Галицький фаховий коледж імені В’ячеслава Чорновола
відділення комп’ютерних технологій
циклова комісія інформатики та комп’ютерних дисциплін

ДОПУСТИТИ ДО ЗАХИСТУ
Завідувач відділення
комп’ютерних технологій
Наталія СТЕФУРАК /_____/
підпис
«___» _____ 2024 р.

ПОЯСНЮВАЛЬНА ЗАПИСКА

до кваліфікаційної роботи
освітньо-професійного ступеня «фаховий молодший бакалавр»
зі спеціальності 122 «Комп’ютерні науки»

на тему: «Кросплатформний музичний плеєр»

Студент групи КН-41	Юрій НИЩОТА	_____ (підпис)
Керівник роботи	Наталія КУЛЬЧИНСЬКА	_____ (підпис)
Консультанти:		
з техніко-економічного обґрунтування	Любов МЕЛЕНЧУК	_____ (підпис)
нормоконтролер	Надія ГАВРИШКІВ	_____ (підпис)

Тернопіль – 2024

Галицький фаховий коледж імені В’ячеслава Чорновола
відділення комп’ютерних технологій
циклова комісія інформатики та комп’ютерних дисциплін

ЗАТВЕРДЖУЮ
Завідувач відділення
комп’ютерних технологій
Наталія СТЕФУРАК /_____/
підпис
«___» _____ 2023 р.

ЗАВДАННЯ
на кваліфікаційну роботу
на здобуття освітньо-професійного ступеня «фаховий молодший бакалавр»
студенту Нищоті Юрію Тарасовичу
(прізвище, ім’я та по-батькові студента)

1. Тема роботи Кросплатформний музичний додаток

затверджена наказом по коледжу від “___”_____ 2023 р., №_____

2. Термін здачі студентом завершеної роботи “___” _____ 2024 р.
3. Вихідні дані до роботи Результати дослідження кросплатформних музичних додатків, ринку продажів послуг музичних додатків. Аналіз технологій реалізації.
4. Перелік питань, які повинні бути розроблені:
а) основна частина розділи Аналіз предметної області та постановка завдань. Проєктування системи. Реалізація та тестування системи.

б) техніко-економічне обгрунтування Аналіз ринку. Розрахунок витрат на проєктування. Обґрунтування необхідності розробки.

5. Перелік графічного матеріалу Діаграма варіантів використання музичного додатку. Діаграма послідовності. DFD діаграма додатку. Фізична модель бази бази даних.

6. Консультанти роботи:_____

Розділ	Косультанти	Підпис, дата	
		Завдання видано	Завдання прийнято
з техніко-економічного обґрунтування	Мелєнчук Л. І. (вчена ступень, звання П.І.Б. консультанта)		

КАЛЕНДАРНИЙ ПЛАН
виконання кваліфікаційної роботи

№ п/п	Найменування етапу	Терміни	
		початку	завершення
1.	Вибір теми кваліфікаційної роботи, ознайомлення з вимогами до кваліфікаційної роботи	16.10.2023	27.10.2023
2.	Аналіз типових рішень і написання відповідного розділу	01.11.2023	30.11.2023
3.	Аналіз та вивчення технологій реалізації і написання відповідного розділу	04.12.2023	12.01.2024
4.	Робота над стурктурою програмного продукту, розробка функціональних вимог. Написання відповідного розділу	15.01.2024	31.01.2024
5.	Налаштування середовища реалізації	01.02.2024	02.02.2024
6.	Проєктування програмного засобу і написання відповідного розділу	04.03.2024	15.03.2024
7.	Реалізація програмного засобу і написання відповідного розділу	18.03.2024	09.04.2024
8.	Вдосконалення модулів	10.04.2024	17.04.2024
9.	Тестування програмного забезпечення і написання відповідного розділу	18.04.2024	22.04.2024

10.	Аналіз економічної частини і написання відповідного розділу	23.04.2024	30.04.2024
11.	Оформлення пояснювальної записки	01.05.2024	27.05.2024
12.	Попередній захист кваліфікаційної роботи	17.06.2024	17.06.2024
13.	Підготовка до захисту кваліфікаційної роботи	18.06.2024	25.06.2024
14.	Захист кваліфікаційної роботи	26.06.2024	26.06.2024

7. Дата видачі завдання “___” _____ 2023 р. Керівник _____/

Завдання прийняв до виконання _____/ /

Реферат

Кваліфікаційна робота. Кросплатформний музичний плеєр. 121с., 31 рисунків, 14 таблиць, 9 джерел, 18 додатків.

Метою кваліфікаційної роботи є розробка кросплатформного музичного додатку для комп’ютерних операційних систем, з використанням сучасних технологій, баз даних та інтерфейсів користувача.

В ході написання кваліфікаційної роботи буде проаналізовано наявні рішення, визначено основні переваги та недоліки. Буде реалізовано макети додатку та модель бази даних.

Для розробки застосунку буде обрано середовище Visual Studio Code. Основна мова програмування – JavaScript, основний фреймворк – ReactJS, серверна частина – NodeJS, для дизайну використано фреймворки TailwindCSS та MaterialUI, для функціоналу авторизації та реєстрації використано Google Firebase. Основна база даних – MySQL, додаткова база даних – Firebase Database. Для реалізації кросплатформності буде використано бібліотеку ElectronJS.

Abstract

Qualification Work. Cross-Platform Music Player. 121 pages, 31 figures, 14 tables, 9 sources, 18 appendices.

The goal of the qualification work is to develop a cross-platform music application for computer operating systems, using modern technologies, databases, and user interfaces.

During the writing of the qualification work, existing solutions will be analyzed, the main advantages and disadvantages will be identified. Application mockups and a database model will be implemented.

The development environment chosen for the application is Visual Studio Code. The main programming language is JavaScript, the main framework is ReactJS, the server-side is NodeJS. For design, the frameworks TailwindCSS and MaterialUI are used, for authentication and registration functionality, Google Firebase is employed. The primary database is MySQL, with an additional Firebase Database. The ElectronJS library will be used to achieve cross-platform compatibility.

ЗМІСТ

СКОРОЧЕННЯ ТА УМОВНІ ПОЗНАКИ

СУБД – Система управління базами даних

ОС – Операційна система

ПЗ – програмне забезпечення

HTML – Hypertext Markup Language

CSS – Cascading Style Sheets

PHP – Hypertext Preprocess

MUI – Material UI

JS – Java Script

API – Application Programming Interface

HTTP – HyperText Transfer Protocol

URL – Uniform Resource Locator

DOM – Document Object Model

DFD – Data Flow Diagram

ВСТУП

Сьогодні, коли майже весь світ використовує цифрові пристрої, важливо забезпечувати потреби людей у якісних додатках. Практично щодня люди прослуховують музику – вдома, на вулиці чи на роботі. Музика супроводжує у багатьох аспектах життя, допомагаючи розслабитися, налаштуватися на роботу або просто насолоджуватися моментом. Для комфортного прослуховування пісень людство створює музичні додатки, які дозволяють швидко та просто знаходити і слухати улюблені треки. Вони стають невід'ємною частиною нашого повсякдення, об'єднуючи користувачів різних вікових категорій та соціальних груп.

Метою цієї кваліфікаційної роботи є створення кросплатформного музичного додатку для комп'ютерних операційних систем, який забезпечить універсальний доступ до музики. Завдання включає розробку зручного та інтуїтивно зрозумілого дизайну, що дозволить користувачам легко орієнтуватись та використовувати додаток. Забезпечення безпеки та приватності користувачів, оптимізацію продуктивності, а також наявність якісного та широкого функціоналу.

Кінцевий результат цього проєкту дозволить закріпити знання про технології реалізації кросплатформних застосунків, почати хороший стартап або ж використати його в якості портфоліо при поданні на роботу розробником. Це буде цінним досвідом, який продемонструє вміння створювати сучасні та функціональні додатки. Крім того, такий проєкт може стати відправною точкою для подальших розробок та вдосконалень, відкриваючи нові горизонти у сфері програмування та розробки програмного забезпечення.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАВДАНЬ

1.1 Дослідження предметної області

Музичний плеєр у вигляді додатку зародився приблизно в ті часи, коли інтернет тільки почав розвиватися і розширюватися по всьому світу. Тоді ще не було ні грандіозних ідей, ні великих та масштабних проєктів. З часом технології почали розвиватися, а з ними і можливості функціоналу до програвачів мультимедіа. Так що ж таке програвач мультимедіа? Програвач мультимедіа – це тип програми, завдяки якій можна відтворювати музику або відеоматеріали. Прикладами таких є: Spotify, Apple Music, YouTube Music тощо. Детальніше про них буде згадано нижче. Завдяки тому, що ця сфера діяльності є відносно новою, на ній не дуже велика конкуренція. Лідери таких додатків створюють максимально комфортні умови для користування. Також придумують багато корисного функціоналу, щоб користувач зміг увімкнути все, що забажає.

Насамперед потрібно розуміти, як має виглядати музичний плеєр у вигляді додатку і як це має працювати. Кожного дня технології тільки покращуються і адаптуються під комфорт користувача. Починаючи від MP3-плеєра до спеціального вебсайту, на якому можна прослуховувати пісні. Користувач повинен мати доступ до великої частини пісень, які є на цій платформі, і можливість прослуховувати їх. Приблизно такий функціонал повинен отримати користувач:

- відтворення музики: старт, пауза, наступна пісня, попередня пісня;
- можливість вмикати різноманітні плейлисти, список улюблених треків, випадкові пісні, рекомендації;
- ввімкнути циклічне програвання музики;
- перегляд різноманітних артистів та їхні альбоми;
- збільшення або ж зменшення гучності;
- перемотування пісні.

Важливо підкреслити, що додатки можна встановити на різні операційні системи, такі як Windows, macOS, Linux, iOS, Android тощо. У майбутньому буде детальніше розглянута ця проблема.

У світі музичних плеєрів як додатків важливе значення має не лише технічний аспект, але й простота та зручність для кінцевого користувача. Окрім базового функціоналу, такого як відтворення, пауза та перемикання треків, зрозумілий та легкий в управлінні інтерфейс грає важливу роль. Музичний плеєр має бути інтуїтивно зрозумілим, надавати широкі можливості доступу до різноманітних функцій, а також дозволяти користувачам налаштовувати параметри відтворення за їхніми особистими уподобаннями. Враховуючи потреби різних користувачів, розробники повинні прагнути до максимальної простоти та комфорту взаємодії з музичним контентом.

1.2 Обґрунтування актуальності розробки

Для того щоб підтвердити актуальність додатків для відтворення музики, нижче наведено ринкову статистику Сполучених Штатів Америки [2]:

– Розмір ринку додатків прослуховування музики може досягти 103,07 мільярдів доларів США до 2030 року, це приблизно 14,4% зросту від 2023 по 2030 рік.

– Приблизно 82 мільйони американців користуються тарифними планами сплачуючи близько 10 доларів в місяць додаткам для прослуховування музики.

– Аналізом експертів очікується, що приблизно до 2027 року кількість користувачів може перевищити число 1,1 мільярдів.

Якщо вірити цьому аналізу, то неважко зрозуміти, що ця сфера є доволі перспективною як у фінансовому, так і в технічному прогресі. Завдяки актуальності прослуховування музики, це може бути хорошим кар’єрним стартом для багатьох початківців і взагалі великим проєктом для різноманітних корпорацій.

Також, якщо враховувати аналіз ринку експертів, сферу музичного програвання очікує тільки успіх. Цей аналіз можна побачити на наведеній нижче схемі, а саме на рисунку 1.1.

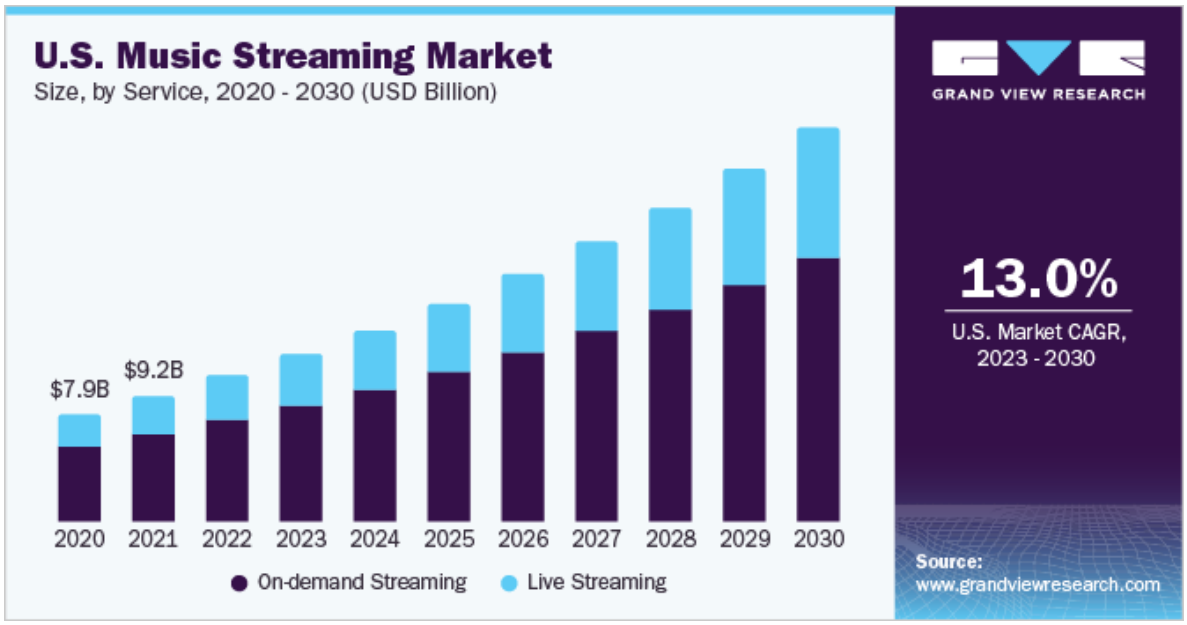


Рисунок 1.1 – Аналіз ринку додатків музичного програвання в Америці від 2020 до 2030 років

У 2022 році на сегмент додатків припадає найбільша частка доходу – 86,4% завдяки перевагі легкої доступності за допомогою додатків замість браузерів і численним функціям, які покращують загальний досвід користувача під час потокового передавання музики. Деякі з його ключових функцій включають настроювані списки відтворення, високоякісне потокове аудіо, відтворення в автономному режимі, параметри обміну в соціальних мережах та інтеграцію з іншими програмами та пристроями. Очікується, що такі особливості посилять зростання сегмента протягом прогнозованого періоду [2]. Тим не менш, потрібно враховувати, що це аналіз лише ринку Сполучених Штатів Америки, і також є велика частина користувачів з інших країн, які так само зацікавлені в активному використанні цих додатків. Аналітику у вигляді діаграми можна побачити нижче на рисунку 1.2.

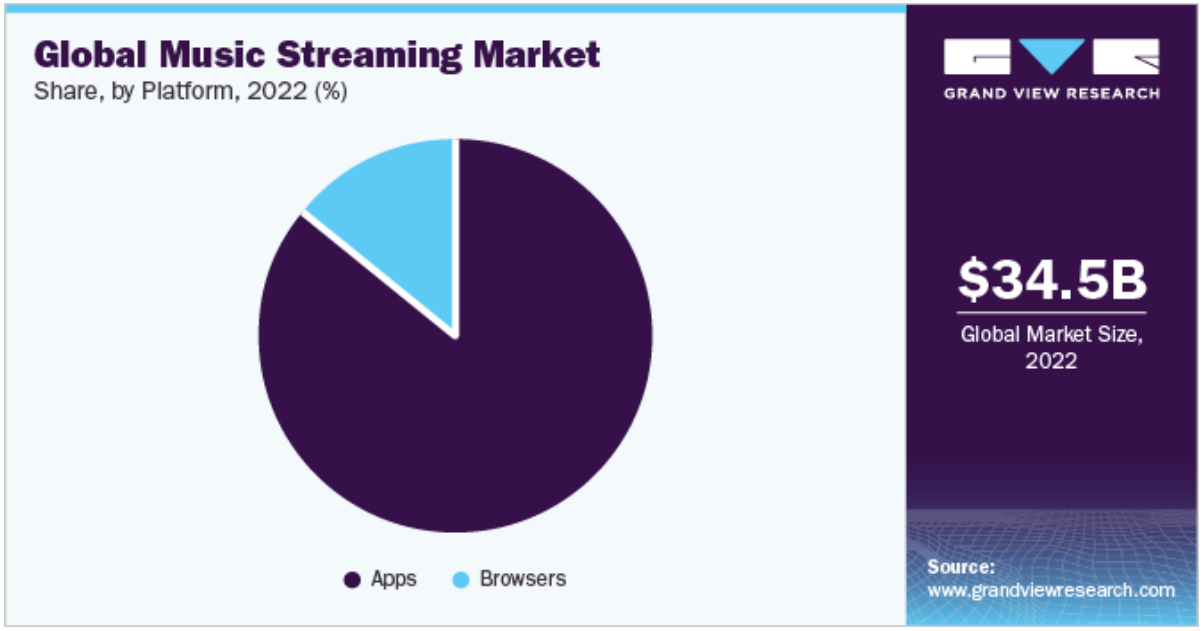


Рисунок 1.2 – Аналіз ринку додатків музичного програвання та вебсайтів в Америці протягом 2022 року

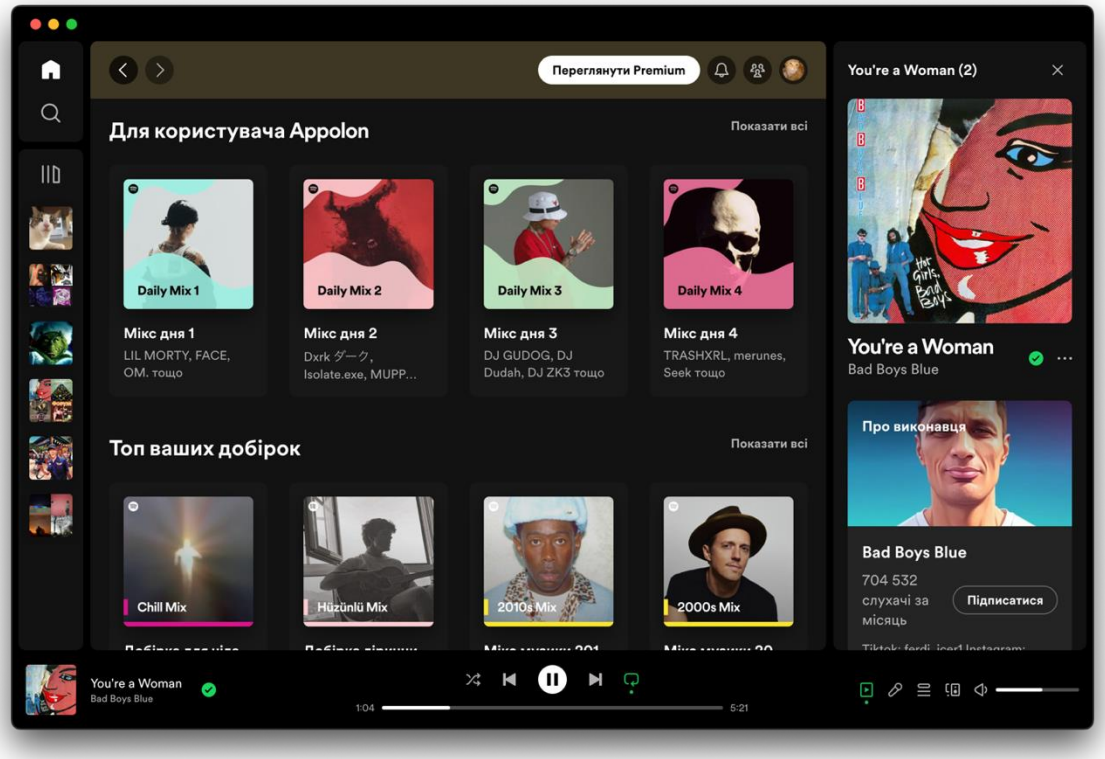
Підсумовуючи, можна дійти висновку, що сфера програмування музичних додатків є доволі перспективною ініціативою на приблизно наступні десять років. Якщо придумати кращий, новий, комфортний функціонал, це дозволить зацікавити нових користувачів на свою платформу.

1.3 Аналіз наявних рішень

Для реалізації свого додатку для програвання музики потрібно проаналізувати декілька платформ, щоб якісно сформувати технічні та інформаційні вимоги.

Spotify – можливо, найвідоміший та найпоширеніший додаток для прослуховування музики, подкастів і відео. Spotify дозволяє користувачам ділитися музикою з іншими та створювати власні списки відтворення [1]. Цей додаток швидко став відомим завдяки своєму комфорту в обранні музики, можливості створення різних плейлистів, списків улюбленої музики, прослуховування різноманітних пісень, відстеження авторів, тощо. Також, завдяки рекламі та тарифному плану, додаток отримує великий прибуток, що є проривом у сфері музичних плеєрів.

Дизайн є доволі простим і інтуїтивно зрозумілим для користування. Користувач з легкістю може скористатися функціоналом завдяки декільком діям. Дизайн того, як виглядає додаток Spotify, можна побачити на рисунку 1.3.



