Міністерство освіти і науки України
Луцький національний технічний університет
Факультет комп'ютерних та інформаційних технологій
Кафедра інженерії програмного забезпечення

КВАЛІФІКАЦІЙНА РОБОТА

ЗА СТУПЕНЕМ ВИЩОЇ ОСВІТИ «БАКАЛАВР»

РОЗРОБКА ВЕБДОДАТКУ ДЛЯ ПРОСЛУХОВУВАННЯ МУЗИКИ НА СТРИМІНГОВІЙ ПЛАТФОРМІ З ВИКОРИСТАННЯМ REACT, NEXT.JS ТА TYPESCRIPT

DEVELOPMENT OF A WEB APPLICATION FOR MUSIC STREAMING USING REACT, NEXT.JS AND TYPESCRIPT

спеціальність 121 «Інженерія програмного забезпечення»
освітня програма «Інженерія програмного забезпечення»

Виконав: здобувач вищої освіти
групи ІПЗ-42
Франовський Дмитро Христович

Керівник:
Ящук Андрій Анатолійович

Кваліфікаційну роботу
допущено до захисту
«10» 06 2025 р.
Гарант освітньої програми:
к.т.н., доцент
Ліщина Наталія Миколаївна

_____

Луцьк – 2025 року

ЛУЦЬКИЙ НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ

Факультет *комп'ютерних та інформаційних технологій*
Кафедра *інженерії програмного забезпечення*
Ступінь вищої освіти *бакалавр*
Галузь знань: *12 «Інформаційні технології»*
Спеціальність: *121 «Інженерія програмного забезпечення»*
Освітня програма: «*Інженерія програмного забезпечення»*

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____
«10» 12 2024 р.

ЗАВДАННЯ

НА КВАЛІФІКАЦІЙНУ РОБОТУ ЗДОБУВАЧУ ВИЩОЇ ОСВІТИ

Франовському Дмитру Христовичу
(прізвище, ім'я, по батькові)

1. Тема кваліфікаційної роботи *«Розробка вебдодатку для прослуховування музики на стримінговій платформі з використанням React, Next.js та TypeScript»*
Керівник роботи: *к.т.н., доцент Ящук А.А.*
затверджені наказом закладу вищої освіти від *«19» грудня 2024 р.* № *474/01-02*
2. Строк подання здобувачем вищої освіти кваліфікаційної роботи бакалавра *«10» червня 2025 р.*
3. Вихідні дані до роботи: *React, Next.js, Supabase, TypeScript, Visual Studio Code, PostgreSQL, методичні вказівки до виконання кваліфікаційної роботи бакалавра.*
4. Зміст розрахунково-пояснювальної записки (перелік питань, що потрібно розробити): *аналіз сучасних аналогів, визначення функціональних та нефункціональних цілей та вимог до розробки, проектування архітектури додатку та структури бази даних, реалізація функціоналу вебдодатку, тестування та перевірка працездатності системи.*
5. Перелік графічного матеріалу: *16 рисунків, 1 таблиця*

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис	
		завдання видав	завдання прийняв
Аналіз предметної області	*Яцук А. А.*		
Специфікація вимог до розробленої системи	*Яцук А. А.*		
Розробка об'єкта проектування	*Яцук А. А.*		
Нормоконтроль	*Вознюк А. В.*		
Гарант ОП	*Ліщина Н. М.*		
Показник запозичень тексту	1.74 %		
Академічна доброчесність	*Яцук А. А.*		

7. Дата видачі завдання *«20» грудня 2024 р.*

№	Назва етапів кваліфікаційної роботи бакалавра	Строк виконання етапів роботи	Примітка
1	Огляд літературних джерел по темі кваліфікаційної роботи бакалавра	до 04.02.2025 р.	вик.
2	Аналіз проблеми розробки та впровадження об'єкту проектування	до 01.03.2025 р.	вик.
3	Обґрунтування вибору шляхів, технологій і засобів вирішення поставленого завдання	до 15.03.2025 р.	вик.
4	Розробка функціонально-структурної схеми роботи об'єкта проектування та проектування бази даних	до 29.03.2025 р.	вик.
5	Практична реалізація об'єкта проектування та розробка бази даних	до 26.04.2025 р.	вик.
6	Тестування та налагодження об'єкта проектування	до 03.05.2025 р.	вик.
7	Здача чистового варіанту кваліфікаційної роботи бакалавра на кафедру	до 10.06.2025 р.	вик.

Здобувач вищої освіти _____ Дмитро ФРАНОВСЬКИЙ

Керівник кваліфікаційної роботи _____ Андрій ЯЩУК

АНОТАЦІЯ

Франовський Д. Х. Розробка вебдодатку для прослуховування музики на стримінговій платформі з використанням React, Next.js та TypeScript. Рукопис. Кваліфікаційна робота бакалавра ОП «Інженерія програмного забезпечення» спеціальності «Інженерія програмного забезпечення». Луцький національний технічний університет. Луцьк, 2025.

Кваліфікаційна робота бакалавра складається зі вступу, трьох розділів, висновків, списку використаних джерел.

У першому розділі здійснено аналіз сучасного стану проблеми, проведено патентний аналіз існуючих музичних стримінгових сервісів, сформульовано актуальність та основні завдання кваліфікаційної роботи.

У другому розділі визначено вимоги до вебдодатку, обґрунтовано вибір технологій та інструментів, описано структуру бази даних, модульну архітектуру системи, засоби безпеки, а також наведено UML-діаграми для моделювання взаємодії компонентів.

У третьому розділі описано практичну реалізацію проєкту: процес розробки інтерфейсу користувача, автентифікації, реалізацію функціоналу пошуку, завантаження та відтворення музики, створення плеєра з адаптивним дизайном та мобільною версією.

У висновках підсумовано результати розробки, оцінено ефективність вибраних рішень, а також вказано напрями подальшого розвитку системи.

Ключові слова: вебдодаток, музичний сервіс, стримінговий сервіс, плеєр, інтерфейс, UI/UX, адаптивний дизайн, React, Next.js, TypeScript, автентифікація, база даних, API, завантаження треків, пошук пісень, JWT токени, RESTful API, Supabase, Zustand, Axios, UML, діаграми, RLS.

ABSTRACT

Franovskyi D. Development of a web application for music streaming using React, Next.js, and TypeScript. Manuscript. Bachelor's qualification thesis of the educational program "Software Engineering" in the specialty "Software Engineering" Lutsk National Technical University. Lutsk, 2025.

The bachelor's qualification thesis consists of an introduction, three chapters, conclusions, and a list of references.

The first chapter presents an analysis of the current state of the problem, a patent analysis of existing music streaming services, and formulates the relevance and main objectives of the qualification work.

The second chapter defines the requirements for the web application, justifies the choice of technologies and tools, describes the database structure, the modular system architecture, security tools, and includes UML diagrams to model the interaction between components.

The third chapter describes the practical implementation of the project: the development process of the user interface, authentication, implementation of search functionality, uploading and playing music, and creation of a player with adaptive design and a mobile version.

The conclusions summarize the development results, evaluate the effectiveness of the chosen solutions, and outline directions for future development of the system.

Keywords: web application, music service, streaming service, player, interface, UI/UX, responsive design, React, Next.js, TypeScript, authentication, database, API, track upload, song search, JWT tokens, RESTful API, Supabase, Zustand, Axios, UML, diagrams, RLS.

ЗМІСТ

ВСТУП

Актуальність теми. Актуальність досліджуваної теми зумовлена зростаючим впливом музики на повсякденне життя людини, оскільки вона відіграє важливу роль – супроводжує в повсякденних ситуаціях, створює настрій, надихає та сприяє розслабленню. Сучасні музичні вебдодатки надають змогу шукати, відтворювати та організовувати улюблені треки, створюючи комфортне середовище для взаємодії з музичним контентом.

Водночас важливо не лише забезпечити базову функціональність, а й створити привабливий інтерфейс, логічну структуру, зручну навігацію та підтримку взаємодії між користувачем і додатком. Особливу увагу слід приділити якості користувацького досвіду, стабільності роботи системи, адаптивності до різних типів пристроїв і загальній інтуїтивності вебінтерфейсу.

Метою даної кваліфікаційної роботи є створення функціонального вебдодатку для прослуховування музики, який забезпечує зручність використання, гнучкість у роботі з музичним контентом та позитивний користувацький досвід.

Об’єктом кваліфікаційної роботи є процес розробки вебзастосунку для прослуховування музики.

Предметом кваліфікаційної роботи є структура, функціональні особливості та інтерфейсна взаємодія інформаційної системи.

Для досягнення поставленої мети були наведені такі цілі:

– сформулювати функціональні та нефункціональні вимоги до майбутньої системи;

– обґрунтувати вибір технологій та інструментів, необхідних для реалізації системи;

– створити архітектуру вебдодатку, розробити базу даних та забезпечити її безпечну структуру;

– реалізувати функціонал вебдодатку;

– провести тестування розробки.

РОЗДІЛ 1
АНАЛІЗ РОЗРОБКИ ВЕБЗАСТОСУНКІВ ДЛЯ ЦИФРОВОГО РОЗПОВСЮДЖЕННЯ МУЗИЧНОГО КОНТЕНТУ І ПОСТАНОВКА ЗАВДАНЬ НА КВАЛІФІКАЦІЙНУ РОБОТУ

1.1 Аналіз сучасного стану проблеми

У сучасному світі музику можна почути майже повсюди: в торгових центрах, на вулиці, в соціальних мережах і ще багато прикладів. І через це, перед людиною постає проблема – а де ж краще слухати її? На вирішення цієї проблеми приходить такі музичні стримінгові сервіси, як Spotify, Apple Music, Youtube Music тощо. Музичні сервіси є сучасним інструментом цифрового розповсюдження, збереження та прослуховування музики. Вони надають користувачам доступ до мільйонів треків у режимі онлайн або офлайн, дозволяючи створювати власні плейлисти, ділитися музикою та отримувати персоналізовані рекомендації.

Патентний аналіз показує, що провідні компанії активно розробляють і захищають інновації у сфері музичних сервісів.

Наприклад, Spotify має патенти на системи аналізу музичних переваг та формування плейлистів, який описує метод аналізу емоційного стану користувача для рекомендації треків, створюючи плейлисти на основі його уподобань [1].

Компанії Apple [2] і Google [3] мають патенти на системи голосового керування, автоматичне створення рекомендацій на основі локації, часу доби або настрою.

Дослідження патентних баз WIPO, USPTO та Google Patents виявили сотні патентів, що стосуються оптимізації потокового передавання, керування бібліотекою користувача, обробки аудіосигналів тощо.

Стримінгові сервіси стали невід’ємною частиною повсякденного життя мільйонів користувачів по всьому світу. Вони мають низку переваг, що і робить їх такими популярними. В першу чергу, це миттєвий доступ до величезної

бібліотеки музики з всіх куточків планети, що дозволяє слухати улюблені пісні без потребності їх завантажувати. Spotify, наприклад, вирізняється потужними алгоритмами рекомендації контенту для користувача, підбираючи пісні за його вподобаннями. Apple Music пропонує високу якість звуку у lossless форматі та об’ємним звучанням. Youtube Music має перевагу в тому, що користувач має можливість слухати не тільки офіційні пісні, але й неофіційний контент, який може завантажити будь-який користувач.

Проте стримінгові сервіси мають низку недоліків. Переважно вони працюють за моделлю freemium (безкоштовне використання з рекламою) або за преміум підпискою. При безкоштовній версії користувачі змушені слухати рекламу, не можуть перемикати треки без обмежень, наприклад Spotify має обмеження в перемиканні 6 пісень за годину. Apple Music взагалі має тільки пробну версію преміум підписки. Таким чином, стримінгові сервіси надають зручний і легальний доступ до великої кількості музичного контенту. Але багато людей досі використовують неліцензійну музику, тобто як це ще називають, займаються піратством [4]. Певно, не всі готові платити за такі послуги, які для прикладу надає Apple Music чи інший стримінговий сервіс.

Спираючись на всі переваги і недоліки цих стримінгових сервісів, я захотів створити власний. Такий музичний сервіс, який підійде для всіх людей: зручний, з величезною кількістю контенту і головне безкоштовний. Обмеження у виді платної підписки створює бар’єр для великої аудиторії, особливо для користувачів із країн з нижчим доходом, які не готові чи не мають можливості щомісяця платити за музику. Безкоштовний музичний сервіс міг би задовільнити попит на легальний доступ до контенту без фінансових витрат і також частково б вирішило проблему піратства. На питання якби можна було б зберегти безкоштовність цього продукту, не витрачаючи при цьому фінансову життєздатність продукту, можна запропонувати альтернативний підхід до монетизації за допомогою співпраці з артистами, донат від користувачів за бажанням або ж підтримка спонсорів. Також чим би відрізнявся цей стримінговий сервіс від конкурентів, це кращим представленням маловідомих

артистів або підтримкою відкритих форматів та музичного архіву незалежних виконавців Це створює потенціал для формування спільноти навколо платформи та привернення уваги творців, які шукають доступні способи розповсюдження своєї музики.

Отже, запуск безкоштовного музичного сервісу має як соціальну, так і ринкову доцільність – він дозволить розширити доступ до музики, стимулювати розвиток нових форматів взаємодії між артистами й слухачами, а також створити конкуренцію, яка може сприяти підвищенню якості послуг на музичному ринку загалом. Саме такий музичний сервіс я і хочу створити, моя мета – це об’єднання людей за допомогою музики.

1.2 Постановка завдання на кваліфікаційну роботу бакалавра

На основі проведеного аналізу предметної області та огляду існуючих рішень було сформульовано перелік основних завдань, необхідних для досягнення мети кваліфікаційної роботи. Основним завданням є створення зручного, адаптивного вебдодатку для прослуховування музики з відкритим доступом до контенту без потреби в платній підписці. Для цього потрібно реалізувати такі цілі:

– провести аналіз наявних музичних сервісів та виявити їхні сильні та слабкі сторони;

– сформулювати функціональні та нефункціональні вимоги до майбутньої системи;

– обґрунтувати вибір технологій та інструментів, необхідних для реалізації системи;

– створити архітектуру вебдодатку, розробити базу даних та забезпечити її безпечну структуру;

– реалізувати функціонал вебдодатку;

– провести тестування розробки.

