БАКАЛАВРСЬКА

РОБОТА

БР.ІП – 12.00.00.000 ІПЗ

Група ІП-21-3

Амброз Олександр

2025

Івано-Франківський національний технічний університет нафти і газу

Інститут інформаційних технологій

Кафедра інженерії програмного забезпечення

Амброз Олександр Степанович

(прізвище, ім'я, по батькові)

(індекс)

БАКАЛАВРСЬКА РОБОТА

Розробка frontend музичного веб-програвача локальної музики

(назва роботи)

Інженерія програмного забезпечення

(назва освітньої програми)

121– Інженерія програмного забезпечення

(шифр і назва спеціальності)

Робота містить результати власних досліджень, використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело:

Здобувач освітнього ступеня Амброз О.С.

(підпис, ініціали та прізвище здобувача)

Науковий керівник Піх Володимир Ярославович, к.т.н., доцент

(підпис, прізвище, ім'я, по батькові, науковий ступінь, вчене звання керівника)

Допущено до захисту
Завідувач кафедри

доц. Бандура В.В.

(посада) (підпис) (дата) (ініціали та прізвище)

Івано-Франківськ – 2025

Івано-Франківський національний технічний університет нафти і газу

Інститут, факультет інформаційних технологій

Кафедра інженерії програмного забезпечення

Ступінь вищої освіти бакалавр

Спеціальність 121 – Інженерія програмного забезпечення

***ЗАТВЕРДЖУЮ*:**
Зав. кафедрою ***ІПЗ***
доц. В.В. Бандура
“ ” 2025 р.

ЗАВДАННЯ

НА БАКАЛАВРСЬКУ РОБОТУ СТУДЕНТОВІ

Амброзу Олександру Степановичу

(прізвище, ім’я, по-батькові)

1. ***Тема проекту (роботи)*** ***“ Розробка frontend музичного веб-програвача локальної музики”***
керівник проекту (роботи) Піх Володимир Ярославович, к.т.н., доцент,

затверджені наказом закладу вищої освіти від “ 28 ” квітня 2025 р. № 264/7

2. ***Строк подання студентом проекту (роботи)*** червень 2025 р.

3. ***Вихідні дані до проекту (роботи)*** Розробити веб-додаток для прослуховування музики

4. ***Зміст розрахунково - пояснювальної записки (перелік питань, які потрібно розробити)***

1. Аналіз предметної області та засобів фронтенд-розробки
2. Аналіз вимог та постановка задачі
3. Проєктування системи
4. Реалізація проєкту
5. Тестування веб-додатку

5. ***Перелік графічного матеріалу (з точним зазначенням обов’язкових креслень)***

1. Діаграма прецедентів – (рис. 3.1, ст.40)
2. Діаграма послідовності – (рис. 3.2, ст.41)
3. Діаграма компонентів – (рис. 3.3, ст.42)
4. Головна сторінка Vibenest – (рис. 4.1, ст.49)
5. Кастомний плеєр – (рис. 4.2, ст.50)

1. ***Консультанти розділів проекту (роботи)***

Розділ	***Консультант***	***Підпис, дата***	
		Завдання видав	***Завдання прийняв***

2. ***Дата видачі завдання*** 15.01.25

Керівник _____ Піх В.Я.
(підпис) (розшифровка підпису)

Завдання прийняв до виконання _____ Амброз О.С.
(підпис) (розшифровка підпису)

КАЛЕНДАРНИЙ ПЛАН

№ з/п	***Назва етапів дипломного проекту (роботи)***	***Строк виконання етапів проекту***	***Примітка***
1	Одержання завдання, ознайомлення з ним, вибір літератури	15.01.25	виконано
2	Робота над теоретичною частиною	25.02.25	виконано
3	Робота над практичною частиною	15.03.25	виконано
4	Тестування та виправлення помилок	15.05.25	виконано
5	Оформлення роботи	22.05.25	виконано

Студент – дипломник _____ Амброз О.С.
(підпис) (розшифровка підпису)

Керівник роботи _____ Піх В.Я.
(підпис) (розшифровка підпису)

АНОТАЦІЯ

Бакалаврська робота містить 74 сторінки, 16 рисунків, 1 таблицю, 1 додаток.

Метою роботи є створення клієнтської частини веб-платформи для прослуховування музики в режимі реального часу, що забезпечує зручність, адаптивність і високу продуктивність під час взаємодії з інтерфейсом.

Об'єктом дослідження виступає процес розробки інтерфейсу веб-застосунку для відтворення аудіо-контенту.

Предмет дослідження охоплює технології frontend-розробки, зокрема використання бібліотек React, Tailwind CSS, Framer Motion, а також принципи побудови UI/UX, обробки подій, маршрутизації та роботи з REST API.

Результати дослідження: реалізовано односторінковий клієнтський застосунок (SPA).

У першому розділі подано огляд еволюції музичних сервісів, аналіз популярних платформ і обґрунтування вибору технологій для розробки клієнтської частини.

У другому розділі зібрано вимоги до інтерфейсу, сформульовано функціональні та нефункціональні характеристики, визначено основні задачі для реалізації фронтенду.

У третьому розділі розроблено архітектуру клієнтської частини на основі SPA-підходу, описано структуру компонентів, управління станом та логіку навігації.

У четвертому розділі описано реалізацію ключових модулів, особливості інтеграції з API, анімацію, адаптивність та механізми взаємодії з користувачем.

У п'ятому розділі проведено тестування інтерфейсу, виявлено проблемні місця, проаналізовано продуктивність і зручність взаємодії на різних пристроях та в різних браузерах.

Висновок: клієнтська частина веб-платформи розроблена відповідно до сучасних вимог веб-інженерії, що забезпечує позитивний користувацький досвід і ефективну роботу з музичним контентом у браузері.

КЛЮЧОВІ СЛОВА: МУЗИЧНА ПЛАТФОРМА, FRONTEND, REACT, REDUX, АУДІОПЛЕЄР, ВЕБ-ІНТЕРФЕЙС, UI/UX, ПОТОКОВЕ ВІДТВОРЕННЯ, TAILWIND CSS, REST API, VITE, АДАПТИВНІСТЬ, ТЕСТУВАННЯ.

ANNOTATION

The bachelor's thesis consists of 74 pages, 16 figures, 1 table, and 1 appendix.

The aim of the research is to create a client-side web platform for listening to music in real-time, providing convenience, adaptability, and high performance during interaction with the interface.

The object of the research is the process of developing the interface of a web application for playing audio content.

The subject of the research covers frontend development technologies, including the use of React, Tailwind CSS, and Framer Motion libraries, as well as principles of UI/UX design, event handling, routing, and working with REST APIs.

Research results: a single-page client-side application (SPA) has been implemented.

The first chapter provides an overview of the evolution of music services, analysis of popular platforms, and justification for choosing the technologies for developing the client-side.

The second chapter gathers the interface requirements, formulates the functional and non-functional characteristics, and defines the main tasks for frontend implementation.

The third chapter describes the architecture of the client-side based on the SPA approach, the structure of components, state management, and navigation logic.

The fourth chapter describes the implementation of key modules, integration with the API, animations, adaptability, and mechanisms for user interaction.

The fifth chapter presents the interface testing, identifies issues, and analyzes performance and usability across different devices and browsers.

Conclusion: the client-side of the web platform was developed in accordance with modern web engineering standards, providing a positive user experience and efficient handling of music content in the browser.

KEYWORDS: MUSIC PLATFORM, FRONTEND, REACT, REDUX, AUDIO PLAYER, WEB INTERFACE, UI/UX, STREAMING, TAILWIND CSS, REST API, VITE, RESPONSIVENESS, TESTING.

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

SPA – «Single Page Application» (односторінковий вебзастосунок, що працює без повного перезавантаження сторінки).

UI – «User Interface» (інтерфейс користувача, графічне середовище для взаємодії з вебзастосунком).

UX – «User Experience» (користувацький досвід, загальне враження від використання інтерфейсу).

API – «Application Programming Interface» (інтерфейс прикладного програмування, набір методів для взаємодії між клієнтом і сервером).

DOM – «Document Object Model» (об'єктна модель документа, яка відображає HTML-структуру вебсторінки у вигляді дерева).

JS – «JavaScript» (мова сценаріїв для створення інтерактивності на вебсторінках).

HTML – «HyperText Markup Language» (мова гіпертекстової розмітки для створення структури вебдокументів).

CSS – «Cascading Style Sheets» (каскадні таблиці стилів, що визначають вигляд і оформлення HTML-елементів).

REST – «Representational State Transfer» (архітектурний стиль побудови API для передачі даних через HTTP).

HTTP – «HyperText Transfer Protocol» (протокол передачі гіпертексту у вебмережах).

React – «React» (бібліотека JavaScript для побудови інтерфейсів користувача на основі компонентів).

Redux – «Redux» (JavaScript-бібліотека для централізованого керування станом застосунку).

Tailwind CSS – «Tailwind CSS» (утилітарний CSS-фреймворк для адаптивної верстки й швидкого створення стилів).

Framer Motion – «Framer Motion» (бібліотека для створення анімацій у React-додатках).

JSON – «JavaScript Object Notation» (формат обміну даними у вигляді тексту, зручний для читання та аналізу).

CRUD – «Create, Read, Update, Delete» (основні операції над даними: створення, читання, оновлення, видалення).

npm – «Node Package Manager» (менеджер пакетів для встановлення бібліотек і залежностей у JavaScript-проєктах).

URL – «Uniform Resource Locator» (уніфікований локатор ресурсу, адреса сторінки або файлу в Інтернеті).

IDE – «Integrated Development Environment» (інтегроване середовище розробки програмного забезпечення).

ЗМІСТ

ВСТУП

У XXI столітті цифрові технології стали невід'ємною складовою культурного, соціального та економічного життя. Особливо швидкий розвиток спостерігається у сфері мультимедійного контенту, де музика займає одне з провідних місць. З переходом на цифрові платформи змінилися не лише способи прослуховування треків, але й очікування користувачів стосовно функціональності та зручності інтерфейсу. У цьому контексті розробка інтуїтивно зрозумілих, динамічних та візуально привабливих вебінтерфейсів для музичних сервісів є актуальною задачею для сучасної IT-галузі.

Вебплатформи для музики мають відповідати високим вимогам до юзабіліті, стабільності, адаптивності й продуктивності. Сучасний користувач очікує не лише можливість миттєвого доступу до треків, але й зручний пошук, просте додавання улюбленого контенту, створення персоналізованих добірок та приємну візуальну взаємодію. Тому розробка клієнтської частини таких застосунків повинна враховувати принципи UX/UI-дизайну, сучасні фреймворки та передові підходи до структурування компонентів, обробки подій, роботи зі станом застосунку та оптимізації рендерингу.

Актуальність теми бакалаврської роботи зумовлена потребою у створенні нових музичних сервісів із високою якістю фронтенд-реалізації. Існуючі рішення, попри широке розповсюдження, не завжди відповідають очікуванням користувачів у частині зручності інтерфейсу, швидкості роботи та адаптивності. Особливо це стосується нових чи експериментальних рішень, які мають потенціал бути впровадженими у стартапи або продукти, орієнтовані на молодіжну аудиторію.

Метою бакалаврської роботи є створення вебплатформи «Vibenest», яка надає користувачам можливість слухати музику в режимі реального часу, додавати треки до особистих добірок, зберігати улюблені композиції, а також взаємодіяти з інтерфейсом за допомогою сучасних засобів візуалізації та

анімацій. При цьому ключовий акцент зроблено на розробку клієнтської частини, що забезпечує високу продуктивність, гнучку архітектуру та зручний користувацький досвід.

Завданнями дослідження є аналіз особливостей сучасних музичних стрімінгових сервісів, вивчення актуальних технологій для реалізації фронтенд-інтерфейсів, проєктування компонетної архітектури вебзастосунку та розробка його ключових функціональних блоків. Особлива увага приділяється створенню адаптивної верстки, забезпеченню інтерактивності через анімації, а також тестуванню користувацького досвіду і оптимізації продуктивності компонентів.

Об’єкт дослідження — процес створення фронтенд-архітектури сучасного вебзастосунку для відтворення музики.

Предмет дослідження — технології, бібліотеки та методи реалізації клієнтської частини вебплатформи з використанням React, Tailwind CSS, Framer Motion, а також підходів до побудови UI/UX-структури, управління станом компонента, маршрутизації, роботи з подіями drag-and-drop і реалізації інтерактивної взаємодії з аудіоконтентом.

Методи дослідження — це моделювання користувацьких сценаріїв, тестування інтерфейсних компонентів, моніторинг продуктивності вебзастосунку, аналіз взаємодії користувачів з інтерфейсом, проведення стрес-тестів на навантаження, а також оцінка адаптивності й доступності. Застосування цих методів допомагає глибше зрозуміти, наскільки зручним і стабільним є клієнтський інтерфейс музичної платформи, а також виявити можливості для покращення користувацького досвіду. Вони дозволяють оптимізувати роботу плеєра, забезпечити плавність анімацій і швидкість реакції на дії користувача, що є ключовим для сучасних мультимедійних сервісів.

Наукова новизна полягає у поєднанні сучасних технологічних рішень (React, Tailwind CSS, Framer Motion) із практичними підходами до побудови інтерфейсу для стрімінгового сервісу, який не лише відтворює аудіо, а й забезпечує високу інтерактивність, візуальну привабливість і швидкодію.

Запропонований підхід до створення вебінтерфейсу дозволяє ефективно поєднувати технічні рішення з принципами UX-дизайну, а також легко масштабувати платформу для впровадження додаткового функціоналу.

Бакалаврська робота охоплює 74 сторінки, містить 16 рисунків 5 розділів, також подано 1 додаток.

РОЗДІЛ 1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ЗАСОБІВ ФРОНТЕНД-РОЗРОБКИ

1.1 Актуальність теми та мета дослідження

У сучасну епоху цифровізації, коли вебзастосунки стали невід'ємною частиною повсякденного життя, значно зростає попит на інтуїтивно зрозумілі, швидкі та візуально привабливі інтерфейси, зокрема в галузі онлайн-сервісів для прослуховування музики. Такі платформи, як Spotify, Apple Music та інші, демонструють важливість якісного фронтенд-рішення для забезпечення зручного доступу до контенту. Водночас вебверсії подібних сервісів не завжди повною мірою відповідають очікуванням користувачів у плані взаємодії, навігації та швидкодії.

Особливої уваги вимагає розробка музичних вебплеєрів, які поєднують сучасні технології клієнтської частини з естетично продуманим інтерфейсом. Успішність таких продуктів залежить не лише від функціоналу, а й від зручності користування, адаптивності до різних пристроїв, інтерактивності та швидкої реакції на дії користувача. З огляду на це, розробка фронтенду для музичного сервісу є актуальним завданням, що поєднує елементи дизайну, вебінженерії та користувацького досвіду.

Метою даної бакалаврської роботи є створення сучасного вебінтерфейсу для музичної платформи Vibenest, який забезпечує зручну взаємодію з основними функціями сервісу: завантаженням треків (у тому числі через drag-and-drop та форму), потоковим відтворенням із кастомним плеєром, формуванням добірок, додаванням до улюблених, а також динамічною навігацією без перезавантаження сторінок. Окрема увага приділяється реалізації анімацій та інтерактивних елементів, що покращують користувацький досвід, а також інтеграції з серверною частиною через REST API.