А також переваги та недоліки цього додатку можна побачити в таблиці 1.1.

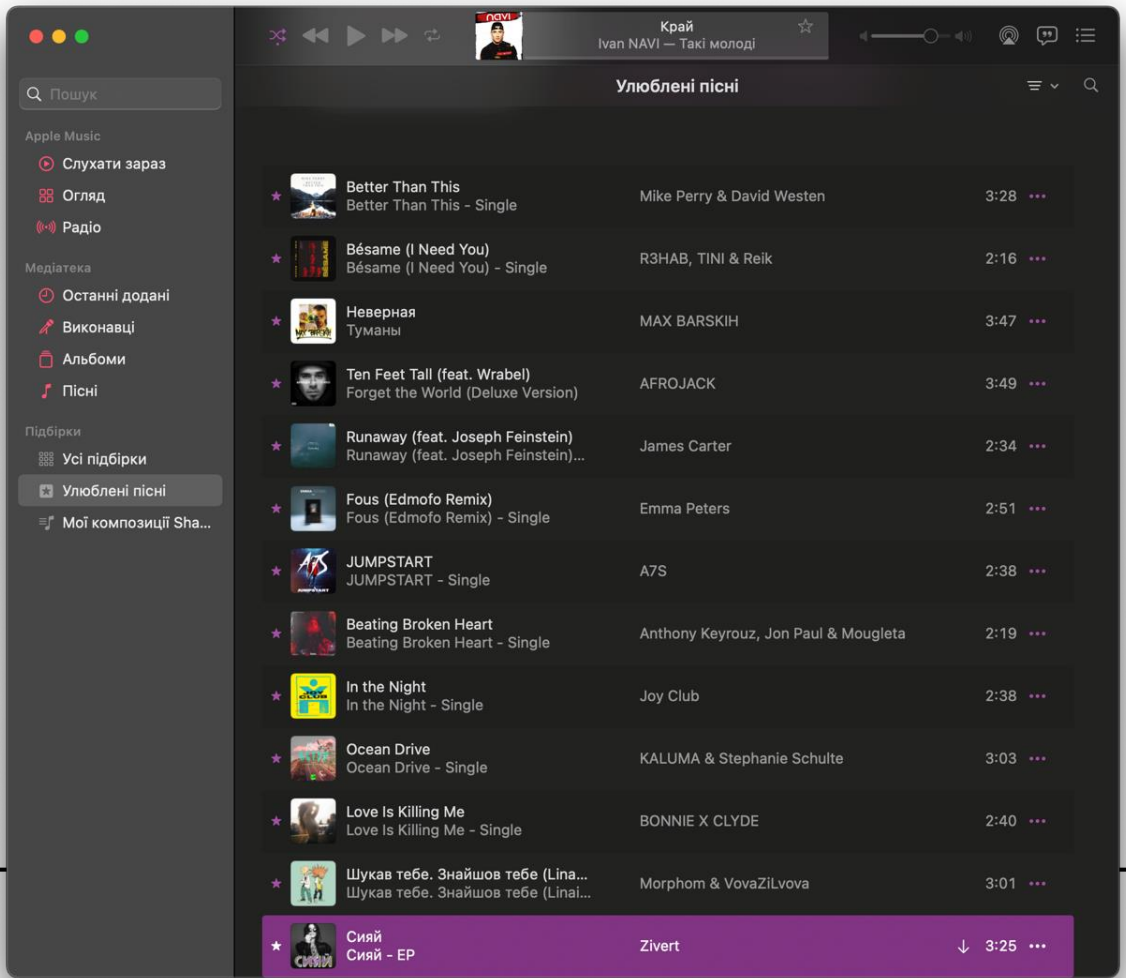
Таблиця 1.1 – переваги та недоліки музичного додатку Spotify

Переваги	Недоліки
Spotify пропонує безкоштовну підписку	Безкоштовний план Spotify має обмеження
Spotify надає великий музичний каталог	Дуже багато реклами в безкоштовному плані
Spotify сумісний з різними платформами	Пісню неможливо завантажити навіть із тарифним планом Spotify Premium

Apple Music — це сервіс потокової передачі музики, який пропонує величезні каталоги, щоб користувач міг насолоджуватися музикою вільніше та

зручніше [5]. На відміну від Spotify, цей додаток створений для операційних систем таких як MacOS або iOS. Також його можна використовувати на Windows або Android. Apple Music є великим конкурентом для Spotify та інших подібних додатків, але через те, що компанія Apple більше спеціалізується на своїх технологіях, ця конкуренція зменшується.

Так само як і попередній додаток, Apple Music є доволі простим і зрозумілим у використанні. Користувач з легкістю може обрати будь-який можливий і доступний функціонал. Переглянути дизайн цього додатку можна на рисунку 1.4.


Край
Ivan NAVI — Такі молоді
Пошук
Apple Music
Слухати зараз
Огляд
Радіо
Медіатека
Останні додані
Виконавці
Альбоми
Пісні
Підбірки
Усі підбірки
Улюблені пісні
Мої композиції Sha...
Улюблені пісні
Better Than This
Better Than This - Single
Mike Perry & David Westen
3:28
Bésame (I Need You)
Bésame (I Need You) - Single
R3HAB, TINI & Reik
2:16
Неверная
Туманы
MAX BARSKIH
3:47
Ten Feet Tall (feat. Wrabel)
Forget the World (Deluxe Version)
AFROJACK
3:49
Runaway (feat. Joseph Feinstein)
Runaway (feat. Joseph Feinstein)...
James Carter
2:34
Fous (Edmofo Remix)
Fous (Edmofo Remix) - Single
Emma Peters
2:51
JUMPSTART
JUMPSTART - Single
A7S
2:38
Beating Broken Heart
Beating Broken Heart - Single
Anthony Keyrouz, Jon Paul & Mougleta
2:19
In the Night
In the Night - Single
Joy Club
2:38
Ocean Drive
Ocean Drive - Single
KALUMA & Stephanie Schulte
3:03
Love Is Killing Me
Love Is Killing Me - Single
BONNIE X CLYDE
2:40
Шукав тебе. Знайшов тебе (Lina...
Шукав тебе. Знайшов тебе (Linai...
Morphom & VovaZiLvova
3:01
Сияй
Сияй - EP
Zivert
3:25

Детальнішу характеристику про переваги та недоліки можна побачити в таблиці 1.2.

Таблиця 1.2 – переваги та недоліки музичного додатку Apple Music

Переваги	Недоліки
Величезний музичний каталог	Не є безкоштовним
Офлайн-прослуховування	Інтерфейс не інтуїтивно зрозумілий
Ціна тарифних планів	Apple Music доступна не в усіх країнах і регіонах

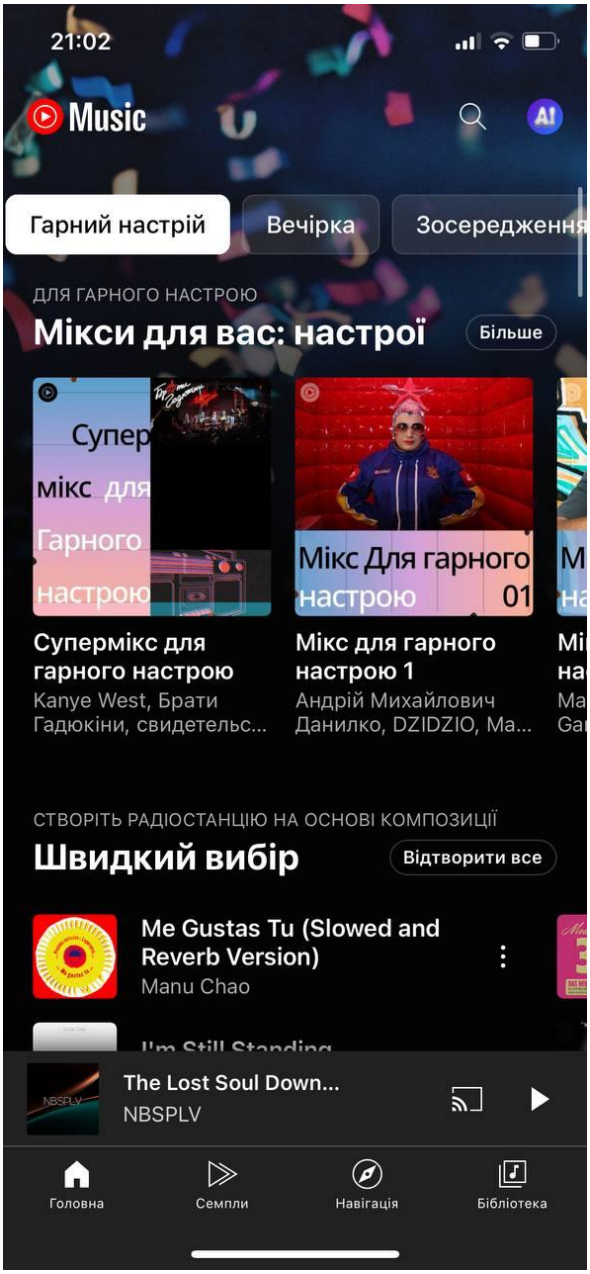
Останній сервіс більш відомий у вигляді вебсайту, але для прикладу також підійде. Цей сервіс хороший тим, що він синхронізований з YouTube, через що можна швидко знайти вподобаний трек або якийсь ексклюзивний ремікс.

YouTube Music — це музичний сервіс, розроблений американською відеоплатформою YouTube, дочірньою компанією Google. Служба розроблена з інтерфейсом користувача, який дозволяє користувачам прослуховувати пісні та музичні відео на YouTube на основі різноманітних жанрів, списків відтворення та рекомендацій [6]. Більше про переваги і недоліки наведено в таблиці 1.3.

Таблиця 1.3 – переваги та недоліки музичного вебсайту Youtube Music

Переваги	Недоліки
Безкоштовний сервіс	Важкий інтерфейс
Офлайн-прослуховування	Погані рекомендації треків
Користувачі можуть побачити кліпи треків	Відсутні підкасти

Дизайн додатку YouTube Music не є дуже простим для користування, проте функціоналу в ньому не менше, а навпаки, більше, ніж у двох останніх додатках. Побачити, як виглядає дизайн цього додатку, можна на рисунку 1.5.



Отже, аналіз наявних рішень є важливим етапом, який надає багато інформації про те, з чого саме повинна складатись інформаційна система музичного програвача. Важливо визначити функціонал, який приблизно повинен бути присутній, а також врахувати вимоги користувачів. Потрібно зробити додаток цікавим і простим у використанні.

1.4 Аналіз вимог та постановка завдання

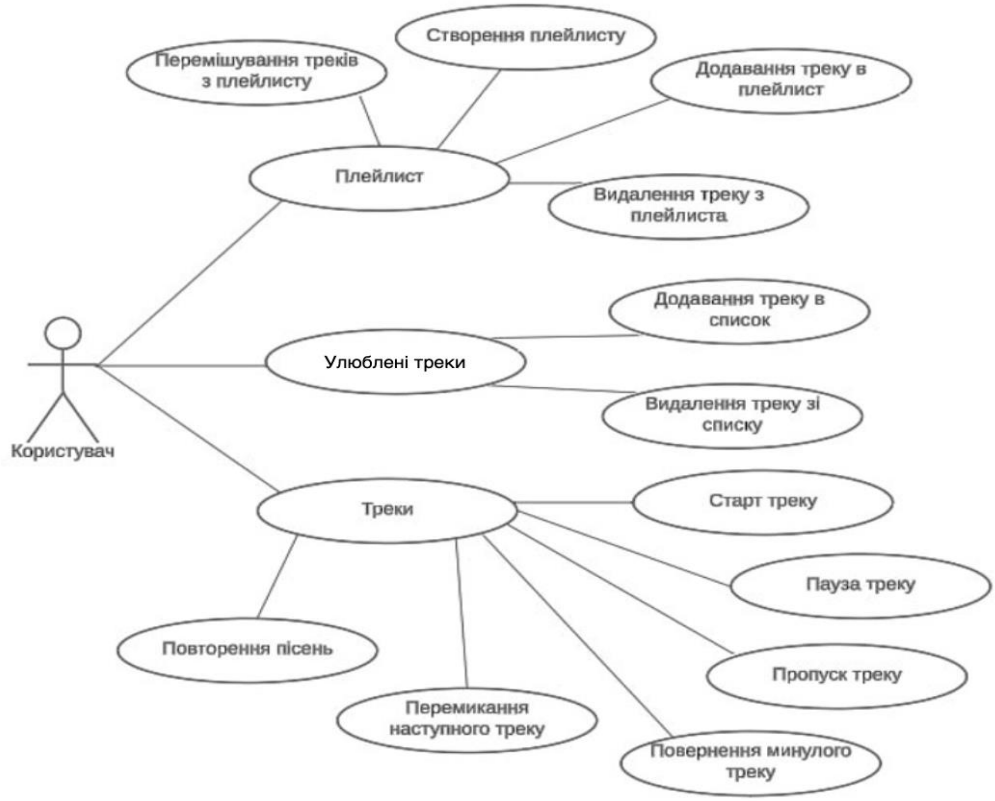
Для того, щоб почати реалізовувати технічну частину додатку, потрібно визначити цілі, вимоги та основну мету, яку потрібно виконати під час виконання завдання. Тим не менш, необхідно розуміти, як саме програмне забезпечення буде бачити користувач, і наскільки буде зручне і просте використання додатку.

Для того, щоб користувач зміг створити свій власний плейлист або обрати улюблені треки, він повинен створити свій власний обліковий запис за допомогою спеціальної форми, яка буде відкриватись одразу ж після відкриття додатку. В іншому випадку, якщо профіль вже існує, то просто авторизуватись в систему. Після чого у користувача буде доступ до прослуховування всіх доступних пісень і можливість знайти їх у спеціальному пошуковому компоненті, переключати їх, змінювати гучність, ставити на паузу або перемотувати сам трек. Користувач зможе створити свій власний плейлист і додавати або видаляти пісні, які забажає. Також буде можливість додавати треки до списку улюблених пісень, який він зможе переслуховувати та редагувати його. Ще однією з реалізованих функцій буде циклічне програвання музики і перемішування плейлистів або улюбленого списку.

Додаток повинен мати привабливий та цікавий дизайн, який буде привертати увагу і нав’язувати користувачу бажання користуватись ПЗ. Необхідно створити навігаційний компонент, завдяки якому користувач зможе відкрити список улюблених треків, плейлисти, тощо. Тим не менш, повинен бути профіль користувача, де користувач зможе залишити про себе особисті дані і за потреби редагувати їх.

Необхідно створити окремий компонент, який буде відповідати за можливість перемикати наступну пісню або попередню, змінювати гучність, вмикати повторення пісні або випадкове програвання пісень. Також додаток повинен містити головну сторінку, на якій будуть відображатись рекомендації для користувача або нові ексклюзивні треки.

Для кращого розуміння того, як має виглядати весь процес та можливий функціонал, була створена діаграма варіантів використання, яка відображена на рисунку 1.6.



Додаток має створювати зручні умови для взаємодії з музичним контентом та пропонувати цікавий візуальний досвід. Підтримка особистих профілів і можливість налаштувань дозволять користувачам насолоджуватися персоналізованим використанням додатку. Усі ці аспекти важливі для створення позитивного враження від користування музичним додатком та забезпечення його популярності серед аудиторії.

2 ПРОЄКТУВАННЯ СИСТЕМИ

2.1 Формалізація вимог

Для реалізації додатку, який буде складатись з різних компонентів, кожен з яких буде розроблятись окремо, потрібно визначити основні сторінки, що міститимуть ці компоненти:

– форма реєстрації та форма авторизації;

– головна сторінка, яка буде містити різноманітні альбоми та плейлисти;

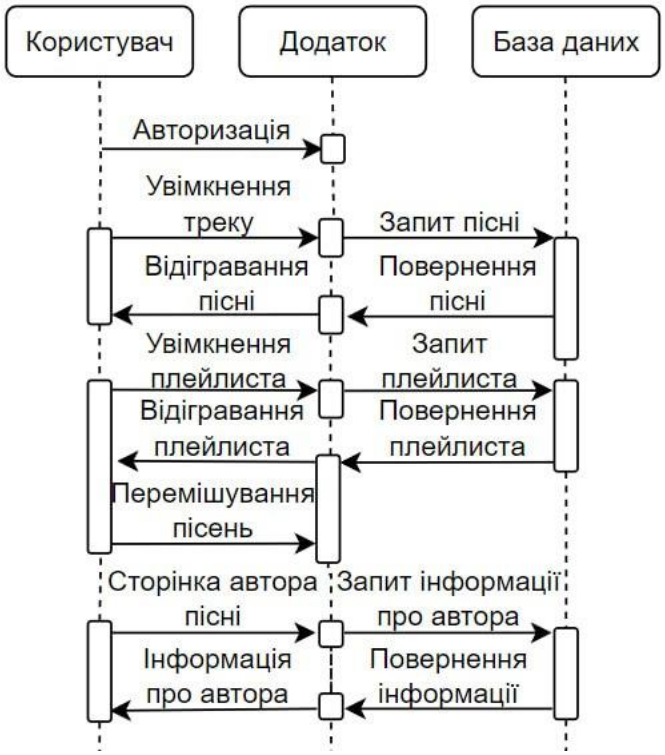
– форма створення плейлиста;

– сторінка пошуку;

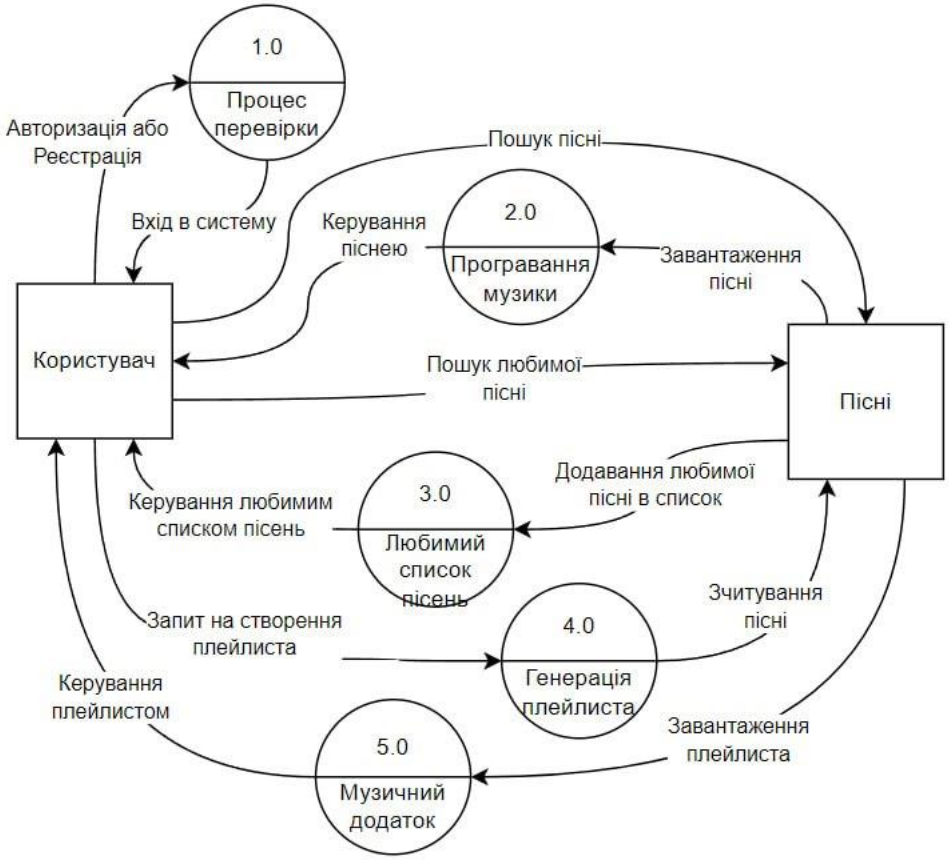
– профіль користувача;

– сторінка з інформацією про певного автора, а саме його пісні та альбоми.

Послідовність процесів та дій відображено на діаграмі послідовностей (рисунок 2.1), завдяки якій можна зрозуміти усі процеси між об’єктами та взаємозв’язки між ними.



Для вивчення шляху проходження даних, основних процесів, що відбуваються, та комунікації між ними, була спроктована діаграма потоків даних (рисунок 2.2). На діаграмі представлено процеси обробки та передачі даних через різні компоненти додатку, що допомагає краще зрозуміти внутрішній механізм системи.



В цьому додатку так званих сторінок не буде, натомість будуть компоненти, які відображатимуть всю необхідну інформацію, яка може змінюватись завдяки певним діям користувача.

Знизу головної сторінки буде прикріплена спеціальна стрічка мультимедіа, яка буде містити в собі інструменти, завдяки яким можна буде зупинити або продовжити трек, пропустити або повернутись на попередній трек назад, перемішати плейлист, збільшити або зменшити гучність, а також побачити фото та назву пісні.