Висновки до розділу 1

У першому розділі було проведено аналіз сучасного стану розвитку музичних стримінгових сервісів, вивчено їхні переваги та недоліки, а також здійснено патентний огляд інновацій у цій сфері. Визначено актуальність створення нового вебдодатку, що забезпечує безкоштовний доступ до музики, підтримку незалежних виконавців і зручний користувацький досвід. На основі проведеного аналізу сформульовано основні завдання кваліфікаційної роботи, спрямовані на розробку функціонального, адаптивного та доступного для користувачів вебдодатку.

РОЗДІЛ 2
СПЕЦИФІКАЦІЯ ВИМОГ ДО РОЗРОБЛЮВАНОЇ СИСТЕМИ

2.1 Аналіз, визначення вимог до розроблюваного програмного забезпечення та проектування програмного забезпечення

Основною ціллю розробки вебдодатку музичного сервісу є забезпечення користувачам швидкого, доступного та приємного способу слухати улюблену музику онлайн. Проект має на меті поєднати ключові переваги сучасних музичних сервісів, а саме зручний інтерфейс, висока продуктивність, забезпечення комфортного користування незалежно від пристрою та зробити вебдодаток доступним для всіх користувачів. Проект орієнтований як на звичайних користувачів, так і для музичних виконавців, які можуть без труднощів завантажити свою творчість на вебсайт.

Аналіз аналогів є важливим аспектом в підготовці до визначення вимог і розробки програмного продукту. Основним і головним елементом в любому музичному додатку є відтворення аудіо. Також важливими складовими є можливість авторизуватися в системі, зберігати пісні в список вподобаних, великий каталог музичних композицій з відображенням основної інформації (назва, автор та обкладинка) та пошук музики. Отже, фокус іде на інтуїтивну, доступну і комфортну взаємодію користувача з системою.

Планування проекту можна поділити на три основні вимоги – функціональну, нефункціональну та стратегічну перспективу. До функціональних вимог відноситься весь функціонал: авторизація, пошук, відтворення музики тощо. Нефункціональними вимогами є швидкодія контенту, надійність і безпека даних користувачі і кросбраузерність. На більш глибокому рівні розробка має також на меті створити платформу, яку в подальшому можна було б розвивати, в цьому і полягає стратегічна перспектива. У перспективі цей проект може слугувати не лише засобом прослуховування музики, а й середовищем для просування власної творчості для незалежних артистів.

Для реалізації цієї мети було обрано об’єктно-орієнтований [5] підхід до моделювання системи з використанням UML-нотацій, що дозволяє наочно відобразити структуру та логіку взаємодії між компонентами додатку. Цей підхід передбачає наявність декількох незалежних, але взаємопов’язаних модулів: авторизація та реєстрація користувача, відображення музичного контенту, відтворення аудіо, збереження пісень у список вподобаних, завантаження своїх пісень, а також підтримка адаптивного дизайну.

Use case діаграми (діаграма прецедентів) використовують для розуміння контексту та сценарію взаємодії користувача з системою, що в подальшому допоможе оптимізувати її, також ця діаграма корисна для планування впровадження функцій [6]. Use case діаграма, яка зображена на рисунку 2.1, допомагає зрозуміти основні функції вебдодатку та спростити розробку. Діаграма прецедентів наочно відображає дії, які користувачі можуть виконувати та ролі в системі із різними можливостями. До дій користувачів відноситься пошук пісень, відтворення аудіо, завантаження своїх пісень на сайт, вподобання треків, а до ролей те, що можуть робити гості, зареєстровані користувачі або адміністратори.

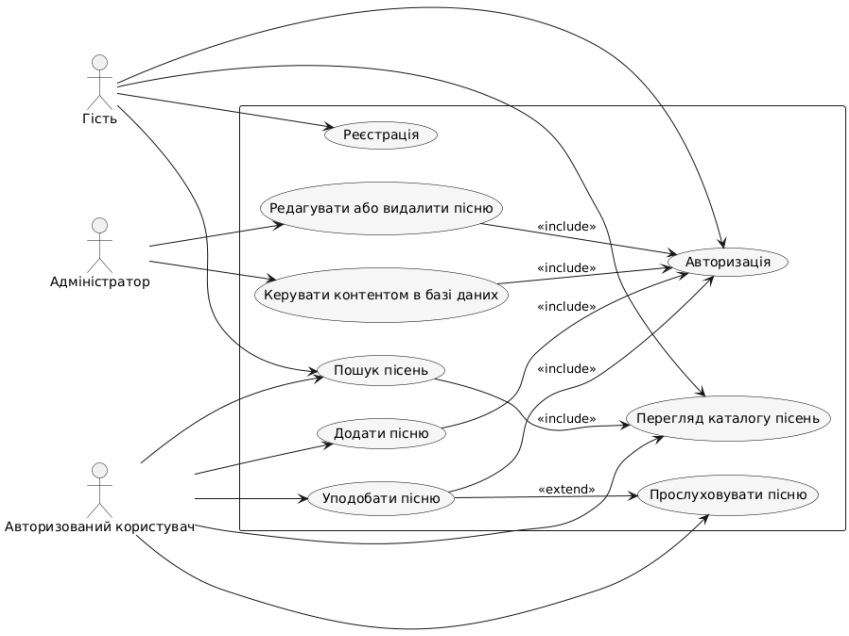

Рисунок 2.1 – Діаграма прецедентів

Діаграма класів [7] в UML-нотаціях – це статична діаграма, яка відображає властивості системи, класи, зв'язки між об'єктами для опису структури системи. Для зображення того, як класи взаємодіють між собою, тобто для кращого розуміння взаємодії користувача із додатком, я зобразив діаграму класів (рис. 2.2). Вона слугує наочним представленням класів в системі та їх взаємозв'язки. В цій діаграмі зображено, як користувач може завантажувати свою музику, додавати її в список вподобань та керувати відтворенням за допомогою плеєра. Діаграма також не тільки дозволяє моделювати базові операції з даними – від отримання списку пісень із бази даних до оновлення стану вподобань, але й чітко визначає функціональність системи через методи класів, наприклад керування відтворенням.

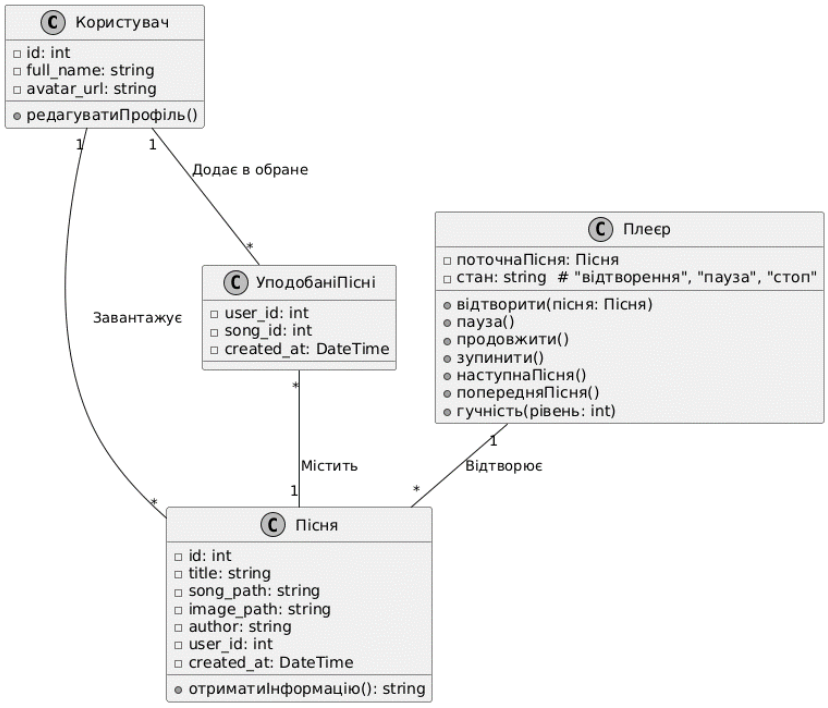

Рисунок 2.2 – Діаграма класів

За допомогою діаграми послідовності можна зобразити взаємодію об'єктів і процесів під час виконання сценарію. Діаграма на рисунку 2.3 зображає типовий сценарій взаємодії – вибір користувачем пісні для відтворення. Через інтерфейс передається запит до бекенду, який повертає дані треку, після чого ініціюється його програвання через плеєр [8].

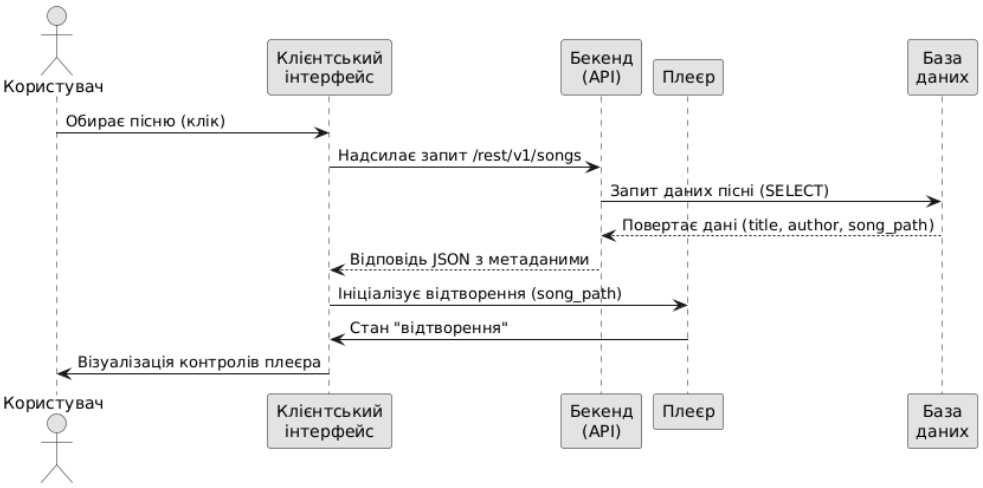

Рисунок 2.3 – Діаграма послідовності

У ході аналізу цілей та вимог було обрано об’єктно-орієнтовану модель з використанням UML-нотації для планування та візуального демонстрування майбутнього функціоналу. Також під час моделювання було побудовано діаграму прецедентів, діаграму класів та діаграму послідовності. Основними результатами моделювання стало чітке розмежування ролей (користувач, гість й адміністратор), діаграми охоплюють ключові процеси і вистроєна зрозуміла схема роботи вебдодатку.

Система орієнтована на взаємодію користувача з інтерфейсом та інтегрується з базою даних через бекенд, що дає змогу організувати надійне зберігання, обробку і передачу інформації в реальному часі.

Початкове введення інформації в базу даних відбувається під час автентифікації користувача та завантаження пісень. Під час створення облікового запису користувач надає свої дані, які автоматично мають заноситись в таблицю users, а при додаванні на сайт своєї пісні – до таблиці songs. А також у процесі взаємодії записуються дані про вподобані пісні і передаються в таблицю liked_songs.

Архівація даних передбачає собою не повне видалення даних, а лише часткове. Це має працювати завдяки soft delete [9], дані будуть помічатись як видалені і попадати в колонку «deleted». Це дозволятиме зберегти історію та

уникнути втрати якогось контенту. Натомість, якщо адміністратор захоче видалити якийсь контент, для цього є bypass, який дозволяє обійти м'якого видалення і видалити рядок назавжди.

Однією з ключових функцій системи має стати здійснення пошуку за запитом користувача. Користувач зможе знаходити треки за назвою або іменем виконавця, а результати будуть фільтруватися в реальному часі. Для цього планується реалізувати взаємодію клієнтської частини з базою даних за допомогою запитів SQL.

Уся робота додатку передбачатиме віддалений доступ до системи через Інтернет за протоколом TCP/IP. Стек протоколів TCP/IP – це набір правил обміну інформацією між комп'ютерами в мережі [10]. Весь доступ до функціоналу здійснюватиметься через браузер, що дозволить користувачеві використовувати додаток незалежно від його геопозиції або типу пристрою.

Особливу увагу потрібно приділити захисту і безпеці даних, вона забезпечуватиметься за допомогою засобів автентифікації. Планується використання JWT-токенів для захищеного збереження сесій, шифрування даних, а також передача інформації виключно через протокол HTTPS. JWT-токени часто використовуються для автентифікації та авторизації в вебдодатках. Як працюватиме автентифікація на основі токенів? Користувач надає свої дані (логін і пароль), сервер видає користувачу токен, в якому знаходиться інформація, яка його ідентифікує та дає дозвіл на доступ до ресурсів. Користувач при кожній взаємодії з контентом, передає свій токен на сервер з кожним запитом до захищених ресурсів, наприклад піснею. Це відбувається залежно від використаного протоколу і API запиту. Потім сервер перевіряє та декодує токен на його цілісність, авторизовані дані та дозволи [11].

Однією з цілей також, як і було описано і зображено вище в діаграмі, є розмежування по правам доступу до функцій платформи. Гості матимуть можливість тільки переглядати бібліотеку пісень, однак не матимуть можливості відтворити їх. Тоді як зареєстровані користувачі матимуть доступ до таких можливостей: відтворювати і зберігати в списку вподобаних пісні та

завантажувати свої пісні. Для реалізації цієї цілі використовуватиметься система політик безпеки RLS. Row-Level Security [12] – це тип керування даними, який дозволяє розробникам та адміністраторам обмежувати дані, до яких користувач має доступ. Без RLS кінцеві користувачі можуть бачити дані, до яких вони не повинні мати доступу.

Для забезпечення цілісності бази даних планується ретельне структурування таблиць. Насамперед, структура бази даних має бути розроблена таким чином, щоб забезпечити чіткий зв’язок сутностей, наприклад кожна пісня зберігатиметься в табличці songs і матиме зовнішній ключ user_id, який буде відображати який саме користувач завантажив цю пісню. Окрім встановлення зовнішніх ключів, ще однією ціллю є застосувати обмежень типів для певних колонок в таблицях, для прикладу є поле created_at, в якому будуть зберігатися дані тільки у форматі часу та дати, а в полі title буде обмеження текстовим типом. Також будуть впроваджені унікальні обмеження для певних атрибутів для того, аби запобігти дублювання даних.

Відображення інформації слід втілити в інтерфейсі користувача за допомогою динамічного рендерингу [13]. Після пошуку або вибору треку, результати повинні одразу з'являтися на сторінці, без оновлення, завдяки React та Next.js. Це дозволить забезпечити високу швидкість взаємодії та зручність для користувача.