Таким чином, дослідження спрямоване на вирішення задачі побудови

функціонального, адаптивного та зручного для користувача вебінтерфейсу з урахуванням сучасних технологічних вимог і стандартів фронтенд-розробки.

1.2 Розвиток музичних технологій

Музика є важливою частиною нашого життя. З часом музика кардинально змінилася, як з точки зору перегляду, так і розповсюдження — від вінілових платівок та радіотрансляцій до цифрових стрімінгових сервісів, які забезпечують доступ до величезної бібліотеки та можливість слухати чудову музику цілий день без перерви.

1.2.1 Вінілові платівки та радіо

У середині 20-го століття вінілові платівки були популярним способом поширення музики. Ці круглі платівки з отворами для програвача дозволяли програвати улюблені платівки вдома. Паралельно радіо транслює музичні програми, тож ви можете слухати нові пісні та улюблені хіти наживо.

1.2.2 Касети та компакт-диски

У 1970-х і 1980-х роках музичні касети та компакт-диски були звичайним явищем. Касети, які могли відтворювати музику по радіо чи іншим чином, стали популярним інструментом. Однак компакт-диски звучали чудово і їх було комфортніше слухати.

1.2.3 Цифровий MP3-плеєр

На початку 21 століття з'явилися MP3-плеєри та цифрові формати, такі як MP3 та AAC. Ці пристрої та формати дозволили людям зберігати та відтворювати тисячі пісень на компактних пристроях, таких як iPod, і брати свою музику з собою, куди б вони не йшли.

1.2.4 Стрімінгові сервіси

Сьогодні такі стрімінгові сервіси, як Spotify, Apple Music та YouTube Music, є найпопулярнішими стрімінговими сервісами для мільйонів користувачів у всьому світі. Ці сервіси надають доступ до великих музичних

бібліотек як на основі підписки, так і безкоштовно з рекламою, що робить його доступним, недорогим та доступним будь-коли та будь-де.

Музичні технології зазнали суттєвого розвитку з появою вінілових платівок та радіо, а також цифрових потокових сервісів, які дозволяють людям слухати музику онлайн. З кожним новим етапом цієї еволюції для меломанів у всьому світі відкриваються нові можливості та перспективи [13].

1.3 Фронтенд застосунку: сутність, роль та базові технології

Фронтенд застосунок, також відомий як користувацький інтерфейс (UI), — це частина вебдодатку, який користувач може використовувати, бачити та взаємодіяти з ним за допомогою кнопок, зображень, інтерактивних елементів, меню навігації та тексту.

Застосунки працюють від серверної частини до фронтенду. Весь процес починається з дизайну. Після визначення дизайну та вмісту сторінки розробник фронтенду може розпочати розробку застосунку.

Після розробки фронтенду застосунку користувач може отримати до нього доступ. Відкривши фронтенд застосунку, користувач одразу починає взаємодіяти зі шрифтами, кольорами та потоком інтерфейсу. Ці елементи керують користувачем навігацією в застосунку, тоді як фронтенд збирає дані з взаємодій.

Фронтенд застосунку є абсолютно критично важливим для візуальних аспектів застосунку. Завдяки використанню HTML, JavaScript та каскадних таблиць стилів (CSS) розробники можуть впроваджувати ефекти та стилістичні налаштування. UI/UX також відіграє важливу роль у розробці фронтенду, оскільки користувачі отримають найкращий досвід завдяки цілеспрямованому впровадженню UI/UX.

Деякими прикладами фронтенд-додатків, що дозволяють використовувати інструменти веб-мов, є HTML, JavaScript та CSS. Ці три

приклади, як згадувалося вище, важливі для дизайну інтерфейсу та доступності.

– HTML закладає фактичну основу для кожного веб-сайту, що існує в Інтернеті, дозволяючи відображати сторінку. Без HTML веб-браузер не має інформації про макет та відображення сторінки. Процес починається з парсингу, який гарантує, що кожен тег у файлі враховано та включено.

– JavaScript базується на тексті та використовується в розробці фронтенд-додатків. Цей фронтенд-додаток є ключем, який забезпечує інтерактивні елементи веб-сторінки, з якими користувач безпосередньо взаємодіє. Відформатований на основі власної унікальної мови, JavaScript динамічно інтерпретує кожен окремий фрагмент коду, один за раз. Діючи як мова програмування, він перевіряє код у режимі реального часу, щоб забезпечити безперебійний процес інтерфейсу.

– CSS визначає макет та стиль веб-сторінок. Цей фронтенд-додаток має вирішальне значення для повного налаштування веб-сторінки, оскільки він дозволяє налаштовувати та змінювати кожен елемент вмісту, включаючи розмір шрифту, інтервали, анімацію.

Крім базових технологій, сучасні фронтенд-додатки використовують бібліотеки та фреймворки, такі як React, Vue, Angular, які спрощують розробку складних інтерфейсів і дозволяють ефективно керувати станом застосунку.

Не менш важливою складовою фронтенду є UI/UX-дизайн, що визначає загальну логіку навігації, послідовність дій користувача та загальне враження від взаємодії. Завдяки правильно організованому UI/UX-дизайну користувач отримує логічно послідовний, зручний і приємний у використанні інтерфейс.

Таким чином, фронтенд є не лише візуальною частиною застосунку, але й ключовим інструментом для взаємодії користувача з функціональністю системи.

1.4 Аналіз популярних web-сервісів для прослуховування музики

1.4.1 Глобальна статистика

Згідно з рисунком 1.1, опублікованим Міжнародною федерацією фонографічної індустрії (IFPI), глобальний дохід від записаної музики у 2024 році досяг 29,6 млрд доларів США, що стало новим історичним максимумом. Порівняно з попереднім роком (2023), індустрія зросла на 5,8%.

Основну частину доходів забезпечує стримінг — 20,4 млрд доларів або понад 68% від загального обсягу. Цей сегмент включає як підписні сервіси (Spotify, Apple Music, YouTube Music тощо), так і стримінг з рекламою. Зростання доходів від стримінгу у 2024 році склало 8% порівняно з 2023 роком.

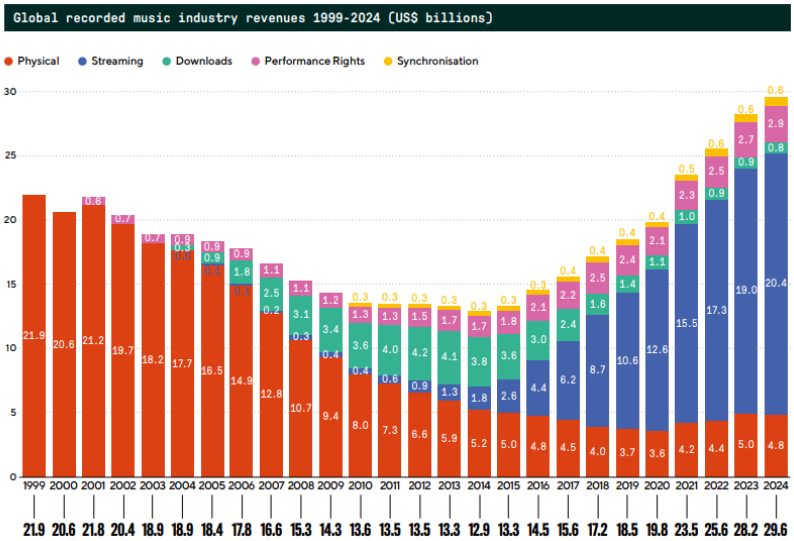


Рисунок 1.1 – Діаграма глобальних доходів індустрії записаної музики

Spotify займає найбільшу частку ринку, з незначним зниженням частки, але стабільним лідерством за кількістю користувачів. За ним йдуть Apple Music, Amazon Music та YouTube Music, причому остання демонструє найшвидше зростання серед усіх західних платформ.

Зростання ринку особливо сильне в Азії, Латинській Америці та Африці, де нові підписки та потокове передавання зростають швидше, ніж на розвинених

ринках.

У галузі також спостерігається високий рівень залученості користувачів, середній слухач витрачає понад 20 годин на тиждень на прослуховування музики, що є новим рекордом.

Це зростання доводить, що потокове передавання залишається ключовим каналом споживання музики в цифрову епоху, задаючи напрямок розвитку галузі в майбутньому [1].

1.4.2 Аналіз Spotify

Шведський сервіс, який вже давно є найпопулярнішим сервісом потокового мовлення музики у світі, значно перевершуючи конкурентів за багатьма показниками. Цей сервіс є справжньою музичною соціальною мережею: тут ви можете додавати друзів, обмінюватися повідомленнями та коментувати музичні новини. На цій платформі вам доступний величезний каталог пісень, зручний інтерфейс та система пошуку, а також рекомендації на основі уподобань користувача. Підтримує практично всі пристрої для відтворення треків, на яких можна налаштувати синхронізацію вашої медіатеки та операційних систем.

Spotify залишається провідним гравцем на світовому ринку музичного стрімінгу й сьогодні. Станом на кінець 2024 року кількість щомісячних активних користувачів платформи перевищила 602 мільйони, з яких понад 236 мільйонів є платними передплатниками. Це являє собою збільшення приблизно на 15% порівняно з 2023 роком, коли кількість преміум-користувачів становила близько 205 мільйонів.

Spotify зберігає лідируючу частку світового ринку музичного стрімінгу – за даними аналітиків MIDiA Research, частка компанії на світовому ринку платного аудіострімінгу у 2024 році становила близько 30%. Ця позиція закріплена завдяки агресивній глобальній експансії, постійному вдосконаленню алгоритмів рекомендацій та пропозиції локалізованого контенту [2].

Інтерфейс Spotify представлено на рисунку 1.3.

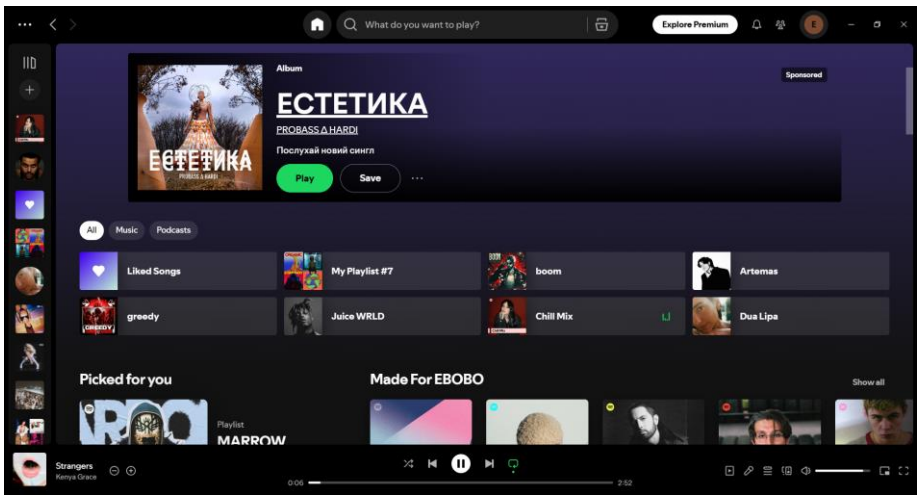


Рисунок 1.2 – Інтерфейс Spotify

Переваги:

– Величезна медіатека — понад 100 мільйонів пісень;

– Точні алгоритми рекомендацій, що враховують уподобання користувача;

– Можливість користуватися сервісом безкоштовно з певними обмеженнями (реклама, обмежений контроль треків);

– Тривалий пробний період Premium (зазвичай 1 місяць, іноді до 3 місяців);

– Високий бітрейт відтворення — до 320 кбіт/с (у Premium);

– Синхронізація відтворення на різних пристроях через функцію "Spotify Connect" або аналогічну.

Недоліки:

– Безкоштовна версія має обмеження: присутня реклама, немає можливості обирати конкретну пісню в деяких плейлистах (особливо на мобільних пристроях);

– Тексти пісень доступні не для всіх треків (хоча ця функція постійно розширюється);

– Ліміт на збереження музики офлайн — до 10 000 треків на один пристрій.

1.4.3 Аналіз YouTube Music

YouTube Music — це музичний стрімінговий сервіс, розроблений компанією Google, який з 2018 року поступово замінює Google Play Music. Він інтегрується з основною платформою YouTube, що дозволяє користувачам не лише слухати офіційні аудіотреки, але й відкривати для себе неофіційні завантаження, живі виступи, ремікси, кавери та інший контент, якого часто немає на інших платформах.

YouTube Music вирізняється гнучким підходом до споживання контенту: користувачі можуть переглядати відеокліпи або слухати лише аудіо, переключаючи режими відтворення. Завдяки використанню даних з основної платформи YouTube, рекомендації сервісу враховують не лише музичні смаки, а й загальні перегляди, що дозволяє краще персоналізувати досвід користувача.

Станом на 2025 рік, YouTube Music разом із YouTube Premium має понад 125 мільйонів підписників, що на 25 мільйонів більше порівняно з 2024 роком. Така динаміка зростання свідчить про стабільне розширення бази користувачів і зростаючу популярність сервісу серед молоді та фанатів відеоконтенту [3].

Інтерфейс YouTube Music зображено на рисунку 1.3.

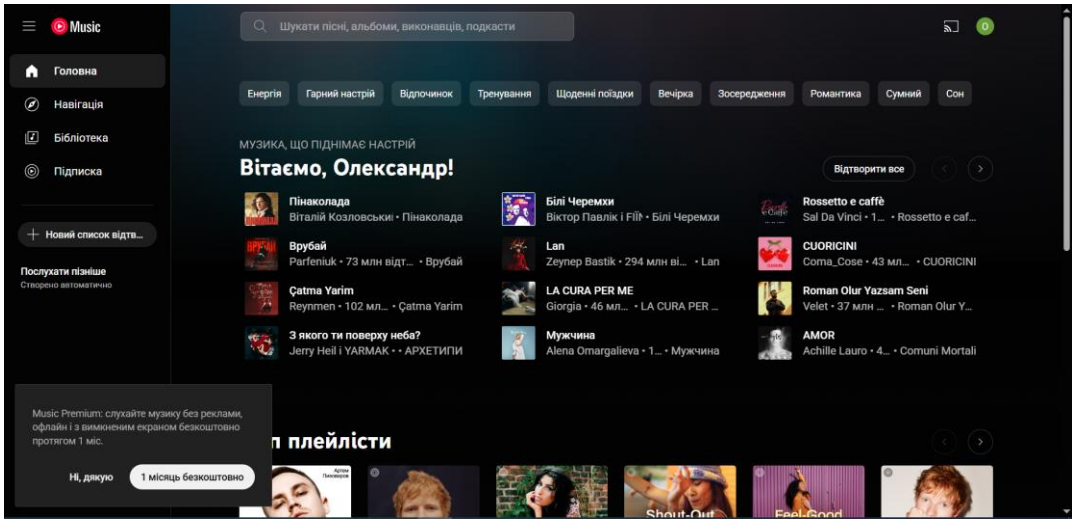


Рисунок 1.3 – Інтерфейс YouTube Music

Переваги:

– Широкий вибір контенту: офіційні треки, кліпи, ремікси, концерти, кавери, фанатські версії;

– Інтеграція з основною платформою YouTube;

– Персоналізовані рекомендації на основі історії переглядів та вподобань;

– Можливість перемикання між відео- та аудіоформатом;

– Офлайн-збереження музики для підписників YouTube Premium;

– Простий та інтуїтивний інтерфейс, зручна навігація;

– Висока якість звуку (до 256 кбіт/с у Premium-версії);

– Доступ до плейлистів на основі активності (місце, час доби, настрій тощо).