Також буде компонент, який буде прикріплений до лівої частини додатку і відображатиме створені плейлисти, кнопку для створення нового плейлиста,

іконку профілю, на яку можна натиснути, щоб перейти на профіль користувача, і кнопку для переходу на сторінку пошуку пісні, автора, плейлиста, альбому тощо.

Головний найбільший компонент буде змінний і буде в собі міняти сторінку в залежності від того, яку хоче бачити користувач. Це може бути список пісень, відображення альбому, плейлиста, інформації про автора, профіль користувача, налаштування, форма створення плейлиста, пошук пісень тощо.

Отже, додаток в цілому буде складатись з трьох компонентів, завдяки яким користувач зможе повноцінно користуватись додатком.

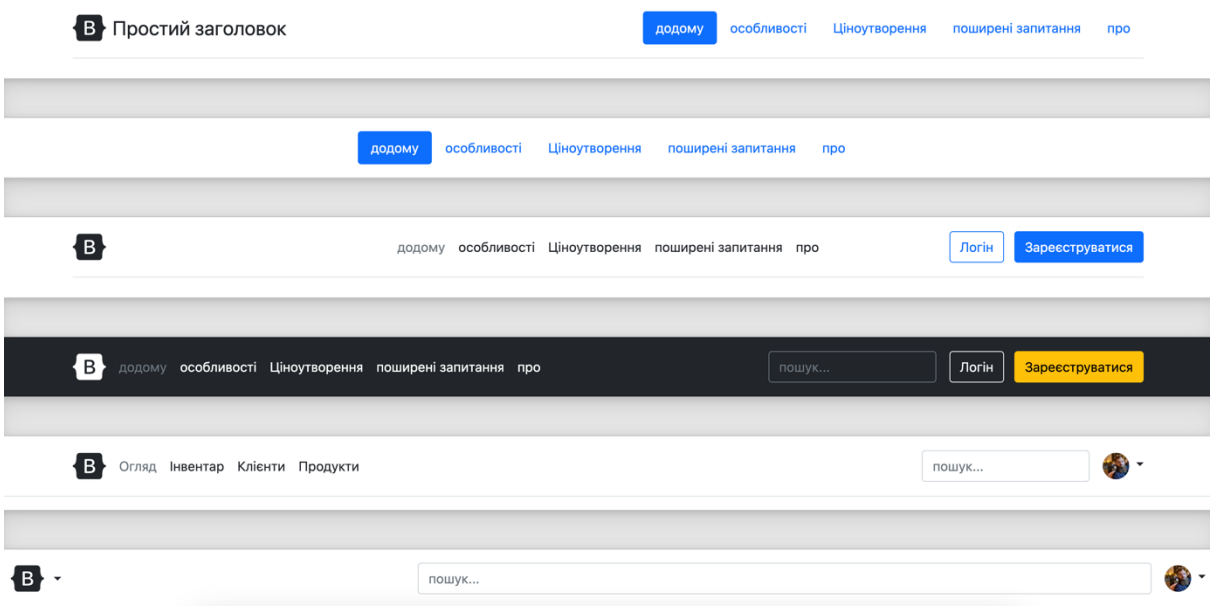
2.2 Аналіз технологій реалізацій

Для реалізації вимог, сформованих у попередньому пункті, потрібно правильно обрати технології та використати їх. Обов’язковою технологією для всіх вебсайтів або вебдодатків є HTML та CSS. HTML – це мова розмітки, яка дозволяє переглядати вебсторінки і створювати розмітки завдяки різноманітним елементам. Вона є, скажімо так, базою для реалізації всіх проєктів, великих чи малих. До цієї мови розмітки додається мова каскадних стилів CSS, завдяки якій можна стилізувати розмітки HTML. Разом HTML і CSS утворюють базу створення всесвітньої інтернет павутини.

Для того, щоб не писати з нуля якийсь продукт, існують так звані фреймворки. Фреймворк – це інструмент, бібліотека, що надає інструменти та шаблони для реалізації вебсайту чи вебдодатку. Для CSS створено декілька фреймворків, які містять готові стилі та шаблони, завдяки чому прискорюється створення продукту. Одним із таких відомих, якісних та простих фреймворків є Bootstrap.

Bootstrap — це безкоштовний фреймворк стилів, який надає широкий вибір інструментів для роботи з CSS та HTML, завдяки якому можна швидко створити

цікавий та простий дизайн. Приклад використання Bootstrap, створення навігаційних полів зображено на рисунку 2.3.



Були визначені наступні переваги та недоліки фреймворку Bootstrap.

Переваги Bootstrap:

– завдяки готовим компонентам та стилям можна швидко створити вебсайт або додаток;

– сайти, побудовані на Bootstrap, легко адаптуються під різні пристрої та розміри екранів;

– Bootstrap робить розробку вебдодатків більш сумісною з різними браузерами.

Недоліки Bootstrap:

– використання стандартних компонентів може призвести до того, що вебсайти виглядають схоже один на одного;

– не всі компоненти Bootstrap можуть бути використані в конкретному проєкті, що може призвести до зайвого коду та повільного завантаження.

Також доволі хороші готові стилі є у фреймворку MUI – це фреймворк з готовими компонентами та стилями, завдяки яким можна створити привабливий вебпродукт.

Переваги MUI:

– базується на креативній бібліотеці з різноманітними дизайнами, що робить інтерфейси стильними та сучасними;

– надає широкий вибір налаштувань та можливостей для кастомізації компонентів.

Недоліки MUI:

– для новачків може бути складно оволодіти всіма можливостями та налаштуваннями MUI;

– для тих, хто шукає унікальний дизайн, однакові стилі використання Material Design можуть бути недоліком.

Для використання MUI в проєкті потрібно встановити та підключити багато залежностей, що може зробити процес налаштування складним. Приклад створення інтерфейсу завдяки MUI, зображено на рисунку 2.4.

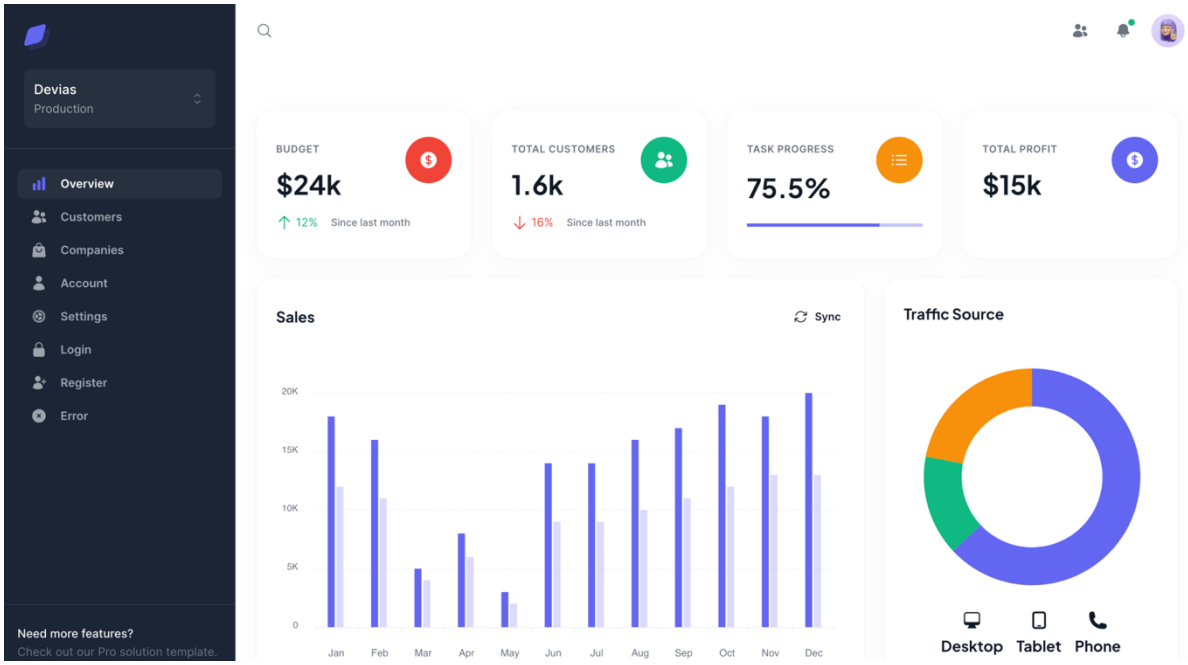


Рисунок 2.4 – Приклад використання фреймворку MUI

Для того, щоб доповнити базові мови для реалізації якісного продукту, можна додати такі мови програмування, як PHP та JavaScript. Вони є

найближчими мовами програмування, завдяки яким можна створити хороший вебсайт або вебдодаток.

JavaScript — динамічна, об'єктно-орієнтована, скриптова прототипна мова програмування. На даний момент ця мова є найвідомішою та найпрактичнішою для створення вебзастосунків.

Переваги JavaScript:

– вивчення та освоєння цієї мови програмування є доволі простим;

– завдяки цій мові програмування є можливість створювати різноманітні ефекти, анімацію, валідувати форми, робити запити на сервер і багато іншого.

Недоліки JavaScript:

– різні браузери можуть різним чином інтерпретувати той самий код JavaScript, що може призвести до несправностей на різних платформах;

– якщо необачно створювати скрипти, це може призвести до безпекових проблем, таких як хакерські атаки на сайт;

– на деяких пристроях або застарілих браузерах JavaScript може працювати повільніше, що може затримувати роботу сторінки.

До мови JavaScript існує велика кількість різноманітних фреймворків, які спрощують роботу та надають готовий код, функціонал, дизайн, залежно від сфокусованості бібліотеки. Одним із таких дуже корисних фреймворків є ReactJS.

ReactJS – це бібліотека JavaScript, яка використовується для побудови інтерфейсів користувача. Вона дозволяє створювати ефективні та масштабовані вебдодатки, орієнтовані на компоненти.

Переваги ReactJS:

– система виведення сторінки або ж DOM. ReactJS використовує віртуальну систему, що пришвидшує, полегшує роботу та збільшує якість додатку;

– компонентна структура програмування, що полегшує програмування та керування кодом і додатком в цілому;

– хороша змістовна та зрозуміла документація.

Недоліки ReactJS:

– для новачків та навіть для досвідчених програмістів можуть виникнути проблеми під час застосування ReactJS;

– необхідність інших додаткових бібліотек. На жаль, на одному ReactJS повноцінно хороший проєкт створити дуже важко, а то й неможливо;

– через те, що технології прогресують, а з ними і цей фреймворк, документація може змінюватись, що вимагає змін програмного коду.

Node.js – це середовище виконання JavaScript, яке дозволяє створювати серверні додатки на базі мови програмування JavaScript.

Основні переваги включають високу швидкодію завдяки спеціальному виконанню різних операцій. Також, практично будь-яка платформа підтримує це середовище. Останнє велике, що варто згадати – це те, що це середовище є дуже корисним для реалізації серверної частини коду.

Однак серед недоліків можна відзначити потенційні проблеми з масштабованістю: якщо буде багато різних запитів, це може призвести до критичних проблем. Також є складність в управлінні асинхронним кодом, що може призвести до зростання складності додатків при розширенні.

PHP – це мова програмування, яка використовується для розробки вебсайтів і додавання динамічної функціональності до вебсторінок. У певний період часу ця мова програмування гордо тримала назву найвідомішої мови програмування у сфері веброзробки.

Головними перевагами PHP є простота у використанні, зрозумілість та легкість вивчення. Оскільки ця мова програмування існує дуже довгий період часу, зібралася велика спільнота розробників та експертів, які допомагають у застосуванні цієї технології.

Щодо недоліків, є суттєві проблеми з безпекою, що може зашкодити програмному коду. Станом на 2024 рік популярність цієї мови програмування поступово згасає, що вказує на неактуальність використання технології.

При створенні свого вебпродукту обов’язково потрібно десь зберігати дані. В цьому допоможе MySQL — реляційна система управління базами даних.

Через те, що ця технологія є доволі старою, вона є безпечною та стабільною, що дозволяє без вагань обрати надійну СУБД. Також, завдяки тривалому існуванню цієї технології, виросла велика спільнота фахівців, які можуть допомогти у використанні MySQL. Висока продуктивність і оптимізованість є беззаперечно великими плюсами цієї технології.

Щодо недоліків, MySQL може виділятися відсутністю документації для роботи з новими типами даних, можливою відсутністю деяких дійсно потрібних функцій, а також з великою кількістю даних можуть виникати певні проблеми.

Отже, потрібно правильно обрати і використовувати технології, щоб продукт виглядав максимально якісно як ззовні, так і всередині програмного коду.

2.3 Проєктування інтерфейсу

Для того, щоб приблизно розуміти, як повинен виглядати додаток, потрібно реалізувати макет. Додаток буде розподілений на декілька компонентів:

– навігація: компонент, в якому буде профіль користувача, головна сторінка, елемент пошуку, улюблені пісні користувача та плейлисти користувача;

– панель відтворення: у цьому компоненті буде фото пісні, назва, полоска збільшення або зменшення гучності, полоска відтворення пісні, а також інструменти, за допомогою яких можна поставити трек на паузу, відтворити попередню або наступну пісню, перемішати список пісень, включити трек на повтор;

– головний компонент: в цьому компоненті буде відображатись вся доступна інформація, плейлисти, списки, пісні, профіль користувача тощо.

Макет додатку має виглядати так само, як показано на рисунку 2.5.

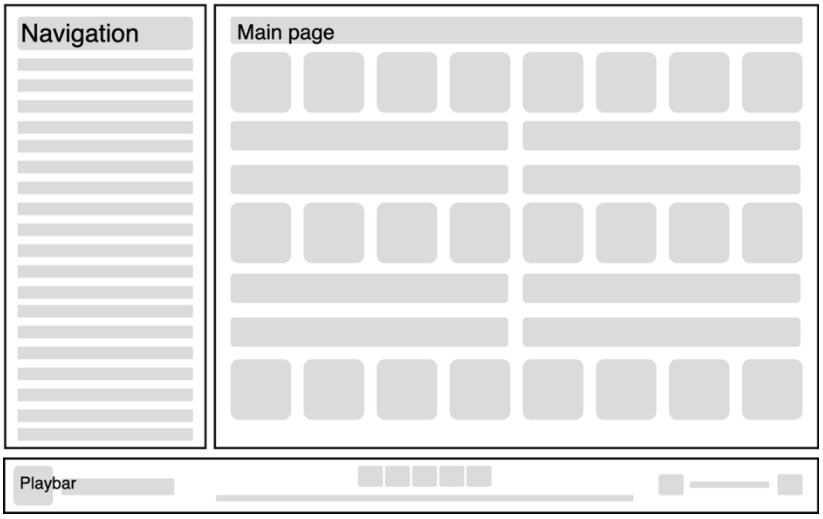


Рисунок 2.5 – Макет вигляду додатку

Також буде окрема сторінка, на якій буде відображена форма, за допомогою якої користувач зможе авторизуватись в додаток або, в разі відсутності профілю, зареєструватись. У низу форми авторизації буде реалізоване посилання, яке буде переключати форму на реєстрацію (рис. 2.6). Далі дизайн буде реалізовуватись, виходячи вже з макету безпосередньо програмним кодом.

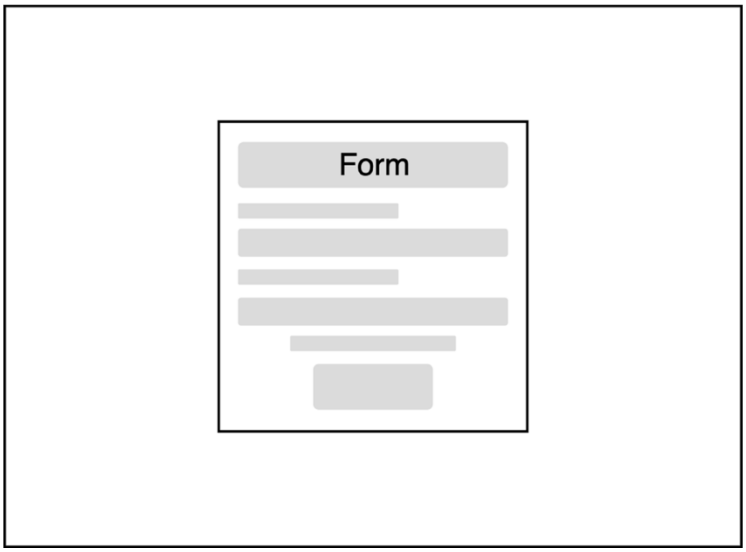


Рисунок 2.6 – Макет форми авторизації та реєстрації

2.4 Проєктування бази даних

Для того, щоб коректно, а найголовніше, правильно зберігалися всі дані з музичного плеєра, потрібно реалізувати модель бази даних. У цьому додатку буде використовуватися система управління базами даних MySQL, яка вже була

згадана раніше. Для початку потрібно визначити основні сутності та їх типи даних.

Головна таблиця буде містити всю інформацію про треки та буде використовуватись в інших таблицях. Детальніше можна побачити в таблиці 2.1.

Таблиця 2.1 – Таблиця з інформацією про пісні

song_id	Int(11)	Ідентифікатор пісні
album_id	Int(11)	Ідентифікатор альбому
author_id	Int(11)	Ідентифікатор автора
song_name	Varchar(255)	Назва пісні
song_image	Varchar(255)	Посилання фотографії пісні
song_path	Varchar(255)	Посилання музики
song_duration	Int(11)	Тривалість пісні

Наступна таблиця буде містити інформацію про плейлисти користувачів. Користувач може створити новий плейлист, і відповідний запис з'являється в цій таблиці. Детальніше можна побачити в таблиці 2.2.

Таблиця 2.2 – Таблиця з плейлистами користувачів

playlist_id	Int(11)	Ідентифікатор улюбленого списку
uid	Varchar(255)	Ідентифікатор користувача
playlist_name	Varchar(1000)	Кількість пісень
playlist_image	Varchar(1000)	Посилання фото плейлиста
song_count	Int(11)	Кількість пісень
playlist_duration	Int(11)	Тривалість списку

Також до минулої таблиці додається проміжна таблиця, яка містить дані про те, які пісні знаходяться в певному плейлисті та їхні ідентифікатори. Детальніше про цю таблицю можна побачити в таблиці 2.3.

Таблиця 2.3 – Таблиця з піснями плейлистів користувачів

playlist_song_id	Int(11)	Ідентифікатор елементів плейлистів користувачів

playlist_id	Int(11)	Ідентифікатор плейлиста користувача
song_id	Int(11)	Ідентифікатор пісні

Наступна таблиця містить записи про улюблений список пісень користувача. Детальніше в таблиці 2.4.

Таблиця 2.4 – Таблиця з списками улюблених пісень користувачів

favorite_list_id	Int(11)	Ідентифікатор улюбленого списку
uid	Varchar(255)	Ідентифікатор користувача
song_count	Int(11)	Кількість пісень
favorite_list_duration	Int(11)	Тривалість списку

До минулої таблиці додається проміжна таблиця, яка містить інформацію про те, які пісні знаходяться в улюблених списках користувачів. Детальніше можна побачити в таблиці 2.5.

Таблиця 2.5 – Таблиця з найулюбленішими піснями користувачів

favorite_list_item_id	Int(11)	Ідентифікатор улюблених пісень користувачів
favorite_list_id	Int(11)	Ідентифікатор улюбленого списку
song_id	Int(11)	Ідентифікатор пісні

Наступна таблиця містить коротку інформацію про користувачів, ідентифікатор та шлях до фотографії користувача. Детальніше в таблиці 2.6.

Таблиця 2.6 – Таблиця з інформацією про користувачів

uid	Varchar(255)	Ідентифікатор користувача
uphoto	Varchar(255)	Адрес фотографії користувача

Необхідно створити таблицю, яка містить інформацію про базові статичні плейлисти, що відображаються на головній сторінці користувачів. Детальніше можна побачити в таблиці 2.7.

Таблиця 2.7 – Таблиця з базовими плейлистами

default_playlist_id	Int(11)	Ідентифікатор плейлиста

name	Varchar(100)	Назва плейлиста
photo	Varchar(300)	Фотографія плейлиста
song_count	Int(11)	Кількість пісень
duration	Int(11)	Тривалість плейлиста

Також потрібно проміжну таблицю, яка буде містити ідентифікатори пісень та базових плейлистів. Детальніше в таблиці 2.8.

Таблиця 2.8 – Таблиця з піснями базових плейлистів

default_playlist_item_id	Int(11)	Ідентифікатор елементів базових плейлистів
default_playlist_id	Int(11)	Ідентифікатор базового плейлиста
song_id	Int(11)	Ідентифікатор пісні

Необхідно створити таблицю, що містить інформацію про альбоми авторів, включаючи фото, назву альбому, ідентифікатор автора та інше. Детальніше можна побачити в таблиці 2.9.

Таблиця 2.9 – Таблиця з альбомами

album_id	Int(11)	Ідентифікатор альбому
author_id	Int(11)	Ідентифікатор автора
album_name	Varchar(1000)	Назва альбому
album_photo	Varchar(1000)	Фото альбому

Необхідно створити останню таблицю, що містить інформацію про авторів, їх фото та назву. Детальніше можна побачити в таблиці 2.10.