Невід’ємною частиною будь-якого вебдодатку або програмного забезпечення в цілому є його інтерфейс, оскільки саме від UI/UX залежить якість взаємодії користувача із системою. Навіть при наявності неймовірного функціоналу чи стабільної роботи бекенду, якщо інтерфейс незручний, незрозумілий чи не привабливий, користувач не зможе ефективно взаємодіяти з додатком, що призведе до розчарування, втрати зацікавленості та ймовірного припинення використання продукту. UI (User Interface) – це візуальне оформлення продукту, він визначає як буде виглядати додаток: типографія та кольори, іконки та кнопки, композиція, форми та відступи, анімація елементів і адаптивність до різних пристроїв [14]. Якщо інтерфейс перенавантажений,

непропорційний або нелогічний – користувач швидко втрачає інтерес і залишає додаток. У свою чергу, UX (User Experience) відповідає за відчуття та емоції, які виникають у користувача при взаємодії з інтерфейсом. Якісний UX зменшує час навчання, підвищує задоволеність і сприяє повторному використанню сервісу. У контексті музичного вебдодатку UX/UI має особливе значення, оскільки користувачі очікують миттєвого доступу до контенту, плавної навігації та зручного керування відтворенням. Якісний інтерфейс повинен мінімізувати кількість дій для досягнення цілі (наприклад, від пошуку до запуску пісні) та водночас створювати позитивне емоційне враження від взаємодії з додатком.

Інтерфейс умовно поділений на декілька функціональних блоків, з якими буде взаємодіяти користувач:

– головна сторінка, на якій знаходитиметься список останніх пісень;

– панель навігації, для мобільних пристроїв вона має бути у «бургер меню» для заощадження місця на екрані [15];

– сторінка пошуку, на якій відображатимуться результати пошуку;

– панель пошуку, дозволяє шукати пісні за назвою або виконавців;

– інтерфейс відтворення представлений у вигляді нижньої панелі плеєра з кнопками управління стану відтворення, кнопками «вперед» і «назад», перемішати чергу відтворення та поставити на повтор, також має писати назву і виконавця пісні, а поруч обкладинка, має бути регулятор гучності звуку;

– сторінка з плейлистами;

– форма завантаження пісень;

– сторінка профілю користувача.

Усі інтерфейси повинні бути реалізовані за принципом адаптивного дизайну. Усі компоненти мають коректно масштабуватися на різних екранах, а у мобільній версії частина функцій переходить у згортаний режим, наприклад, як раніше було згадано, «бургер меню» і плеєр, який може розгортатися для повного управління відтворенням.

Щодо інформаційних потоків, система має функціонувати за принципом «клієнт-сервер». Користувач відправляє запит на сервер, наприклад, пошук

пісні, де він обробляється, і готовий результат відправляється клієнтові, який відобразить результати пошуку. Усі обміни відбуваються через RESTful API із застосуванням безпечного протоколу HTTPS [16].

2.2 Вибір засобів, методів і алгоритмів вирішення поставленого завдання

Для того аби втілити в реальність всі цілі, які були поставлені перед розробкою, потрібно обрати інструменти, які в цьому допоможуть. Для створення сучасного вебдодатку існує величезна кількість інструментів та технологій, які забезпечують як клієнтську, так і серверну частину, зберігання даних, обробку запитів, автентифікацію користувачів та інші важливі функції. На початковому етапі розробки було проаналізовані найбільш поширені рішення, зокрема фреймворки для frontend частини (табл. 2.1), засоби створення серверної логіки та інструменти для автентифікації.

Таблиця 2.1 Аналіз поширених рішень для розробки вебдодатків

Критерій	React + Next.js	Angular	Vue.js
1	2	3	4
Синтаксис	JSX	HTML-шаблони і TypeScript	HTML-шаблони + Composition API
Підхід до структури	Компонентний, гнучкий	MVVM, сувора структура з модулями, сервісами, пайпами	Компонентний
Підтримка TypeScript	Добра, через Next.js	Вбудована	Недопрацьована
Початок роботи	Простий, низький поріг входу	Складний старт, потребує вивчення великого обсягу документації	Простий старт, проте може буте плутаним через нестабільність версій
Екосистема	Дуже велика, гнучка (Redux, React Query, Next.js тощо)	Велика, все «з коробки» (Forms, Router, CLI, DI, тестування)	Помірна, часто обмежена, нестабільна підтримка в нових версіях

Продовження таблиці 2.1

1	2	3	4
Документація	Якісна, доступна українською	Повна	Мінімальна
Спільнота, підтримка	Спільнота дуже величезна, легко знайти допомогу	Велике	Спільнота менше, складно гуглити проблеми
Масштабованість	Висока при правильній архітектурі	Висока, добре масштабований завдяки DI, модулям	Посередня, залежить від досвіду та структурування проєкту
Недоліки	Не включає багато чого «з коробки», потреба підбирати бібліотеки вручну	Висока складність, вимогливість до знань, великий поріг входу	Часті проблеми з сумісністю версій, слабка підтримка TypeScript, обмежена документація

Остаточний вибір був зроблений на користь React у поєднанні з Next.js, що зумовлено не лише технічними перевагами цієї зв'язки, а й наявним практичним досвідом в роботі з цією технологією. Раніше React та Next.js вже використовувались у кількох власних проєктах, що дозволило швидко зорієнтуватися в структурі додатку, уникати типових помилок, ефективно використовувати вже знайомі інструменти. Завдяки цьому вдалося зосередитись на реалізації основних функціональних можливостях та UI без додаткових витрат часу на освоєння нового фреймворку.

У процесі розробки потрібно обрати ефективний інструмент для стилізування інтерфейсу користувача. Від правильного вибору залежить швидкість розробки, зручність коду, адаптивність дизайну та загальний користувацький досвід. Існує велика кількість рішень для стилізації: від класичного CSS до його сучасних фреймворків. Тому доцільно зробити порівняння сучасних і популярних інструментів, щоб обґрунтувати вибір того, який найкраще підходить для реалізації всіх цілей та дизайну певних компонентів.

Чистий CSS при використанні у full-stack розробці дає повний контроль над стилями та не вимагає сторонніх бібліотек, що спрощує структуру проєкту

та зменшує його розмір, це особливо корисно для невеликих або індивідуальних стилізованих проектів. Проте, у великих додатках це ускладнює підтримку, уповільнює розробку через велику кількість шаблонного коду та ускладнює повторне використання стилів і реалізацію тем. Тому чистий CSS поступається сучасним фреймворкам за зручністю масштабування.

Tailwind CSS – це CSS-фреймворк, яка використовує підхід «utility-first», у якому стилі застосовуються безпосередньо у класах HTML/JSX елементів [17]. Перевагами Tailwind CSS є: швидка розробка UI завдяки готовим утилітам, повна кастомізація через файл конфігурації, висока продуктивність (не використані класи видаляються на етапі продакшен-збірки), вбудована підтримка адаптивності, dark mode, hover/focus ефектів, чудова інтеграція з React та Next.js. Недоліки є, але вони не сильно значні, адже з часом до цього звикаєш, а саме до великої кількості класів в коді і ще потрібен час до звикання до утилітарного підходу.

Bootstrap – це відкритий та безкоштовний HTML, CSS та JS фреймворк для швидкої верстки адаптивних дизайнів сайтів та вебдодатків. Він простий у використанні, є можливість налаштувати під свій проект, усі компоненти виконані в одному стилі, сумісний з усіма сучасними браузерами. Проте, Bootstrap має свої недоліки, серед яких більшість розмір кінцевих css та js-файлів проекту і складність використання для верстки сайтів з унікальним дизайном [18].

Bulma – це сучасний CSS-фреймворк, який дозволяє ефективно створювати адаптивні та семантично чисті інтерфейси. Як зазначено в блозі Pieces [19], основна мета Bulma – забезпечити розробникам простий, гнучкий інструмент для побудови інтерфейсів без необхідності писати багато власного CSS-коду. До переваг належать простота використання, модульність та інтуїтивна система модифікаторів. Однак Bulma має і недоліки: він не включає JavaScript-компонентів, що потребує додаткової інтеграції для динамічних елементів, а також може мати проблеми сумісності з застарілими браузерами.

Material UI (MUI) – це популярна бібліотека компонентів для React, яка реалізує принципи дизайну Material Design від Google [20]. Перевагами цієї бібліотеки є швидкий старт (велика кількість вже готових елементів, таких як кнопки, форми, діалоги тощо), гнучка кастомізація завдяки підтримки темізації, інтеграція з React. Недоліки: значний розмір бібліотеки, через велику кількість вже заготовлених елементів дизайну, складний для новачків, якщо проєкт не передбачає використання принципів Material Design, адаптація MUI під інший стиль може вимагати додаткових зусиль [21].

Chakra UI – це бібліотека компонентів для React, яка дозволяє швидко створювати адаптивні, доступні та стилізовані інтерфейси за допомогою JSX-пропсів. Вона має просту структуру, підтримує темну/світлу тему та відповідає стандартам доступності [22]. Серед переваг – зручність використання, гнучка система стилізації та хороша документація [23]. До недоліків можна віднести обмежену кастомізацію деяких компонентів і залежність від Emotion, що може збільшити розмір проєкту [24], менше контролю, ніж у Tailwind.

Порівняльний аналіз показав, що Tailwind CSS є оптимальним вибором для розробки кастомізованих інтерфейсів у поєднанні з React/Next.js. Цей фреймворк забезпечує високу швидкість верстки, адаптивність, а також чистий і контрольований дизайн без перевантаження зайвим CSS-кодом. У контексті створення музичного вебдодатку, де особливо важлива гнучкість стилізації, адаптація під мобільні пристрої та швидкість розробки, Tailwind став найкращим рішенням.

Zustand – бібліотека для керування глобальним станом у React-додатках. У порівнянні з Redux, вона не потребує зайвої конфігурації, middleware чи boilerplate-коду. У додатку Zustand використовується для:

– керування станом плеєра (поточний трек, статус відтворення, гучність тощо);

– зберігання даних про користувача після авторизації;

– синхронізації UI між різними компонентами.

У розробці сучасних вебдодатків існує багато технологій для створення бекенду, таких як Node.js, Django, Laravel, Firebase і Supabase. Вибір залежить від типу проєкту, вимог до швидкості розробки, масштабованості та структури даних.

Node.js є популярним середовищем виконання JavaScript на сервері, яке використовує подієво-орієнтовану, неблокуючу модель вводу/виводу. Це забезпечує високу продуктивність і дозволяє обробляти велику кількість одночасних з'єднань без блокування основного потоку виконання. Використання однієї мови для фронтенду і бекенду спрощує розробку та підтримку коду. Водночас, через однопоточну природу Node.js він не підходить для обчислювально важких задач, які можуть блокувати цикл подій. Також варто враховувати ризики, пов'язані з безпекою сторонніх пакетів у великій екосистемі npm, що потребує ретельного управління залежностями і оновленнями [25].

Django – фреймворк на Python, який забезпечує багато готового функціоналу «з коробки»: ORM, автентифікацію, адміністративну панель. Це робить Django привабливим вибором для швидкої побудови CRUD-додатків. Проте інтеграція з JavaScript-орієнтованими фронтендами, такими як React або Next.js, часто ускладнює процес розробки [26].

Laravel – це популярний PHP-фреймворк, який вирізняється чистою та зрозумілою архітектурою на основі MVC. Він надає широкий набір вбудованих інструментів для швидкої розробки вебдодатків, зокрема маршрутизацію, автентифікацію та ORM для роботи з базою даних. Велика спільнота та детальна документація роблять Laravel зручним для розробників. Однак для новачків може бути складним у вивченні, а через багатофункціональність він іноді поступається в продуктивності легшим фреймворкам [27].

Supabase – це сучасна платформа Backend-as-a-Service (BaaS), яка базується на потужній реляційній базі даних PostgreSQL. Вона надає широкий спектр інструментів для роботи зі структурованими даними, включно з вбудованою автентифікацією користувачів, підтримкою REST і GraphQL API, а також real-time підписками для обробки подій у режимі реального часу. Завдяки

цим можливостям, Supabase дозволяє розробникам швидко та ефективно створювати надійний та масштабований бекенд без необхідності складного налаштування серверної інфраструктури [28]. Це особливо корисно для проєктів, які потребують швидкої інтеграції та адаптивності у розвитку функціоналу.

Supabase було обрано для розробки, як бекенд-сервіс, що забезпечує RESTful API, автентифікацію та роботу з базою даних PostgreSQL. Такий підхід дозволив зосередитись на логіці додатку без необхідності самостійно налаштовувати серверну інфраструктуру.

Axios – це бібліотека для виконання HTTP запитів. У проекті Axios використовується для взаємодії з Supabase API, а саме з отриманням списку пісень, додаванням і видаленням пісень в список вибраних, обробкою взаємодії та обробкою помилок з боку бекенду.

Розроблена інформаційна система має модульну архітектуру, що забезпечує чітке розділення функціональності, спрощує масштабування, а також підвищує зручність супроводу та подальшого розвитку. Кожен модуль відповідає за окремий аспект роботи системи – наприклад, автентифікацію, управління аудіофайлами, відображення інтерфейсу користувача тощо – та взаємодіє з іншими модулями через чітко визначені інтерфейси. Такий підхід дозволяє легко змінювати або оновлювати окремі частини системи без ризику порушення її цілісності.

Для кращого розуміння внутрішніх процесів системи було створено наочні візуалізації у вигляді діаграм, що демонструють логіку обміну даними та поведінку системи при виконанні основних дій. Зокрема, на діаграмі послідовності (рис. 2.4) показано типовий сценарій використання: процес автентифікації користувача, завантаження списку пісень із сервера, відтворення пісні та управління нею.