Недоліки:

– Без підписки неможливо слухати музику у фоновому режимі чи офлайн;

– Нижчий бітрейт у безкоштовній версії;

– Рекомендації можуть бути менш точними, якщо основний акаунт YouTube використовується не лише для музики;

– Обмежена кількість функцій у безкоштовній версії (наявність реклами, неможливість перемотки деяких треків);

– Порівняно менша кількість ексклюзивного музичного контенту в аудіоформаті, ніж у Spotify чи Apple Music.

1.4.4 Порівняльна таблиця сервісів для прослуховування музики

Для проведення порівняльного аналізу основні характеристики найпопулярніших музичних платформ були зведені в узагальнену таблицю 1.1.

Таблиця 1.1

Порівняльна таблиця сервісів для прослуховування музики

Функціонал	Spotify	YouTube Music
Завантаження власної музики	Немає можливості завантаження власних файлів у веб-версії, але є в десктоп-додатку (через локальну бібліотеку).	Можна завантажувати власні аудіофайли до хмарного сховища для прослуховування в будь-якому місці.
Офлайн прослуховування	Доступне лише з платною підпискою Premium.	Офлайн-режим доступний при оформленні підписки YouTube Music Premium.
Безкоштовна версія	Є, з рекламою, обмеженнями на вибір пісень і якість звуку.	Є, з рекламою, без офлайн-доступу та з обмеженнями функціоналу.
Підтримка текстів пісень	Тексти доступні не для всіх композицій, але поступово розширюється підтримка.	Тексти пісень відображаються у більшості треків, часто інтегровані з відео.
Створення та управління плейлистами	Розвинений функціонал, можна створювати власні, ділитися, слідкувати за іншими плейлистами.	Також повноцінне формування плейлистів, зручно інтегрується з відеоконтентом.
Алгоритми рекомендацій	Дуже точні, побудовані на глибокому аналізі уподобань користувача.	Рекомендації базуються як на музичних смаках, так і на переглядах відео.
Бітрейт звуку	До 320 кбіт/с (Ogg Vorbis) для Premium, нижча якість у безкоштовній версії.	До 256 кбіт/с (AAC) для Premium, у безкоштовній версії нижча якість.
Інтеграція з іншими сервісами	Підтримує Spotify Connect для синхронізації між пристроями.	Тісно інтегрований з YouTube, що дає доступ до відео та аудіо в одному додатку.
Підтримка пристроїв	Доступний на більшості платформ: мобільні, десктоп, смарт-тв, ігрові консолі.	Широка підтримка, особливо серед Android-пристроїв і пристроїв із доступом до YouTube.

1.4.5 Результати аналізу існуючих рішень

У ході аналізу популярних музичних сервісів було виявлено, що більшість із них — це масштабні платформи, створені великими міжнародними компаніями, які пропонують стрімінг музики з хмарних сховищ. Серед українських або більш "кастомних" веб-застосунків із подібним функціоналом практично відсутні приклади, які б дозволяли користувачам взаємодіяти зі своїми власними аудіофайлами без необхідності завантажувати їх у хмару.

Особливістю розробленого застосунку Vibenest є можливість завантаження локальних пісень (наприклад, із SD-картки або жорсткого диска). Уся взаємодія відбувається безпосередньо в браузері за допомогою drag-and-drop або форми завантаження, після чого треки відтворюються у кастомному аудіоплеєрі.

Таким чином, Vibenest виступає як веб-інтерфейс для прослуховування власної музики онлайн, без посередництва великих платформ — що і відрізняє його від існуючих аналогів.

1.5 Висновок по розділу

У цьому розділі було розглянуто, як працюють популярні музичні платформи, на що звертають увагу користувачі та чому деякі сервіси стають більш успішними за інші. Це дало змогу краще зрозуміти, які функції справді важливі, а які — другорядні. Особливу увагу було приділено інтерфейсу: він має бути не просто красивим, а зручним, швидким і зрозумілим.

Крім того, було з’ясовано, що сучасні сервіси значною мірою залежать від правильної взаємодії фронтенду з API, що дозволяє забезпечити стабільну роботу та швидку реакцію на дії користувача. Це стало орієнтиром для подальшої розробки власного інтерфейсу для платформи Vibenest.

Таким чином, аналіз існуючих рішень дав змогу не тільки надихнутись ідеями, але й чітко визначити, чого саме потрібно уникати й на чому зосередитись під час реалізації власного проєкту.

РОЗДІЛ 2. АНАЛІЗ ВИМОГ ТА ПОСТАНОВКА ЗАДАЧІ

2.1 Збір та аналіз вимог користувачів

Успішна розробка будь-якої інформаційної системи починається зі збирання та аналізу вимог користувачів. Це фундаментальний етап, який дозволяє чітко зрозуміти очікування, потреби та поведінкові сценарії потенційних користувачів. У випадку створення фронтенд-інтерфейсу для музичної платформи Vibenest, цей етап набуває особливого значення, оскільки мова йде про продукт, який повинен бути не лише функціональним, а й естетично привабливим, інтуїтивно зрозумілим та адаптивним до різних типів пристроїв.

Процес збору вимог передбачав використання кількох взаємодоповнюючих методів. Перш за все, було проведено інтерв'ю з потенційними користувачами, які планують завантажувати свою музику. Основними питаннями були: які функції користувачі очікують бачити, як вони уявляють ідеальний музичний сервіс, які елементи інтерфейсу вважають зручними або, навпаки, недоречними.

Окрім безпосереднього контакту з користувачами, проводився аналіз існуючих рішень. Зокрема, були розглянуті популярні музичні сервіси, їх інтерфейси, навігаційні патерни, шляхи користувачів, використання рекомендаційних алгоритмів, фільтрації та пошуку треків. Цей аналіз дав змогу виявити як загальноприйняті практики, так і унікальні дизайнерські рішення, які варто врахувати або уникати.

Особливу увагу приділено побудові сценаріїв використання (user stories), що охоплюють типові дії: реєстрація, пошук музики, прослуховування, створення плейлистів, завантаження композицій. Кожен сценарій дозволяє уявити, як саме користувач взаємодіє з системою, які дії він виконує і які очікує результати.

Таким чином, зібрані дані лягли в основу подальшої формалізації вимог до інтерфейсу, забезпечуючи орієнтацію на реальні очікування аудиторії.

2.2 Формулювання функціональних та нефункціональних вимог

На основі попередньо зібраних і проаналізованих потреб користувачів, а також з урахуванням особливостей предметної області, було здійснено формулювання вимог до функціональності та якості майбутньої музичної платформи Vibenest. Чітке розділення на функціональні та нефункціональні вимоги дозволяє краще структурувати процес проєктування інтерфейсу та визначити пріоритети під час розробки.

2.2.1 Функціональні вимоги

2.2.1.1 Реєстрація та авторизація користувачів

Однією з ключових функцій інтерфейсу музичної платформи є можливість швидкої реєстрації нового користувача за допомогою унікального нікнейму. Такий підхід дозволяє уникнути традиційної авторизації через електронну пошту або інші сервіси, спрощуючи процес початку користування сервісом. Це рішення знижує поріг входу для нових відвідувачів і особливо актуальне для цільової аудиторії, яка цінує швидкість та мінімальну кількість кроків при реєстрації.

На рівні інтерфейсу це реалізовано через окремий екран, де користувач вводить бажаний нікнейм. Коли користувач натискає кнопку підтвердження, запит надсилається до серверної частини, яка вже перевіряє унікальність введеного імені. Якщо такий нікнейм вже існує в системі, відображається повідомлення про помилку — користувачеві пропонується спробувати інший варіант. Таким чином, усі перевірки виконуються на стороні сервера, що значно спрощує реалізацію на фронтенді.

Після успішної реєстрації користувача автоматично перенаправляє на головну сторінку, де він одразу може розпочати взаємодію з платформою. Це

забезпечує плавний перехід без необхідності додаткових підтверджень або повторного входу. Такий сценарій дозволяє не лише заощадити час користувача, але й створює враження легкості й доступності ресурсу.

Зовнішній вигляд сторінки реєстрації витримано в загальному стилі платформи — з акцентом на мінімалізм, чистий дизайн та зрозумілу типографіку. Усі елементи на сторінці логічно впорядковані, а навігація зводиться до мінімуму: поле введення нікнейму та кнопка підтвердження. Це відповідає принципу “nothing extra” — тільки найнеобхідніше для досягнення мети.

Ще однією особливістю є адаптивність: інтерфейс рівнозначно добре виглядає на десктопах, планшетах і мобільних пристроях. Завдяки цьому користувачі можуть зареєструватися з будь-якого пристрою, що позитивно впливає на зручність і збільшує загальну доступність системи.

У межах цієї функціональності не передбачено складних сценаріїв підтвердження або багаторівневої перевірки особистості. Така простота є частиною загальної філософії проєкту — дати можливість будь-кому швидко створити профіль і почати взаємодію з платформою без зайвих дій. У цьому контексті нікнейм виконує роль ключового ідентифікатора користувача в системі.

Таким чином, функція реєстрації за нікнеймом є не лише необхідною для доступу до системи, а й важливою складовою користувацького досвіду. Простота, швидкість і чіткий сценарій взаємодії формують позитивне перше враження від роботи з платформою, що напряму впливає на лояльність нових користувачів.

2.2.1.2 Перегляд списку пісень та їх прослуховування

Одна з основних функцій музичної платформи Vibenest — це можливість користувачів переглядати список доступних пісень та слухати їх у режимі онлайн. Це ядро всієї системи, оскільки саме навколо музичного контенту вибудовується інтерфейс, логіка взаємодії та емоційний досвід користувача. У межах цієї вимоги платформа повинна надавати доступ до динамічного переліку

пісень, відображених у зрозумілому та привабливому вигляді.

Інтерфейс реалізований у вигляді стрічки або сітки пісень, де кожен елемент містить базову інформацію: назву треку, нік автора, а також кнопку для відтворення. Це дозволяє швидко орієнтуватися в наявному контенті та взаємодіяти з ним без зайвих переходів. Кожен запис має свою унікальну візуальну стилізацію, що допомагає формувати індивідуальність треків і підтримує загальний настрій платформи.

З технічної точки зору, дані про пісні завантажуються з бекенду через REST API. Після отримання відповіді фронтенд формує відповідне візуальне представлення. Плеєр для прослуховування вбудований у кожен елемент треку і дозволяє програвати звук без необхідності відкривати окрему сторінку. Це робить взаємодію максимально швидкою та зручною, особливо на мобільних пристроях.

У процесі прослуховування користувач має змогу ставити трек на паузу, перемотувати його та бачити загальний прогрес. Крім того, відтворення одного треку автоматично зупиняє інші, що запобігає накладанню звуку. Це важлива деталь, яка сприяє зручності й позитивному сприйняттю системи, особливо для нових користувачів, які тільки ознайомлюються з функціональністю.

Ще одним важливим аспектом цієї функції є адаптивність. Перегляд та прослуховування доступні як на десктопах, так і на мобільних пристроях. Всі елементи динамічно перебудовуються залежно від ширини екрана, зберігаючи зручність і читабельність. Це забезпечує стабільний користувацький досвід незалежно від типу пристрою.

Список треків не є статичним: він може оновлюватися в режимі реального часу після додавання нових пісень іншими користувачами. Це формує відчуття живого середовища, де постійно з’являється новий контент. Таким чином, платформа стає не просто засобом прослуховування, а справжньою спільнотою, в якій відображаються музичні смаки її учасників.

Також важливо підкреслити, що прослуховування відбувається

безпосередньо з сервера, що дозволяє уникати необхідності завантаження треків на пристрій користувача. Це не тільки полегшує доступ, але й захищає авторські права, адже користувачі не мають прямого доступу до файлів пісень.

Загалом, ця функція поєднує в собі ключові принципи інтерфейсного дизайну — простоту, швидкість, інтерактивність — і забезпечує базову цінність платформи: можливість слухати улюблену музику в зручному форматі. Її реалізація є фундаментом для подальших покращень, таких як фільтрація за жанрами, рекомендації або плейлисти, які можуть бути додані на наступних етапах розвитку проєкту.

2.2.1.3 Додавання музичних треків користувачами

Можливість додавати власні музичні композиції — це один із ключових функціональних елементів платформи Vibenest, який формує саму суть сервісу як спільноти творчих людей. Дана функція дозволяє користувачам безпосередньо наповнювати платформу новим контентом, що робить її живою, динамічною і відкритою для музичних експериментів.

На інтерфейсному рівні реалізовано окремий компонент з підтримкою drag-and-drop механізму для зручного завантаження аудіофайлів. Користувач може перетягнути файл треку у визначену зону або вибрати його вручну зі свого пристрою. На цьому етапі відсутня необхідність у верифікації через email, що спрощує доступ та знижує бар’єр для нових користувачів.

У рамках функціональності реалізовано базову перевірку структури даних на фронтенді: перевіряється, чи додано файл і чи заповнені обов’язкові поля (назва треку та виконавець). Проте складної валідації або перевірки формату аудіо на даному етапі не впроваджено. Це дозволяє максимально спростити перший реліз функції, зберігаючи при цьому цілісність введених даних.

Після натискання кнопки "Додати" трек відправляється на сервер через відповідний API-запит. У відповіді бекенд надсилає підтвердження про успішне збереження файлу, після чого інтерфейс автоматично оновлює список пісень,

додаючи новий трек у стрічку. Таким чином, користувач одразу бачить результат своєї дії, що створює ефект миттєвого зворотного зв’язку.

Дизайн інтерфейсу створений таким чином, щоб процес додавання був інтуїтивним та зрозумілим. Усі елементи логічно згруповані, використовується помірна кількість підказок і візуальних індикаторів. Завдяки цьому навіть користувач, який вперше взаємодіє з платформою, може самостійно додати трек без потреби в сторонніх інструкціях.

З технічної сторони важливим є те, що трек надсилається на сервер у вигляді multipart/form-data, що дозволяє коректно передавати як сам файл, так і метадані. У відповідь від сервера можна обробити різні типи помилок: наприклад, якщо файл не зчитано або не збережено.

У перспективі дана функція може бути розширена за рахунок додавання попереднього прослуховування треку до його публікації, можливості вказувати жанр, опис. Проте навіть у базовому вигляді вона виконує свою головну роль — надає користувачам простий і зручний інструмент для самовираження та створення контенту на платформі.

2.2.1.4 Пошук, вибране та формування плейлистів

Однією з важливих функціональних можливостей фронтенд-інтерфейсу платформи Vibenest є забезпечення користувачеві зручного способу взаємодії з музичним контентом — зокрема, можливість шукати треки, додавати їх до улюбленого та створювати власні плейлисти. Усі ці функції об’єднані в єдину логіку персоналізації, яка формує індивідуальний музичний простір для кожного користувача.

Пошук композицій реалізовано через зручний текстовий інтерфейс, який реагує на введення в реальному часі. Це дозволяє швидко знаходити необхідні треки за назвою або іменем виконавця, навіть якщо користувач пам’ятає лише частину назви. Пошукове поле розміщене у верхній частині сторінки, що робить його постійно доступним незалежно від поточного розділу платформи. Завдяки інтерактивній реалізації результатів, користувач бачить знайдені варіанти одразу

в процесі введення запиту, що суттєво підвищує зручність.