Таблиця 2.10 – Таблиця з авторами

author_id	Int(11)	Ідентифікатор автора
author_name	Varchar(1000)	Назва автора
author_photo	Varchar(1000)	Фото автора

Вище були наведені таблиці з основними сутностями, які будуть застосовуватись в інформаційній системі додатку музичного плеєра. Також потрібно навести приклад фізичної бази даних зі зв'язками, який можна побачити на рисунку 2.7.

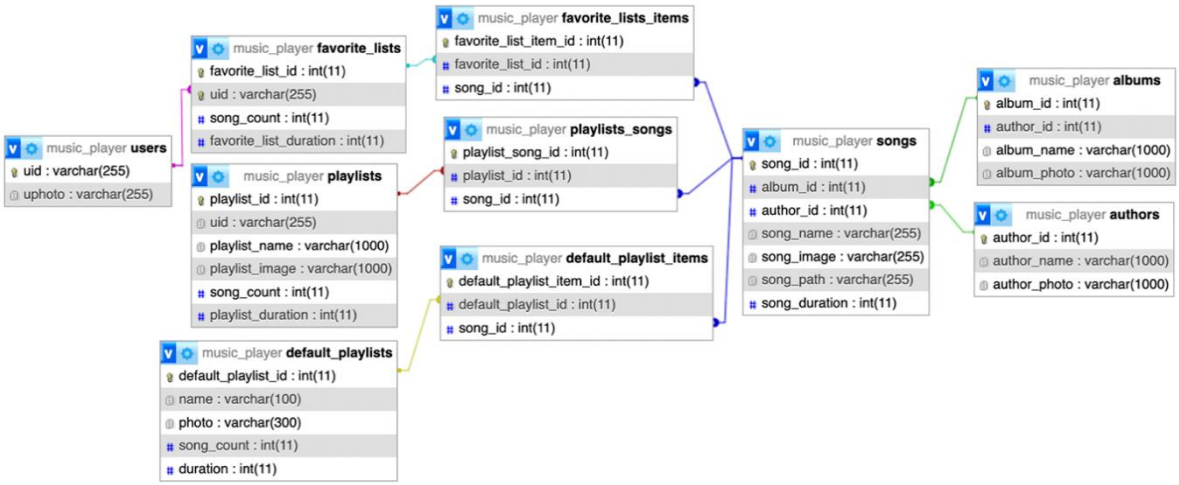


Рисунок 2.7 – Фізична модель бази даних

Використання сервісу Firebase від Google дозволить надати функціонал для реєстрації та авторизації користувачів. У політиці цього сервісу зазначено, що цей функціонал можна використовувати лише у разі використання їхньої бази даних. Це зроблено для запобігання обману та крадіжки паролів зловмисниками. При створенні користувача його інформація буде додаватись до вже існуючої колекції users. Ця інформація буде використовуватись для подальшого порівняння даних при реєстрації та авторизації. Адекватне проєктування моделі бази даних є ключовим етапом в розробці будь-якої інформаційної системи. Правильно спроєктована база даних дозволить забезпечити ефективне зберігання та обробку даних, забезпечуючи при цьому їхню цілісність.изначені відносини між таблицями та зв'язки між сутностями допоможуть уникнути проблем з дублюванням даних та забезпечать ефективний доступ до інформації.

3 РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ СИСТЕМИ

3.1 Обґрунтування вибору технологій реалізації

Основною мовою програмування для реалізації музичного додатку була обрана JavaScript. Ця мова на даний момент є найбільш відомою серед

розробників та активно використовується в різних сферах веброзробки. JavaScript надає всі необхідні інструменти та фреймворки для створення функціональних та інтерактивних вебдодатків. Крім того, завдяки широкій спільноті та численним ресурсам для навчання та підтримки, JavaScript є відмінним вибором для реалізації будь-яких проєктів, включаючи музичні додатки.

В якості основного фреймворка було обрано ReactJS. Цей фреймворк дозволяє створювати високоякісні та динамічні інтерфейси, що є великою перевагою при розробці музичного додатку. Завдяки компонентній архітектурі, ReactJS сприяє зручному та ефективному управлінню станом додатку, що робить його ідеальним вибором для створення складних користувацьких інтерфейсів. ReactJS також підтримує швидку ітерацію та легке тестування компонентів, що важливо для підтримки високої якості продукту.

Для ролі основної бази даних була обрана реляційна СУБД MySQL. MySQL має зрозумілий та простий функціонал, що дозволяє швидко налаштувати та керувати базами даних. Завдяки її масштабованості та надійності, MySQL є відмінним вибором для зберігання великої кількості даних. Крім того, велика кількість документації та підтримка спільноти роблять MySQL зручною для інтеграції з іншими технологіями, що використовуються у додатку. Для реалізації функціоналу авторизації та реєстрації додатково була використана база даних сервісу Firebase, яка надає готові рішення для зберігання та управління профілями користувачів.

Для реалізації дизайну музичного додатку був використаний новітній фреймворк стилів TailwindCSS. На момент реалізації додатку цей фреймворк набув великої популярності та широко використовується серед розробників завдяки своїй простоті та ефективності. TailwindCSS дозволяє швидко створювати стильні та адаптивні інтерфейси, надаючи гнучкі інструменти для кастомізації дизайну. Відсутність необхідності написання кастомних стилів значно скорочує час розробки і спрощує підтримку коду. Крім того, TailwindCSS

легко інтегрується з іншими технологіями, такими як ReactJS, що робить його відмінним вибором для створення сучасних вебдодатків. У деяких компонентах музичного додатку також був використаний фреймворк стилів MUI, який забезпечує широкий спектр готових до використання компонентів.

Для забезпечення кросплатформної роботи додатку був обраний фреймворк ElectronJS. ElectronJS дозволяє інтегрувати вебпродукти у десктопні додатки, забезпечуючи можливість запуску додатку на різних операційних системах, таких як Windows, macOS та Linux. Завдяки сумісності з ReactJS, ElectronJS надає зручний та потужний інструмент для створення високоякісних кросплатформних додатків. Велика документація та активна спільнота допомагають розробникам швидко освоїти цей фреймворк та вирішувати виникаючі питання, що робить процес розробки більш ефективним.

Для реалізації авторизації та реєстрації в музичному додатку використовується сервіс Google – Firebase. Цей сервіс надає готові методи та хуки для створення і управління профілями користувачів, що значно спрощує розробку відповідного функціоналу. Firebase забезпечує високу надійність та безпеку зберігання даних користувачів, а також легко інтегрується з іншими технологіями, використаними у додатку. Завдяки використанню Firebase, розробники можуть зосередитися на інших аспектах додатку, не витрачаючи багато часу на реалізацію базових функцій авторизації та реєстрації.

Всі необхідні бібліотеки та залежності будуть встановлюватися за допомогою пакетного менеджера npm. npm є стандартним інструментом для управління пакетами у світі JavaScript, що дозволяє легко знаходити, встановлювати та оновлювати бібліотеки, необхідні для проєкту. Використання npm забезпечує зручність у керуванні версіями пакетів та їхніх залежностей, що допомагає уникнути конфліктів та забезпечує стабільну роботу додатку.

Програмний код буде реалізовуватися в текстовому редакторі Visual Studio Code. Visual Studio Code є потужним та зручним інструментом для розробки,

який надає широкий спектр функцій для ефективної роботи з кодом. Завдяки вбудованим функціям автодоповнення, інтегрованому терміналу та підтримці численних розширень, Visual Studio Code дозволяє швидко та якісно реалізовувати код.

3.2 Реалізація бази даних

Для початку потрібно створити окремий файл “server.js”, який буде головним серверним файлом для проєкту. У цьому файлі буде налаштовано підключення до системи управління базами даних phpMyAdmin, створено API для обробки запитів до бази даних, а також реалізовано обробку запитів з клієнтської частини додатку. Реалізація цих завдань буде здійснюватися за допомогою фреймворка Node.js. Node.js забезпечує ефективне та асинхронне оброблення запитів, що дозволяє створювати швидкі та масштабовані серверні додатки.

API – це механізми, які дозволяють пересилати певну інформацію між програмними компонентами. Вони забезпечують стандартизований спосіб взаємодії між різними частинами системи, полегшуючи обмін даними та функціональність. У цьому випадку, API будуть відповідати за отримання, створення, оновлення та видалення даних у базі даних, забезпечуючи необхідний функціонал для музичного додатку.

Оскільки немає можливості створити окрему URL для кожної пісні, фотографії автора та альбомів, необхідно зберігати всі ці файли у папці проєкту. Шляхи до цих файлів будуть зберігатися у базі даних, що дозволить легко керувати доступом до медіа-ресурсів. Такий підхід забезпечує централізоване управління файлами та їх доступність для клієнтської частини додатку.

Для того, щоб створити підключення до бази даних, потрібно встановити спеціальне програмне забезпечення під назвою “XAMPP” та запустити всі можливі сервери. XAMPP надає повний набір інструментів для налаштування локального сервера, включаючи Apache, MySQL та PHP, що робить його ідеальним для розробки та тестування вебдодатків. Apache – це популярний

вебсервер, який оброблює HTTP-запити. Він підтримує різноманітні модулі для розширення функціональності, включаючи захист, аутентифікацію та управління сеансами. HTTP-запити — це запити, які веббраузери надсилають на сервери, щоб отримати інформацію, вебсторінку або інші ресурси, такі як зображення чи дані. Далі, в серверному файлі проєкту потрібно імпортувати бібліотеку MySQL та налаштувати підключення до бази даних за допомогою наступного програмного коду:

```
import mysql from 'mysql';
const db = mysql.createConnection({
    host: 'localhost',
    user: 'root',
    password: '',
    database: 'music_player'
})
```

Для реалізації запитів щодо повернення даних з бази даних, потрібно створити API, у яких буде описано запити для отримання необхідної інформації. Реалізовані запити будуть детально описані у додатку А, забезпечуючи чітке уявлення про структуру та функціональність API.

Щоб зберігати фото профілю користувачів, була використана бібліотека “multer”. Ця бібліотека дозволяє зберігати фото в локальному сховищі музичного додатку, забезпечуючи зручний спосіб управління завантаженням файлів. За допомогою спеціального API, одночасно з додаванням фотографії в сховище, буде зберігатися шлях цієї фотографії у базі даних та виводитися під час авторизації користувача. Весь програмний код для зберігання фото профілю користувача в локальному сховищі описано також у додатку А, що забезпечує детальну документацію цього процесу.

Щодо підключення до бази даних профілів сервісу Firebase, це буде описано в наступному пункті при реалізації клієнтської частини.

3.3 Опис розробки

Для того, щоб створити проєкт, потрібно в консолі або в терміналі операційної системи ввести команду “npm create vite@latest” та назву проєкту. Vite – це інструмент, який буде використовуватися з ReactJS. Цей інструмент використовує спеціальні методи для швидкої та простої компіляції проєкту.

Додаток буде розподілений на дві сторінки:

1) сторінка з формою авторизації та реєстрації;
2) сторінка з головним контентом та функціоналом.

Кожна сторінка міститиме кілька компонентів, відображення яких залежатиме від дій користувача.

Головний файл “App.jsx” міститиме два шляхи: перший буде компілювати компонент з формами авторизації та реєстрації, другий – компонент головної сторінки. Програмний код цього файлу виглядатиме таким чином:

```
import MainPage from './pages/MainPage';
import LoginPage from './pages/LoginPage';
import './index.css';

function App() {
  return (
    <Routes>
      <Route path={'/'} element={<LoginPage />}></Route>
      <Route path={'/main'} element={<MainPage />}></Route>
    </Routes>
  )
}

export default App
```

Цей компонент буде імпортований в інший не менш важливий файл, який компілюватиме всю програму. Імпортований компонент “App” буде обгорнутий в спеціальні контексти. Контексти – це спеціальний функціонал ReactJS, який дозволяє створювати змінні стану та методи, і за допомогою хуків

використовувати їх в інших компонентах, що використовують цей контекст. Тому файл “App” у файлі “main.jsx” буде обгорнутий у кілька провайдерів, які надаватимуть контексти всім компонентам проєкту. Провайдер - це шаблон, що передає дані від батьківського компонента до всіх його дочірніх компонентів у дереві React.

Програмний код файлу “main.jsx” виглядатиме наступним чином:

```
import AudioProvider from './context/AudioContext.jsx';
import { FirebaseProvider } from
'./firebase/firebaseProvider.jsx';
import { AuthProvider } from './firebase/authProvider.jsx';

ReactDOM.createRoot(document.getElementById('root')).render(
  <React.StrictMode>
    <FirebaseProvider>
      <AuthProvider>
        <AudioProvider>
          <BrowserRouter>
            <App />
          </BrowserRouter>
        </AudioProvider>
      </AuthProvider>
    </FirebaseProvider>
  </React.StrictMode>
)
```

Сторінка з авторизацією і реєстрацією спочатку генеруватиме форму авторизації. Її можна буде змінити, клацнувши на кнопку "Немає профілю? Зареєструйтесь тут", після чого буде згенеровано компонент реєстрації. Також у файлі контексту авторизації Firebase буде постійна перевірка того, чи користувач вже авторизований. Якщо користувач авторизований, сторінка зміниться на іншу, яка міститиме стрічку мультимедіа, навігаційне меню і головну сторінку, яку користувач зможе змінити, виконавши певні дії.

Програмний код сторінки авторизації буде відображений у додатку Б. Компонент з формою реєстрації відображений у додатку В. Реалізація провайдера, який перевірятиме, чи користувач авторизований, буде в додатку Г. Провайдер, який використовуватиме функціонал авторизації та реєстрації, буде описаний у додатку Ґ.

У додатках Б та В також реалізовано дизайн за допомогою стильового фреймворку TailwindCSS. Для використання стилів потрібно вказати готові стилі фреймворку в атрибуті “class” елемента.

Варто також зазначити, що для того, щоб скористатися функціоналом Firebase, потрібно створити проєкт і базу даних на офіційному сайті Firebase.com. Після чого необхідно отримати спеціальний ключ в json форматі проєкту, щоб все працювало коректно.

Головна сторінка як раніше було сказано буде поділена на три частини:

1) навігаційне меню;
2) стрічка мультимедіа;
3) блок контенту.

Навігаційне меню змінюватиме блок контенту залежно від того, куди перейде користувач за допомогою меню. Стрічка мультимедіа постійно відображатиметься в нижній частині додатка, а меню – в лівій частині.

Кожна кнопка в меню запрограмована на перевизначення стану відображення контенту. Тобто, якщо користувач клацне на відображення плейлиста, то стан, який відповідає за відображення контенту, змінюється і рендериться інший компонент. В коді головної сторінки застосовується конструкція “switch case”, яка змінює рендеринг компонентів. Конструкція switch порівнює вираз із випадками, переліченими всередині неї, а потім виконує відповідні інструкції.

У компоненті меню через метод “map” відображаються всі плейлисти користувача. Користувач також може створити свій плейлист. Метод “map” використовує вказану функцію для кожного елемента масива і повертає змінений

готовий масив. Кожного разу, коли користувач клацає на іконку створення плейлиста, використовується API, де зчитується ідентифікатор користувача, після чого в базу даних додається новий плейлист з усією необхідною інформацією.

Стрічка мультимедіа є одним з найбільш багатофункціональних компонентів цього проєкту. Потрібно правильно реалізувати систему програвання музики. Стрічка складається з декількох частин:

Ліва частина мультимедіа:

– відображається фото пісні;
– назва пісні.

Середня частина мультимедіа:

– повзунок тривалості пісні, який показує поточний прогрес відтворення та дозволяє переміщатися по треку;
– перемішування пісень;
– перемикання на наступну пісню;
– перемикання на попередню пісню;
– зациклення пісні або списку пісень;
– кнопка паузи/відтворення, яка ставить на паузу або запускає пісню.

Права частина мультимедіа:

– повзунок гучності, який дозволяє збільшувати або зменшувати гучність пісні.

Для кожної кнопки створений окремий метод, які реалізовані в контекстному файлі “AudioContext.jsx”. Також після кожного нового включення треку в окремому стані зберігається тип списку пісень. Тобто, якщо користувач ввімкнув пісню з власного плейлиста, то наступна пісня буде відтворюватись саме з цього плейлиста.

Детальніший програмний код головної сторінки можна побачити в додатку Д. Програмний код компонента меню відображений в додатку Е. Код компоненти стрічки мультимедіа можна побачити в додатку Є, а файл

AudioContext.jsx, який містить весь функціонал відтворювання пісень зображений на додатку Ж. Весь дизайн компонентів описаний у додатках.

Наступним кроком є створення компонентів, які будуть містити головний контент.

Потрібно реалізувати наступні компоненти:

– профіль користувача – цей компонент міститиме фото профілю користувача, яке можна видалити або завантажити нове, а також кнопку для виходу з профілю;

– головний компонент – міститиме базові плейлисти та список хітів;

– компонент пошуку – міститиме пошукову стрічку, де можна знайти будь-яку пісню, автора або альбом;

– улюблений список – компонент, який міститиме улюблені треки користувача;

– компонент плейлиста користувача – цей компонент буде відображати пісні з певного плейлиста, обраного користувачем;

– базовий плейлист – компонент, який міститиме вже сформований список пісень і буде відображатись як рекомендація для кожного користувача додатку;

– компонент автора – цей компонент міститиме фото та назву користувача, його відомі треки та альбоми;

– компонент альбома – цей компонент міститиме фото, назву альбома, кількість пісень і список треків самого альбому.

Також потрібно створити окремий компонент під назвою “track.jsx”, який буде отримувати інформацію таку як: фото, назва, шлях, тривалість, ідентифікатор тощо, і використовувати ці дані в окремій компоненті. Коли потрібно вивести список, необхідно передати список пісень і застосувати компонент “track” до кожного елемента масиву за допомогою спеціального методу “map”, який виконує дії над певним масивом.

У компонент “track” потрібно додати іконку улюбленої пісні, клацнувши на яку, користувач додасть цю пісню до списку улюблених пісень.

Також у цього компонента буде випадний список, через який можна додати цю пісню до будь-якого плейлиста користувача, видалити цю пісню з плейлиста (доступно тільки на сторінці плейлиста користувача), або перейти до альбому пісні.

Базовий плейлист буде відображати своє фото та назву. Клацнувши на них, користувача перенесе на сторінку зі списком цього плейлиста. Програмний код того, як виглядатиме елемент базового плейлиста, буде виглядати наступним чином:

```
const DefaultPlaylist = (props) => {
  const { name, photo } = props;

  return (
    <div className="flex items-center justify-center">
      <img src={photo} alt={name} className="w-36 h-36
object-cover" />
      <div className="w-32 h-36 flex items-center justify-
center transform rotate-90 -ml-12">
        <p className="text-xl text-white hover:text-blue-
600">{name}</p>
      </div>
    </div>
  );
};

export default DefaultPlaylist;
```

Так само, як і базовий плейлист, буде відображатись список авторів у пошуковій стрічці. Тому елемент компонента автора буде виглядати наступним чином:

```
const Author = (props) => {
```

```
    const { author_name, author_photo } = props;

  return (
   <div className="flex flex-col justify-center m-3">
    <img src={author_photo} alt={author_name} className="w-36"/>
    <p className="text-3xl">{author_name}</p>
   </div>
    );
   };

   export default Author;
```

Останній необхідний компонент, який буде приймати масив інформації, — це компонент альбому. Програмний код буде виглядати наступним чином:

```
   const Album = (props) => {
     const { album_name, album_photo } = props;

     return (
       <div className="flex flex-col justify-center m-3">
         <img src={album_photo} alt={album_name} className="w-
36" />
         <p className="text-3xl text-white hover:text-blue-
600">{album_name}</p>
       </div>
     );
   };
   export default Album;
```

Також було створено окремі функції JavaScript, які форматують тривалість пісні. У базі даних тривалість пісні зберігається в секундах. Відображення повинно бути у хвилинах і секундах. Якщо це відображення тривалості плейлиста або улюбленого списку, то повинні також відображатися години.