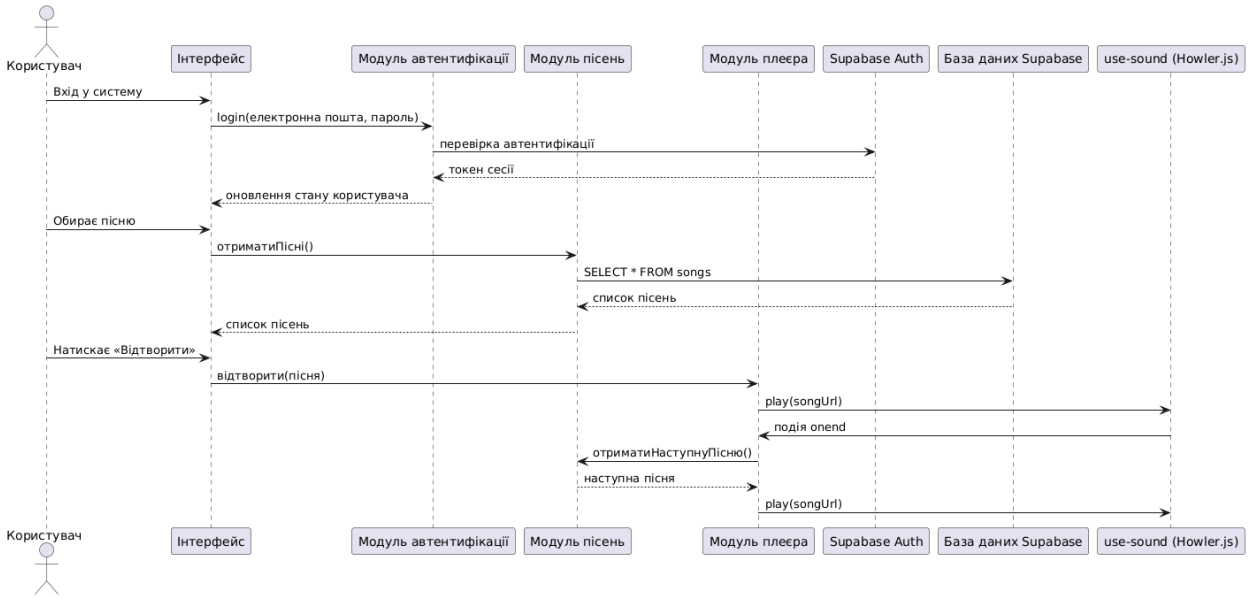

Рисунок 2.4 – Діаграма послідовності дій користувача при відтворенні пісні

Взаємодія починається з логіну користувача, який ініціює перевірку даних через Supabase Auth. Після отримання токена сесії система оновлює стан користувача. Далі користувач обирає пісню – додаток надсилає запит до модулю пісень, який отримує дані з бази даних Supabase. Коли користувач натискає кнопку «Відтворити», пісня передається у модуль плеєра, де викликається функція play з бібліотеки use-sound від Howler.js.

По завершенню пісні спрацьовує подія onend, після чого завантажується й автоматично починає відтворюватись наступна композиція.

Діаграма компонентів (рис. 2.5) демонструє загальну архітектуру клієнтської частини вебдодатку. Інтерфейс користувача взаємодіє з чотирма основними модулями: автентифікації, плеєра, пісень і пошуку. Ці модулі взаємодіють із серверною частиною де використовуються сервіси Supabase: Supabase Auth для автентифікації, база даних Supabase для збереження та пошуку інформації про пісні, а також бібліотека use-sound (на базі Howler.js), яка керує відтворенням аудіо.

Діаграма дозволяє побачити розподіл відповідальностей між клієнтом і сервером, а також логіку взаємодії між модулями застосунку та зовнішніми сервісами.

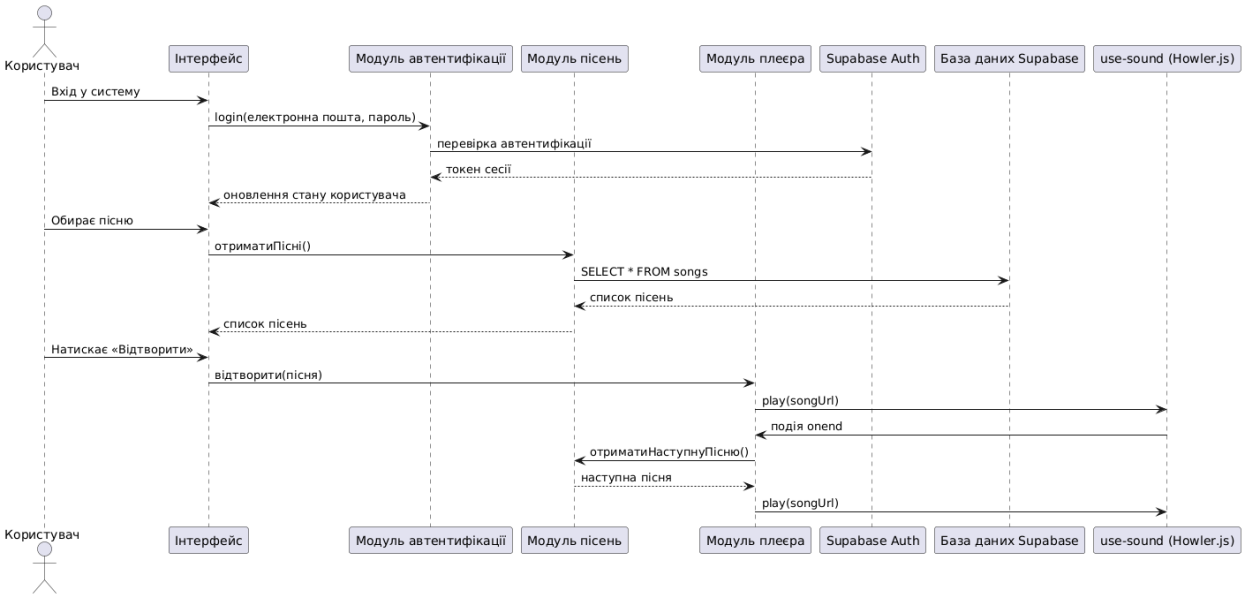

Рисунок 2.5 – Архітектура клієнтської частини

Висновки до розділу 2

У цьому розділі було проведено аналіз цілей та вимог до розроблюваної інформаційної системи, що дозволило визначити ключові критерії вибору інструментів, методів і алгоритмів для її реалізації.

Здійснено огляд існуючих засобів програмної розробки з урахуванням поставлених задач, а також їх можливостей, переваг і обмежень. На основі проведеного аналізу обґрунтовано вибір конкретних технологій, які найкраще відповідають функціональним вимогам системи та забезпечують ефективне вирішення прикладних задач.

Надано опис обраних інструментів і методів, які планується використати в процесі розробки. Також визначено структуру алгоритмічного забезпечення та типи діаграм, що використовуються для моделювання системи.

РОЗДІЛ 3
РОЗРОБКА ВЕБДОДАТКУ МУЗИЧНОГО СТРИМІНГОВОГО СЕРВІСУ

3.1 Практична реалізація об’єкта проєктування

Розробка розпочалася з створення у Visual Studio Code папки проекту за допомогою команди npx create-next-app@latest. TailwindCSS встановлюється при створенні і відразу є в проекті. Потім необхідно встановити всі необхідні залежності, налаштувати файли конфігурації, такі як: tailwind.config.js, tsconfig.json, .env, а також створити базову структуру папок для майбутнього застосунку.

Перед початком розробки потрібно створити акаунт і проєкт у Supabase. Після цього Supabase автоматично розгортає сервер PostgreSQL з базовими інструментами, включно з SQL-редактором. Далі створюються необхідні таблиці. Наприклад, таблиця songs (ліст. 3.1), яка зберігає завантажені пісні, містить такі поля: id (UUID, первинний ключ), user_id (зовнішній ключ на auth.users), title, author, image_path, song_path (усі типу text) і created_at.

Лістинг 3.1 – Таблиця songs

```
create table songs (
  id uuid primary key default gen_random_uuid(),
  user_id uuid references auth.users on delete cascade,
  title text,
  author text,
  image_path text,
  song_path text,
  created_at timestamp with time zone default timezone('utc', now())
);
```

кінець лістингу 3.1

У Supabase було налаштувано авторизацію через email та OAuth. У проєкті Next.js було вказано необхідні змінні середовища з параметрами Supabase. Для зберігання пісень і зображень було створено bucket у Storage. Щоб захистити дані, увімкнено RLS і додано політики, які дозволяють користувачу бачити лише власні записи. Після цих налаштувань, Supabase був готовий до інтеграції з

фронтендом. При створенні дається унікальний ідентифікатор проекту, публічний та секретний API ключі, які потім переносяться в .env.local.

Розробка сайту розпочинається зі створення статичної версії інтерфейсу. На цьому етапі повторюється зовнішній вигляд макету зроблений в Figma (рис. 3.1) – верстаються основні блоки, розміщується контент, підбираються шрифти, кольори та інші стилі відповідно до дизайну. По ходу розробки, було замінено шрифти і логотип, але загальний вигляд залишився яким і був.

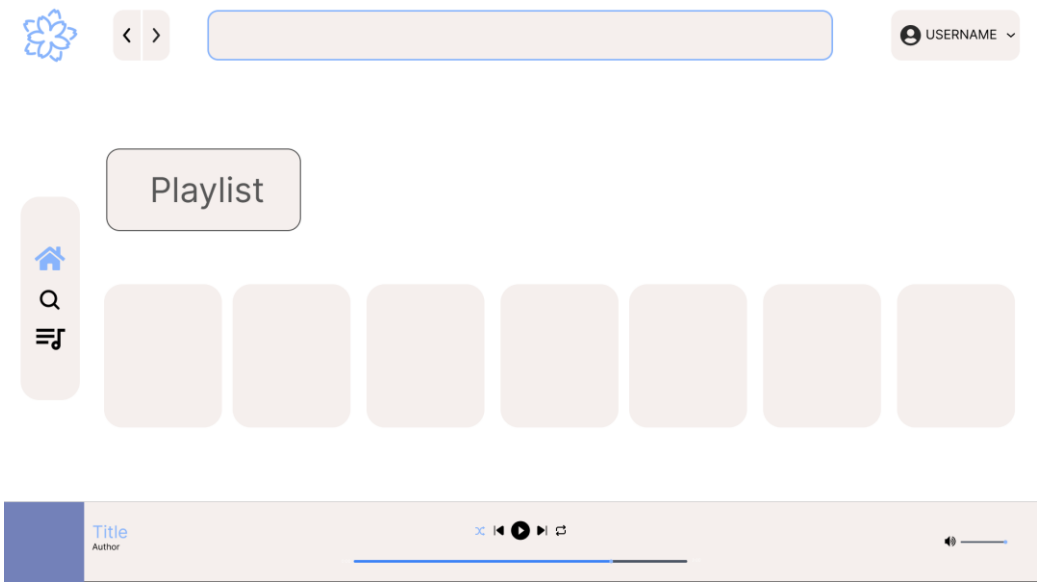

Рисунок 3.1 – Макет інтерфейсу вебдодатку

Для початку було створено папку Components, в подальшому в якій будуть знаходиться всі компоненти додатку, аби структура проекту була «чистою» і з нею було легко взаємодіяти, а потім було створено компонент Sidebar. Це панель навігації по сайту, яка знаходиться на макеті зліва. Він містить в собі три кнопки: іконка будинку відповідає за перенаправлення користувача на домашню сторінку, якщо він не знаходиться на ній, кнопка пошуку перенаправляє на сторінку, на якій користувач може шукати пісні за їх назвою чи автором, і кнопка плейлистів, яка перенаправляє на відповідну сторінку.

Компонент Sidebar використовує хук usePathname з Next.js для визначення поточного шляху. На основі цього шляху формується масив routes, який містить

об’єкти з інформацією про іконку, мітку, шлях та активність кожного пункту меню (ліст. 3.2).

Лістинг 3.2 – Формування масиву навігації

```
const pathname = usePathname();

const routes = useMemo(
  () => [
    { icon: HiHome, label: "Home", href: "/", active: pathname === 
"/" },
    { icon: HiSearch, label: "Search", href: "/search", active: 
pathname === "/search" },
    { icon: TbPlaylist, label: "Playlists", href: "/playlists", 
active: pathname === "/playlists" },
  ],
  [pathname]
);
```

кінець лістингу 3.2

У розмітці Sidebar складається з двох частин: сама бокова панель (прихована на малих екранах) та основний блок контенту (<main>), розташований праворуч (ліст. 3.3).

Лістинг 3.3 – Розмітка компонента Sidebar

```
<div className="flex h-full">
  <div className="hidden md:flex flex-col w-[100px]">
    <Logo />
    <div className="flex flex-col">
      {routes.map((item) => (
        <SidebarItem key={item.label} {...item} />
      ))}
    </div>
  </div>
  <main className="flex-1">{children}</main>
</div>
```

кінець лістингу 3.3

Компонент SidebarItem відповідає за рендер одного пункту меню. Він приймає іконку, адресу, мітку та активний стан. Активний пункт підсвічується кольором --accent-color, інші мають колір --inactive-text. Усе стилізується через

tailwind-merge (ліст. 3.4). У результаті реалізовано просту й зручну бокову панель, яка дозволяє користувачеві переміщатися між сторінками без перезавантаження основного контенту.

Лістинг 3.4 – SidebarItem

```
const SidebarItem = ({ icon: Icon, href, active }) => {
  return (
    <Link
      href={href}
      className={twMerge(
        "flex items-center gap-x-4 text-md font-medium",
        active ? "text-[var(--accent-color)]" : "text-[var(--
inactive-text)]"
      )}
    >
      <Icon size={40} />
    </Link>
  );
};
```

кінець лістингу 3.4

Supabase був використаний для автентифікації, оскільки його легко налаштувати, а також він підтримує вхід через email/пароль і популярні соцмережі, як-от Google і GitHub (рис. 3.2). В React було реалізовано модальне вікно для входу з компонентом <Auth>, який автоматично показує кнопки соцмереж і керує сесією.

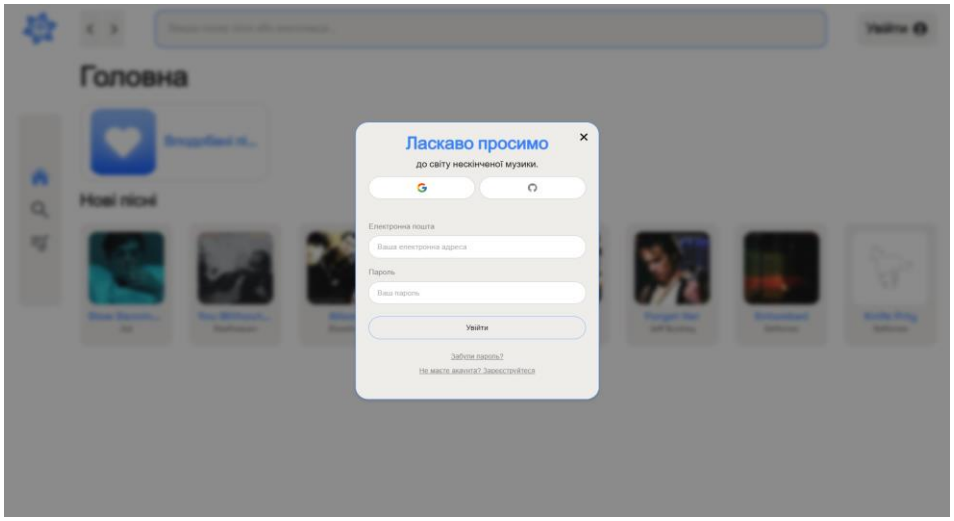

Рисунок 3.2 – Модальне вікно авторизації

У коді через useSessionContext() йде відстеження за сесією: коли користувач увійшов, перевіряється email, модальне вікно закривається і оновлюється сторінка. Модальне вікно відкривається/закривається за станом з кастомного хука useAuthModal(). Завдяки авторизації через Supabase, було забезпечену безпечну і приємну на вигляд авторизацію (ліст. 3.5).