Після знаходження потрібного треку користувач може швидко додати його до улюблених. Для цього використовується знайома і зрозуміла візуально іконка — сердечко, яка розташована поряд із назвою композиції. Якщо користувач натискає на нього, іконка змінює колір, візуально підтверджуючи дію. Обрані улюблені композиції зберігаються у спеціальному розділі, доступному з головного меню. Таким чином користувач може в будь-який момент повернутись до улюблених треків, не витрачаючи час на повторний пошук.

Окрім індивідуального збереження композицій, користувач має змогу створювати власні музичні добірки у вигляді плейлистів. Це дозволяє структуровано організовувати улюблену музику за певними критеріями — наприклад, за настроєм, жанром, подією або навіть частиною доби. Інтерфейс створення плейлистів передбачає мінімальні зусилля: користувач вводить назву, після чого може одразу додавати до нього треки з усього доступного каталогу, включно з улюбленими. Додавання до плейлистів також виконується через просту взаємодію — обрання відповідного списку зі спливаючого меню або підтвердження натисканням на відповідну кнопку.

Важливою особливістю реалізації є логічна взаємозв’язка між пошуком, улюбленими треками та плейлистами. Всі дії користувача взаємодіють між собою в межах однієї системи: наприклад, трек, знайдений через пошук, може одразу бути доданий до улюбленого або до плейлиста без потреби переходити на інші сторінки. Це дозволяє суттєво зменшити кількість кроків для досягнення цілі та зробити взаємодію більш органічною.

З технічної точки зору, усі ці дії реалізовані без необхідності перезавантаження сторінки. Інтерфейс реагує на кожну дію користувача миттєво — як-от натискання на сердечко або підтвердження створення плейлиста. Таке реагування забезпечує комфортне користування сервісом навіть на пристроях із обмеженими ресурсами.

2.2.2 Нефункціональні вимоги

Нефункціональні вимоги визначають не стільки що робить система, як якою вона є під час використання. У контексті фронтенд-розробки музичної платформи Vibenest, це стосується якості, швидкості, зручності та адаптованості інтерфейсу. Саме ці характеристики безпосередньо впливають на досвід користувача та сприйняття платформи в цілому. Розглянемо ключові нефункціональні вимоги, релевантні до клієнтської частини проєкту.

2.2.2.1 Адаптивність інтерфейсу

Одним із основних очікувань сучасних користувачів є можливість повноцінного використання платформи на будь-якому пристрої — чи то смартфон, планшет, чи комп'ютер із великим екраном. Тому адаптивність інтерфейсу є критично важливою нефункціональною вимогою. У випадку з Vibenest реалізація адаптивності здійснюється за допомогою Tailwind CSS — сучасного утилітного CSS-фреймворка. Елементи інтерфейсу автоматично змінюють свій розмір, розташування та логіку поведінки в залежності від ширини вікна браузера.

Наприклад, у мобільній версії панель керування треками має зменшену висоту, а меню ховається за «бургер»-іконкою. Це дозволяє уникнути захаращення екрана і зосередити увагу користувача на основному функціоналі — прослуховуванні музики. У десктопній версії доступно більше контенту одразу, а додаткові елементи, як-от список пісень чи панель фільтрів, залишаються постійно видимими. Таке рішення створює логічну і зручну навігацію, не змінюючи користувацьку логіку на різних пристроях.

2.2.2.2 Продуктивність

Ще однією ключовою вимогою до фронтенду є продуктивність, тобто швидкість завантаження сторінок, швидке реагування на дії користувача та плавність анімацій. Інтерфейс Vibenest реалізований таким чином, щоб забезпечити миттєвий відгук на кліки, зміни треків, додавання пісень або навігацію між розділами. Це досягається шляхом оптимізації компонентів, lazy

loading-у, використання кешування даних і зменшення обсягу HTTP-запитів.

Також важливо, що на фронтенді застосовуються мінімалістичні візуальні елементи з плавними переходами, які не лише приємні для ока, але й не обтяжують ресурси пристрою. Усе це створює ефект «живої» взаємодії, без затримок чи зависань, що особливо цінно для мобільних користувачів із повільним інтернет-з'єднанням.

2.2.2.3 Ергономічність і зрозумілість

Зручність використання — ще один визначальний критерій якості фронтенд-інтерфейсу. Платформа має бути інтуїтивно зрозумілою навіть для тих, хто вперше нею користується. Це досягається завдяки чіткій структурі елементів, логічному розташуванню кнопок, контрастному оформленню активних зон та текстів. У Vibenest застосовано принципи UX-дизайну, які допомагають уникнути ситуацій, коли користувач не розуміє, що від нього очікується.

При додаванні пісні через drag-and-drop користувач одразу бачить анімацію додавання, що дає чіткий візуальний фідбек. Такий підхід допомагає зменшити кількість помилок при роботі з інтерфейсом і підвищує загальне задоволення від користування.

2.2.2.4 Сумісність із браузерами

Фронтенд має працювати однаково стабільно в усіх сучасних браузерах. Це важлива вимога, яка запобігає втраті частини аудиторії через технічні проблеми на стороні користувача. Інтерфейс Vibenest протестовано в найпопулярніших браузерах, таких як Google Chrome, Mozilla Firefox, Microsoft Edge та Safari.

Для цього використовуються кросбраузерні стилі та JavaScript-функції, які гарантовано підтримуються більшістю браузерних рушіїв. Водночас відмовлено від використання нестабільних або експериментальних API. Це дозволяє досягти уніфікованого досвіду незалежно від пристрою чи ОС, що позитивно впливає на рівень довіри до платформи.

2.2.2.5 Масштабованість

Платформа розробляється з урахуванням можливого зростання — як по функціоналу, так і по числу користувачів. Тому фронтенд має бути масштабованим. У реалізації Vibenest це означає, що кожен компонент інтерфейсу спроєктований як незалежний модуль із чітко визначеними властивостями та поведінкою.

Таке розділення дозволяє легко додавати нові можливості, не змінюючи вже існуючий код. Наприклад, у майбутньому можна реалізувати окремий розділ для рекомендацій або створення плейлистів, не порушуючи логіку основної сторінки. Масштабованість також передбачає структуровану архітектуру фронтенд-проєкту, що полегшує його підтримку і розвиток у довготривалій перспективі.

2.3 Постановка задачі проєктування системи

На основі зібраних і проаналізованих вимог до користувацького інтерфейсу музичної платформи Vibenest постає необхідність чітко сформулювати задачі, які мають бути вирішені в процесі її проєктування. Основна мета — створити сучасний, інтуїтивно зрозумілий, швидкодіючий та адаптивний фронтенд, що забезпечуватиме комфортну взаємодію користувача з системою, незалежно від типу пристрою чи умов використання.

Проєктування інтерфейсу повинно охоплювати весь набір функціональних можливостей, визначених у попередньому підрозділі. Зокрема, це авторизація через введення унікального ніку, додавання нових треків, перегляд списку доступної музики, можливість додавати улюблені композиції до окремого списку, пошук за назвою або автором треку, створення та перегляд персональних плейлистів, а також базова навігація по додатку. Усі ці функції мають бути представлені у зрозумілому вигляді з урахуванням принципів UI/UX-дизайну.

Особливу увагу варто приділити адаптивності. Оскільки частина користувачів взаємодіє з платформою через мобільні пристрої, інтерфейс повинен гнучко перебудовуватись залежно від розміру екрану, зберігаючи при цьому повну функціональність. Це означає, що компоненти, елементи керування та меню повинні бути оптимізовані як для десктопів, так і для смартфонів.

Ще одним важливим аспектом є забезпечення високої продуктивності. Платформа має швидко реагувати на дії користувача — наприклад, миттєво перемикати треки, без затримок відкривати сторінки та не перевантажувати браузер. Це вимагає продуманого структурування компонування даних на клієнті, обмеження непотрібних рендерів та ефективної взаємодії з API.

Також варто передбачити гнучкість для подальшого розвитку платформи. З цією метою потрібно обрати таку архітектуру фронтенду, яка дозволить безболісно додавати нові компоненти, змінювати існуючі або масштабувати проєкт за потреби. Наприклад, використання компонентного підходу в React дозволить ефективно керувати складністю інтерфейсу та забезпечити його підтримуваність у довгостроковій перспективі.

Окремо слід врахувати потребу у відповідності до нефункціональних вимог, зокрема: забезпечення кросбраузерної сумісності, ергономічності, візуальної привабливості та логічної побудови елементів. Інтерфейс повинен бути комфортним для користувачів з різним рівнем цифрової грамотності.

Таким чином, задача проєктування інтерфейсу для Vibenest охоплює цілу низку взаємопов’язаних технічних та дизайнерських рішень. Важливо не лише реалізувати заплановану функціональність, а й зробити це з урахуванням сучасних вимог до швидкості, адаптивності, доступності та естетичної цілісності інтерфейсу.

2.4 Висновок по розділу

У другому розділі було здійснено комплексний аналіз вимог до

користувацького інтерфейсу платформи Vibenest. Ретельно розглянуто способи збору інформації від майбутніх користувачів та сформовано перелік функціональних і нефункціональних вимог, що відповідають очікуванням цільової аудиторії. Основна увага приділялася тим можливостям, які реалізуються безпосередньо на фронтенді, з урахуванням сучасних стандартів у сфері веброзробки.

Функціональні вимоги визначили набір ключових дій, які користувач може виконувати в межах інтерфейсу — від входу до системи до взаємодії з музичним контентом. Зокрема, користувач має змогу додавати улюблені треки, створювати власні плейлисти, шукати музику за назвою або виконавцем, а також завантажувати нові композиції через зручну форму додавання. Ці функції забезпечують гнучкість і персоналізацію взаємодії з платформою, що є важливою складовою сучасного музичного сервісу.

Окремо були проаналізовані нефункціональні характеристики, які впливають на якість взаємодії: швидкодія, адаптивність до різних розмірів екранів, сумісність з поширеними браузерами, ергономіка, зрозуміла навігація та масштабованість інтерфейсу з можливістю подальшого розширення функціоналу.

На підставі зібраної інформації сформульовано чітке завдання для етапу проєктування. Воно охоплює побудову логічної, зручної та гнучкої інтерфейсної структури, що дозволяє задовольнити очікування користувачів і водночас залишає простір для майбутнього розвитку проєкту.

Проведений аналіз створює необхідне підґрунтя для подальшої розробки інтерфейсу, забезпечуючи її послідовність, обґрунтованість та технічну цілісність. Отже, другий розділ закладає основу для переходу до безпосереднього конструювання компонентів користувацького інтерфейсу, які відповідатимуть поставленим вимогам і забезпечуватимуть високий рівень взаємодії з платформою.

РОЗДІЛ 3. ПРОЄКТУВАННЯ СИСТЕМИ

3.1 Розробка архітектури програмного забезпечення

Архітектура програмного забезпечення є одним із визначальних аспектів проєктування вебплатформи, оскільки саме вона задає структуру, правила взаємодії між компонентами та технічну основу для подальшого розвитку системи. У випадку музичної платформи Vibenest, архітектура має враховувати специфіку роботи з динамічними аудіофайлами, взаємодію користувачів із контентом, авторизацію, управління станом додатку та вимоги до високої продуктивності. Оскільки основний фокус даної роботи спрямований на фронтенд, архітектурні рішення були прийняті з урахуванням потреб сучасного інтерфейсу користувача.

Для клієнтської частини було обрано архітектурний підхід на основі концепції SPA (Single Page Application), що дозволяє реалізувати єдиний динамічний інтерфейс, який оновлюється без повного перезавантаження сторінки. Це підвищує швидкодію, зменшує навантаження на сервер та забезпечує приємний досвід користування. Такий підхід також дає можливість реалізувати більш складну логіку відображення елементів, змін стану та обробки подій без зайвої складності в навігації. З технічної точки зору, SPA значно спрощує реалізацію взаємодії з API-сервісами, оскільки усі запити до сервера виконуються асинхронно, без оновлення сторінки.

Для реалізації SPA було використано бібліотеку React — одну з найпопулярніших і найефективніших технологій для створення інтерфейсів. React дозволяє розробляти застосунок на основі компонентної моделі, де кожен елемент (наприклад, картка пісні, форма додавання треку, плеєр чи панель навігації) існує як незалежна частина інтерфейсу. Це не тільки покращує повторне використання коду, але й полегшує тестування та модифікацію окремих частин застосунку без необхідності втручання у всю систему.

Компоненти можна ізолювати, перевикористовувати та навіть передавати між проєктами, що покращує масштабованість.

Для ефективного управління даними у межах додатку було вирішено використовувати вбудовану систему керування станом React — Context API, а також хуки useState та useEffect для керування локальним станом і життєвим циклом компонентів. Context API дозволяє уникнути так званого «prop drilling» — передачі даних через декілька рівнів вкладеності компонентів. Це особливо важливо, коли йдеться про глобальні об’єкти, такі як облікові дані користувача, поточний список треків, інформацію про плейлісти або улюблені композиції. Такий підхід забезпечує централізовану логіку роботи з даними, що підвищує контрольованість системи.

Для взаємодії з бекенд-сервером, де зберігаються всі основні дані, застосовується REST API. Кожен запит (отримання списку треків, додавання нового треку, оновлення списку улюблених) ініціюється клієнтом і відправляється через HTTP-протокол. Для виконання запитів було обрано бібліотеку Axios, яка надає зручний інтерфейс для створення запитів, обробки помилок, підключення токенів авторизації, встановлення заголовків і автоматичної обробки відповідей. Axios є оптимальним рішенням для фронтенд-застосунків завдяки своїй гнучкості та активній підтримці.

Маршрутизація у Vibenest реалізована за допомогою React Router — популярної бібліотеки для побудови навігаційної структури у SPA-додатках. Вона дозволяє визначити логічну карту сайту, при якій кожна частина інтерфейсу має свою адресу (URL). Наприклад, домашня сторінка, розділ з треками, плейлісти або форма додавання музики мають власні шляхи, які не потребують повного перезавантаження. Це забезпечує не лише зручність навігації для користувача, а й закладає фундамент для майбутньої SEO-оптимізації через SSR (Server-Side Rendering) або SSG (Static Site Generation), якщо така потреба з’явиться.

Стилистична частина інтерфейсу реалізована на основі Tailwind CSS —

утилітарного фреймворку, який дозволяє швидко формувати стильові рішення без необхідності створювати окремі стилі для кожного компонента. Tailwind надає велику кількість готових класів для форматування відступів, розмірів, кольорів, шрифтів, а також підтримує темізацію та адаптивність. Це дозволяє пришвидшити розробку, уникнути дублювання стилів та зберігати консистентність інтерфейсу.

Архітектура застосунку також враховує питання безпеки. Зокрема, важливою частиною є робота з авторизованим доступом до API. Наприклад, певні дії — як-от додавання музики або збереження улюблених треків — доступні лише авторизованим користувачам. У цьому випадку використовуються токени авторизації, які зберігаються на боці клієнта (наприклад, у localStorage) та автоматично підставляються в заголовки запитів. Це дозволяє захистити API від несанкціонованих запитів і забезпечити персоналізовану взаємодію.

Ще одним важливим аспектом є структура проєкту. Вона побудована згідно з принципами модульності та розділення відповідальностей. Кожен розділ додатку має власну директорію, в якій зберігаються відповідні компоненти, стилі, сервіси для роботи з API, утиліти та типи. Такий підхід дозволяє легко орієнтуватися в коді, швидко знаходити потрібні частини та мінімізувати ризики помилок під час розробки або рефакторингу.