Тому для форматування тривалості треку був використаний такий програмний код:

```
import moment from "moment";

export default seconds => {
    if (seconds === undefined) {
        return "00:00";
    } else {
        const duration = moment.duration(seconds, 'seconds');
        const hours = duration.hours();
        const format = hours > 0 ? "HH:mm:ss" : "mm:ss";
        return moment.utc(seconds * 1000).format(format);
    }
};
```

Для відображення часу плейлиста або улюбленого списку користувача був використаний такий програмний код:

```
import moment from "moment";

export default seconds => {
    if (seconds === undefined) {
        return "00:00";
    } else {
        const duration = moment.duration(seconds, 'seconds');
        const hours = duration.hours();
        const minutes = duration.minutes();
        const format = hours > 0 ? `${hours} год ${minutes}
хв` : `${minutes} хв`;
        if (duration.seconds() > 0) {
            return `${format} ${duration.seconds()} сек`;
        } else {
            return format;
        }
    }
```

```
};
```

Програмний код головної сторінки музичного додатку буде досить простим. Він буде використовувати два методи “map” та змінні стану з файла “AudioContext.jsx”. Програмний код буде виглядати наступним чином:

```
import { AudioContext } from "../context/AudioContext";
import "../css/main.css";
import Track from "./track";
import DefaultPlaylist from "./defaultPlaylist";
const TrackList = () => {
    const { songs, defaultPlaylists, onChangeMenuItem,
onChangeDefPlaylist } = useContext(AudioContext);
    return (
        <div className='bg-[#212121] h-full p-2 text-white'>
            <div className='bg-[#292929] rounded-2xl h-full
p-3 flex flex-col'>
                <div className="flex flex-row m-3">
                    {defaultPlaylists.map((playlist, index)
=> (
                        <div onClick={() => {
onChangeMenuItem('def'); onChangeDefPlaylist(playlist); }}
key={`${playlist.default_playlist_id}-${index}`}>
                            <DefaultPlaylist key={index}
{...playlist} /></div>))}</div>
                <p className="text-4xl m-3 -mt-1">Хіти</p>
                <div className="overflow-auto border-t-2
border-white">
                    {songs.map((track, index) => (
                        <Track key={index} {...track}
playlistType="standartList" />
                    ))}
                </div>
            </div>
```

```
    );
};
export default TrackList;
```

Програмний код компонента треку буде відображений в додатку Т. Код компоненти випадного списку треків буде в додатку С. Програмний код відображення списку базового плейлиста та всієї необхідної для цього інформації можна побачити в додатку І. Відображення програмного коду списку пісень плейлиста користувача можна побачити в додатку Р. Програмний код відображення списку пісень альбому можна побачити в додатку У. Код відображення списку пісень автора буде в додатку К. Програмний код, який відображує улюблені пісні користувача, можна побачити в додатку Л.

Потрібно створити функціонал, який дозволить користувачу змінювати назву власного плейлиста. При натисканні на назву плейлиста, повинне відкритись модальне вікно з формою, де в полі буде писати стара назва плейлиста, яку можна змінити та зберегти. Реалізація програмного коду модального вікна буде відображена в додатку М.

Щоб здійснити останній крок і адаптувати проєкт як кросплатформний додаток, спочатку необхідно встановити бібліотеку “electron”. Це можна зробити, виконавши команду "npm install electron" у терміналі. Після цього треба налаштувати скрипт запуску в файлі “package.json”. У об'єкті скрипта вписати команду “’start’: ‘electron .’”, яка дозволить запустити проєкт як додаток через термінал. Проте перед цим необхідно створити файл “electron.cjs” та внести у нього наступний код:

```
const { app, BrowserWindow } = require('electron');

const createWindow = () => {
    const win = new BrowserWindow({
        width: 1300,
        height: 800,
```

```
            resizable: false
        })
        win.loadURL('http://localhost:6969')
    }

    app.whenReady().then(() => {
        createWindow();
    })

    app.on('activate', () => {
        if(BrowserWindow.getAllWindows().length===0)
createWindow()
    })

    app.on('window-all-closed', () => {
        if (process.platform !== 'darwin') app.quit()
    })
```

Для запуску повноцінної програми спочатку потрібно запустити сервер за допомогою команди “node server.js”. Після цього можна запустити локальний режим розробника проєкту, введенням команди “npm run dev”. І на останок, щоб запустити окреме вікно музичного додатку, треба ввести команду “npm start”.

3.4 Тестування програмного забезпечення

Тестування є дуже важливим етапом розробки програмного забезпечення. На цьому кінцевому етапі знаходяться всі проблеми програми: баги та помилки. Визначаються основні проблеми та результати роботи програми, яка була створена. Спочатку потрібно перевірити музичний додаток на його кросплатформеність та функціональність. Для цього за допомогою фреймворку Electron, який був згаданий раніше, було запущено програму в комп’ютерних операційних системах Windows та MacOS (рис. 3.1, рис. 3.2).

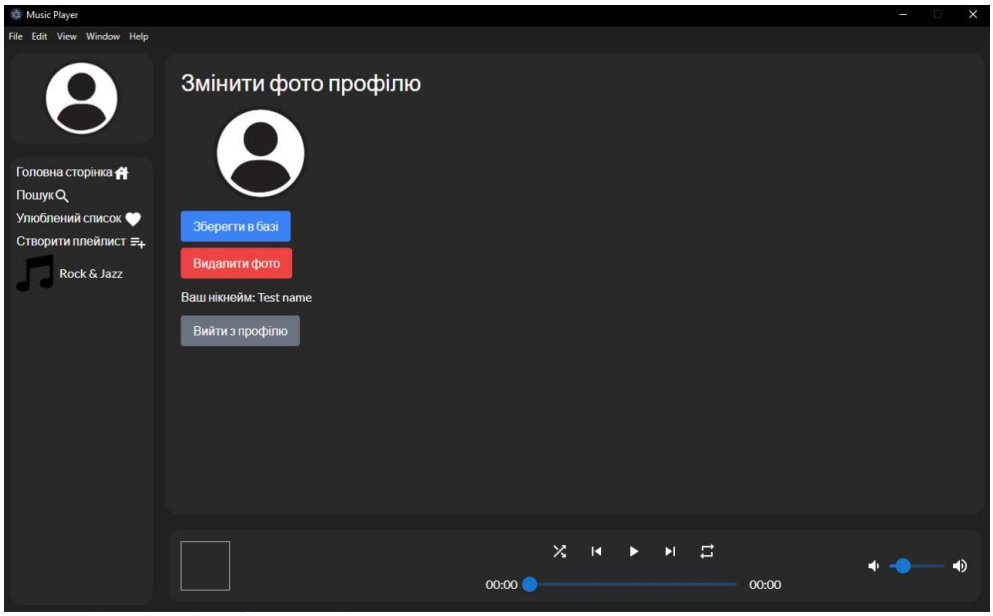


Рисунок 3.1 – Тестування запуску програми на ОС Windows

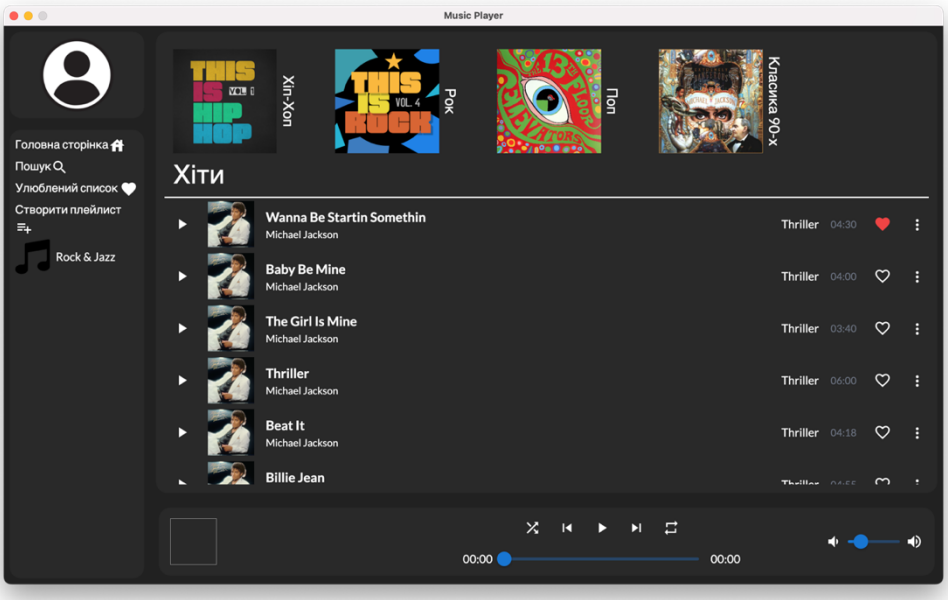


Рисунок 2.2 – Тестування запуску програми на MacOS

Після успішного запуску можна перейти до наступного етапу перевірки функціоналу. Необхідно перевірити наступний функціонал:

– перевірка реєстрації;
– перевірка базового плейлиста;

– перевірка зміни фото профілю;
– перевірка улюбленого списку;
– перевірка пошуку;
– перевірка користувацького плейлиста;
– перевірка зміни назви користувацького плейлиста;
– перевірка стрічки мультимедіа.

Спробуємо створити нового користувача та навмисно неправильно ввести електронну адресу (рис. 3.3). Форма виявляє помилку і не дозволяє виконати наступні дії.

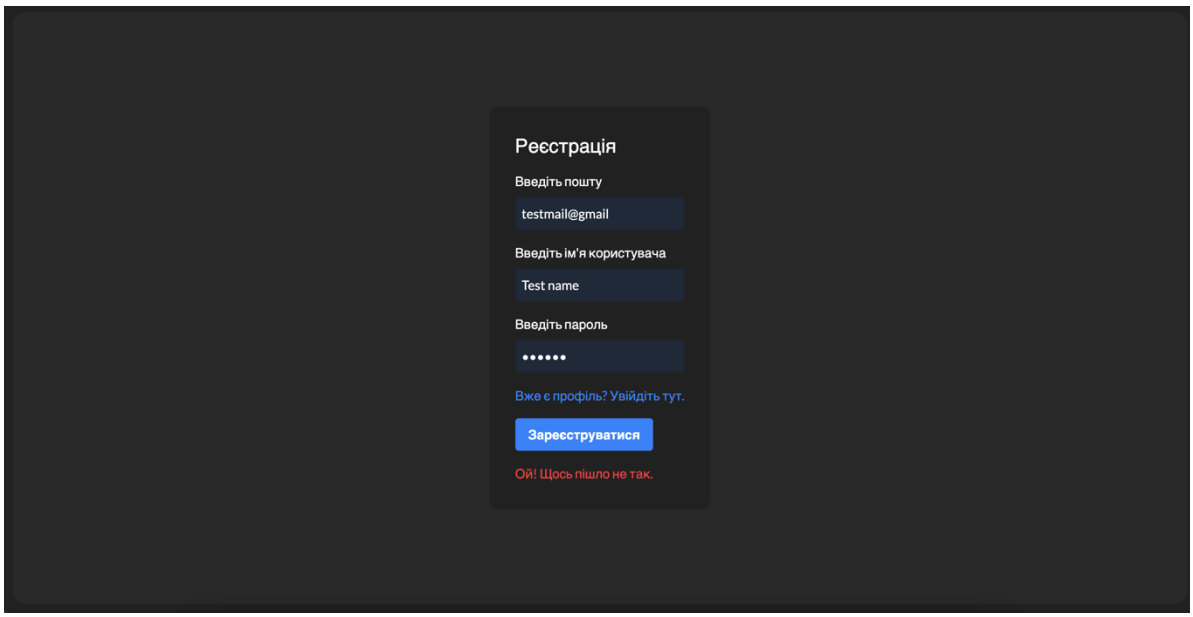


Рисунок 3.3 – Тестування форми реєстрації

Якщо спробувати ввести коректні дані, система пропустить реєстрацію і одразу після реєстрації автоматично авторизує користувача (рис. 3.4, рис. 3.5).

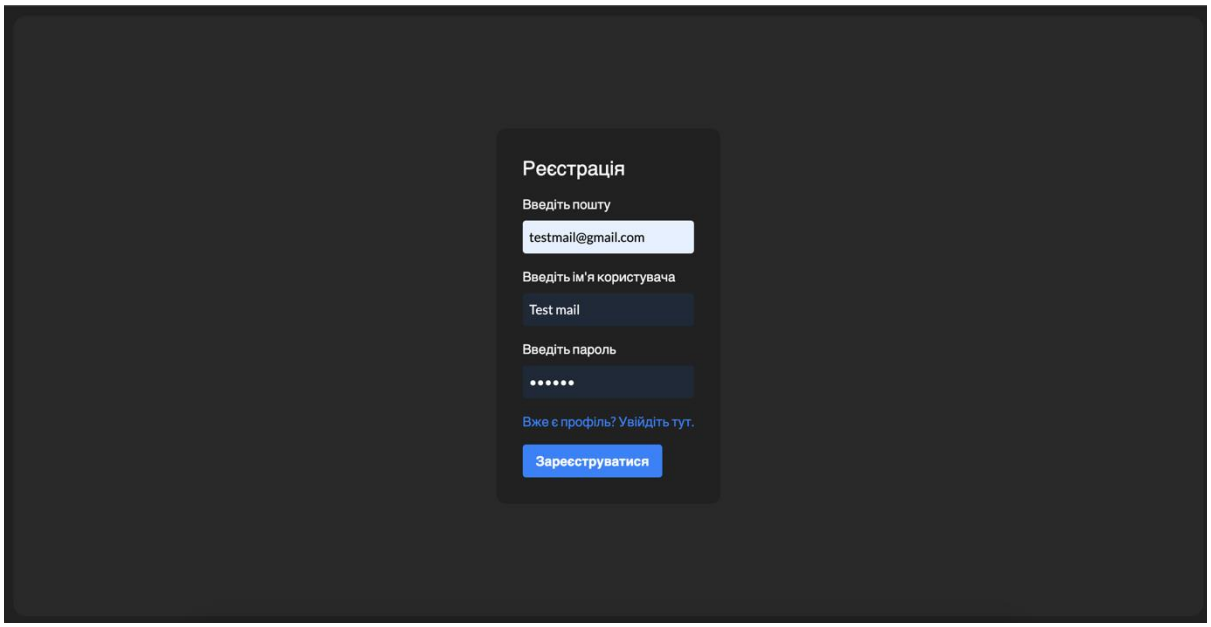


Рисунок 3.4 – Тестування форми реєстрації

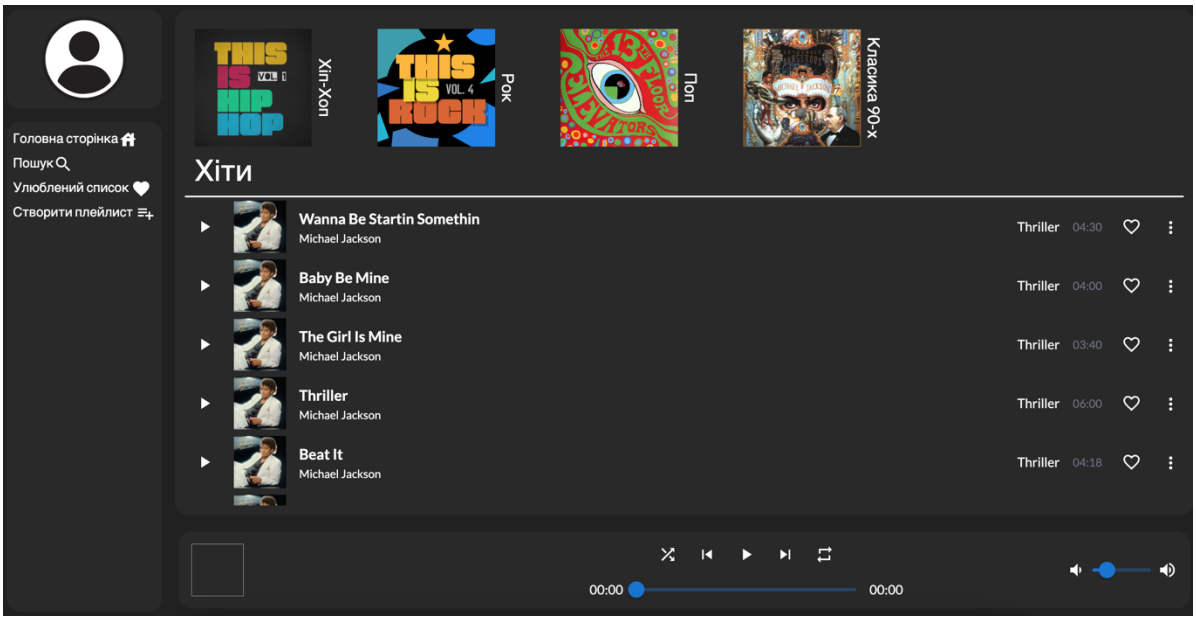


Рисунок 3.5 – Успішна авторизація

Одразу після успішної авторизації користувача переадресовують на головну сторінку, де можна побачити базові плейлисти та список пісень. Наступним кроком спробуємо змінити фотографію профілю користувача. Для цього потрібно натиснути на базову фотографію профілю, після чого

користувача переадресує на сторінку профілю. Далі необхідно на цій сторінці натиснути на іконку та завантажити нову фотографію (рис. 3.6, рис. 3.7).

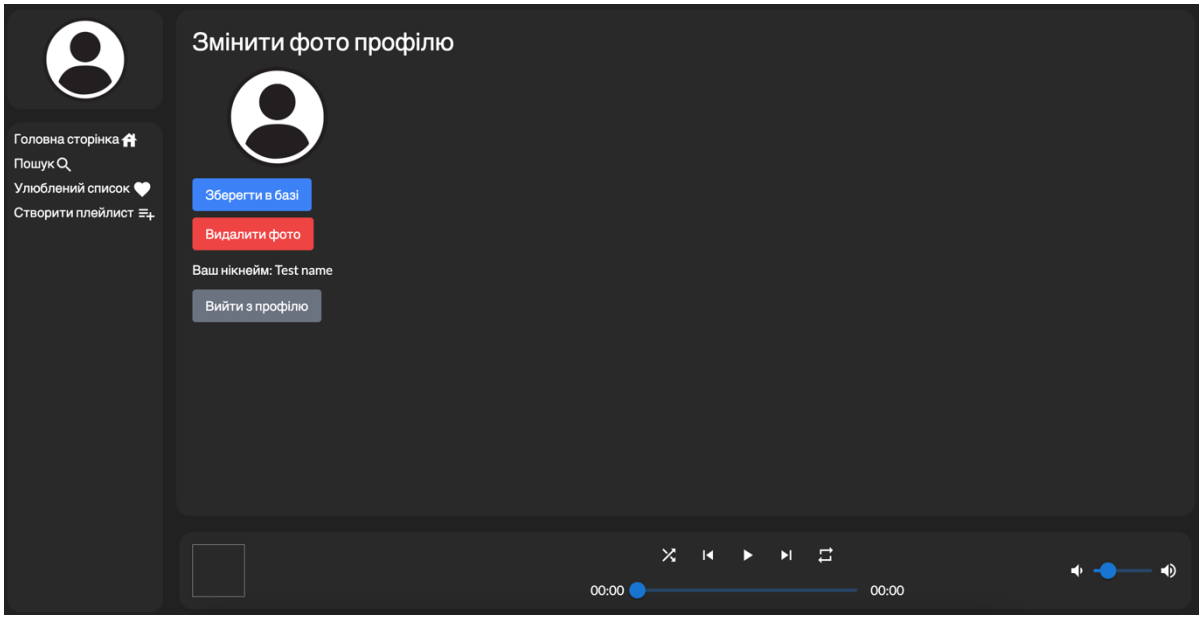


Рисунок 3.6 – Профіль користувача

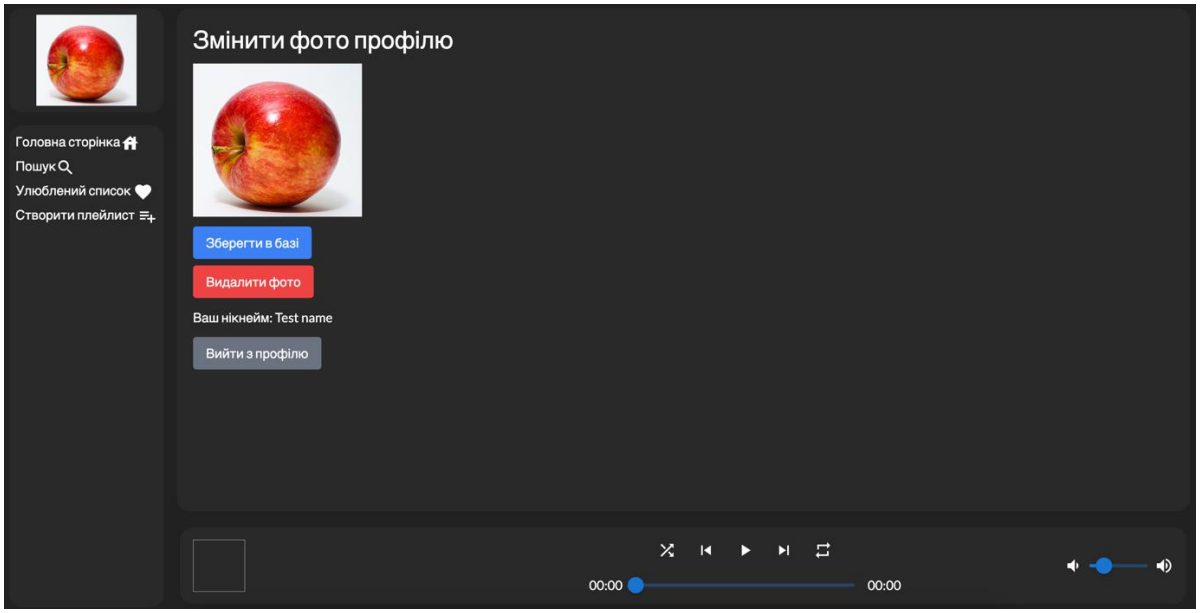


Рисунок 3.7 – Тестування завантаження нової фотографії

Фотографію також можна видалити, натиснувши на кнопку “Видалити фото”. Всі процеси виконуються одразу на очах користувача, що дозволяє не оновлювати сторінку після певних змін.