Лістинг 3.5 – Авторизація в компоненті AuthModal

```
const { session } = useSessionContext();
const { onClose, isOpen } = useAuthModal();
const supabaseClient = useSupabaseClient();
const router = useRouter();
useEffect(() => {
  if (session) {
    if (!session.user?.email) {
      console.error("No email received from Spotify! Supabase
requires an email.");
    }
    onClose();
    router.refresh();
  }
}, [session, router, onClose]);

return (
  <Modal isOpen={isOpen} onChange={(open) => !open && onClose()}>
    <Auth
      supabaseClient={supabaseClient}
      providers={["google", "github"]}
      theme="dark"
      socialLayout="horizontal"
      appearance={{
        theme: ThemeSupa,
        variables: {
          default: {
            colors: { brand: "#131313", brandAccent: "#88b4fc" },
            radii: { inputBorderRadius: "20px", borderRadiusButton:
"40px", buttonBorderRadius: "20px" },
          },
        },
      }}
    />
  </Modal>
);
```

кінець лістингу 3.5

Загалом, реалізація пошуку – це один із найважливіших UX-елементів мого застосунку. При вебдодатку було реалізувано функціонал пошуку треків за допомогою бази даних Supabase та можливостей Next.js. Основна мета полягала в тому, щоб користувач міг швидко знаходити пісні за назвою або автором, отримуючи при цьому зручний та інтуїтивно зрозумілий інтерфейс.

Робота розпочалася з реалізації серверної частини. Було створено асинхронну функцію getSongsByQuery, яка звертається до таблиці songs у Supabase. Вона приймає рядок запиту та фільтрує дані за допомогою оператора ilike, що дозволяє здійснювати нечіткий пошук одразу по двох колонках – title та author (ліст. 3.6).

Лістинг 3.6 – Використання даних із таблички songs

```
const { data, error } = await supabase
  .from("songs")
  .select("*")
  .or(`title.ilike.%${query}%,author.ilike.%${query}%`)
  .order("created_at", { ascending: false });
```

кінець лістингу 3.6

На клієнті було створено компонент SearchInput, який відповідає за введення пошукового запиту. При натисканні клавіші Enter, відбувається перехід на сторінку /search з параметром query (ліст. 3.7).

Лістинг 3.7 – Фрагмент коду панелі пошуку

```
const handleSearchSubmit = () => {
  if (value.trim()) {
    const url = qs.stringifyUrl({ url: "/search", query: { query:
value } });
    router.push(url);
  }
};
```

кінець лістингу 3.7

Результати відображаються через компонент SearchContent, який рендерить список пісень у вигляді адаптивної сітки, яка відображається на сторінці пошуку (ліст. 3.8).

Лістинг 3.8 – Рендер пісень на сторінці

```
{songs.map((song) => (
  <div key={song.id}>
    <SearchItem onClick={(id: string) => onPlay(id)} data={song} />
  </div>
))}
```

кінець лістингу 3.8

У компоненті SearchItem реалізовано hover-ефект з кнопкою відтворення та кнопкою лайку. Це дозволяє взаємодіяти з кожною піснею без додаткових кліків. Результат пошуку зображено на рисунку 3.3.

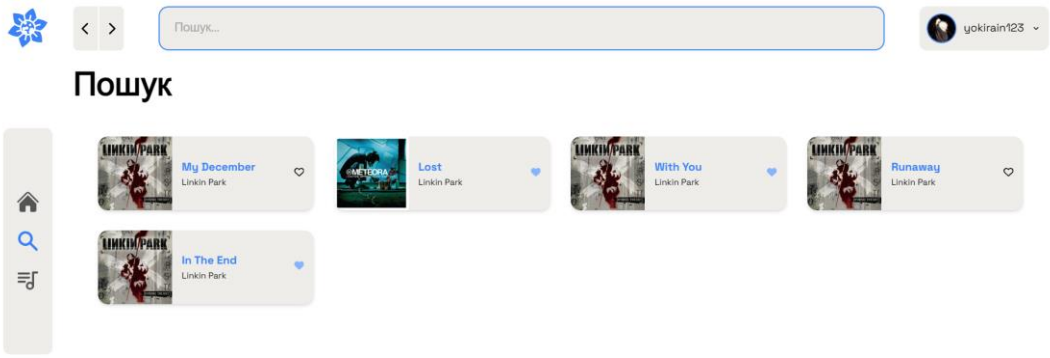

Рисунок 3.3 – Результат на сторінці пошуку

Перед тим, як було почато розробку, одною з головних цілей для себе було виділено реалізацію відтворення пісень. Це має бути повноцінний програвач, не просто кнопка «play» чи «pause», а інтерактивний, динамічний компонент з можливістю перемикати пісні, контролювати гучність, з відображенням прогресу відтворення і інформації про пісню, а також функціоналом перемішки або повторення пісень.

Роботу над реалізацією плеєра було розпочато зі створення хука usePlayer (ліст. 3.9). Його основне завдання – забезпечити глобальне сховище стану за допомогою бібліотеки Zustand, яка є простою, легкою та продуктивною. Під час побудови функціоналу плеєра виникла потреба в забезпечені доступу із будь-якої частини додатку до інформації про те, яка пісня є активною в даний момент, який список треків зараз програється, а також до прямого посилання на аудіофайл. Щоб уникнути складної передачі стану через багато рівнів вкладених компонентів [29], я виніс цю логіку у глобальний стан за допомогою Zustand.

Лістинг 3.9 – Константа usePlayer

```
import { create } from "zustand";

interface PlayerStore {
  ids: string[];
  activeId?: string;
  songUrl: string;
  setId: (id: string) => void;
  setIds: (ids: string[]) => void;
  reset: () => void;
  setSongUrl: (url: string) => void;
}

const usePlayer = create<PlayerStore>((set) => ({
  ids: [],
  activeId: undefined,
  songUrl: "",
  setId: (id: string) => set({ activeId: id }),
  setIds: (ids: string[]) => set({ ids: ids }),
  reset: () => set({ ids: [], activeId: undefined, songUrl: "" }),
  setSongUrl: (url: string) => set({ songUrl: url }),
}));

export default usePlayer;
```

кінець лістингу 3.9

Усередині хука usePlayer було визначено інтерфейс PlayerStore, який задає структуру мого стану. У ньому передбачено три основні поля: масив ids, де зберігаються ідентифікатори треків у поточному плейлисті; змінну activeId, яка вказує на ідентифікатор треку, що зараз відтворюється; та songUrl, який містить

пряме посилання на mp3-файл із Supabase. Крім цього, інтерфейс містить чотири методи, необхідні для роботи зі станом: setId, setIds, setSongUrl та reset.

Функція setId використовується тоді, коли користувач натискає на певну пісню – вона встановлює відповідний ідентифікатор як активний. setIds дозволяє зберегти список усіх доступних пісень, наприклад, коли користувач відкриває певний альбом або отримує результат пошуку – це забезпечує можливість перемикання треків у межах цього списку. За допомогою setSongUrl зберігається URL-адреса аудіофайлу, яка динамічно генерується з Supabase і потім передається у плеєр. Нарешті, метод reset дозволяє повністю очистити стан, наприклад, коли відтворення припинено або користувач залишає сторінку.

Таким чином, хук usePlayer виконує роль централізованого сховища для стану аудіоплеєра. Він дозволяє зручно зберігати та змінювати інформацію про поточну пісню, список треків і посилання на аудіофайл. Це значно спрощує керування плеєром у всьому додатку та дозволяє уникнути зайвої передачі даних через компоненти.

Компонент Player, який відповідає за показ плеєра знизу екрану, з’являється лише тоді, коли користувач обирає якусь пісню і в глобальному стані, який був описаний вище буде присутній активний ідентифікатор і буде успішно завантажено URL пісні. Спочатку компонент отримує активний songId з хука usePlayer, після чого за допомогою action функції useGetSongById (ліст. 3.10) знаходить відповідну пісню, а далі викликає асинхронну функцію loadSongUrl, щоб отримати посилання для відтворення.

Лістинг 3.10 – Компонент useGetSongById

```
"use client";

import { Song } from "@/types"
import { useSessionContext } from "@supabase/auth-helpers-react"
import { useEffect, useMemo, useState } from "react"
import toast from "react-hot-toast"

const useGetSongById = (id?: string) => {
    const [isLoading, setIsLoading] = useState(false)
    const [song, setSong] = useState<Song | undefined>(undefined)
```

```
    const {supabaseClient} = useSessionContext()

    useEffect(() => {
        if (!id) {
            return
        }

        setIsLoading(true)


        const fetchSong = async () => {
            const {data, error} = await supabaseClient
            .from('songs')
            .select('*')
            .eq('id', id)
            .single()

            if (error) {
                setIsLoading(false)
                return toast.error(error.message)
            }
            setSong(data as Song)
            setIsLoading(false)
        }
        fetchSong()
    }, [id, supabaseClient])
    return useMemo(() => ({
        isLoading,
        song
    }), [isLoading, song])
}

export default useGetSongById
```

кінець лістингу 3.10

Також важливо описати як працює loadSongUrl, зображена на лістингу 3.11. Функція loadSongUrl отримує публічну URL-адресу пісні з Supabase Storage. Вона приймає об’єкт song, створює клієнт Supabase, і через метод getPublicUrl повертає посилання на аудіофайл, який зберігається в бакеті «songs». Якщо пісня не передана, повертає порожній рядок.

Лістинг 3.11 – Константа loadSongUrl()

```
import { Song } from "@/types";
import { createClientComponentClient } from "@supabase/auth-helpers-
nextjs";
```

```
const loadSongUrl = async (song: Song): Promise<string> => {
  const supabase = createClientComponentClient();

  if (!song) return "";

  const { data: songData } = supabase.storage
    .from("songs")
    .getPublicUrl(song.song_path);

  return songData.publicUrl;
};


export default loadSongUrl;
```

кінець лістингу 3.11

Отримання посилання пісні відображено в першому useEffect, який реагує на зміну пісні в лістингу 3.12.

Лістинг 3.12 – Використання useEffect

```
useEffect(() => {

  if (song) {
    const fetchSongUrl = async () => {
      const url = await loadSongUrl(song);
      setSongUrl(url);
    };

    fetchSongUrl();
  }

}, [song]);
```

кінець лістингу 3.12

Другий useEffect відслідковує чи пісня і її URL доступні, і лише тоді вмикає видимість плеєра. Якщо хоча б щось із цього відсутнє, компонент нічого не рендерить (ліст. 3.13).

Лістинг 3.13 – Хук useEffect

```
useEffect(() => {

  setIsVisible(!!(song && songUrl));
```

```
}, [song, songUrl]);

if (!isVisible || !song || !songUrl) return null;
```

кінець лістингу 3.13

На більших екранах компонент Player завжди відображається у вигляді вузької панелі, закріпленої в нижній частині сторінки. Вона показує інтерфейс керування відтворенням через компонент PlayerContent і не має режиму розгортання – плеєр завжди доступний у фіксованому вигляді.

На менших екранах (мобільних пристроях) реалізовано два режими: компактний і розгорнутий. За замовчуванням плеєр відображається у компактному вигляді – це вузький блок з назвою треку й виконавцем, який рендериться через компонент PlayerItem. При натисканні на цей блок плеєр розгортається на весь екран, показуючи повний інтерфейс керування відтворенням через PlayerContent. Перемикання між цими режимами здійснюється за допомогою стану isExpanded (ліст. 3.14).

Лістинг 3.14 – Розгортання плеєра на всю сторінку

```
<div className={`transition-all duration-300 overflow-hidden ${
  isExpanded ? "max-h-screen" : "max-h-0"
} md:max-h-[100px] md:overflow-visible`}>
```

кінець лістингу 3.14

Таким чином, компонент адаптується до типу пристрою: на більших екранах завжди показується у згорнутому, але повністю функціональному вигляді, а на менших – дозволяє користувачу розгортати або згортати плеєр залежно від потреб.

Компонент PlayerContent було створено як основну функціональну частину плеєра для музичного сервісу. Метою було реалізувати повний контроль над програванням: перемикання треків, пауза, регулювання гучності, підтримка режимів повтору і перемішки, відображення прогресу, а також підтримка мобільної адаптації.

Так як, вже було реалізовано відтворення пісні при виборі її, то на цьому етапі потрібно було, аби при натисканні кнопки Play/Pause трек відповідно запускався або зупинявся. Було налаштувано use-sound так, щоб він автоматично оновлював стан setIsPlaying, коли відбувалося відтворення або пауза, а саму логіку кнопки винесено в окрему функцію handlePlay, яка перевіряє, чи є зараз відтворення (ліст. 3.15). Завдяки цьому користувач може керувати програванням треку інтуїтивно і без затримок.

Лістинг 3.15 – Відтворення або пауза треку

```
const [play, { pause, stop, sound }] = useSound(songUrl, {
  volume: volume / 100,
  format: ["mp3"],
  onplay: () => setIsPlaying(true),
  onpause: () => setIsPlaying(false),
});

const handlePlay = useCallback(() => {
  if (!sound) return;
  isPlaying ? pause() : play();
}, [isPlaying, pause, play, sound]);
```

кінець лістингу 3.15

Після цього було реалізовано можливість перемикання між піснями (ліст. 3.16). Є список player.ids, який містить всі доступні треки, та player.activeId – ідентифікатор поточного треку. Ці значення використано, щоб знайти індекс активного треку і визначити наступний або попередній. Як це працює? currentIndex визначає індекс активної пісні, nextSong обирає наступний елемент, якщо він є і якщо поточна пісня остання вона повертається до першої в списку, previousSong діє аналогічно, але програє попередній трек.