У результаті, архітектура фронтенд-застосунку Vibenest є гнучкою, модульною та сучасною. Вона відповідає всім базовим вимогам до продуктивного вебінтерфейсу, забезпечуючи швидкість, масштабованість та зручність використання. Завдяки обраному технологічному стеку система готова до подальшого розвитку та інтеграції з новими можливостями — як на рівні інтерфейсу, так і в логіці обробки користувацьких дій.

3.2 Моделювання бізнес-процесів та системних компонентів

Моделювання бізнес-процесів і компонентів є критично важливим етапом при проєктуванні будь-якого програмного забезпечення. Воно дозволяє візуалізувати логіку взаємодії користувача з системою, зрозуміти послідовність подій, визначити відповідальність кожного модуля та передбачити потенційні вузькі місця. Для музичної платформи Vibenest цей етап мав на меті чітко сформулювати, як саме відбуватиметься додавання, перегляд, прослуховування та управління треками на фронтенді, а також взаємодія з API й іншими компонентами.

У процесі розробки платформи було застосовано підхід до моделювання бізнес-процесів за допомогою UML-діаграм. Серед основних типів діаграм, що використовувалися, — діаграма прецедентів (Use Case Diagram), діаграма послідовності (Sequence Diagram) та діаграма компонентів (Component Diagram). Вони дають змогу побачити платформу з різних кутів зору — з точки зору користувача, логіки роботи інтерфейсу та технічної структури компонентів відповідно.

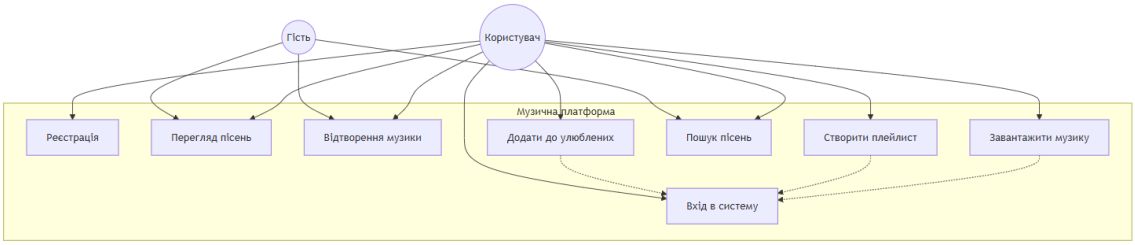


Рисунок 3.1 - Діаграма прецедентів

На рисунку 3.1 зображено основні дії, які може виконувати користувач. Серед них: реєстрація, авторизація, перегляд списку треків, прослуховування композицій, додавання до улюблених, створення плейлистів, пошук треків за ключовими словами та додавання власної музики. Кожна з цих дій реалізована

як окремий сценарій використання, який взаємодіє з відповідними компонентами інтерфейсу та API.

Важливим було також виокремити ролі — звичайний користувач та адміністратор (у перспективі). На поточному етапі система реалізує лише базову роль користувача, однак у майбутньому можлива реалізація додаткових рівнів доступу для модерування контенту, керування користувачами тощо.

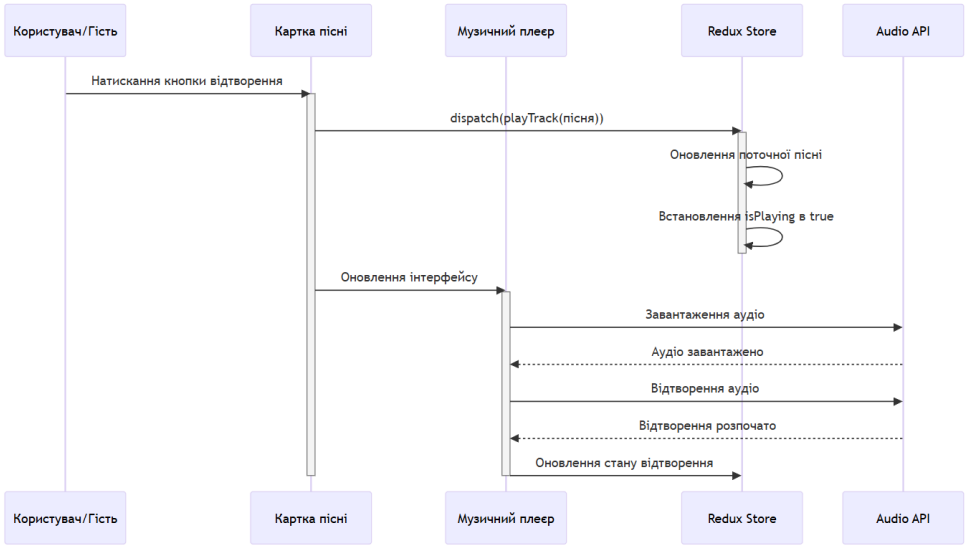


Рисунок 3.2 - Діаграма послідовності

На рисунку 3.2 зображено діаграму послідовності яка допомогла змоделювати порядок взаємодії між користувачем, фронтенд-інтерфейсом і сервером. Наприклад, сценарій додавання треку включає кілька етапів: користувач заповнює форму з інформацією про пісню, обирає аудіофайл, надсилає запит, система обробляє цей запит і повертає статус операції. Усі ці етапи реалізуються через відповідні методи API і відображаються у вигляді покрокової взаємодії.

Подібно описується сценарій відтворення треку: користувач натискає на кнопку «Play», фронтенд надсилає запит до сервера за URL аудіофайлу, отримує потік даних та відтворює його за допомогою HTML5 audio API. Така деталізація дає змогу розробникам виявити й потенційно оптимізувати ділянки, де може

виникати затримка або непередбачувана поведінка інтерфейсу.

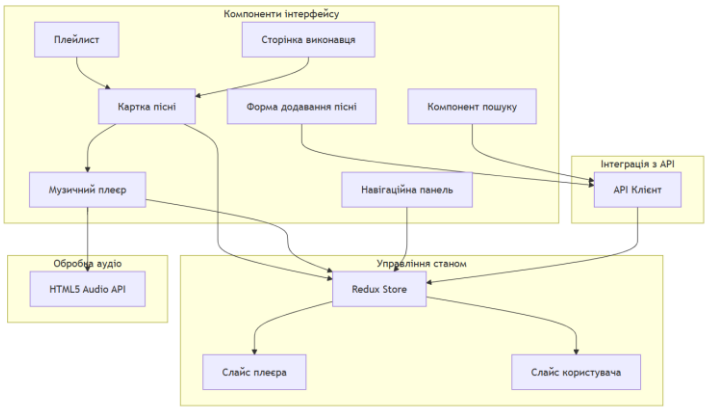


Рисунок 3.3 - Діаграма компонентів

На рисунку 3.3 зображена структура діаграми компонентів застосунку представлена у вигляді взаємозалежних блоків. Основні з них:

– Компонент автентифікації — відповідає за збереження інформації про користувача та перевірку його статусу при взаємодії з функціями, які потребують авторизації.

– Компонент списку треків — відображає отримані з API дані у вигляді плиток або списку з назвами, виконавцями та кнопками взаємодії.

– Компонент програвача — контролює відтворення, паузу, перемотку, відображення прогресу треку.

– Компонент пошуку — реалізує логіку фільтрації контенту за назвою, автором або тегами.

– Компонент плейлистів — дозволяє створювати та зберігати списки улюблених треків.

– Форма додавання треків — містить поля для завантаження аудіофайлу та введення супровідної інформації.

Кожен з цих компонентів функціонує незалежно, проте взаємодіє із загальним станом застосунку через глобальний контекст або передачу пропсів.

3.2.1 Бізнес-логіка платформи

На рівні бізнес-логіки фронтенд виступає посередником між діями

користувача та бізнес-методами, реалізованими на сервері. Наприклад, коли користувач додає трек до улюблених, відбувається перевірка автентифікації, надсилається запит з ID треку на сервер, після чого оновлюється інтерфейс. Всі ці процеси повинні відбуватися швидко, безперебійно та з повідомленнями про успіх чи помилку — це й становить ключову вимогу до UX частини застосунку.

Моделювання процесів також дозволяє підготувати платформу до масштабування: у разі зростання кількості користувачів або ускладнення функціоналу (наприклад, стрімінг у реальному часі, колаборативні плейлисти, інтеграція з іншими сервісами) загальна структура залишиться стабільною, завдяки чітко розмежованим ролям і функціям компонентів.

3.3 Вибір та обґрунтування технологій та інструментів розробки

3.3.1 React

React — це JavaScript-бібліотека для створення інтерфейсів користувача, яка стала основною технологією для розробки фронтенду проєкту Vibenest. Однією з ключових переваг React є його компонентна архітектура, яка дозволяє розбивати інтерфейс на логічно ізольовані частини, що значно полегшує підтримку та масштабування коду. Компоненти можна повторно використовувати, що знижує кількість дублювання та пришвидшує розробку.

React використовує концепцію Virtual DOM, яка дозволяє ефективно оновлювати лише ті частини сторінки, які змінилися. Це підвищує продуктивність програми та забезпечує плавний інтерфейс навіть при великій кількості динамічних елементів. Такий підхід зменшує навантаження на браузер, роблячи додаток швидким та чутливим до дій користувача.

Бібліотека має активну спільноту, багатий набір додаткових бібліотек та рішень, таких як React Router для маршрутизації, Redux або Zustand для керування станом. Це дозволяє швидко інтегрувати нові функції без потреби реалізовувати все з нуля. Високий рівень підтримки та регулярні оновлення

роблять React безпечним і перспективним вибором.

Інтерфейс Vibenest вимагав високої гнучкості в налаштуванні зовнішнього вигляду та поведінки компонентів. Завдяки JSX — синтаксису, що дозволяє писати HTML у JavaScript — розробка інтерфейсу стала швидшою та інтуїтивно зрозумілою. Це дозволило скоротити час розробки MVP та оперативно вносити зміни в дизайн.

React також зручно інтегрується з TypeScript, що дозволяє використовувати статичну типізацію, знижуючи ймовірність помилок під час розробки. Наявність типів покращує зручність IDE, автодоповнення та навігацію в коді. Це особливо важливо при роботі в команді, де чітка структура та передбачуваність коду зменшує кількість багів.

Досвід розробників у використанні React був додатковим аргументом на користь його вибору. Знання внутрішньої логіки бібліотеки та знайомство з її екосистемою дозволили швидко налаштувати середовище розробки та приступити до реалізації ключового функціоналу [7].

3.3.2 Angular

Angular — це повноцінний фреймворк від Google, який також популярний для розробки SPA-додатків. Основна відмінність Angular від React полягає в його комплексному підході: Angular включає в себе всі необхідні засоби для розробки, такі як маршрутизація, сервіс для запитів до API, ін’єкцію залежностей тощо. Це робить Angular більш жорстко структурованим.

З одного боку, такий підхід дозволяє уникнути фрагментації, оскільки всі елементи працюють за єдиною логікою та стандартами. Але з іншого боку, розробник повинен дотримуватись суворих правил і шаблонів, що може обмежувати гнучкість при реалізації нестандартних рішень. Для проєкту Vibenest, де важлива була кастомізація та експериментальний підхід до UI/UX, це стало мінусом.

Angular використовує власний шаблонізатор (HTML-подібну мову розмітки), що ускладнює інтеграцію з інструментами, орієнтованими на чистий

JSX чи JavaScript. Крім того, крива навчання Angular значно вища, особливо для початківців або невеликих команд. Це також вплинуло на вибір — команда була більш досвідчена в React, ніж в Angular.

Також Angular значно більший за обсягом, ніж React, і вимагає більше часу на компіляцію та завантаження. Для платформи, де важливий швидкий рендер і легка вага сторінки, це стало обмеженням. Angular може бути хорошим варіантом для великих корпоративних застосунків, але для Vibenest він був надмірним.

Попри це, варто визнати, що Angular має високий рівень безпеки, вбудовану підтримку модульності та добру документацію. Для команд, які шукають повний стек у межах одного фреймворка, Angular залишається надійним вибором. Але для нашої задачі його обмеження перевищили переваги [8].

3.3.3 Vite

Vite — сучасний інструмент для збірки, який забезпечує миттєвий запуск дев-сервера та гаряче оновлення сторінки при зміні коду. Це досягається завдяки використанню нативних ES-модулів та розділенню збірки на етапи девелопменту і продакшну. У результаті старт проєкту відбувається майже миттєво.

Головна перевага Vite — це його швидкість. У порівнянні з традиційними бандлерами, такими як Webpack, Vite дозволяє починати розробку без необхідності чекати на довгі процеси збірки. Це позитивно впливає на загальну продуктивність розробників і знижує час між написанням коду та його перевіркою в браузері.

Vite також має просту конфігурацію — більшість речей працює «з коробки». За потреби його легко розширити за допомогою плагінів, що робить його зручним для проєктів будь-якого масштабу. Важливо, що він чудово інтегрується з React і підтримує Tailwind CSS, ESLint та TypeScript.

Ще одним плюсом є зрозуміла структура проєкту: Vite не вимагає великої

кількості налаштувань і дозволяє зосередитись на розробці самого застосунку. Для команди Vibenest це означало можливість зосередитися на UX/UI без зайвих технічних перешкод.

Vite використовує Rollup на етапі продакшн-збірки, що забезпечує високу продуктивність, малі розміри файлів і кращу оптимізацію коду. Це критично для швидкого завантаження застосунку на мобільних пристроях і в умовах повільного інтернету [9].

3.3.4 Webpack

Webpack довгий час був головним інструментом для збірки JavaScript-додатків. Його основна перевага — гнучкість і повний контроль над конфігурацією. У Webpack можна задати практично будь-яку поведінку, що робить його універсальним інструментом для великих і нестандартних проєктів.

Проте гнучкість має свою ціну — конфігурація Webpack є складною і потребує часу на вивчення. У випадку Vibenest це стало проблемою, адже метою було швидке розгортання MVP. Витрати на налаштування Webpack не виправдалися б, враховуючи масштаби та цілі проєкту.

Webpack має більший час старту і повільніше оновлення при зміні файлів у порівнянні з Vite. Це знижує продуктивність команди, особливо на етапі інтенсивної розробки. Для великих застосунків Webpack може залишатись доцільним, але не в випадку невеликої музичної платформи.

Однак Webpack залишається дуже потужним інструментом — з великою кількістю плагінів, інтеграцій і підтримкою практично всіх сучасних підходів до розробки. Його використовують у багатьох великих компаніях, що робить його досі актуальним, попри появу більш легких альтернатив [10].

3.3.5 Tailwind CSS

Tailwind CSS — це утилітарний CSS-фреймворк, що дозволяє будувати унікальні інтерфейси без потреби в написанні окремих стилів. В основі лежить ідея повторного використання коротких класів безпосередньо в HTML/JSX, що дозволяє швидко змінювати вигляд елементів.

Головною перевагою Tailwind є його гнучкість — він не накладає жодних обмежень на дизайн, як це робить Bootstrap. Це дозволило створити для Vibenest унікальний візуальний стиль, який легко масштабувався на всі сторінки застосунку. Адаптивна верстка реалізована за допомогою зручних брейкпоінтів.

Важливою характеристикою є також автоматична оптимізація CSS — Tailwind видаляє всі невикористані стилі під час фінальної збірки, що значно зменшує розмір файлів. Завдяки цьому сайт завантажується швидше, що позитивно впливає на користувацький досвід.

Використання Tailwind дозволило дизайнерам і розробникам ефективно співпрацювати — стилі інтерфейсу були зрозумілими без потреби звертатися до зовнішніх CSS-файлів. Це прискорило процес розробки і знизило кількість помилок при верстці [11].