Наступним кроком потрібно перевірити улюблений список користувача та його функціонал. Перейшовши на головну сторінку, потрібно натиснути на іконку сердечка будь-якого треку, щоб сформувати власний список. Після цього слід перейти на сторінку улюбленого списку та перевірити його наявність (рис. 3.8).

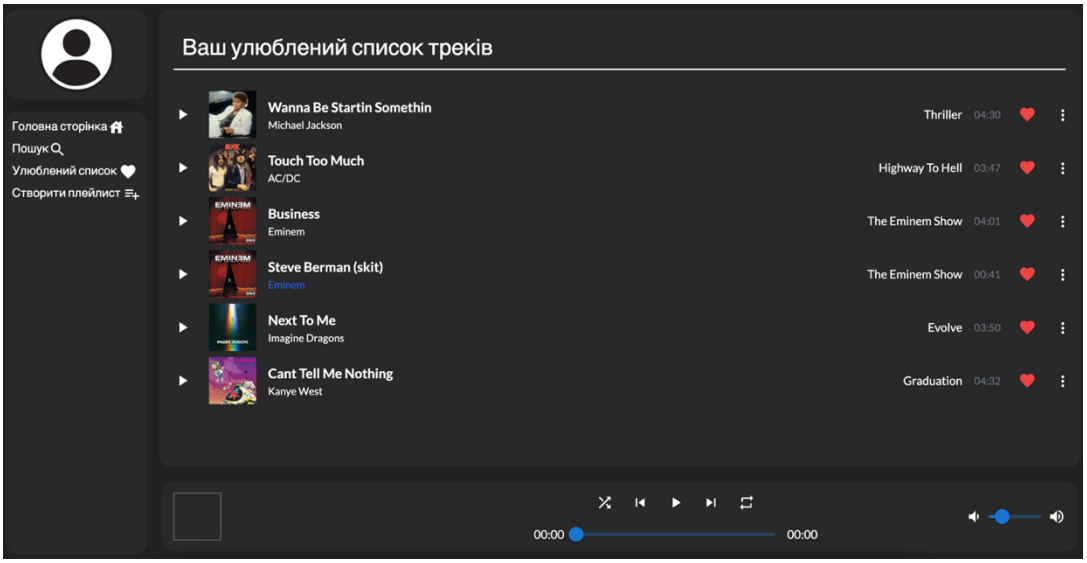


Рисунок 3.8 – Тестування функціоналу улюбленого списку

Наступним кроком необхідно перевірити працездатність базового плейлиста. Для цього потрібно перейти на головну сторінку та натиснути на будь-який плейлист, після чого повинно відобразитися список пісень цього плейлиста (рис. 3.9).

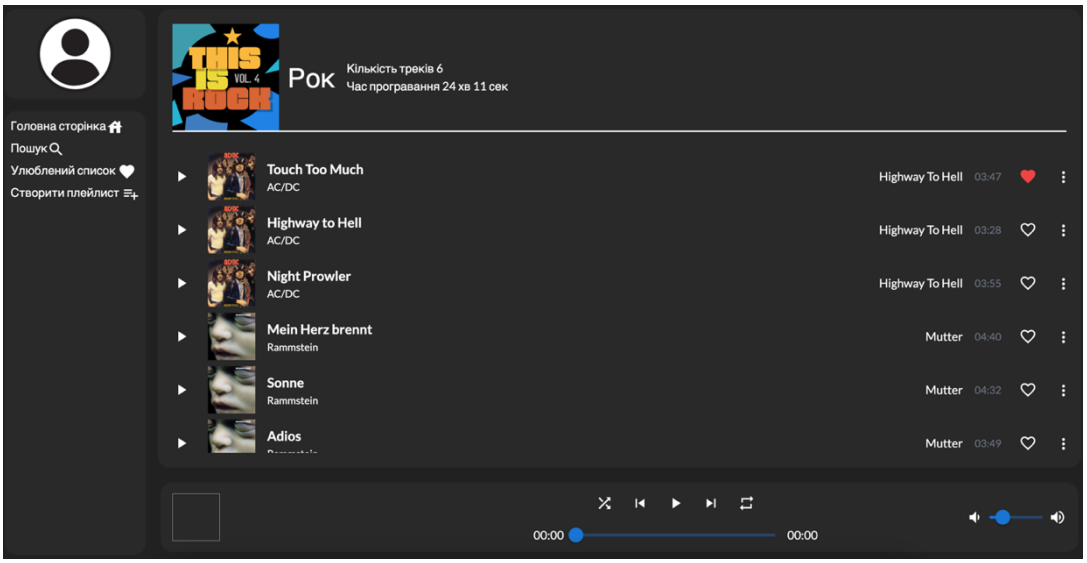


Рисунок 3.9 – Тестування базового плейлиста

Далі потрібно перевірити функціонал пошуку. Перейшовши за допомогою меню на сторінку пошуку, можна побачити три кнопки, які шукають окремо пісню, автора та альбом (рис. 3.10, рис. 3.11, рис. 3.12).

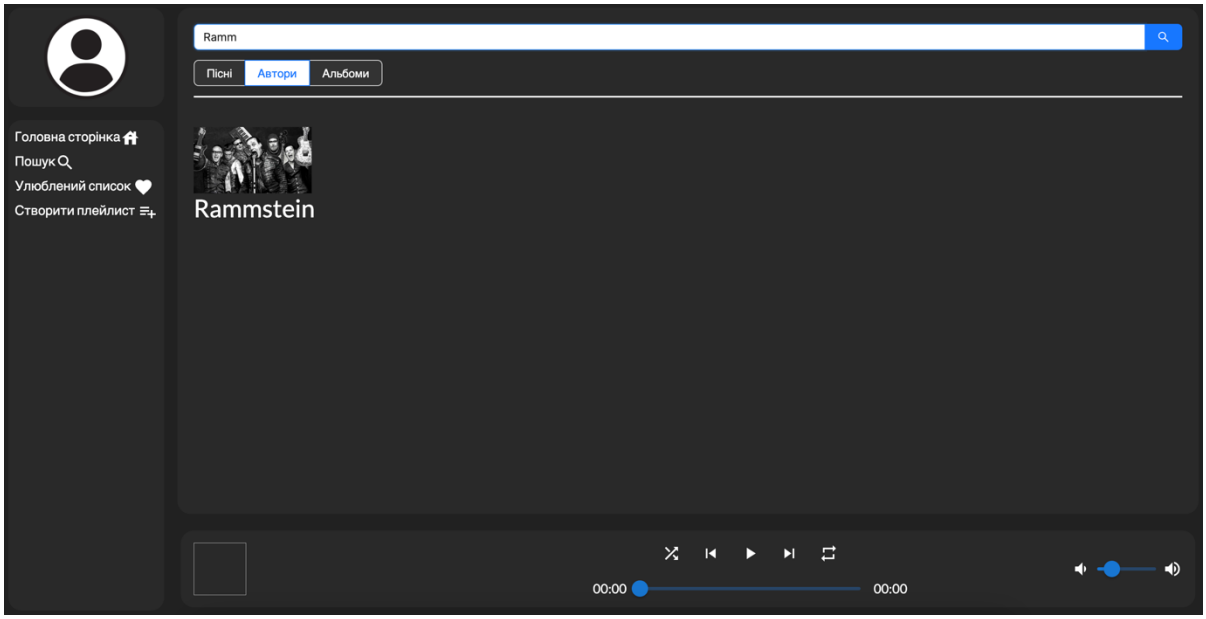


Рисунок 3.10 – Тестування пошуку автора

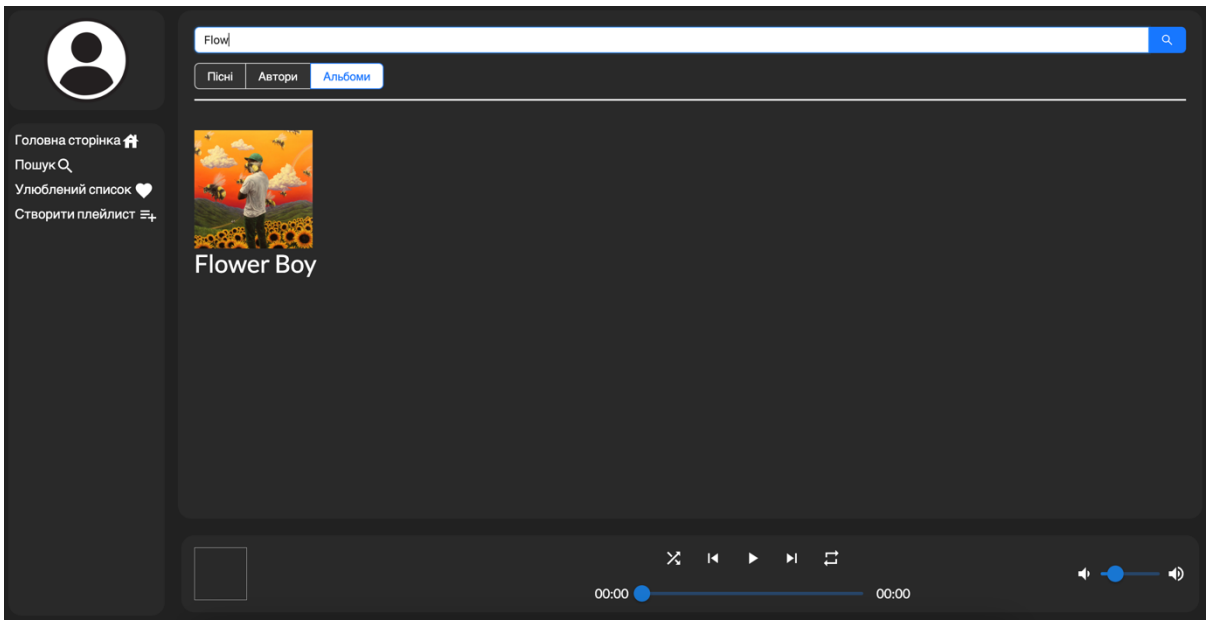


Рисунок 3.11 – Тестування пошуку альбому

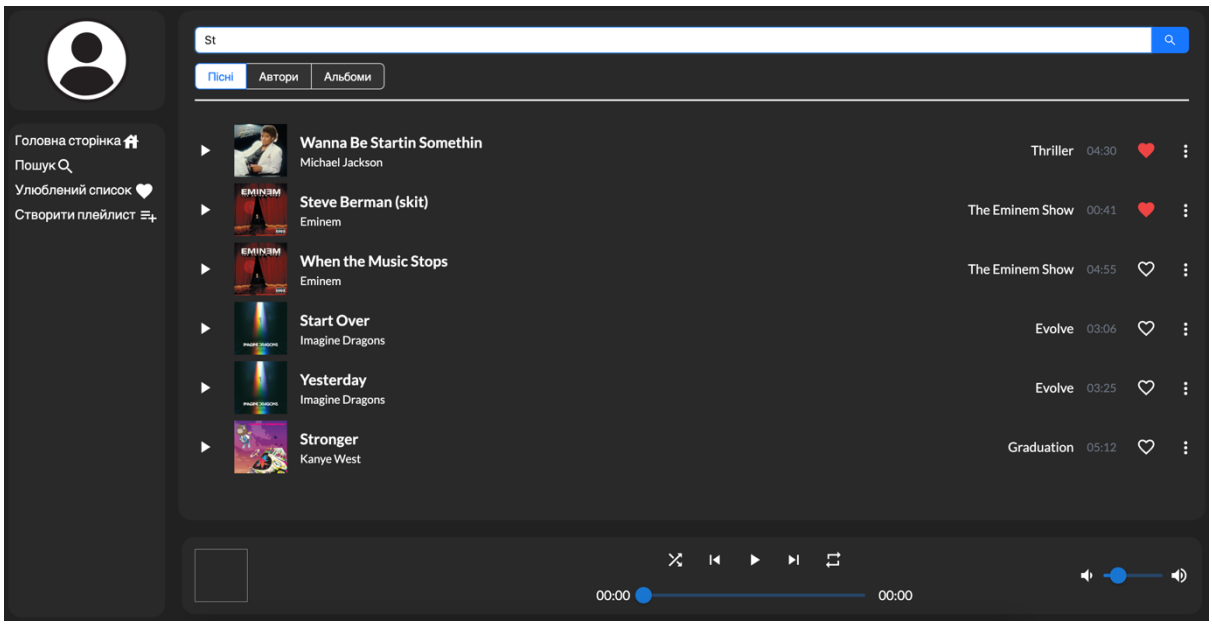


Рисунок 3.12 – Тестування пошуку пісні

Наступним кроком потрібно перевірити функціонал користувацького плейлиста. Для цього потрібно в меню натиснути на іконку створення нового плейлиста. Після цього повинен створитися новий плейлист за шаблоном. Натиснувши на плейлист, можна змінити його назву, відкривши модальне вікно з формою для зміни (рис. 3.13, рис. 3.14).

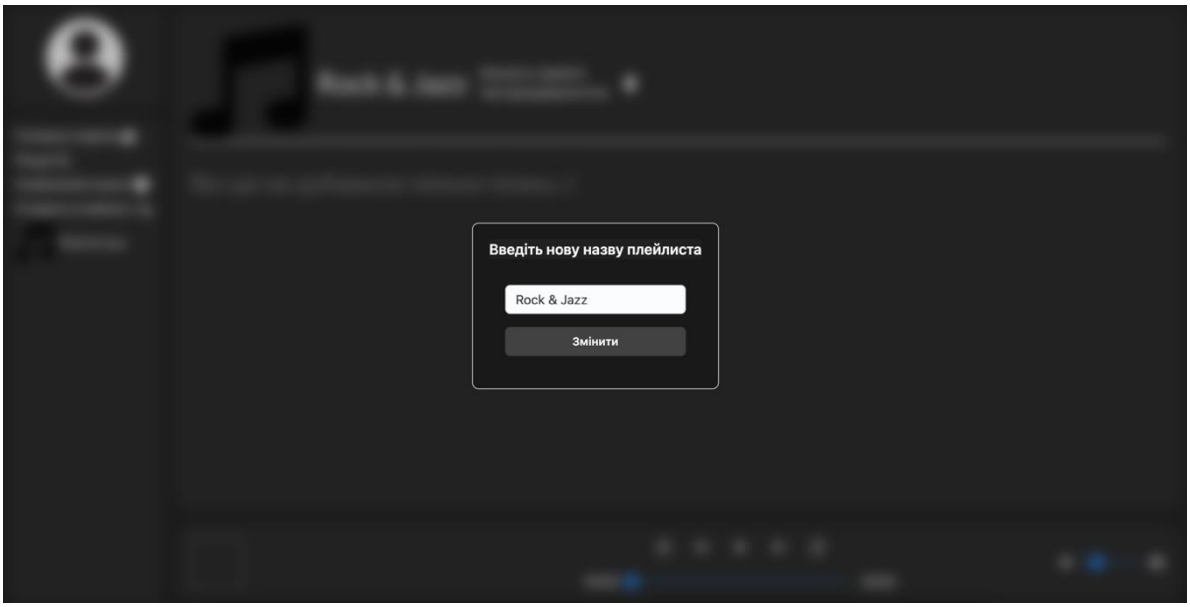


Рисунок 3.13 – Модальне вікно для зміни назви плейлиста

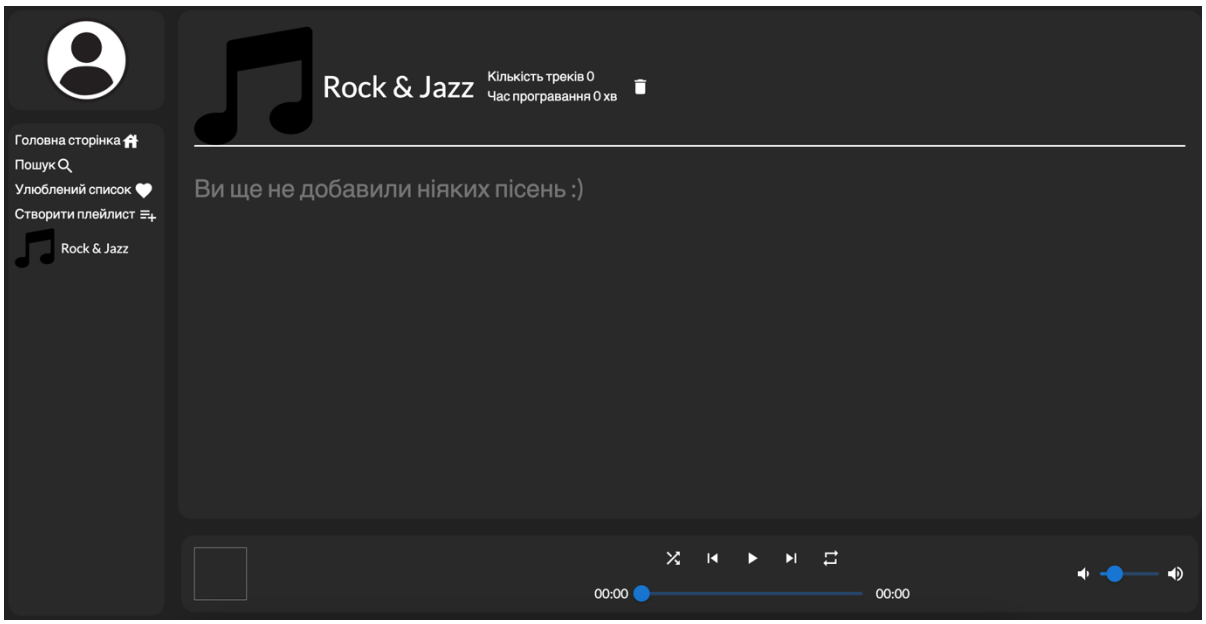


Рисунок 3.14 – Перевірка зміни назви користувацького плейлиста

Щоб перевірити функціонал додавання пісні в плейлист, спробуємо додати одну й ту саму пісню двічі. Це допоможе перевірити, чи відображаються дубльовані пісні у плейлисті та чи виникають які-небудь проблеми під час додавання. Також, важливо переконатися, що система правильно обробляє такі сценарії, забезпечуючи коректну роботу функціоналу (рис. 3.15, рис. 3.16).

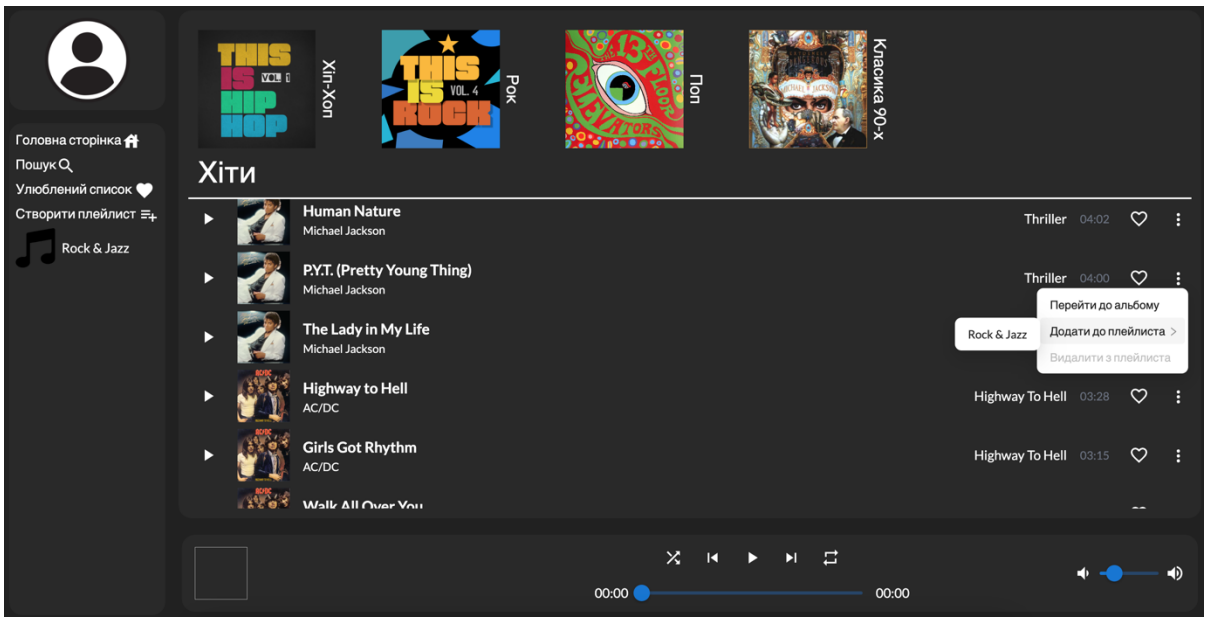


Рисунок 3.15 – Перевірка додавання пісні в плейлист

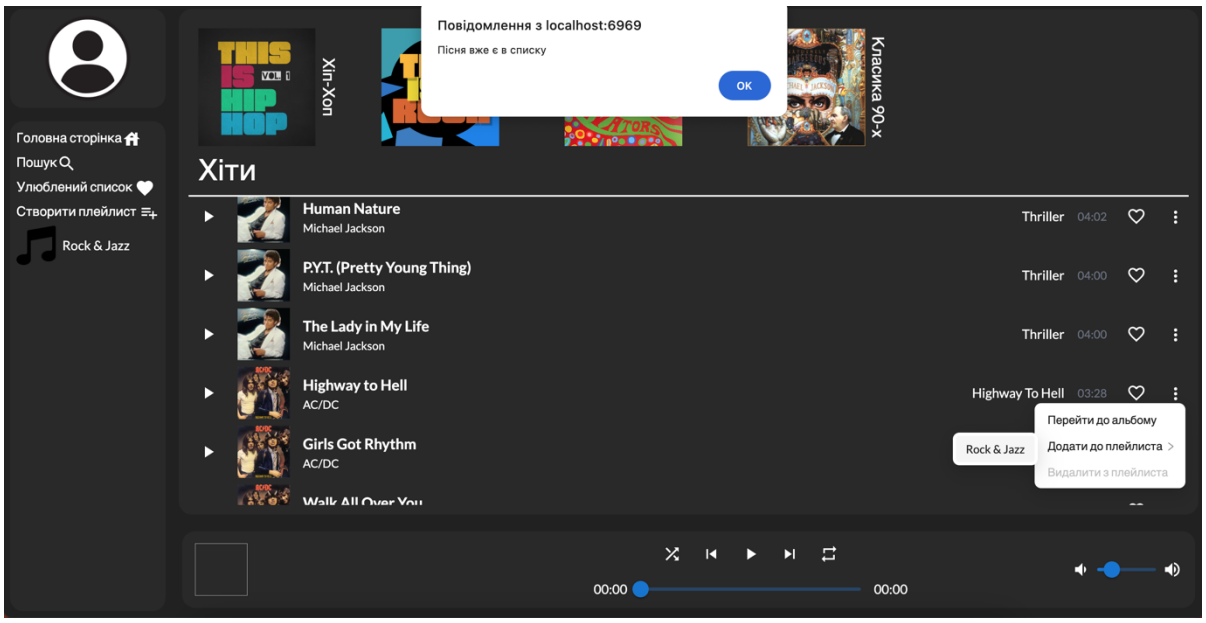


Рисунок 3.16 – Перевірка додавання пісні в плейлист

Отже, система контролює процес дублювання пісні в плейлисті і повертає попередження користувачу. Далі потрібно перейти в користувацький плейлист і перевірити, чи присутній список пісень (рис. 3.17).