Лістинг 3.16 – Відтворення та пауза пісні

```
const onPlayNext = useCallback(() => {
  if (player.ids.length === 0) return;
  const currentIndex = player.ids.findIndex((id) => id ===
player.activeId);
  const nextSong = player.ids[currentIndex + 1] ?? player.ids[0];
  player.setId(nextSong);
```

```
}, [player]);

const onPlayPrevious = useCallback(() => {
  if (player.ids.length === 0) return;
  const currentIndex = player.ids.findIndex((id) => id ===
player.activeId);
       const previousSong = player.ids[currentIndex - 1] ??
player.ids[player.ids.length - 1];
       player.setId(previousSong);
     }, [player]);
```

кінець лістингу 3.16

Далі було реалізовано регулювання гучності, це зображено в лістингу 3.17. Спочатку було створено стан volume і зчитано його значення з localStorage, щоб зберігати налаштування між сесіями. Значення гучності зберігається як число (від 0 до 100). Якщо нічого не збережено – використовується значення 100 (максимальна гучність). Потім використано хук useEffect, щоб при зміні гучності зберігати її в localStorage і змінювати гучність поточного об’єкту sound. sound.volume(volume / 100) – бібліотека use-sound очікує значення від 0 до 1, тому значення volume і ділиться на 100.

Лістинг 3.17 – Регулювання гучності

```
const [volume, setVolume] = useState(
  Number(localStorage.getItem("volume")) || 100);

useEffect(() => {
  localStorage.setItem("volume", String(volume));
  if (sound) {
    sound.volume(volume / 100);
  }
}, [volume, sound]);
```

кінець лістингу 3.17

Після гучності, було розроблено відображення прогресу відтворення треку та можливість перемотки. Спочатку було створено useEffect, у якому використано requestAnimationFrame, щоб оновлювати currentTime і duration кожного кадру, поки пісня грає. sound.seek() повертає поточну позицію пісні в секундах, sound.duration() – це загальна тривалість пісні, це оновлює прогрес

плавно і в режимі реального часу. Також реалізувано можливість перемотки за допомогою повзунка: коли користувач рухає повзунок, sound.seek(newTime) переміщує відтворення на нову позицію і пісня, в свою чергу, відтворюється на тому моменті. Фрагмент коду зображений у лістингу 3.18.

Лістинг 3.18 – Прогрес пісні та перемотка

```
useEffect(() => {
  let animationFrameId: number;
  const updateProgress = () => {

    if (sound && isPlaying) {
      const time = sound.seek() as number;
      setCurrentTime(time);
      setDuration(sound.duration() || 0);
      animationFrameId = requestAnimationFrame(updateProgress);
    }
  };

  if (isPlaying) {
    animationFrameId = requestAnimationFrame(updateProgress);
  }

  return () => cancelAnimationFrame(animationFrameId);
}, [isPlaying, sound]);

const handleSeek = (e: React.ChangeEvent<HTMLInputElement>) => {
  const newTime = parseFloat(e.target.value);
  if (sound) {
    sound.seek(newTime);
    setCurrentTime(newTime);
  }
};
```

кінець лістингу 3.18

Після реалізації основних можливостей аудіоплеєра, було додано підтримку двох додаткових режимів повтору та перемішування. Режим повтору дозволяє відтворювати один і той самий трек безперервно. Для цього використано подію onend, яка спрацьовує після завершення треку. Якщо активовано режим «one», пісня одразу повторно запускається. Якщо ж повтор не увімкнений, плеєр переходить до наступного треку.

У режимі перемішування при переході до наступного треку випадковим чином обирається новий індекс треку, що дозволяє зробити прослуховування менш передбачуваним (ліст. 3.19).

Лістинг 3.19 – Відтворення пісень в режимі перемішування

```
const onPlayNext = useCallback(() => {
    if (player.ids.length === 0) return;
    if (isShuffling) {
      const randomIndex = Math.floor(Math.random() *
player.ids.length);
      player.setId(player.ids[randomIndex]);
    } else {
      const currentIndex = player.ids.findIndex((id) => id ===
player.activeId);
      const nextSong = player.ids[currentIndex + 1] ??
player.ids[0];
      player.setId(nextSong);
    }
  }, [player, isShuffling]);
```

кінець лістингу 3.19

На фінальному етапі було проведено адаптацію аудіоплеєра для коректного відображення та зручного користування на мобільних пристроях. Зокрема, створено окремий компонент MobilePlayerItem, який активується при розгортанні плеєра у повноекранному режимі (рис. 3.4).

Рисунок 3.4 – Розгорнутий плеєр на мобільних пристроях

Цей компонент забезпечує оптимізований інтерфейс, що враховує обмежений розмір екрану смартфонів та планшетів, покращуючи зручність використання та управління відтворенням музики у мобільному середовищі. Для привабливості було зроблено розмитий фон, який синхронізований з обкладинкою пісні. Також для кращого читання назви пісні і її виконавця на фоні розмитої обкладинки, був зроблений темніший фон.

У компоненті UploadModal (рис. 3.5) реалізована повна логіка завантаження пісні та обкладинки на сайт, з подальшим збереженням інформації в базу даних Supabase.

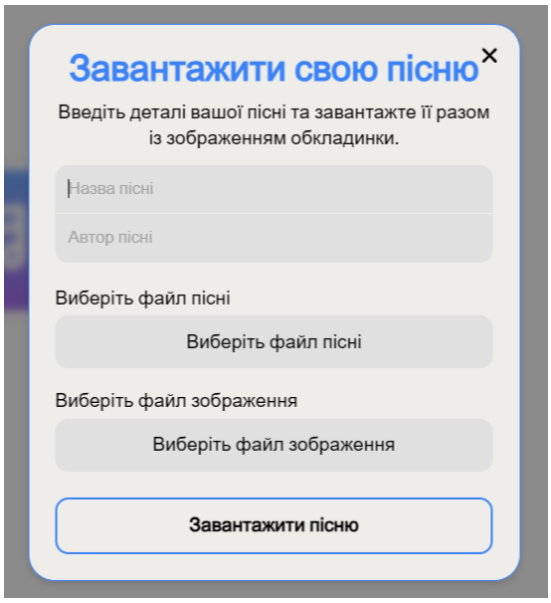

Рисунок 3.5 – Модальне вікно завантаження пісні

Ось як усе це працює: коли користувач відкриває модальне вікно та заповнює форму, він вводить назву пісні, автора, обирає файл пісні у форматі .mp3 і зображення (обкладинку пісні). Після натискання кнопки «Upload song» спрацьовує функція onSubmit (ліст. 3.20), яка обробляє ці дані.

Лістинг 3.20 – Логіка завантаження пісні

```
const onSubmit: SubmitHandler<FieldValues> = async (values) => {
    try {
      setIsLoading(true);
```

```
      const imageFile = values.image?.[0];
      const songFile = values.song?.[0];
      const uniqueID = uniqid();

      const { data: songData, error: songError } = await
supabaseClient.storage
        .from("songs").upload(`song-${values.title}-${uniqueID}`,
songFile, {cacheControl: "3600", upsert: false,});

      const { data: imageData, error: imageError } =
        await supabaseClient.storage
          .from("images").upload(`images-${values.title-
${uniqueID}`, imageFile, {cacheControl: "3600", upsert: false,});

      const { error: supabaseError } = await supabaseClient
        .from("songs").insert({user_id: user.id, title:
values.title, author: values.author, image_path: imageData.path,
song_path: songData.path});

      router.refresh();
      setIsLoading(false);
      toast.success("Song created!");
      reset();
      uploadModal.onClose();
    } catch (error) {
      toast.error("Something went wrong");
    } finally {
      setIsLoading(false);
    }
    setSelectedSong(null);
    setSelectedImage(null);
  };
```

кінець лістингу 3.20

Спочатку перевіряється, чи є обидва файли та чи авторизований користувач. Далі для кожного файлу створюється унікальна назва за допомогою uniqid, щоб уникнути конфліктів у сховищі. Потім обидва файли завантажуються в Supabase Storage – пісня в bucket songs, а зображення в bucket images. Якщо обидва завантаження проходять успішно, шлях до кожного з файлів (.path) зберігається разом із додатковою інформацією (назва, автор, ID користувача) в таблицю songs у базі даних Supabase (рис. 3.6).

id int8	created_at timestamptz	title text	song_path text	image_path text	author text	user_id uuid
34	2025-05-03 20:34:11.5839+00	Fantasy	song-Fantasy-ma8okox4	images-Fantasy-ma8okox4	DyE	00b4f9d6-64f4-4dd3-bfc5-fd6
33	2025-05-03 20:31:59.84132+00	Sanctuary	song-Sanctuary-ma8ohv69	images-Sanctuary-ma8ohv69	Joji	00b4f9d6-64f4-4dd3-bfc5-fd6
32	2025-05-03 20:25:44.131016+0	Sextape	song-Sextape-ma8o9v2h	images-Sextape-ma8o9v2h	Deftones	00b4f9d6-64f4-4dd3-bfc5-fd6
31	2025-05-03 20:25:20.273396+0	Show Me How	song-Show Me How-ma8o9fah	images-Show Me How-ma8o9fah	Man I Trust	00b4f9d6-64f4-4dd3-bfc5-fd6
30	2025-05-03 20:22:07.273163+0	California World	song-California World-ma8o58dt	images-California World-ma8o58dt	Lil Peep	00b4f9d6-64f4-4dd3-bfc5-fd6
29	2025-05-03 20:20:18.692995+0	Oceans 2001	song-Oceans 2001-ma8o2ujh	images-Oceans 2001-ma8o2ujh	Yung Lean	00b4f9d6-64f4-4dd3-bfc5-fd6
28	2025-05-03 20:19:11.716936+00	Good Days	song-Good Days-ma8o1cr6	images-Good Days-ma8o1cr6	SZA	00b4f9d6-64f4-4dd3-bfc5-fd6
27	2025-05-03 20:17:19.691603+0	Oil	song-Oil-ma8nz1ew	images-Oil-ma8nz1ew	Aircraft	00b4f9d6-64f4-4dd3-bfc5-fd6
26	2025-05-03 20:15:22.274541+0	Xtal	song-Xtal-ma8nwinh	images-Xtal-ma8nwinh	Aphex Twin	00b4f9d6-64f4-4dd3-bfc5-fd6
25	2025-05-03 20:13:50.89138+00	Celestica	song-Celestica-ma8nufs1	images-Celestica-ma8nufs1	Crystal Castles	00b4f9d6-64f4-4dd3-bfc5-fd6
24	2025-05-03 20:12:22.575146+0	Swimming Pools (Drank)	song-Swimming Pools (Drank)-ma8nsn7o	images-Swimming Pools (Drank)-ma8nsn7	Kendrick Lamar	00b4f9d6-64f4-4dd3-bfc5-fd6
23	2025-05-03 20:10:25.501612+0	1000 Blunts	song-1000 Blunts-ma8nq21m	images-1000 Blunts-ma8nq21m	$uicideboy$	00b4f9d6-64f4-4dd3-bfc5-fd6
22	2025-05-03 20:07:58.200183+0	Penthouse Shordy	song-Penthouse Shordy-ma8nmyns	images-Penthouse Shordy-ma8nmyns	Dom Corleo	00b4f9d6-64f4-4dd3-bfc5-fd6
21	2025-05-03 20:06:27.467599+0	Doomsday	song-Doomsday-ma8nkved	images-Doomsday-ma8nkved	MF DOOM	00b4f9d6-64f4-4dd3-bfc5-fd6
20	2025-05-03 20:05:10.275917+0	Long Time (Intro)	song-Long Time (Intro)-ma8njan6	images-Long Time (Intro)-ma8njan6	Playboi Carti	00b4f9d6-64f4-4dd3-bfc5-fd6
19	2025-05-03 20:02:12.018764+0	3005	song-3005-ma8nfi8x	images-3005-ma8nfi8x	Childish Gambino	00b4f9d6-64f4-4dd3-bfc5-fd6
18	2025-05-03 19:35:01.219095+0	Fashion Killa	song-Fashion Killa-ma8mglt7	images-Fashion Killa-ma8mglt7	A$AP Rocky	00b4f9d6-64f4-4dd3-bfc5-fd6
17	2025-05-03 19:32:06.3692+00	Lipgloss	song-Lipgloss-ma8mcvsq	images-Lipgloss-ma8mcvsq	Charlie xcx, cupcakKe	00b4f9d6-64f4-4dd3-bfc5-fd6
16	2025-01-27 19:31:00.392547+0	Be Quiet And Drive (Far Away)	song-Be Quiet And Drive (Far Away)-m6f	images-Be Quiet And Drive (Far Away)-m	Deftones	00b4f9d6-64f4-4dd3-bfc5-fd6
15	2024-11-08 17:38:25.821911+00	Resonance	song-Resonance-m390sxue	images-Resonance-m390sxue	Home	00b4f9d6-64f4-4dd3-bfc5-fd6
13	2024-10-17 17:06:46.928076+0	Timeless	song-Timeless-m2djzft3	images-Timeless-m2djzft3	Playboi Carti, The Weeknd	00b4f9d6-64f4-4dd3-bfc5-fd6

Рисунок 3.6 – Завантажені пісні в базі даних

Якщо все проходить без помилок, модальне вікно закривається, форма очищається, з’являється повідомлення про успіх і оновлюється сторінка. Якщо виникає помилка – вона відображається через toast.error.

Таким чином, після завантаження пісні її метадані потрапляють у таблицю, а сам файл – у хмарне сховище Supabase, і стає доступним для відтворення у застосунку.

Компонент LikeButton реалізує функціонал вподобання пісні. Цей компонент – кнопка, яка дає користувачу можливість вподобати або зняти вподобання з пісні. Коли сторінка завантажується, код перевіряє, чи користувач вже вподобав цю пісню, і показує відповідну іконку серця: заповнену, якщо пісня вподобана, або порожню, якщо ні.

Якщо пісня ще не вподобана, значення isLiked за замовчуванням дорівнює false, отже, коли користувач натискає кнопку вподобання, виконується взаємодія з базою даних в Supabase, а саме пісня і її ідентифікатор (song_id) та ідентифікатор користувача добавляється в таблицю liked_songs, і після цього значення isLiked буде true – що відобразиться в іконці серця. Якщо ж навпаки, користувач хоче прибрати пісню з вподобаних, то дані song_id і user_id

видаляються з таблиці. Про ці дії також зверху приходить сповіщення про те, що пісня тепер вподобана або ні. Як це працює зображено в лістингу 3.21.