3.3.6 Bootstrap

Bootstrap — один з найстаріших та найпопулярніших CSS-фреймворків, який надає готові компоненти для побудови інтерфейсу. Він особливо корисний для швидкого прототипування, але виявився менш гнучким, ніж Tailwind.

Bootstrap базується на попередньо визначених шаблонах, що обмежує кастомізацію. Для Vibenest, де важливо було створити індивідуальний стиль, це стало проблемою. Щоб змінити вигляд компонентів, часто доводиться перевизначати стилі, що створює дублювання коду та ускладнює підтримку.

Хоча Bootstrap дозволяє створювати адаптивні сайти, його система брейкпоінтів менш зручна у порівнянні з Tailwind. Крім того, компоненти Bootstrap часто виглядають однаково, що може призвести до відчуття шаблонності інтерфейсу.

Перевагою Bootstrap залишається велика спільнота, доступність документації та підтримка багатьох платформ. Але для проєкту з унікальним UI, Tailwind став кращим вибором через свою модульність і свободу стилізації [12].

3.4 Висновок по розділу

У цьому розділі було детально розглянуто основні етапи проєктування фронтенд-частини музичної платформи Vibenest. Перш за все, було визначено архітектурний підхід до побудови програмного забезпечення, який ґрунтується на компонентній структурі, розділенні логіки на контейнери та презентаційні компоненти, використанні REST API для взаємодії з серверною частиною.

У межах підрозділу 3.2 було здійснено моделювання ключових бізнес-процесів та системних компонентів із використанням UML-діаграм. Це дозволило структуровано представити логіку взаємодії користувача з системою, ролі різних компонентів та їхню взаємозалежність. Розроблені моделі забезпечили чітке уявлення про внутрішню організацію платформи та стали основою для її подальшої реалізації.

У підрозділі 3.3 було детально проаналізовано сучасні інструменти та технології розробки, серед яких особливу увагу приділено бібліотеці React, а також іншим допоміжним засобам. Обрані технології були ретельно зважені з огляду на ефективність, швидкість розробки, підтримку масштабованості, зручність інтеграції та відповідність потребам платформи. Зроблений вибір дозволив закласти надійну технічну базу для реалізації всіх необхідних функціональних і нефункціональних вимог.

Таким чином, проєктування системи як ключовий етап розробки забезпечило стратегічне бачення архітектури майбутнього застосунку, визначило основні програмні компоненти та їхні ролі, а також дозволило зробити зважений вибір технологічного стеку.

РОЗДІЛ 4. РЕАЛІЗАЦІЯ ПРОЄКТУ

4.1 Розробка інтерфейсу вебдодатку Vibenest

Інтерфейс платформи Vibenest розроблявся з урахуванням сучасних підходів до UI/UX-дизайну та специфіки музичних вебзастосунків. Основним інструментом для створення макетів стала Figma, яка забезпечила зручне середовище для проєктування прототипів та узгодження дизайну з логікою роботи системи. Візуальна концепція була витримана в темних кольорах із яскравими акцентами, що дозволило досягти сучасного стилю з характерною музичною атмосферою. Усі екрани створювались відповідно до обраної структури додатку: від головної сторінки до плеєра, пошуку, додавання треків і плейлистів.

На рисунку 4.1 зображена головна сторінка, яка є першою точкою взаємодії користувача із сервісом. Її композиція вибудована таким чином, щоб одразу після входу користувач бачив актуальні треки, рекомендації та найпопулярніші добірки. Візуальні елементи тут мають чітку ієрархію: на передньому плані обкладинки треків, що супроводжуються назвою, ім’ям виконавця та іконкою "сердечка", яка дозволяє додавати трек до улюбленого. Анімація при наведенні на картку створює відчуття динамічності й взаємодії, а натискання миттєво активує глобальний плеєр, який залишається доступним незалежно від сторінки.

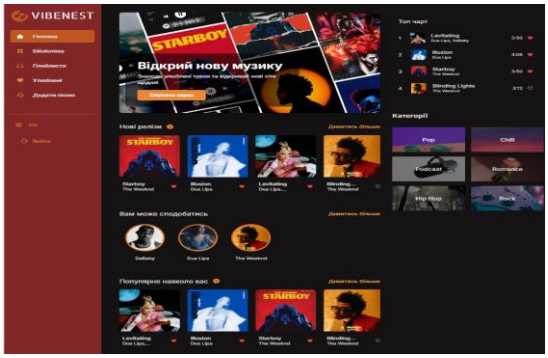


Рисунок 4.1 – Головна сторінка Vibenest

На рисунку 4.2 зображено глобальний плеєр який розташований у нижній частині екрана і залишається на місці при переході між розділами. У ньому відображається назва активного треку, обкладинка, ім’я виконавця, а також присутні основні елементи керування: відтворення, пауза, перемикання композицій, прогрес-бар і регулювання гучності. Цей компонент реалізовано на базі глобального стану Redux, що дозволяє синхронізувати його з іншими частинами застосунку без зайвих оновлень сторінки. Зміна треку або дії користувача одразу ж відображаються у плеєрі, створюючи безшовний досвід використання.

Рисунок 4.2 – Кастомний плеєр

Пошукова система, яка зображена на рисунку 4.3, стала окремим функціональним блоком, що дозволяє користувачеві швидко знайти потрібний трек або виконавця. Пошуковий інтерфейс побудовано з використанням debounce-затримки, що оптимізує запити до серверу й запобігає зайвому навантаженню. Результати пошуку відображаються у вигляді знайомих карток, з яких трек можна одразу запустити або додати до улюблених. Сторінка пошуку підтримує адаптивність і логічно інтегрована в загальну навігаційну систему застосунку.

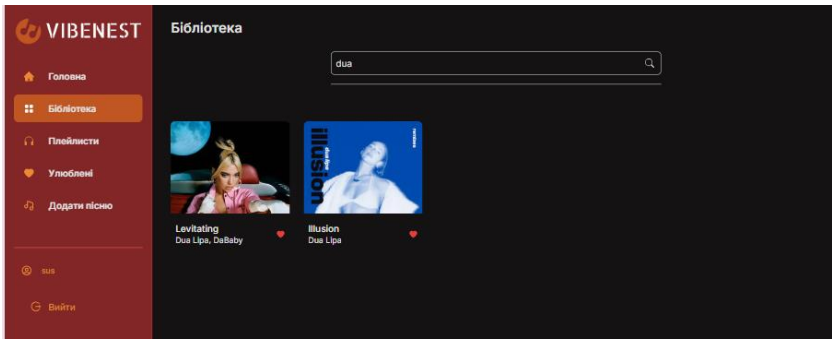


Рисунок 4.3 – Сторінка пошуку музики за назвою або виконавцем

Важливим розділом платформи є сторінка додавання нового треку, яка зображена на рисунку 4.4. Вона доступна лише авторизованим користувачам і реалізована у двох варіантах: через класичну форму з полями для назви, виконавця, тривалості, а також через зону drag-and-drop, яка дозволяє завантажити пісню простим перетягуванням файлу. Форма містить підказки, перевірку правильності введених даних та відображення статусу завантаження. У разі помилки користувач бачить відповідне повідомлення, а після успішного завантаження — підтвердження.

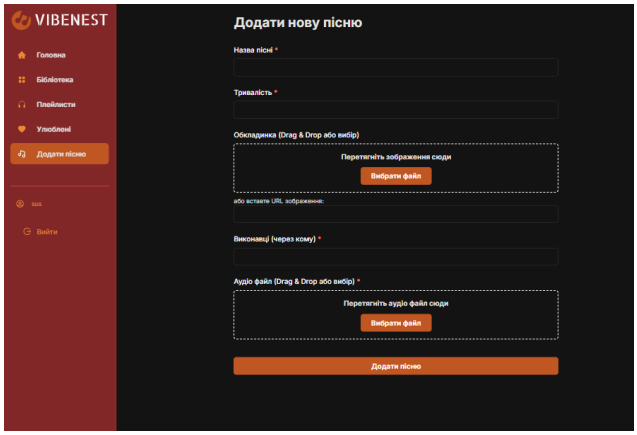


Рисунок 4.4 – Інтерфейс додавання нового треку через форму

Розділи плейлистів і улюбленого контенту створюють простір для персоналізації музичного досвіду. Користувачі можуть формувати власні добірки з доступних треків, переглядати їх у зручному форматі карток, відкривати окремі плейлисти (зображено на рисунку 4.5), а також додавати чи вилучати пісні з них. Всі дії синхронізуються з сервером і миттєво відображаються в інтерфейсі без перезавантаження сторінки. Візуальна частина добірок оформлена у стилі загального інтерфейсу, що забезпечує єдину стилістику й впізнаваність платформи.

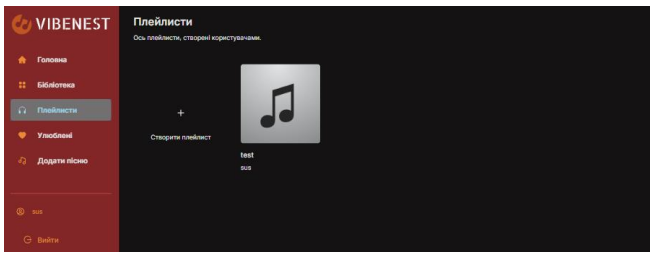


Рисунок 4.5 – Сторінка з власними плейлистами користувача

Інтерфейс також адаптований для різних типів пристроїв. На настільних комп’ютерах навігація реалізована у вигляді бічної панелі з текстовими підписами та іконками. На планшетах ця панель стискається до іконок, а на мобільних — трансформується у приховане меню або нижню панель. Такий підхід дозволив зберегти повну функціональність навіть на екранах малого розміру та зробити платформу однаково зручною як для користувачів десктопів, так і для смартфонів.

4.2 Опис розробки програмного коду

Компонент SongCard, який зображено на рисунку 4.6, відповідає за відображення окремого музичного треку в інтерфейсі користувача. Він приймає об'єкт song та audioRef як пропси, а також використовує хуки useDispatch та useSelector для взаємодії з глобальним станом через Redux. Зокрема, з Redux Store витягуються дані про поточний трек (currentTrack), статус відтворення (isPlaying), а також інформація про користувача й токен доступу. Це дозволяє компоненту не тільки показувати релевантну інформацію, але й реагувати на події — наприклад, відтворення треку при кліку на картку.

```
const SongCard = ({ song, audioRef }) => {
    const dispatch = useDispatch();
    const { currentTrack, isPlaying } = useSelector((state) => state.player);
    const { user, token } = useSelector((state) => state.user);

    const toast = useToast();

    const playSong = () => {
        window.dispatchEvent(new Event('user-interacted'));
        if (isCurrentTrack) {
            dispatch(setPlaying(!isPlaying));
        } else {
            dispatch(playTrack(song));
            dispatch(setTrackList({ list: [song] }));
        }
    };
```

Рисунок 4.6 - Фрагмент коду: структура компонента SongCard

Основна функція playSong активується при взаємодії користувача з компонентом. Спочатку викликається глобальна подія user-interacted, яка використовується для активації відтворення звуку в браузері. Далі відбувається оновлення глобального стану — через диспетчер Redux викликаються дії playTrack(song) та setTrackList({ list: [song] }), які відповідно задають поточну пісню та список для відтворення. Це гарантує, що при натисканні на конкретну пісню, плеєр відтворює саме її, а інші частини інтерфейсу автоматично синхронізуються.

Такий підхід дозволяє уникнути дублювання логіки у плеєрі та картках треків — вся логіка керування зосереджена в одному місці, і реакція на дії користувача завжди послідовна. Крім того, у компоненті можна легко перевіряти, чи поточний трек співпадає з відображеним, і на основі цього змінювати візуальні ефекти (наприклад, виділяти активний трек). Це забезпечує гнучкий і реактивний UI. Компонент є ізольованим, легко тестується й може бути повторно використаний у різних частинах додатку, наприклад у результатах пошуку або плейлистах.

Для реалізації функціоналу додавання треку до списку улюблених пісень реалізована функція handleLike (що зображено на рисунку 4.7), яка працює асинхронно. Спочатку за допомогою axios (у вигляді client) відправляється PATCH-запит на адресу /songs/like/${song?._id}. При цьому в заголовках запиту передається токен авторизації користувача. Це дозволяє серверу перевірити права доступу й оновити стан користувача у базі даних — зокрема, перелік треків у списку улюблених.

Рисунок 4.7 - Фрагмент коду: додавання треку до улюблених

У разі успішної відповіді від сервера, оновлені дані користувача одразу передаються в Redux через дію setUser(response.data). Таким чином, всі компоненти, що використовують ці дані (наприклад, список улюблених треків або іконка "серце" в SongCard), миттєво оновлюються, без перезавантаження сторінки. Крім того, через бібліотеку toast користувачеві відображається повідомлення "Список улюблених оновлено", що підвищує інформативність взаємодії.

Якщо ж виникає помилка (наприклад, недійсний токен або сервер недоступний), то вона перехоплюється блоком catch, і користувач отримує повідомлення про помилку. Це дозволяє зберігати контроль над користувацьким досвідом навіть у непередбачуваних ситуаціях. Загалом функція handleLike є важливою частиною інтерфейсу, оскільки забезпечує персоналізацію та взаємодію користувача з музичним контентом.

Реалізація через axios і toast дає змогу легко розширити функціонал — наприклад, реалізувати видалення з улюблених, або змінити тип повідомлень, не змінюючи бізнес-логіку. Також цей код може бути винесений у загальний хук або сервіс, якщо така логіка буде повторюватись у різних місцях додатку, що забезпечує чистоту і повторне використання коду.

Для зберігання та керування станом плеєра створено окремий playerSlice у Redux, який зображено на рисунку 4.8. У початковому стані initialState визначено такі змінні: currentTrack, isPlaying, currentIndex, trackList і repeatStatus. Це дозволяє точно описати, що саме зараз відтворюється, чи активне відтворення, яка пісня грає у списку, а також який режим повторення обрано. Структура зберігається в глобальному сховищі, доступному з будь-якої частини додатку.

Рисунок 4.8 - Фрагмент коду: playerSlice

У об’єкті reducers описані функції, що змінюють конкретні частини стану. Наприклад, setCurrentTrack змінює активний трек, setPlaying — встановлює статус відтворення (грає/не грає). Ці редьюсери використовуються у відповідних компонентах інтерфейсу, що дає змогу інтерактивно керувати станом плеєра на основі дій користувача — наприклад, пауза, наступний трек, зміна гучності.

Завдяки Redux можна уникнути "пробросу" пропсів через кілька рівнів компонентів. Компоненти просто підписуються на частину стану, яка їм потрібна, і реагують лише на її зміну. Це зменшує навантаження на головний компонент і дозволяє ефективно оновлювати лише ті частини інтерфейсу, які залежать від певних змін. Наприклад, MusicPlayer слухає currentTrack і isPlaying, а SongCard може використовувати currentTrack для підсвітки активної пісні.

Цей підхід також полегшує розробку нових функцій — додавання shuffle, repeat або керування гучністю, адже весь стан керується централізовано. Якщо з’явиться необхідність зберігати історію прослуховувань або кешувати останній стан, це також легко реалізувати в межах одного slice. Таким чином, структура playerSlice є критично важливою для стабільної роботи аудіо-плеєра у Vibenest.