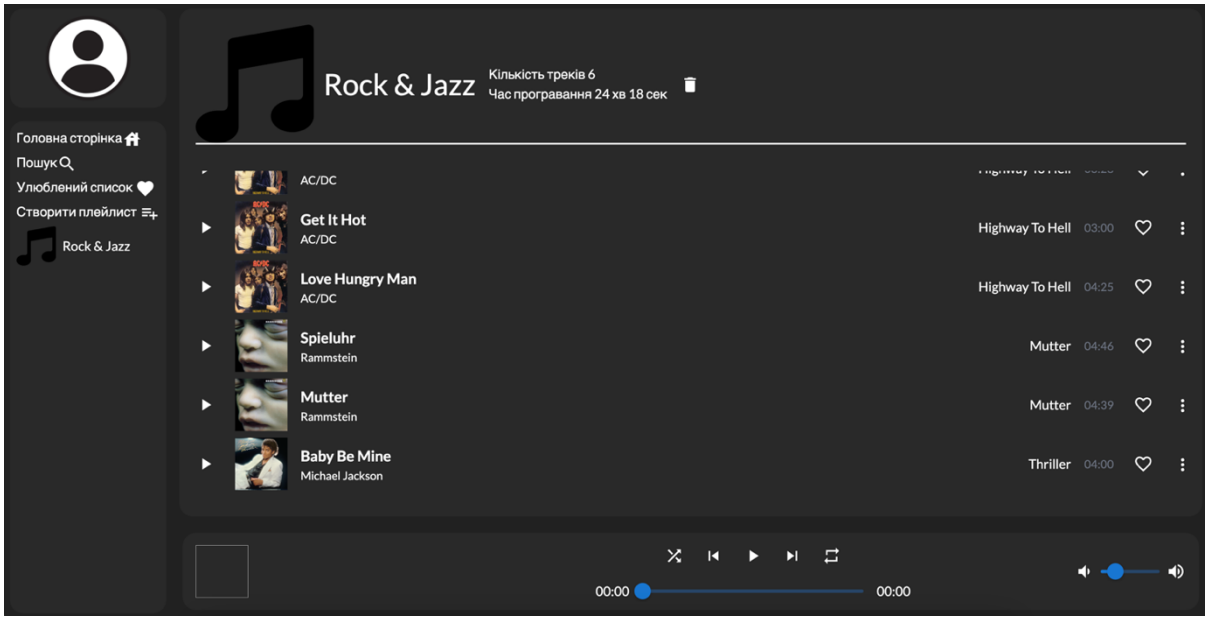


Рисунок 3.17 – Користувацький плейлист

Наступним та останнім кроком буде перевірка видалення пісень з плейлиста за допомогою випадного меню кожного треку (рис. 3.18).

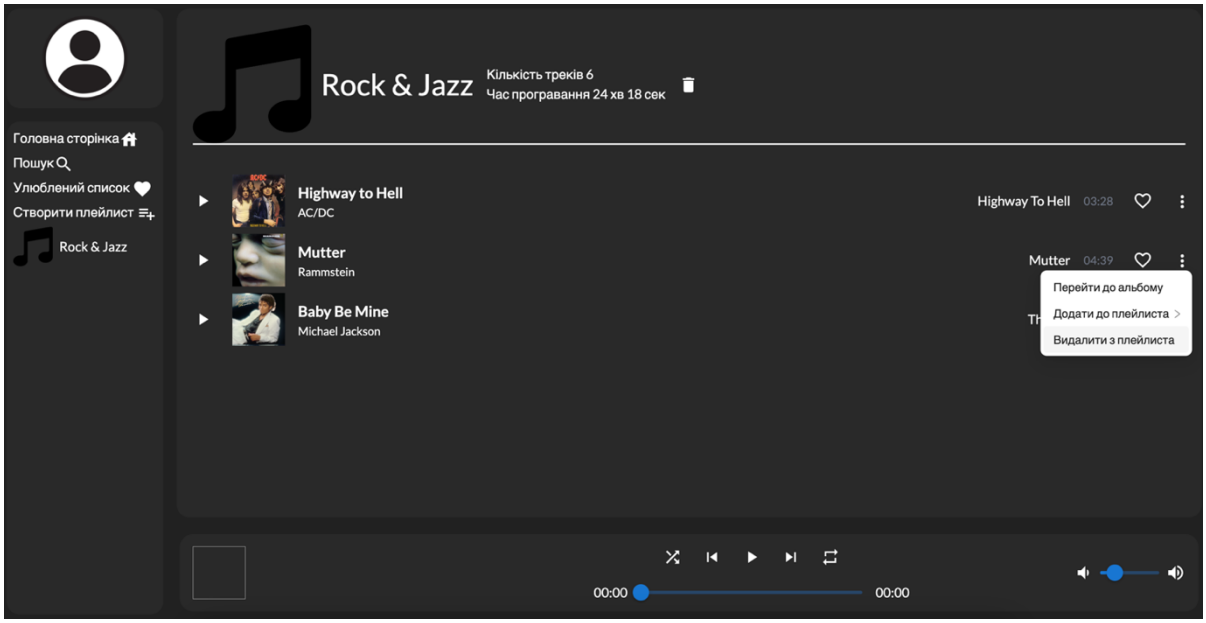


Рисунок 3.18 – Тестування видалення пісень з плейлиста

Після завершення всіх кроків тестування ПЗ підтверджено його високу якість та готовність до випуску на ринок.

4 ТЕХНІКО-ЕКОНОМІЧНЕ ОБҐРУНТУВАННЯ

4.1 Аналіз ринку

Додаток музичного плеєра, який був реалізований, не є новим та ексклюзивним на ринку. Однак, цей проєкт має потенціал стати успішним стартапом, оскільки музичні плеєри завжди користуються популярністю. Можливим ринком збуту можна визначити країни Європи та США, де користувачі постійно шукають нові зручні рішення для прослуховування музики. Також вартим уваги є азійський ринок, де високий попит на різноманітні технологічні новинки, що може сприяти стрімкому зростанню популярності додатку. Латинська Америка та Індія також представляють перспективні ринки завдяки постійному зростанню інтересу до цифрових продуктів, що може значно збільшити попит на музичний плеєр.

Очікується, що попит буде високим, оскільки користувачі завжди цінують комфорт та простоту у використанні додатків для прослуховування музики. На початковому етапі виходу продукції можна залучити та зацікавити користувачів безкоштовним тарифом використання та відсутністю реклами. Ця стратегія дозволить створити початкову базу користувачів та стимулювати їх випробувати новий продукт. Якщо користувач не захоче оплачувати тариф, можна залишити безкоштовний план, на якому час від часу будуть програватися рекламні засоби, що дозволить монетизувати продукт навіть серед тих, хто не готовий платити за преміум-підписку.

Для ефективної монетизації та залучення вигідних рекламних пропозицій потрібно обрати досвідченого менеджера, який буде контролювати потік реклами та обирати лише найбільш вигідні пропозиції. Це дозволить забезпечити стабільний дохід від реклами без шкоди для досвіду користувачів.

Основними конкурентами на ринку є такі компанії, як Spotify, Apple Music та YouTube Music. Кожен з цих плеєрів вже має велику аудиторію, яка активно використовує їх додатки. Однією з можливих стратегій перехоплення активної

аудиторії конкурентів є зменшення ціни тарифного плану, що може залучити користувачів, які шукають більш вигідні умови.

Додатково, варто розглянути можливість впровадження унікальних функцій та інновацій, які відрізнятимуть додаток від конкурентів та приваблюватимуть користувачів. Це можуть бути покращена якість звуку, ексклюзивні плейлисти, інтеграція з іншими популярними сервісами або нові соціальні функції, що дозволять користувачам ділитися музикою та враженнями. Завдяки такому підходу, музичний плеєр зможе знайти свою нішу на ринку та забезпечити стійке зростання популярності.

4.2 Розрахунок витрат на проєктування

Щоб правильно врахувати всі нюанси податків, бухгалтерії та законодавства України при розрахунку загальної вартості проєкту, необхідно скласти таблицю розрахунку заробітної плати проєктувальників. Нижче наведена прикладна таблиця (табл. 4.1), яка включає основні елементи, необхідні для коректного розрахунку заробітної плати.

Таблиця 4.1 – Розрахунок заробітної плати проєктувальників

Посада	Оклад	Відрахування	Кількість		Сума
	грн/міс	грн/міс	чол.	місяців	з/п, грн
Розробник	15000	2925	1	2	24150
	Усього зарплати:				24150

Для початку розрахунку загальної вартості проєкту, використовуючи основний оклад у 15000 гривень, потрібно розрахувати:

- Податок на доходи фізичних осіб: 15000 грн * 18% = 2700 грн.
- Військовий збір: 15000 грн * 1,5% = 225 грн.
- Єдиний внесок: 15000 грн * 22% = 3300 грн.

Відрахування – 2925 грн. (2700 грн. + 225 грн.).

Виплата працівникові – 12075 грн. (15000 грн. - 2700 грн. - 225 грн.).

Наступним кроком потрібно вирахувати:

– Відрахування на соціальні потреби: 15000 грн. * 22% = 3300 грн.

– Контрагентські роботи і послуги: 15000 грн. * 15% = 2250 грн.

– Витрати на відрядження: 3000 грн.

– Інші прямі витрати: 15000 грн. * 45% = 6750 грн.

– Усього прямих витрат: 39450 грн. (24150 грн. + 3300 грн. + 2250 грн. + + 3000 грн. + 6750 грн.).

– Накладні витрати: 39450 грн * 30% = 11835 грн.

– Планові накопичення: (39450 грн. + 11835 грн.) * 20% = 10257 грн.

– Усього, кошторисна вартість проєкту: 39450 грн. + 11835 грн. + + 10257 грн. = 61542 грн.

– Податок на додану вартість: 61542 грн. * 20% = 12308,5 грн.

– Загалом, договірна ціна розробки Зп: 24150 грн. + 3300 грн. + 2250 грн. + 3000 грн. + 6750 грн. + 39450 грн. + 11835 грн. + 10257 грн. + 61542 грн. + + 12308,5 грн. = 119441 грн.

Детальний опис розрахунків, їхні найменування, обґрунтування, або ж іншими словами кошторис витрат на проєктування можна побачити в додатку П таблиця П.1.

Отже, загальна вартість проєкту складає 174842,5 гривень, включаючи всі основні та додаткові витрати на заробітну плату, соціальні відрахування, послуги контрагентів, витрати на відрядження, інші прямі та накладні витрати, планові накопичення та податок на додану вартість.

4.3 Обґрунтування необхідності розробки

Кросплатформний музичний плеєр дозволить користувачам швидко і зручно знаходити та слухати улюблені треки, забезпечуючи при цьому високий рівень комфорту.

Крім того, розробка музичного додатку відкриває можливості для отримання вигоди через монетизацію. Це може включати рекламу, платні підписки, продаж ексклюзивного контенту або інші форми доходу, які можуть допомогти розробнику отримувати стабільний прибуток. Створення такого додатку не лише задовольнить потреби користувачів, але й відкриє нові можливості для отримання прибутку або ж стане перспективним бізнес-проєктом.

Музика також відіграє важливу роль у підвищенні продуктивності та покращенні навчального процесу. Дослідження показують, що прослуховування музики може сприяти концентрації, покращенню настрою та зниженню стресу, що є важливими аспектами для ефективної роботи чи навчання. Музичний додаток, що пропонує різноманітні плейлисти для різних видів діяльності, таких як навчання, робота або відпочинок, може стати незамінним інструментом для користувачів. Він допоможе створити сприятливу атмосферу для виконання завдань та досягнення особистих чи професійних цілей. Таким чином, розробка такого додатку не лише задовольняє базову потребу в доступі до музики, але й сприяє підвищенню ефективності та якості життя користувачів, що ще більше підтверджує необхідність його створення.

Таким чином, необхідність розробки кросплатформного музичного додатку є доволі високою.

ВИСНОВКИ

При виконанні кваліфікаційної роботи було реалізовано кросплатформний музичний додаток. Визначено головні цілі, вимоги та функціонал, які були реалізовані в додатку. Також було проведено порівняльний аналіз існуючих музичних додатків, виявлено їхні переваги та недоліки у контексті сучасних потреб користувачів. Розроблений додаток зосереджений на вирішенні цих проблемних питань, пропонуючи новітні технології і підходи до музичного відтворення та взаємодії з користувачем.

Було використано JavaScript як основну мову програмування, а ReactJS — основний фреймворк. Додатково, для реалізації дизайну використано фреймворки TailwindCSS та Material-UI, що дозволили створити привабливий та якісний дизайн.

Для зберігання користувацьких фотографій профілів було використано бібліотеку "multer", яка дозволяє зберігати певні дані в локальному сховищі. Для зберігання важливіших даних було використано систему управління базами даних MySQL, що дозволило зберігати всю необхідну інформацію для роботи музичного додатку. Також було використано функціонал авторизації та реєстрації сервісу Firebase від Google. Додатково було використано базу даних Firebase Database, яка забезпечує зберігання та шифрування даних користувачів до їхнього профілю.

Було проаналізовано ринок схожих додатків, розраховано вартість проєкту та необхідну суму для його реалізації. Визначено основні нюанси при реалізації музичного додатку та можливі недоліки. На кінцевому етапі створення додатку його було детально перевірено на наявність помилок або багів.

Перш за все, музичний додаток був створений для закріплення знань після вивчення згаданих технологій і може бути використаний як хороший стартап або елемент портфоліо при пошуку пропозицій роботи розробника.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Стаття про актуальність створення музичних додатків. Ekreative.com: вебсайт. URL: https://www.ekreative.com/blog/what-does-it-take-to-develop-a-great-music-app/ (дата звернення: 24.02.2024);
2. Аналіз статистика ринку музичних додатків. Grand View Research: вебсайт. URL: https://www.grandviewresearch.com/industry-analysis/music-streaming-market (дата звернення: 24.02.2024);
3. Стаття аналіз музичних додатків сьогодення. LinkedIn: вебсайт. URL: https://www.linkedin.com/pulse/what-makes-your-on-demand-music-streaming-app-tnn2f/ (дата звернення: 24.02.2024);
4. Переваги та недоліки додатку Spotify. Techquintal: вебсайт. URL: https://www.techquintal.com/advantages-and-disadvantages-of-spotify/ (дата звернення: 25.02.2024);
5. Переваги та недоліки Apple Music. NoteBurner.com: вебсайт. URL: https://www.noteburner.com/apple-music-tips/pros-and-cons-of-apple-music.html (дата звернення: 25.02.2024);
6. Переваги та недоліки Youtube Music. FreeYourMusic: вебсайт. URL: https://freeyourmusic.com/blog/youtube-music-review (дата звернення: 26.02.2024);
7. JavaScript. Вікіпедія : вебсайт. URL: https://uk.wikipedia.org/wiki/JavaScript (дата звернення: 31.03.2024);
8. PHP. Вікіпедія : вебсайт. URL: https://uk.wikipedia.org/wiki/PHP (дата звернення: 31.03.2024);
9. MySQL. Вікіпедія : вебсайт. URL: https://uk.wikipedia.org/wiki/MySQL (дата звернення: 31.03.2024).