Лістинг 3.21 – Добавлення і видалення пісні в таблиці liked_songs

```
try {
      if (isLiked) {
        const { error } = await supabaseClient
          .from("liked_songs")
          .delete()
          .eq("user_id", user.id)
          .eq("song_id", songId);

        if (error) {
          toast.error(error.message);
        } else {
          setIsLiked(false);
          toast.success("Song unliked!");
        }
      } else {
        const { error } = await supabaseClient
          .from("liked_songs")
          .insert({ song_id: songId, user_id: user.id });

        if (error) {
          toast.error(error.message);
        } else {
          setIsLiked(true);
          toast.success("Song liked!");
        }
      }
```

кінець лістингу 3.21

На рисунку 3.7 візуально зображено, як пісня із таблиці songs додається в таблицю liked_songs.

Рисунок 3.7 – Вподобана пісня в базі даних

Було реалізовано функціонал зміни теми інтерфейсу (рис. 3.8), який дозволяє користувачу переключатися між світлою та темною темами. Зміна теми відбувається динамічно без перезавантаження сторінки, завдяки використанню хуків React та бібліотеки next-themes. Інтерфейс керування зміною теми оформлено у вигляді кнопки з відповідною іконкою, що відображає поточний режим – сонце для світлої теми і місяць для темної. Це покращує зручність використання додатку і дозволяє адаптувати вигляд інтерфейсу під уподобання користувача.

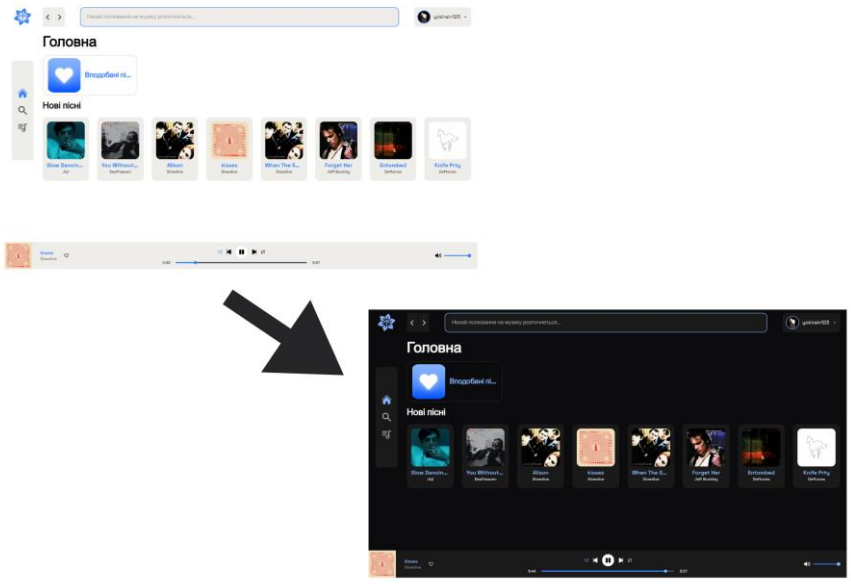

Рисунок 3.8 – Зміна теми

3.2 Тестування та налагодження інформаційно-комп'ютерної системи

Під час розробки застосунку було виявлено кілька технічних проблем, пов'язаних із рендерингом списків компонентів у React та використанням cookies у середовищі Next.js. У даному тексті розглянуто конкретні помилки, їхні причини та способи усунення.

Першою виявленою помилкою було попередження з боку React (ліст. 3.22).

Лістинг 3.22 – Помилка, яка виникала

```
Encountered two children with the same key, ``.
```

кінець лістингу 3.22

Це спостерігалося під час рендерингу списку компонентів за допомогою методу .map(), коли як ключ передавалося значення item.label. Проблема полягала в тому, що значення label у масиві routes було порожнім рядком для кожного елемента, що призводило до використання однакових ключів для всіх елементів списку. Такий підхід суперечить вимогам React щодо унікальності ключів, унаслідок чого фреймворк не міг коректно відстежувати зміни в списку під час оновлень.

Для усунення цієї помилки було модифіковано масив routes, кожному елементу якого було призначено унікальне значення в полі label – зокрема, «Home», «Search», «Playlists» (ліст. 3.23).

Лістинг 3.23 – Масив routes

```
const routes = useMemo(() => [
{ icon: HiHome, label: "Home", active: pathname === "/", href: "/"},
{ icon: HiSearch, label: "Search", active: pathname === "/search",
     href: "/search"},
{ icon: TbPlaylist, label: "Playlists", active: pathname ===
     "/playlists", href: "/playlists",},
    ], [pathname]);
```

кінець лістингу 3.23

Крім того, значення key={item.label} було замінено на key={item.href}, оскільки href для кожного елемента мав унікальне та стабільне значення, що відповідало вимогам React до ключів і було більш придатним для цієї ролі. У випадках, коли список не був динамічним, від використання методу .map() узагалі відмовилися, замість цього кожен компонент було прописано окремо з жорстко заданим значенням key (ліст. 3.24). Такий підхід повністю усунув проблему з ключами та ліквідував усі пов’язані з нею попередження й помилки під час рендерингу.

Лістинг 3.24 – Використання key

```
<ListItem
    key={`liked-${index}`}
    {...item}
    image="/images/like.png"
    name={"Liked"}
    href={"liked"}
/>
```

кінець лістингу 3.24

Наступна помилка виникла під час роботи з API cookies у середовищі Next.js. Було зроблено спробу отримати доступ до cookies за допомогою оператора await, що спричиняло помилку з повідомленням про необхідність використання await. Як згодом з’ясувалося, це повідомлення було некоректним – функція cookies() не є асинхронною, і відповідно, не потребує обгортання в await.

Додатково помилка полягала в тому, що виклик cookies() здійснювався у клієнтському компоненті, який містив директиву «use client». Це суперечило вимогам Next.js, оскільки функція cookies() доступна виключно в серверному контексті – у серверних компонентах або у route handlers/API routes.

Для виправлення ситуації було вилучено await перед викликом cookies() і залишено лише cookies: cookies(). Також виклик функції було переміщено до серверного компонента відповідно до вимог фреймворку. Крім того, було видалено await перед викликом createServerComponentClient(...), оскільки ця функція є синхронною.

У результаті, як показано в лістингу 3.25, логіка роботи з cookies почала функціонувати стабільно, а швидкодія сайту покращилася. Помилки більше не виникали.

Лістинг 3.25 – Виправлене використання cookies

```
const supabase = createServerComponentClient({
    cookies: cookies,
  });
```

кінець лістингу 3.25

Ще одна проблема, яка спричиняла попередження в React, була пов’язана з використанням JSX-фрагментів (<> </>) усередині методу .map() без зазначення атрибута key. Оскільки такі фрагменти також вважаються повноцінними елементами списку, React очікував наявність унікального ключа для кожного з них, інакше не міг ефективно виконувати оновлення DOM.

Для усунення цієї проблеми порожній фрагмент було замінено на React.Fragment з відповідним атрибутом key, або, за потреби, фрагмент було замінено на обгортку у вигляді елемента div з ключем. У випадках, коли використання фрагментів не було критично необхідним, їх було повністю вилучено з розмітки задля уникнення зайвої вкладеності.

У результаті проведеного аналізу та усунення помилок було досягнуто підвищення стабільності коду. Було поглиблено розуміння принципів роботи reconciler-алгоритму в React, а також забезпечено коректне розмежування клієнтської та серверної логіки у межах архітектури Next.js. Здійснені виправлення сприяли запобіганню потенційним помилкам у майбутньому та покращили якість програмного рішення.

3.3 Розробка бази даних

Під час проєктування бази даних для музичного стримінгового вебзастосунку було використано платформу Supabase як основний інструмент. Такий вибір був обумовлений наявністю зручного інтерфейсу для роботи з PostgreSQL, вбудованою системою автентифікації та гнучким механізмом керування доступом.

Проєктування бази даних розпочалося з визначення основних сутностей, необхідних для реалізації функціоналу застосунку. До таких сутностей було віднесено користувачів, пісні та вподобані треки. На основі цього було створено таблиці users, songs та liked_songs.

Таблиця users містила базову інформацію про користувачів, зокрема повне ім’я та аватар. Поле id виступало основним ідентифікатором і мало зв’язок із

системною таблицею auth.users, яка відповідає за автентифікацію користувачів. Візуалізовану структуру цієї схеми було подано на рисунку 3.9.

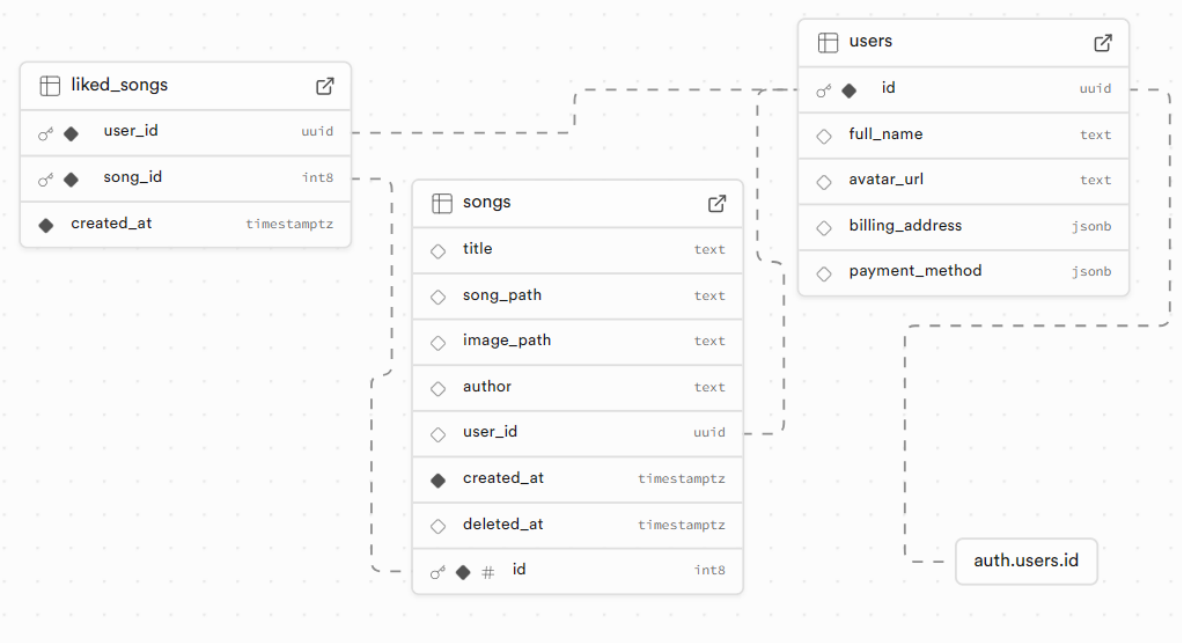

Рисунок 3.9 – Схема бази даних

Таблиця songs зберігає дані про пісні – назву, шлях до аудіофайлу, шлях до обкладинки, автора пісні, а також user_id, який вказує, хто завантажив цю пісню. Щоб реалізувати можливість «м'якого видалення», було додано поле deleted_at і створено спеціальний тригер trigger_soft_delete_songs (ліст. 3.26), який не видаляє запис фізично, а просто встановлює дату видалення. Це дозволяє зберігати історію і в разі потреби – відновити пісню або провести аналітику.

Лістинг 3.26 – Код тригеру для «м'ягкого видалення»

```
DROP TRIGGER IF EXISTS "trigger_soft_delete_songs" ON
"public"."songs";

CREATE TRIGGER "trigger_soft_delete_songs"

BEFORE DELETE
ON "public"."songs"

FOR EACH ROW
EXECUTE FUNCTION "public"."soft_delete_songs"();
```

кінець лістингу 3.26

Таблиця liked_songs реалізує зв’язок «багато до багатьох» між користувачами та піснями. Вона містить лише три поля – user_id, song_id та дату створення запису. Саме ця таблиця дозволяє зберігати, які пісні сподобалися конкретному користувачеві.

Окрему увагу було приділено налаштуванню політик доступу (RLS). Supabase дозволяє дуже гнучко визначати, хто і які дії може виконувати над даними. Було надано права ролі public на перегляд списку пісень, проте можливість прослуховування та додавання нових пісень було обмежено лише автентифікованими користувачами. Аналогічним чином для таблиці liked_songs було дозволено перегляд пісень усім користувачам, тоді як додавання або видалення пісень у вподобаннях було дозволено лише користувачам, які увійшли у свій акаунт. Для таблиці users було реалізовано політику, згідно з якою кожен користувач міг переглядати та оновлювати виключно власні персональні дані (рис. 3.10).

Загалом, Supabase дуже спростив процес розробки бази даних. Це дало можливість швидко створювати таблиці, тестувати запити, писати функції та тригери, і при цьому не потрібно було налаштовувати окремий сервер чи вручну конфігурувати доступ. Усе це дозволило зосередитись на логіці самого застосунку й швидко отримати працюючу систему з автентифікацією, захищеним доступом до даних і зручним API для взаємодії з фронтендом.

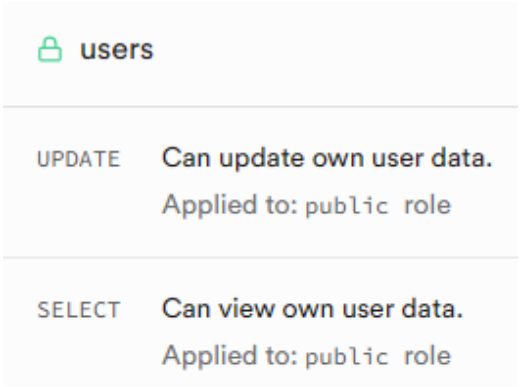

Рисунок 3.10 – RLS політики для таблиці users

3.4 Робота з API

У проєкті було реалізовано гнучку та ефективну систему API, яка взаємодіє з базою даних Supabase, забезпечуючи зручний доступ до даних про пісні з прив’язкою до конкретного користувача. Це API підтримує функції отримання, сортування та фільтрації інформації, що дає змогу формувати персоналізований контент відповідно до уподобань користувачів. Вся логіка була розроблена з урахуванням основних принципів безпеки, масштабованості та адаптивності, що дозволяє коректно працювати як на стороні сервера, так і на клієнті. Для серверного рендерингу в середовищі Next.js було використано клієнтську обгортку createServerComponentClient з бібліотеки @supabase/auth-helpers-nextjs, що забезпечує безпечне отримання інформації про сесію користувача без необхідності передачі токенів на фронтенд. Аутентифікація здійснюється через cookies і заголовки Next.js, що підвищує рівень захисту даних та покращує користувацький досвід.