4.3 Використані алгоритми та структури даних

4.3.1 Структури даних

4.3.1.1 Масиви

Масиви (Arrays) у проєкті Vibenest відіграють ключову роль у зберіганні

та обробці колекцій даних. Вони використовуються для представлення списків треків, плейлистів, улюблених композицій і навіть черги на відтворення. Завдяки своїй простоті та універсальності, масиви дозволяють зручно реалізовувати ітерацію, сортування та фільтрацію елементів, що критично важливо для музичного застосунку з динамічним контентом.

Наприклад, Redux Store зберігає масив trackList, у якому містяться всі треки, що доступні для відтворення. Під час перемикання пісень у плеєрі застосовуються операції з індексами масиву, а під час пошуку — метод filter, який дозволяє відібрати елементи за ключовим словом. Це робить масиви незамінними при реалізації таких функцій, як пошук, створення добірок і плейлистів.

Ще одна перевага масивів — підтримка мутабельних та немутатбельних операцій. Для уникнення змін у глобальному стані напряму, розробники часто використовують методи, які створюють копії масиву, наприклад map, slice чи оператор розпакування [...]. Це дозволяє зберігати чистоту стану та уникати неочікуваної поведінки в React-компонентах.

Крім того, масиви використовуються при генерації UI-компонентів у React. Наприклад, компонент SongList отримує масив треків і відображає кожен за допомогою компонента SongCard, використовуючи метод map. Таким чином, масиви виконують не лише логічну, а й візуальну роль у побудові інтерфейсу.

У ситуаціях, коли потрібно перевірити наявність елемента в списку (наприклад, чи є трек в улюблених), застосовується метод includes, який повертає булеве значення. Це дозволяє швидко визначити дію — додати елемент чи видалити. Загалом, масиви є універсальною структурою, яка забезпечує як гнучкість, так і продуктивність.

4.3.1.2 Об'єкти

Об'єкти (Objects) активно використовуються для опису складних сутностей, таких як пісня, користувач або плейлист. Кожен об’єкт містить набір властивостей, що логічно групують інформацію. Наприклад, об’єкт song зберігає

дані про назву треку, виконавців, тривалість, зображення обкладинки та унікальний ідентифікатор.

```
const song = {
  _id: "123",
  title: "Song Title",
  artistes: ["Artist 1", "Artist 2"],
  duration: "3:45",
  coverImage: "url",
};
```

Такий формат даних дає змогу легко працювати з ними у компонентах інтерфейсу. Наприклад, при відображенні картки треку можна одразу звернутись до властивостей song.title або song.coverImage. Завдяки цьому код залишається читабельним та структурованим.

Об'єкти також використовуються у Redux Store — кожен slice зберігає свої об'єкти, наприклад user, що містить дані про поточного користувача, та currentTrack, який описує активний трек у плеєрі. Це дозволяє ефективно організовувати глобальний стан.

Ще однією перевагою об'єктів є можливість швидко змінювати лише потрібне поле без впливу на інші. Це особливо корисно в Redux-редьюсерах, де, наприклад, можна оновити тільки user.favorites, не змінюючи решту об'єкта user.

Завдяки структурованості, об’єкти легко піддаються серіалізації — їх можна передати у запиті на сервер, зберегти в базі даних або у локальному сховищі (localStorage). Це універсальна форма представлення даних, яка добре інтегрується з API.

4.3.1.3 Redux Store

Redux Store є центральною структурою керування станом у застосунку Vibenest. Він дозволяє зберігати глобальні дані, які доступні у будь-якому компоненті, без потреби у пробросі пропсів через кілька рівнів вкладеності.

Структура Store має вигляд дерева, де кожен slice відповідає за певну

частину стану. Наприклад:

```
{
  player: {
    currentTrack: { ... },
    isPlaying: true,
    trackList: [ ... ],
  },
  user: {
    id: "user123",
    favorites: [ ... ],
  }
}
```

Завдяки Redux, компоненти можуть підписуватись лише на ті дані, які їм потрібні. Це оптимізує рендеринг і дозволяє змінювати стан централізовано. Для цього використовуються редьюсери, які змінюють частину стану згідно з викликаною дією (action).

Також Redux забезпечує передбачуваність поведінки застосунку. Усі зміни відбуваються через явно оголошені дії, що дозволяє легко відслідковувати помилки, відлагоджувати логіку і забезпечити стабільність застосунку.

Під час реалізації Vibenest було створено окремі slice для користувача (userSlice), плеєра (playerSlice) та бібліотеки треків. Це дозволяє масштабувати систему, додаючи нові функції без зміни вже існуючих частин.

4.3.2 Алгоритми

Одним із ключових аспектів створення інтуїтивно зрозумілого та функціонального інтерфейсу є правильна реалізація алгоритмів, які відповідають за обробку дій користувача. У Vibenest велика увага приділяється пошуку, навігації між треками, персоналізації, збереженню налаштувань та валідації введених даних. Розглянемо більш детально основні алгоритми, використані у проєкті.

4.3.2.1 Пошук треків за назвою або виконавцем

Щоб користувач міг швидко знаходити потрібну композицію серед великої кількості пісень, було реалізовано простий, але ефективний пошуковий алгоритм. Після введення запиту, застосунок виконує фільтрацію списку треків у режимі реального часу. Для цього використовується метод filter, який перебирає всі елементи масиву та перевіряє, чи містить назва треку або ім'я виконавця пошукове слово.

```
const filteredSongs = songs.filter(song =>
  song.title.toLowerCase().includes(searchTerm.toLowerCase()) ||
  song.artistes.some(artist => artist.toLowerCase().includes(searchTerm.toLowerCase()))
);
```

Цей підхід дозволяє знаходити треки як за частковим збігом у назві, так і за відповідністю імені одного з виконавців. Використання toLowerCase() гарантує нечутливість до регістру, що покращує точність пошуку. Результати оновлюються динамічно, без потреби у перезавантаженні сторінки або натисканні кнопки пошуку.

4.3.2.2 Додавання або видалення треку з улюбленого

Кожен користувач має можливість додавати треки до переліку улюблених. Для цього використовується простий алгоритм перевірки наявності ідентифікатора поточного треку в масиві favorites, що зберігається у глобальному стані користувача. Якщо трек уже є в цьому списку — він видаляється; якщо відсутній — додається.

```
const isFavorite = user.favorites.includes(song._id);
const toggleFavorite = () => {
  if (isFavorite) {
    user.favorites = user.favorites.filter(id => id !== song._id);
  } else {
    user.favorites.push(song._id);
```

```
    }
};
```

Такий механізм забезпечує швидкий відгук інтерфейсу, дозволяючи користувачу взаємодіяти зі своїми улюбленими треками без зайвих дій. У поєднанні з Redux оновлення списку відбувається миттєво, а елемент інтерфейсу змінює свій стан — наприклад, іконка «серце» стає заповненою.

4.3.2.3 Логіка перемикання треків у плеєрі

Навігація між треками у Vibenest реалізована за допомогою індексації у масиві trackList. Поточний індекс треку (currentIndex) дозволяє швидко визначити наступний або попередній трек. Також враховано крайові випадки, наприклад, повернення до першої пісні після останньої.

```
const nextTrack = () => {
  if (currentIndex >= trackList.length - 1) {
    setCurrentTrack(trackList[0]);
    setCurrentIndex(0);
  } else {
    setCurrentTrack(trackList[currentIndex + 1]);
    setCurrentIndex(currentIndex + 1);
  }
};
```

Цей підхід не тільки забезпечує зручність перемикання, але й дозволяє легко розширити функціонал у майбутньому — наприклад, реалізувати випадкове відтворення (shuffle), повтор одного треку або циклічне відтворення всього списку.

4.3.2.4 Збереження рівня гучності та позиції відтворення

Щоб зробити використання плеєра комфортнішим, реалізовано збереження рівня гучності у localStorage. Після перезавантаження сторінки користувач починає прослуховування на тій самій гучності, яка була задана раніше. Це забезпечує безперервність досвіду.

```
localStorage.setItem('player-volume', newVolume);
const savedVolume = localStorage.getItem('player-volume');
if (audioRef.current && savedVolume !== null) {
  audioRef.current.volume = parseFloat(savedVolume);
}
```

Таке рішення є зручним і для користувачів із різними уподобаннями — наприклад, ті, хто слухає музику через навушники, часто мають індивідуальні налаштування звуку. Крім того, збереження в локальному сховищі не потребує серверної взаємодії, а тому реалізується максимально швидко.

4.3.2.5 Валідація форм при додаванні треку

Щоб уникнути помилок при заповненні форм, на фронтенді реалізована перевірка правильності введених даних. Наприклад, поле тривалості перевіряється на відповідність формату хвилини:секунди, а назва треку не повинна бути порожньою.

```
const isValidDuration = (duration) => /^\d+:\d{2}$/.test(duration);
if (!formData.title || !isValidDuration(formData.duration)) {
  setError("Будь ласка, заповніть всі поля коректно");
  return;
}
```

Цей алгоритм забезпечує базову валідацію без використання сторонніх бібліотек. Проте його легко можна розширити у майбутньому — наприклад, додати перевірку допустимого формату файлів, обов’язкову наявність виконавця, або навіть попередній перегляд обкладинки. Валідація на клієнтській стороні зменшує кількість невдалих запитів до API та покращує UX.

4.3.2.6 Оптимізація рендерингу компонентів

Щоб уникнути зайвих оновлень інтерфейсу й підвищити продуктивність, у проєкті активно застосовуються React-хуки для оптимізації. Наприклад, useCallback використовується для збереження функцій між рендерами, що особливо важливо, коли функції передаються в дочірні компоненти.

```
const handlePlayClick = useCallback((e) => {
   e.preventDefault();
   e.stopPropagation();
   onPlay();
}, [onPlay]);
```

Також використовуються React.memo та useMemo, що дозволяє уникати повторних обчислень або повторного створення об’єктів, коли в цьому немає необхідності. Це особливо важливо при роботі з довгими списками треків або інтерактивними елементами.

Завдяки цьому навіть на слабших пристроях інтерфейс залишається плавним і стабільним, а час відгуку на дії користувача мінімізується.

4.4 Висновок по розділу

У четвертому розділі було розглянуто ключові аспекти реалізації інтерфейсної частини музичної платформи Vibenest. Детально описано архітектуру побудови клієнтської частини застосунку, зокрема — використання бібліотеки React, що забезпечила компонентний підхід, гнучкість у створенні UI та ефективну роботу зі станом за допомогою Redux.

У межах розділу охарактеризовано структуру основних компонентів інтерфейсу: картки треку, музичного плеєра, форми додавання пісень, панелі навігації тощо. Значна увага приділена алгоритмам взаємодії з користувачем — пошуку треків, додаванню до улюблених, перемиканню пісень, обробці локальних параметрів (наприклад, збереження гучності та позиції треку). Це дозволяє забезпечити зручний, швидкий і персоналізований користувацький досвід.

Також проаналізовано використані структури даних, зокрема масиви для зберігання списків треків, об'єкти для опису одиниць контенту, а також сховище Redux як центральну структуру для управління станом. Завдяки цим рішенням

вдалося досягти цілісності логіки інтерфейсу та його узгодженості з очікуваннями користувача.

Окремо розглянуто реалізацію графічного дизайну, заснованого на принципах UI/UX. На основі макетів з Figma розроблено інтерфейс із дотриманням принципів мінімалізму, контрастності та адаптивності. Вся візуальна частина узгоджена з логікою взаємодії, що дозволяє швидко орієнтуватися в застосунку навіть новим користувачам.

Таким чином, реалізована клієнтська частина Vibenest демонструє не лише сучасний технічний підхід до розробки SPA-додатків, але й врахування потреб користувача в зручності, швидкості й естетиці під час взаємодії з музичною платформою.

РОЗДІЛ 5. ТЕСТУВАННЯ ВЕБ-ДОДАТКУ

5.1 Стратегії та методи тестування

У процесі розробки інтерфейсу музичної платформи Vibenest тестування стало невід'ємною частиною циклу створення якісного програмного забезпечення. Воно охоплює як окремі компоненти інтерфейсу, так і логіку зміни стану додатку, що реалізована за допомогою Redux. Основна мета тестування полягала не лише у виявленні помилок, а й у забезпеченні стабільності функціоналу, особливо в тих місцях, де відбувається взаємодія з користувачем або зміна даних. Підхід до тестування був сформований таким чином, щоб виявляти проблеми ще на етапі розробки та швидко реагувати на них.

У проекті використовувались три основні інструменти для написання та запуску тестів. Першим з них є Jest — потужний фреймворк для юніт-тестування, який дозволяє запускати окремі тести або весь набір одночасно, створювати фіктивні (mocked) версії функцій, перевіряти виклики функцій, обробляти асинхронний код та перевіряти значення, які повертаються. Його гнучкість і простота інтеграції з іншими бібліотеками зробили Jest основою тестового середовища Vibenest.

Другим компонентом стеку є React Testing Library, яка була обрана завдяки своїй орієнтації на тестування поведінки користувача, а не реалізації. Цей підхід дозволяє проводити тестування на рівні DOM-елементів — як користувач бачить і взаємодіє з додатком. Наприклад, бібліотека дозволяє емулювати натискання кнопок, введення тексту у поля форм, зміну фокусу, навігацію тощо. Важливо, що тести не мають залежності від внутрішньої структури компонентів, тому при їхній зміні тести не потребують переписування — що суттєво знижує витрати часу на підтримку коду.

Окрему частину тестування було зосереджено на формі додавання нового треку — одній із ключових функцій для користувача. Тут було застосовано

Puppeteer — інструмент для e2e (end-to-end) тестування, який дозволяє запускати повноцінний браузер в анонімному режимі й взаємодіяти з інтерфейсом майже як справжній користувач. За його допомогою перевірялись такі сценарії, як: заповнення полів форми, перевірка валідації (наприклад, правильність тривалості у форматі хв:сек), завантаження аудіофайлів, відображення помилок та повідомлень про успішне додавання. Таким чином вдалося протестувати не лише окремі функції, а й цілісну поведінку системи.

Структура організації тестів у проєкті також була чітко продумана. Для кожного модуля (наприклад, компонента або Redux-slice) створювався відповідний файл з тестами, який розміщувався поряд із основним файлом модуля. Такий підхід дозволяє розробникам легко орієнтуватися у коді, одразу бачити, чи є покриття тестами, і оперативно вносити зміни. Наприклад, поряд із файлом SongCard.jsx знаходиться SongCard.test.jsx, а для playerSlice.js є відповідний playerSlice.test.js.

У React-компонентах тестування охоплювало два рівні: рендеринг та взаємодію. На рівні рендерингу перевірялась наявність необхідних елементів — наприклад, назви пісні та виконавця. Прикладом може бути тест, що перевіряє, чи відображається пісня “Test Song” у компоненті SongCard. Додатково симулювались дії користувача: клік на картку треку, після якого перевіряється, чи запустилось відтворення. Такі тести дозволяють гарантувати, що інтерфейс правильно реагує на користувача навіть при зміні внутрішньої реалізації компонентів.

Окремим аспектом стало тестування логіки Redux. Було протестовано, як змінюється глобальний стан при виклику дій (setCurrentTrack, setPlaying тощо). Для цього використовувалась функція playerReducer у зв’язці з Jest, яка дозволяла перевірити, що, наприклад, при виклику setCurrentTrack(track) у стані з’являється саме цей трек. Таке тестування дозволяє впевнено розширювати функціонал Redux-стану, не боячись зламати існуючу логіку.

