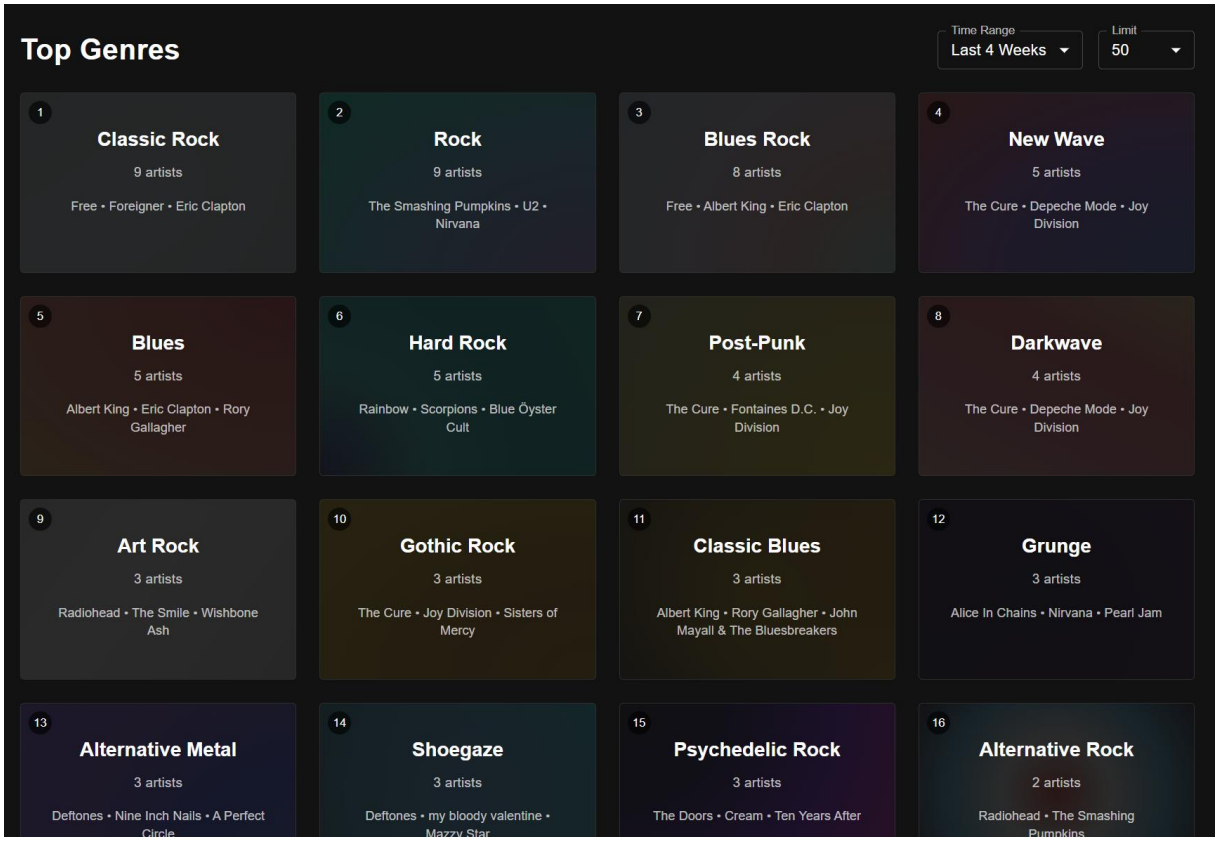


Рис. 14. Сторінка «Genres»