Основним запитом до API є функція getSongs, яка виконує GET-запит до таблиці songs у базі Supabase. Вона отримує всі пісні, сортує їх за датою створення (поле created_at) у спадному порядку та повертає результат для відображення глобального списку треків. Цей метод не залежить від конкретного користувача і використовується для формування загального каталогу музики.

Для реалізації персоналізованого доступу до даних було створено окремий запит getSongsByUserId. Цей метод значно складніший: спочатку створюється клієнт Supabase, після чого виконується виклик supabase.auth.getUser() для визначення ідентифікатора поточного користувача, отриманого з сесійних cookies. Далі виконується запит SELECT до таблиці songs із фільтрацією за user_id, що дозволяє отримати лише ті пісні, які належать конкретному користувачу. Цей підхід гарантує, що користувач бачить лише власні вподобані композиції, що підвищує зручність і персоналізацію інтерфейсу.

Щодо взаємодії з API, в логах Supabase зберігається повна історія виконаних запитів (рис. 3.11). Наприклад, кожен GET-запит до /rest/v1/songs

напряму звертається до таблиці songs для отримання метаданих треків. Водночас запити до /storage/v1/object/public/songs/... спрямовані до об’єктного сховища Supabase, де зберігаються безпосередньо файли пісень. Таким чином, метадані музичних композицій – такі як назва, автор, URL файлу – зберігаються у базі даних, тоді як аудіофайли та обкладинки розміщуються у публічному сховищі (bucket). Логи API демонструють, що для відтворення пісень спочатку здійснюється запит до об’єктного сховища для отримання URL файлу, а вже потім окремо завантажується аудіо або зображення, що дозволяє оптимізувати передачу даних і підвищити швидкість роботи додатку.

Загалом, реалізована система API забезпечує надійну, безпечну і масштабовану платформу для роботи з музичними даними, що відповідає сучасним вимогам веброзробки та користувацьких сервісів.

Окремо в логах видно і OPTIONS запити – це CORS preflight’и [30], які браузер надсилає автоматично перед деякими типами GET/POST-запитів, щоб перевірити, чи дозволяється взаємодія. Вони важливі для безпечного обміну даними.

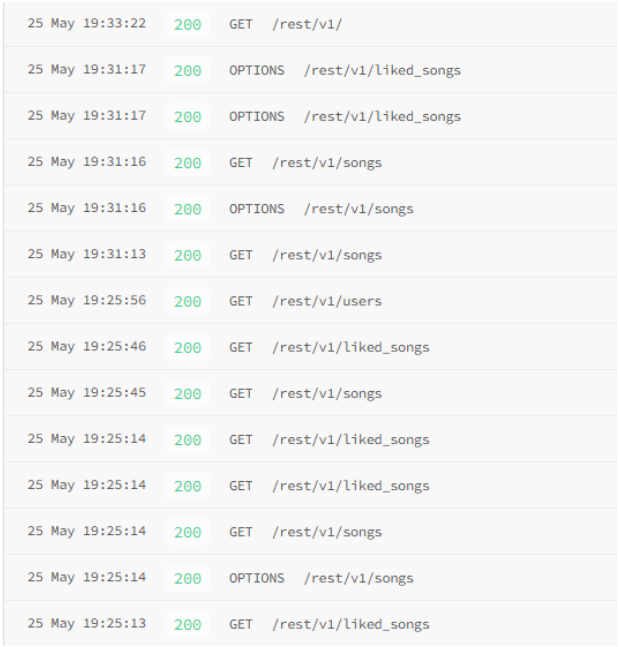

Рисунок 3.11 – Логи API викликів

Все це разом дає змогу створити інтерактивний та безпечний досвід для користувача: пісні завантажуються швидко, користувач бачить тільки свої дані, а авторизація на бекенді гарантує, що ніхто не отримає доступу до чужої інформації. Також передбачено обробку помилок на всіх етапах – якщо користувач не авторизований або якщо виникає проблема з запитом до бази, система повертає порожній масив і виводить помилку в консолі, не порушуючи загальну логіку застосунку.

3.5 Розгортання проекту на хостингах

Щоб розгорнути проект на звичайному хостингу, де немає підтримки Node.js, спочатку потрібно зробити його статичним. Для цього у файлі next.config.js потрібно додати такий рядок (ліст. 3.27).

Лістинг 3.27 – next.config.js

```
const nextConfig = {
  output: 'export',
};
module.exports = nextConfig;
```

кінець лістингу 3.27

Це дає можливість експортувати сайт у вигляді статичних файлів, які надалі можуть бути завантажені на сервер. Далі потрібно запустити білд і після того, як він запустився, потрібно ввести команду npm run export, яку перед цим було додано в конфіг. У результаті цього з’являється нова папка out, в якій знаходиться вже готова статична версія сайту з HTML, CSS, JavaScript, з всіма сторінками, стилями та скриптами. Тобто це вже готовий сайт, який можна відкривати у браузері без сервера. Після цього потрібно вибрати FTP-клієнт, підключитись до хостингу і ввести хост, логін, пароль і порт, які дає хостинг-провайдер. Як правило, всі файли сайту, знаходяться в створеній експортом папці out, їх потрібно закинути в папку public_html. Для цього просто потрібно

виділити файли і перетягнути їх в public_html на сервері. Коли всі файли будуть завантажені, сайт працюватиме і його можна відкривати за своїм доменом.

Єдине, що потрібно зробити з бекенд частиною після розгортання – це зайти в Supabase, і в розділі Authentication, і потім URL Configuration замінити посилання на сайт на те, де зараз розміщений сайт. Supabase автоматично оновить усі потрібні callback’и, наприклад, редіректи після логіну, реєстрації, відновлення паролю тощо).

Висновки до розділу 3

У третьому розділі представлено практичну реалізацію музичного стримінгового вебдодатку з інтерфейсом на React, Next.js і Tailwind CSS, функціоналом реєстрації, автентифікації, пошуку, завантаження та відтворення музики, з використанням Supabase з RLS-політиками, управлінням станом через Zustand, інтеграцією з API, адаптивним дизайном і розгортанням проєкту на хостингу, що дозволило створити повнофункціональний прототип музичного сервісу, який відповідає сучасним вимогам зручності, безпеки та масштабованості.

ВИСНОВКИ

У межах виконання даної кваліфікаційної роботи було досягнуто всіх поставлених цілей щодо розробки вебдодатку для прослуховування музики.

По-перше, було проведено аналіз наявних музичних сервісів – зокрема, Spotify, Apple Music та YouTube Music. Це дозволило виявити їхні сильні сторони, таких як широкий каталог, зручність інтерфейсу, рекомендаційні системи та слабкі місця, наприклад, складність у завантаженні власного контенту, обмеження без підписки тощо, що стало підґрунтям для формування вимог до майбутньої системи.

По-друге, було сформульовано функціональні та нефункціональні вимоги до вебдодатку. До функціональних належали: реєстрація та авторизація користувачів, пошук та відтворення треків, додавання власної музики, створення списків уподобань. До нефункціональних – адаптивність інтерфейсу, безпека зберігання даних, продуктивність та зручність користування.

Далі, було обґрунтовано вибір технологій і засобів розробки, серед яких: React, Next.js, Tailwind CSS для фронтенду, Supabase для бекенду та бази даних, а також Zustand для керування станом додатку. Вибір цих інструментів зумовлений їх гнучкістю, продуктивністю та активною підтримкою спільноти.

У процесі реалізації було створено архітектуру вебдодатку, спроєктовано структуру бази даних з урахуванням безпеки зберігання конфіденційної інформації, а також реалізовано повний функціонал вебзастосунку, відповідно до попередньо визначених вимог.

Успішно реалізовано базу даних, що забезпечує надійне та безпечне зберігання конфіденційної інформації, а також розроблено архітектуру вебдодатку, яка гарантує його стабільність і масштабованість. Повністю впроваджено функціонал вебзастосунку відповідно до визначених вимог, що забезпечує коректну роботу користувацького інтерфейсу та взаємодію з серверною частиною.

На завершальному етапі було проведено тестування системи, виявлено та усунуто низку технічних помилок, що дозволило забезпечити стабільну та надійну роботу додатку.

Таким чином, розроблений вебдодаток є сучасним, функціональним рішенням, що відповідає актуальним вимогам до музичних стримінгових сервісів і має потенціал до подальшого розвитку.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Willings S. Spotify gets patent for emotion-based music recommendations. MusicTech. URL: https://musictech.com/news/spotify-patent-detect-emotion-recommend-music/?utm_source=chatgpt.com (date of access: 01.06.2025).

2. Patently Apple. Apple's Back Tap Patent also explores using new Voice Recognition Technology to control a Future TV by Apple & more. URL: https://www.patentlyapple.com/2021/01/apples-back-tap-patent-also-explores-using-new-voice-recognition-technology-to-control-a-future-tv-by-apple-more.html. (date of access: 05.02.2025).

3. US8260778B2 - Mood based music recommendation method and system – Google Patents. Google Patents. URL: https://patents.google.com/patent/US8260778B2/en (date of access: 05.02.2025).

4. Як зрозуміти, що ти музичний пірат і крадеш у артиста. МУЗВАР. URL: https://muzvar.com.ua/iak-zrozumity-shcho-ty-muzychnyi-pirat-i-kradesh-zarobitok-uliublenoho-artysta-rozmova-z-muzychnym-marketolohom-leonidom-lastochkinym/ (дата звернення: 06.02.2025)

5. Bhagwat S. What is Object Oriented Model in DBMS? Scaler Topics. URL: https://www.scaler.com/topics/object-oriented-model-in-dbms/?_x_tr_hist=true (date of access: 12.03.2025).

6. Каграманова Ю. Як будувати UML-діаграми. Розбираємо три найпопулярніші варіанти. dou.ua. URL: https://dou.ua/ forums/topic/40575/ (дата звернення: 12.03.2025).

7. Моралес Дж. What is UML Class Diagram including UML Class Diagram Maker. MindOnMap. URL: https://www.mindonmap.com/uk/ blog/what-is-uml-class-diagram/ (date of access: 12.03.2025).

8. Maxym Z. Діаграма послідовності (Sequence Diagrams). Maxym Zosym. URL: https://www.maxzosim.com/sequence-diagrams/ (дата звернення: 12.03.2025).

9. Учасники проектів Вікімедіа. М’яке видалення (шаблон проєктування). Вікіпедія. URL: https://uk.wikipedia.org/wiki/М’яке_видалення_(шаблон_проєктування) (дата звернення: 13.03.2025).

10. TCP IP: історія розвитку протоколів і основні принципи роботи. FoxmindEd. URL: https://foxminded.ua/tcp-ip/ (дата звернення: 13.03.2025).

11. Токени аутентифікації. QATestLab. URL: https://training.qatestlab.com/blog/technical-articles/authentication-tokens/ (дата звернення: 14.03.2025).

12. Pedersen G. How and When to Use Power BI Row-Level Security. phData. URL: https://www.phdata.io/blog/power-bi-row-level-security/ (date of access: 15.03.2025).

13. Гордієнко К. JavaScript SEO: коротко про основні принципи. Serpstat Blog. URL: https://serpstat.com/uk/blog/korotko-pro-javascript-seo/ (дата звернення: 15.03.2025).

14. Скорина І. UI/UX-дизайн: інтро в професію. SKVOT. URL: https://skvot.io/uk/blog/into-into-ux-ui-designer-profession (дата звернення: 17.03.2025).

15. Гамбургер меню для сайту. Переваги та найкращі практики. FoxmindEd. URL: https://foxminded.ua/hamburher-meniu-dlia-saitu/ (дата звернення: 17.03.2025).

16. Bigelow S. J., Gillis A. S. What is REST API (RESTful API)? Search App Architecture. URL: https://www.techtarget.com/searchapparchitecture/definition/RESTful-API (date of access: 17.03.2025).

17. Tailwind CSS Create T3 App. Create T3 App. URL: https://create.t3.gg/uk/usage/tailwind/ (date of access: 18.03.2025).

18. Що таке Bootstrap? IT Master. URL: https://itmaster.biz.ua/programming/web-prohramuvannia/bootstrap.html (дата звернення: 18.03.2025).

19. Understanding the Bulma CSS framework: a complete 2025 guide. Pieces for Developers. URL: https://pieces.app/blog/understanding-bulma (date of access: 17.03.2025).

20. ReactJS – Material UI: A Beginner’s Guide. W3schools. URL: https://w3schools.tech/tutorial/reactjs/reactjs_material_ui (date of access: 17.03.2025).

21. Varshavsky Y. Material UI overview: Key benefits and limitations. LinkedIn. URL: https://www.linkedin.com/pulse/material-ui-overview-key-benefits-limitations-yuri-varshavsky-ikpef/ (date of access: 17.03.2025).

22. Chakra UI adoption guide. Overview, examples, and alternatives. LogRocket Blog. URL: https://blog.logrocket.com/chakra-ui-adoption-guide/ (date of access: 17.03.2025).

23. Chakra UI Review 2025 – Tips, Alternatives & More. Toksta. URL: https://www.toksta.com/products/chakra-ui (date of access: 17.03.2025).

24. Samatov I. Chakra UI: React library built for speed. Iskander Samatov. URL: https://isamatov.com/chakra-ui-react/ (date of access: 17.03.2025).

25. Semah B. What Exactly is Node.js? Explained for Beginners. freeCodeCamp.org. URL: https://www.freecodecamp.org/news/what-is-node-js/ (date of access: 18.03.2025).

26. Django Tutorials – Real Python. Real Python. URL: https://realpython.com/tutorials/django/ (date of access: 18.03.2025).

27. Laravel. Is It Really Clean and Classy? SitePoint. URL: https://www.sitepoint.com/laravel-really-clean-and-classy/ (date of access: 18.03.2025).

28. Supabase Docs. Supabase. URL: https://supabase.com/docs (date of access: 18.03.2025).

29. Code of Relevancy. What is prop drilling in React? DEV Community. URL: https://dev.to/codeofrelevancy/what-is-prop-drilling-in-react-3kol (date of access: 19.04.2025).

30. Sytnyk D. Використання CORS в Next.js для обробки cross-origin запитів. Medium. URL: https://medium.com/@dmitry.sytnyk/використання-cors-в-next-js-для-обробки-cross-origin-запитів-d8fe2f8eae8a (дата звернення: 19.04.2025).