Загалом обрана стратегія — поєднання юніт-тестів, тестування

інтерфейсу та e2e — дозволила охопити як низькорівневу логіку додатку, так і його поведінку з точки зору користувача. Завдяки цьому вдалося вчасно виявити низку дрібних помилок, пов’язаних із відображенням або реакцією на події, а також переконатися, що платформа залишається стабільною при подальшому розширенні функціоналу.

5.2 Результати тестування системи

Після завершення основної розробки інтерфейсу було виконано кілька типів тестування для перевірки стабільності та відповідності функціональності вебзастосунку Vibenest. Тестування охоплювало ключові області взаємодії: роботу Redux-сховища, поведінку окремих компонентів у типових сценаріях, а також енд-ту-енд тестування дій користувача через браузерне середовище. Всі перевірки проводились у середовищі розробника до розгортання проєкту.

Особливу увагу було приділено перевірці коректної роботи Redux-slice'ів, зокрема playerSlice. Основною метою було переконатись, що зміна стану відбувається згідно з очікуваною логікою — наприклад, при виклику дії setCurrentTrack справді оновлюється активний трек, а при setPlaying — змінюється статус відтворення. Для цього створено окремі модульні тести, які працюють без UI, перевіряючи лише логіку змін стану.

У прикладах тестів використовувались реальні об’єкти з початковим станом, що дозволило відтворити типову поведінку користувача. Тестування Redux-ред’юсерів підтвердило, що сховище оновлюється передбачувано, без побічних ефектів, і є стабільним у межах визначених сценаріїв.

Окремо тестувались ключові компоненти, зокрема SongCard, який відповідає за візуалізацію треків і взаємодію з користувачем. Модульні тести охоплювали перевірку відображення назви пісні, імені виконавця, а також реакцію на кліки.

Було перевірено, що після кліку на компонент правильно викликаються

відповідні дії Redux (через мокування dispatch) і запускається відтворення пісні. Для цього використовувалась бібліотека React Testing Library, яка імітує взаємодію через DOM. Такий підхід дозволяє перевірити поведінку компонентів максимально наближено до реального сценарію використання.

Модульні тести також враховували можливі крайові випадки — наприклад, відсутність даних або неправильне передання пропсів. Це дозволило виявити та усунути потенційні помилки на ранньому етапі.

На завершальному етапі було проведено енд-ту-енд тестування найважливішого сценарію — додавання нового треку через форму. Для цього застосовано Puppeteer — інструмент, який дає змогу автоматизувати взаємодію з браузером. У рамках тесту скрипт автоматично відкривав сторінку, вводив назву композиції, ім’я виконавця, додавав обкладинку та аудіофайл. Після цього форма проходила валідацію та надсилалась, а результат відображався у списку треків.

Це дозволило переконатися, що весь ланцюжок дій — від введення даних до появи пісні в інтерфейсі — працює стабільно. Завдяки Puppeteer вдалося відтворити реальні дії користувача, перевірити обробку введених даних і реакцію інтерфейсу на події, зокрема при помилках у формі чи неправильному файлі. Усі кроки тесту пройдені успішно, що підтверджується автоматичним логом у консолі, де видно результати модульних і функціональних тестів з використанням Jest та React Testing Library (зображено на рисунку 5.1).

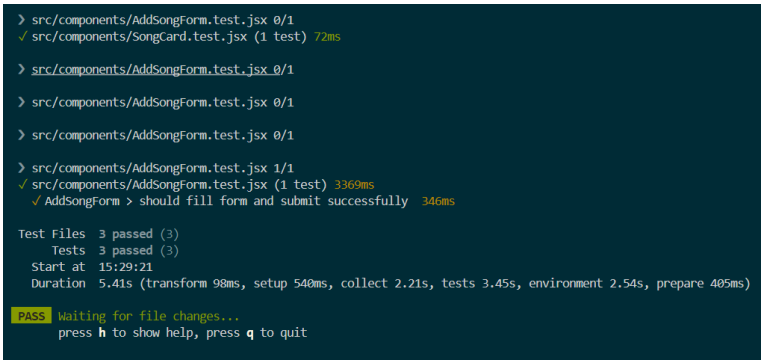
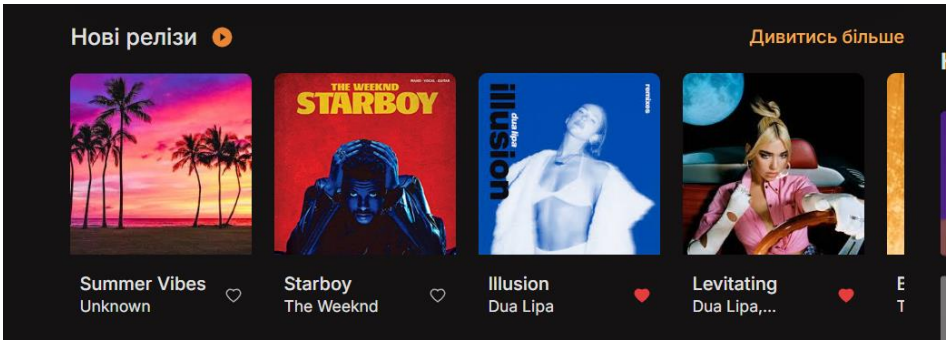


Рисунок 5.1 – Консольне повідомлення про успішне проходження модульних тестів

На рисунку 5.2 зображено код автоматичного тестування і результат. У рамках тесту скрипт автоматично відкривав сторінку, вводив назву композиції, ім’я виконавця, додавав обкладинку та аудіофайл. Після цього форма проходила валідацію та надсилалась, а результат відображався у списку треків. Це дозволило переконатися, що весь ланцюжок дій — від введення даних до появи пісні в інтерфейсі — працює стабільно.

```
test('should fill form and submit successfully', async () => {
    await page.waitForSelector('input[name="title"]', { timeout: 10000 });

    await page.type('input[name="title"]', 'Summer Vibes');
    await page.type('input[name="artistes"]', 'Unknown');

    const coverInput = await page.$('input[type="file"][accept="image/*"]');
    const coverPath = path.resolve(__dirname, '__mocks__/assets/test-image.jpg');
    await coverInput.uploadFile(coverPath);

    const audioInput = await page.$('input[type="file"][accept="audio/*"]');
    const audioPath = path.resolve(__dirname, '__mocks__/assets/test-track.mp3');
    await audioInput.uploadFile(audioPath);

    await page.waitForFunction(() => {
        const input = document.querySelector('input[name="duration"]');
        return input && input.value.trim().length > 0;
    }, { timeout: 15000 });

    await page.click('button[type="submit"]');
```

а)



б)

Рисунок 5.2 – Автоматизоване додавання треку: а) фрагмент коду e2e-тесту з Puppeteer; б) інтерфейс застосунку після успішного додавання пісні

Завдяки Puppeteer вдалося відтворити реальні дії користувача, перевірити обробку введених даних і реакцію інтерфейсу на події, зокрема при помилках у формі чи неправильному файлі. Усі кроки тесту пройдені успішно, що

підтверджується автоматичним логом у консолі.

5.3 Висновки до розділу

У п’ятому розділі було детально розглянуто питання тестування функціональних компонентів музичної платформи Vibenest. Основну увагу приділено модульному тестуванню, що здійснювалося із застосуванням інструментів Jest та React Testing Library. Такий підхід дав змогу перевірити працездатність окремих елементів інтерфейсу та логіки додатку, не залучаючи серверну частину.

Перевагою обраної стратегії стало те, що тести охопили ключові взаємодії користувача з інтерфейсом: рендеринг даних, натискання на елементи, запуск відтворення музики, оновлення стану Redux. Це дозволило переконатися у стабільності базового функціоналу ще до повного завершення розробки. Тестування проводилося в умовах, наближених до реального використання — з імітацією DOM-структури та типових сценаріїв взаємодії.

Важливим є і той факт, що структура тестів була логічно організована: кожен модуль або компонент мав власний файл з тестами, розміщений безпосередньо поряд із основним кодом. Це спрощувало підтримку та оновлення перевірок у процесі розробки.

Таким чином, реалізоване тестування забезпечило базовий рівень надійності системи, виявило потенційні проблеми ще на ранніх етапах та дозволило зробити перші висновки щодо якості клієнтської частини. Це створює міцне підґрунтя для майбутнього розширення функціоналу та переходу до більш глибокого рівня перевірок — зокрема, інтеграційного та e2e-тестування за участі реального сервера.

ВИСНОВКИ

У ході виконання бакалаврської роботи було проведено всебічний аналіз сучасних музичних веб-платформ з метою визначення основних функціональних і технічних вимог до фронтенд-інтерфейсу. Аналіз дозволив сформувати чітке уявлення про очікування користувачів щодо зручності, швидкодії та сучасного дизайну інтерфейсу.

Для реалізації фронтенд-частини платформи Vibenest була обрана бібліотека React, що забезпечила створення компонентної архітектури. Такий підхід дозволив структурувати код, забезпечити високу модульність і гнучкість у подальшій розробці та підтримці. Кожен компонент відповідає за окремий функціональний блок інтерфейсу, що підвищує повторне використання та масштабованість.

Розроблено і реалізовано основний функціонал користувацького інтерфейсу, який включає:

- Механізм реєстрації та аутентифікації користувачів з перевіркою введених даних.
- Пошук музичних треків із зручним інтерфейсом відображення результатів.
- Завантаження та відтворення музичних композицій за допомогою кастомного плеєра з повним набором контролів.
- Можливість створення та перегляду персональних добірок.
- Односторінкову (SPA) навігацію, що забезпечує швидку і безперервну взаємодію з додатком без перезавантаження сторінок.

Інтерактивність і обмін даними з сервером реалізовано через REST API, що забезпечує надійну та ефективну взаємодію фронтенду з бекендом.

Особливу увагу було приділено тестуванню користувацького інтерфейсу на предмет функціональності, адаптивності та продуктивності. Тестування проводилося на різних пристроях і розмірах екранів, що підтвердило

стабільність відображення та коректність роботи додатку у різних умовах.

Оптимізація коду та ресурсів сприяла покращенню швидкодії інтерфейсу, зменшенню часу завантаження і підвищенню загальної плавності роботи. Використання React у поєднанні з сучасними підходами до управління станом і маршрутизації зробило застосунок масштабованим і готовим до подальшого розширення функціоналу.

Таким чином, фронтенд-частина проєкту реалізована на високому технічному рівні, відповідає сучасним стандартам веб-розробки і забезпечує комфортну роботу користувачів з цифровою музичною платформою. Розроблена архітектура та функціональність можуть бути використані як основа для подальшого розвитку мультимедійних інтерфейсів.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. IFPI GLOBAL MUSIC REPORT 2025 [Електронний ресурс] – Режим доступу: https://www.ifpi.org/wp-content/uploads/2024/03/GMR2025_SOTI.pdf

2. Simplebeen. Spotify User Statistics [Електронний ресурс]. – Режим доступу:

https://simplebeen.com/spotify-user-statistics

3. Music Business Worldwide [Електронний ресурс]. – Режим доступу: https://www.musicbusinessworldwide.com/youtube-music-hits-125m-paid-subscribers-adding-2m-subs-per-month-on-average-over-the-past-year/

4. Основні вимоги до дизайну сайту [Електронний ресурс] – Режим доступу: https://ua.waykun.com/articles/osnovni-vimogi-do-dizajnu-sajtu.php

5. The State of UX Design in 2025 [Електронний ресурс] – Режим доступу: https://www.wix.com/blog/ux-design-trends

6. UI/UX design [Електронний ресурс] – Режим доступу: https://www.coursera.org/articles/ui-vs-ux-design

7. React – A JavaScript library for building user interfaces [Електронний ресурс] – Режим доступу: https://reactjs.org/

8. Angular – The modern web developer’s platform [Електронний ресурс]. – Режим доступу: https://angular.dev/

9. Vite – Next Generation Frontend Tooling [Електронний ресурс] – Режим доступу: https://vitejs.dev/

10. Webpack – Module bundler [Електронний ресурс]. – Режим доступу: https://webpack.js.org/

11. Tailwind CSS – Rapidly build modern websites [Електронний ресурс] – Режим доступу: https://tailwindcss.com/

12. Bootstrap – The most popular HTML, CSS, and JS library [Електронний ресурс]. – Режим доступу: https://getbootstrap.com/docs/

13. Від вінілу до стрімінгу: розвиток технологій розповсюдження та

прослуховування музики [Електронний ресурс] – Режим доступу: https://zn.ua/ukr/novosti-kompaniy/vid-vinilu-do-striminhu-rozvitok-tekhnolohij-rozpovsjudzhennja-ta-proslukhovuvannja-muziki.html

14. Методичні рекомендації та вимоги щодо виконання та оформлення бакалаврських робіт для студентів спеціальності 121 “Інженерія програмного забезпечення. В. Я. Піх, М. М. Піх, В.В. Бандура – Івано- Франківськ: ІФНТУНГ, 2024. – 52с.

15. Музика // Вікіпедія [Електронний ресурс]. – Режим доступу: https://uk.wikipedia.org/wiki/Музика

16. Вітт С. Як музика стала вільною. Цифрова революція та перемога піратства. – Київ: Наш Формат, 2017. – 360 с.

17. React Testing Library – Simple and complete testing utilities [Електронний ресурс]. – Режим доступу: https://testing-library.com/docs/react-testing-library/intro/

18. Mock Service Worker (MSW) [Електронний ресурс]. – Режим доступу: https://mswjs.io/

19. JavaScript documentation / Mozilla Developer Network [Електронний ресурс]. – Режим доступу: https://developer.mozilla.org/en-US/docs/Web/JavaScript

20. Figma — Collaborative interface design tool [Електронний ресурс]. – Режим доступу: https://www.figma.com/

21. Spotify Developer Documentation [Електронний ресурс]. – Режим доступу: https://developer.spotify.com/documentation/web-api

22. YouTube Music Help / Google Support [Електронний ресурс]. – Режим доступу: https://support.google.com/youtubemusic

23. Tailwind UI Patterns and Best Practices [Електронний ресурс]. – Режим доступу: https://tailwindui.com/

24. GitHub – Інструмент командної розробки [Електронний ресурс]. – Режим доступу: https://github.com/

25. Flanagan D. JavaScript: The Definitive Guide: Master the World's Most-

Used Programming Language. – 7th ed. – Beijing; Boston: O'Reilly Media, 2020. – 706 с.

26. Rescigno R. Tailwind CSS: A guide to using the popular utility-first CSS framework. – Paperback. – [S.l.]: Independently published, 2023. – 138 с.

27. S. Payla P., Hartson R. The UX Book: Agile UX Design for a Quality User Experience. – 2nd ed. – Amsterdam: Morgan Kaufmann, 2019. – 744 с.

28. Tidwell J. Designing Web Interfaces: Principles and Patterns for Rich Interaction. – Sebastopol: O'Reilly Media, 2009. – 352 с.

29. Wieruch R. The Road to React: With React 18 and React Hooks: Required Knowledge: JavaScript. – 5th ed. – Berlin: Leanpub, 2023. – 340 с.

30. Learn ReactJS from Scratch [Електронний ресурс] – Режим доступу: https://ngeducate.blogspot.com/p/learn-reactjs-from-scratch.html

Додаток А

https://github.com/romannung/vibenest

БІБЛІОГРАФІЧНА ДОВІДКА

Тема дипломної роботи: “Розробка frontend музичного веб-програвача локальної музики”

Обсяг пояснювальної записки: 74 аркушів

Дата закінчення роботи 10 червня 2025р.

Підпис _____