ДОДАТКИ

Додаток А

Серверний файл проєкту з підключенням до бази даних, створенням API і використання локальних сховищ

```
import express from 'express';
import mysql from 'mysql';
import cors from 'cors';
import multer from 'multer';
import fs from 'fs';
const app = express();
app.use(express.json());
app.use(cors())
const db = mysql.createConnection({
    host: 'localhost',
    user: 'root',
    password: '',
    database: 'music_player'
})
const imagesDir = './public/images/';
if (!fs.existsSync(imagesDir)) {
    fs.mkdirSync(imagesDir, { recursive: true });
}
const avatarStorage = multer.diskStorage({
    destination: function (req, file, cb) {
        cb(null, imagesDir);
    },
    filename: function (req, file, cb) {
        const uniqueSuffix = `${Date.now()}-
${Math.round(Math.random() * 1E9)}`;
        cb(null, `${uniqueSuffix}_${file.originalname}`);
    }
});
const avatarUpload = multer({
    storage: avatarStorage,
```

```
    fileFilter: function (req, file, cb) {
        if (file.mimetype === 'image/jpeg' || file.mimetype ===
'image/png') {
            cb(null, true);
        } else {
            cb(new Error('Only JPEG and PNG files are allowed!'),
false);
        }
    }
});
app.post('/upload', avatarUpload.single('file'), (req, res) => {
    const uid = req.body.uid;
    const newPhoto = req.file.filename;
    const sqlSelect = 'SELECT uphoto FROM users WHERE uid = ?';
    const sqlUpdate = 'UPDATE users SET uphoto = ? WHERE uid = ?';
    db.query(sqlSelect, uid, (err, result) => {
        if (err) return res.json({ Error: "err" });
        const currentPhoto = result[0].uphoto;
        if (currentPhoto) {
            fs.unlink(`./public/images/${currentPhoto}`, (err) =>
{
                if (err) return res.json({ Error: "err delete" });
            });
        }
        db.query(sqlUpdate, [newPhoto, uid], (err, result) => {
            if (err) return res.json({ Error: "err update" });
            return res.json({ Status: "Success" });
        });
    });
});
app.post('/deleteAvatar', async (req, res) => {
    const uid = req.body.uid;
    const sqlSelect = 'SELECT uphoto FROM users WHERE uid = ?';
    const sqlDeletePhoto = 'UPDATE users SET uphoto = "" WHERE uid
= ?';
```

```
    db.query(sqlSelect, uid, (err, data) => {
        if (err) return res.json({ error: err.message });
        const currentPhoto = data[0]?.uphoto;
        if (currentPhoto) {
            fs.unlink(`./public/images/${currentPhoto}`, (err) =>
{
                if (err) return res.json({ error: "Error deleting
photo from storage" });
                db.query(sqlDeletePhoto, uid, (err, result) => {
                    if (err) return res.json({ error: "Error
updating database" });
                    return res.json({ Status: "Photo deleted
successfully" });
                });
            });
        } else {
            return res.json({ Status: "No photo to delete" });
        }
    });
});
app.post('/createUser', (req, res) => {
    const uid = req.body.uid;
    const sqlInsertUser = 'INSERT INTO users (uid, uphoto) VALUES
(?, "")';
    const sqlInsertFavoriteList = 'INSERT INTO favorite_lists
(uid, song_count, favorite_list_duration) VALUES (?, 0, 0)';
    db.query(sqlInsertUser, [uid], (err) => {
        if (err) return res.json({ error: err.message });
        db.query(sqlInsertFavoriteList, [uid], (err) => {
            if (err) return res.json({ error: err.message });
            return res.status(201).json();
        });
    });
});
```

```
app.post('/loadAvatar', async (req, res) => {
    const uid = req.body.uid;
    const sql = 'SELECT uphoto FROM users WHERE uid = ?';
    db.query(sql, uid, (err, data) => {
        if (err) return res.json({ error: err.message });
        return res.json({ uphoto: data[0].uphoto });
    });
});
app.get('/songsAndAuthors', (req, res) => {
    const sql = 'SELECT songs.*, authors.author_name AS
song_author FROM songs LEFT JOIN authors ON songs.author_id =
authors.author_id';
    db.query(sql, (err, data) => {
        if (err) return res.json(err);
        return res.json(data);
    });
});
app.post('/userFavoriteSongs', (req, res) => {
    const uid = req.body.uid;
    const sql = `SELECT songs.*, authors.author_name AS
song_author FROM songs LEFT JOIN favorite_lists_items ON
songs.song_id = favorite_lists_items.song_id LEFT JOIN
favorite_lists ON favorite_lists.favorite_list_id =
favorite_lists_items.favorite_list_id LEFT JOIN authors ON
songs.author_id = authors.author_id WHERE favorite_lists.uid = ?`;
    db.query(sql, [uid], (err, data) => {
        if (err) return res.json(err);
        return res.json(data);
    });
});
app.post('/addToFavorites', (req, res) => {
    const { uid, song_id } = req.body;
    const checkIfExistsQuery = 'SELECT * FROM favorite_lists_items
WHERE favorite_list_id IN (SELECT favorite_list_id FROM
favorite_lists WHERE uid = ?) AND song_id = ?';
```

```
    db.query(checkIfExistsQuery, [uid, song_id], (err, result) =>
{
        if (err) return res.json({ error: err.message });
        if (result.length > 0) return res.json({ message: 'Song
already exists in favorites' });
        const insertQuery = 'INSERT INTO favorite_lists_items
(favorite_list_id, song_id) VALUES ((SELECT favorite_list_id FROM
favorite_lists WHERE uid = ?), ?)';
        db.query(insertQuery, [uid, song_id], (err, result) => {
            if (err) return res.json({ error: err.message });
            const updateFavoriteListQuery = `UPDATE favorite_lists
SET song_count = song_count + 1,favorite_list_duration =
favorite_list_duration + (SELECT song_duration FROM songs WHERE
song_id = ?) WHERE uid = ?`;
            db.query(updateFavoriteListQuery, [song_id, uid],
(err, result) => {
                if (err) return res.json({ error: err.message });
                return res.json({ message: 'Song added to
favorites successfully' });
            });
        });
    });
});
app.post('/removeFromFavorites', (req, res) => {
    const { uid, song_id } = req.body;
    const checkIfExistsQuery = 'SELECT * FROM favorite_lists_items
WHERE favorite_list_id IN (SELECT favorite_list_id FROM
favorite_lists WHERE uid = ?) AND song_id = ?';
    db.query(checkIfExistsQuery, [uid, song_id], (err, result) =>
{
        if (err) return res.json({ error: err.message });
        if (result.length === 0) return res.json({ message: 'Song
not found in favorites' });
        const removeFromFavoritesQuery = 'DELETE FROM
favorite_lists_items WHERE favorite_list_id IN (SELECT
```

```
favorite_list_id FROM favorite_lists WHERE uid = ?) AND song_id =
?';
        db.query(removeFromFavoritesQuery, [uid, song_id], (err,
result) => {
            if (err) return res.json({ error: err.message });
            const updateFavoriteListQuery = `UPDATE favorite_lists
SET song_count = song_count - 1, favorite_list_duration =
favorite_list_duration - (SELECT song_duration FROM songs WHERE
song_id = ?) WHERE uid = ?`;
            db.query(updateFavoriteListQuery, [song_id, uid],
(err, result) => {
                if (err) return res.json({ error: err.message });
                return res.json({ message: 'Song removed from
favorites successfully' });
            });
        });
    });
});
app.post('/favoriteInfo', (req, res) => {
    const uid = req.body.uid;
    const get = 'SELECT * FROM favorite_lists WHERE uid = ?';
    db.query(get, [uid], (err, data) => {
        if (err) return res.json({ error: err.message });
        return res.json(data);
    });
});
app.post('/getPlaylists', (req, res) => {
    const uid = req.body.uid;
    const sql = 'SELECT * FROM playlists WHERE uid = ?';
    db.query(sql, [uid], (err, data) => {
        if (err) return res.json({ error: err.message });
        return res.json(data);
    });
});
```

```
app.post('/createPlaylist', (req, res) => {
    const uid = req.body.uid;
    const playlistImage = 'https://cdn-icons-
png.flaticon.com/512/483/483041.png';
    const sql = 'INSERT INTO playlists (uid, playlist_name,
playlist_image, song_count, playlist_duration) VALUES (?,
"Плейлист", ?, 0, 0)';
    db.query(sql, [uid, playlistImage], (err, result) => {
        if (err) return res.json({ error: err.message });
        return res.json({ message: 'Playlist created successfully'
});
    });
});
app.post('/deletePlaylist', (req, res) => {
    const playlist_id = req.body.playlistid;
    const deletePlayListSongs = 'DELETE FROM playlists_songs WHERE
playlist_id = ?';
    db.query(deletePlayListSongs, [playlist_id], (err, result) =>
{
        if (err) return res.json({ error: err.message });
        const deletePlayList = 'DELETE FROM playlists WHERE
playlist_id = ?';
        db.query(deletePlayList, [playlist_id], (err, result) => {
            if (err) return res.json({ error: err.message });
            return res.json({ message: 'Playlist and associated
songs deleted successfully' });
        });
    });
});
app.post('/getPlaylistSongs', (req, res) => {
    const { playlist_id } = req.body;
    const sql = `SELECT songs.*, authors.author_name AS
song_author FROM playlists_songs JOIN songs ON
playlists_songs.song_id = songs.song_id LEFT JOIN authors ON
```

```
songs.author_id = authors.author_id WHERE
playlists_songs.playlist_id = ?`;
    db.query(sql, [playlist_id], (err, data) => {
        if (err) return res.json({ error: err.message });
        return res.json(data);});});
app.post('/addSongToPlaylist', (req, res) => {
    const { playlist_id, song_id } = req.body;
    const checkIfExistsQuery = 'SELECT * FROM playlists_songs
WHERE playlist_id = ? AND song_id = ?';
    db.query(checkIfExistsQuery, [playlist_id, song_id], (err,
result) => {
        if (err) return res.json({ error: err.message });
        if (result.length > 0) return res.json({ message: 'Пісня
вже є в списку' });
        const insertQuery = 'INSERT INTO playlists_songs
(playlist_id, song_id) VALUES (?, ?)';
        db.query(insertQuery, [playlist_id, song_id], (err,
result) => {
            if (err) return res.json({ error: err.message });
            const updatePlaylistQuery = `UPDATE playlists SET
song_count = song_count + 1, playlist_duration = playlist_duration
+ (SELECT song_duration FROM songs WHERE song_id = ?) WHERE
playlist_id = ?`;
            db.query(updatePlaylistQuery, [song_id, playlist_id],
(err, result) => {
                if (err) return res.json({ error: err.message });
                return;
            });
        });
    });
});
app.post('/removeSongFromPlaylist', (req, res) => {
    const { playlist_id, song_id } = req.body;
    const deleteQuery = 'DELETE FROM playlists_songs WHERE
playlist_id = ? AND song_id = ?';
```

```
    const updatePlaylistQuery = `UPDATE playlists SET song_count =
song_count - 1, playlist_duration = playlist_duration - (SELECT
song_duration FROM songs WHERE song_id = ?) WHERE playlist_id =
?`;
    db.query(deleteQuery, [playlist_id, song_id], (err, result) =>
{
        if (err) return res.json({ error: err.message });
        db.query(updatePlaylistQuery, [song_id, playlist_id],
(err, result) => {
            if (err) return res.json({ error: err.message });
            return res.json({ message: 'Song removed from playlist
successfully' });
        });
    });
});
app.post('/checkSongInPlaylist', (req, res) => {
    const { playlist_id, song_id } = req.body;
    const sql = 'SELECT * FROM playlists_songs WHERE playlist_id =
? AND song_id = ?';
    db.query(sql, [playlist_id, song_id], (err, data) => {
        if (err) return res.json({ error: err.message });
        return res.json({ isInPlaylist: data.length > 0 });
    });
});
app.post('/updatePlaylistName', (req, res) => {
    const playlist_id = req.body.id;
    const newName = req.body.inputValue;
    if (!newName || newName.trim() === '') { return; }
    const sqlUpdate = 'UPDATE playlists SET playlist_name = ?
WHERE playlist_id = ?';
    db.query(sqlUpdate, [newName, playlist_id], (err, result) => {
        if (err) return res.json({ Error: "Error updating playlist
name" });
        return res.json({ Status: "Name updated successfully" });
    });
```

```
});
app.post('/getPlaylistName', (req, res) => {
    const playlist_id = req.body.id;
    const sqlGet = 'SELECT playlist_name FROM playlists WHERE
playlist_id = ?';
    db.query(sqlGet, [playlist_id], (err, data) => {
        if (err) return res.json({ Error: "Error updating playlist
name" });
        return res.json(data);
    });
})
app.post('/authors', (req, res) => {
    const sql = 'SELECT * FROM authors';
    db.query(sql, (err, data) => {
        if (err) return res.json({ error: err.message });
        return res.json(data);
    });
});
app.post('/albums', (req, res) => {
    const sql = 'SELECT * FROM albums';
    db.query(sql, (err, data) => {
        if (err) return res.json({ error: err.message });
        return res.json(data);
    });
});
app.post('/songsByAuthor', (req, res) => {
    const author_id = req.body.author_id;
    const sql = `SELECT songs.*, authors.author_name AS
song_author, albums.album_name AS album_name FROM songs LEFT JOIN
authors ON songs.author_id = authors.author_id LEFT JOIN albums ON
songs.album_id = albums.album_id WHERE songs.author_id = ?`;
    db.query(sql, [author_id], (err, data) => {
        if (err) return res.json({ error: err.message });
        return res.json(data);
    });
```

```
});
app.post('/songsByAlbum', (req, res) => {
    const album_id = req.body.album_id;
    const sql = `SELECT songs.*, authors.author_name AS
song_author, albums.album_name AS album_name FROM songs LEFT JOIN
authors ON songs.author_id = authors.author_id LEFT JOIN albums ON
songs.album_id = albums.album_id WHERE songs.album_id = ?`;
    db.query(sql, [album_id], (err, data) => {
        if (err) return res.json({ error: err.message });
        return res.json(data);
    });
});
app.post('/defaultPlaylists', (req, res) => {
    const sql = 'SELECT * FROM default_playlists';
    db.query(sql, (err, data) => {
        if (err) return res.json({ error: err.message });
        return res.json(data);
    });
});
app.post('/defaultPlaylistSongs', (req, res) => {
    const sql = `SELECT songs.*,
default_playlist_items.default_playlist_id AS default_playlist_id,
authors.author_name AS song_author FROM songs JOIN
default_playlist_items ON songs.song_id =
default_playlist_items.song_id LEFT JOIN default_playlists ON
default_playlist_items.default_playlist_id =
default_playlists.default_playlist_id LEFT JOIN authors ON
songs.author_id = authors.author_id`;
    db.query(sql, (err, data) => {
        if (err) return res.json({ error: err.message });
        return res.json(data);
    });
});
app.listen(8081, () => {
    console.log('http://localhost:8081');})
```

Додаток Б

Сторінка авторизації

```
import { useState, useEffect } from 'react';
import { useAuthContext } from '../firebase/authProvider';
import { Register } from '../components/Register';
import { useNavigate } from 'react-router-dom';
const LoginPage = () => {
    const { login, profile } = useAuthContext();
    const navigate = useNavigate();
    const [email, setEmail] = useState('');
    const [password, setPassword] = useState('');
    const [loginRunning, setLoginRunning] = useState(false);
    const [errorMessage, setErrorMessage] = useState();
    const [showRegisterForm, setShowRegisterForm] =
useState(false);
    const toggleShowRegisterScreen = () => {
        setShowRegisterForm((currVal) => !currVal);
        setEmail('');
        setPassword('');
        setErrorMessage(null);
    };
    const handleButtonClick = async () => {
        setLoginRunning(true);
        let success = await login(email, password);
        setLoginRunning(false);
        if (!success) {
            setErrorMessage("Ой! Щось пішло не так.");};};};
    useEffect(() => {
        if (profile) {
            navigate('/main');
        }
    }, [profile, navigate]);
    return (
```

```
        <div className="bg-[#212121] flex flex-row h-screen w-
screen">
            <div className='bg-[#292929] rounded-2xl w-full flex
items-center justify-center m-3'>
                <div className="flex flex-row m-3">
                    {showRegisterForm ? (
                        <Register
toggleShowRegisterScreen={toggleShowRegisterScreen} />
                    ) : (
                        <div className="bg-[#212121] p-8 text-
white rounded-xl">
                            <p className="text-2xl mb-4">Логін</p>
                            <div id="form">
                                <div className="form-group mb-4">
                                    <label className="block mb-
2">Введіть пошту</label>
                                    <input
                                        type='text'
                                        name='email'
                                        value={email}
                                        onChange={(e) =>
setEmail(e.target.value)}
                                        className="w-full p-2
rounded bg-gray-800 text-white"
                                    />
                                </div>
                                <div className="form-group mb-4">
                                    <label className="block mb-
2">Введіть ваш пароль</label>
                                    <input
                                        type='password'
                                        name='password'
                                        value={password}
                                        onChange={(e) =>
setPassword(e.target.value)}
```

```
                                        className="w-full p-2
rounded bg-gray-800 text-white"
                                    />
                                </div>
                                <div className="form-group mb-4">
                                    <button
                                        className="text-blue-500
hover:underline" onClick={toggleShowRegisterScreen}>Немає профілю?
Зареєструйтесь тут.
                                    </button>
                                </div>
                                <div className="form-group">
                                    <button
                                        className="bg-blue-500
hover:bg-blue-700 text-white font-bold py-2 px-4 rounded"

onClick={handleButtonClick}
                                        disabled={loginRunning}
                                    >
                                        {loginRunning ?
'Зачекайте...' : 'Увійти'}
                                    </button>
                                    {errorMessage && (
                                        <p className="error-
message pt-4 text-red-500">{errorMessage}</p>
                                    )}
                                </div>
                            </div>
                        </div>
                    )}
                </div>
            </div>
        </div>
    );};
export default LoginPage;
```

Додаток В

Компонент реєстрації

```
import { useState } from 'react';
import { useAuthContext } from '../firebase/authProvider';
export const Register = ({ toggleShowRegisterScreen }) => {
  const { register } = useAuthContext();
  const [displayName, setDisplayName] = useState('');
  const [email, setEmail] = useState('');
  const [password, setPassword] = useState('');
  const [registrationRunning, setRegistrationRunning] =
useState(false);
  const [errorMessage, setErrorMessage] = useState();
  const handleButtonClick = async () => {
    setRegistrationRunning(true);
    let theDisplayName = displayName;
    if (theDisplayName?.length <= 0) {
      theDisplayName = 'NO DISPLAY NAME PROVIDED';}
    let success = await register(email, password, theDisplayName);
    if (!success) {
      setErrorMessage("Ой! Щось пішло не так.");
      setRegistrationRunning(false);
    }
  };
  return (
    <div className="bg-[#212121] p-8 text-white rounded-xl">
      <p className="text-2xl mb-4"> Реєстрація</p >
      <div id="form">
        <div className="form-group mb-4">
          <label className="block mb-2">Введіть пошту</label>
          <input
            type='text'
            name='email'
            value={email}
            onChange={(e) => setEmail(e.target.value)}
```

```
            className="w-full p-2 rounded bg-gray-800 text-
white"/></div>
        <div className="form-group mb-4">
          <label className="block mb-2">Введіть ім'я
користувача</label>
          <input
            type='text'
            name='displayName'
            value={displayName}
            onChange={(e) => setDisplayName(e.target.value)}
            className="w-full p-2 rounded bg-gray-800 text-
white"/></div>
        <div className="form-group mb-4">
          <label className="block mb-2">Введіть пароль</label>
          <input
            type='password'
            name='password'
            value={password}
            onChange={(e) => setPassword(e.target.value)}
            className="w-full p-2 rounded bg-gray-800 text-
white"/></div>
        <div className="form-group mb-4">
          <button className="text-blue-500 hover:underline"
onClick={toggleShowRegisterScreen}>
            Вже є профіль? Увійдіть тут.
          </button></div><div><button
            className="bg-blue-500 hover:bg-blue-700 text-white
font-bold py-2 px-4 rounded"
            onClick={handleButtonClick}
            disabled={registrationRunning}>
            {registrationRunning ? 'Зачекайте...' :
'Зареєструватися'}</button>
          {errorMessage && (
            <p className="text-red-500 error-message pt-
4">{errorMessage}</p>)}</div></div ></div>);};
```

Додаток Г

Провайдер Firebase

```
import { createContext, useContext, useEffect, useState } from
'react';
import { initializeApp } from 'firebase/app';
import { getAuth } from 'firebase/auth';
import { getFirestore } from 'firebase/firestore';
import myFirebaseConfig from './firebaseConfig.json';

export const FirebaseContext = createContext({});

const FirebaseProvider = (props) => {
  const myApp = initializeApp(myFirebaseConfig);
  const myAuth = getAuth(myApp);
  const myFS = getFirestore(myApp);
  const [firebaseInitializing, setFirebaseInitializing] =
useState(true);
  useEffect(() => { setFirebaseInitializing(false); }, []);

  if (firebaseInitializing) {
    return <h1>Loading...</h1>;
  }
  const value = { myApp, myAuth, myFS };
  const { children } = props;
  return (
    <FirebaseContext.Provider value={value}>
      {children}
    </FirebaseContext.Provider>
  );};
const useFirebaseContext = () => {
  const context = useContext(FirebaseContext);
  return context;
};
export { FirebaseProvider, useFirebaseContext };
```

Додаток Ґ

Провайдер використання функціоналу авторизації та реєстрації

```
import { createContext, useContext, useEffect, useState } from
'react';
import { createUserWithEmailAndPassword, onAuthStateChanged,
signInWithEmailAndPassword, signOut, } from 'firebase/auth';
import { doc, onSnapshot, setDoc, serverTimestamp } from
'firebase/firestore';
import { useFirebaseContext } from './firebaseProvider';
import axios from 'axios';

export const AuthContext = createContext({});
const PROFILE_COLLECTION = 'users';

const AuthProvider = (props) => {
  const [user, setUser] = useState(null);
  const [profile, setProfile] = useState(null);
  const [authLoading, setAuthLoading] = useState(true);
  const [authErrorMessages, setAuthErrorMessages] = useState();
  const [userPhoto, setUserPhoto] = useState();
  const [userPlaylists, setUserPlaylists] = useState([]);
  const { myAuth, myFS } = useFirebaseContext();

  useEffect(() => {
    if (myAuth) {
      let unsubscribe = onAuthStateChanged(myAuth, (user) => {
        if (user) {
          setUser(user);
          if (!sessionStorage.getItem('initialized')) {
            logoutFunction();
            sessionStorage.setItem('initialized', 'true');
          }
        }
        setAuthLoading(false);
```

```
      });
      return unsubscribe;
    }
  }, [myAuth]);

  useEffect(() => {
    if (myAuth) {
      let unsubscribe = onAuthStateChanged(myAuth, (user) => {
        if (user) {
          setUser(user);
        }
        setAuthLoading(false);
      });
      return unsubscribe;
    }
  }, [myAuth]);

  useEffect(() => {
    let unsubscribe = null;
    const listenToUserDoc = async (uid) => {
      try {
        let docRef = doc(myFS, PROFILE_COLLECTION, uid);
        unsubscribe = await onSnapshot(docRef,
          (docSnap) => {
            let profileData = docSnap.data();
            console.log('Got user profile:', profileData,
docSnap);
            if (!profileData) { setAuthErrorMessages([`No profile
doc found in Firestore at: ${docRef.path}`]); }
            setProfile(profileData);
          },
          (firestoreErr) => {
            console.error(`onSnapshot() callback failed with:
${firestoreErr.message}`, firestoreErr);
```

```
            setAuthErrorMessages([firestoreErr.message, 'Have you
initialized your Firestore database?']);
          }
        );
      } catch (ex) {
        console.error(`useEffect() calling onSnapshot() failed
with: ${ex.message}`);
        setAuthErrorMessages([ex.message]);
      }
    };

    if (user?.uid) {
      listenToUserDoc(user.uid);
      return () => {
        unsubscribe && unsubscribe();
      };
    } else if (!user) {
      setAuthLoading(true);
      setProfile(null);
      setAuthErrorMessages(null);
    }
  }, [user, setProfile, myFS]);

  /**
   * @param {string}
   * @param {string}
   * @param {string}
   * @returns {boolean}
   */
  const registerFunction = async (email, password, username = '')
=> {
    let userCredential;
    try {
      userCredential = await
createUserWithEmailAndPassword(myAuth, email, password);
```

```
    } catch (ex) {
      setAuthErrorMessages([console.log(ex.message)]);
      return false;
    }

    try {
      let user = userCredential.user;
      let uid = user.uid;
      let userDocRef = doc(myFS, 'users', user.uid);
      let userDocData = { uid: user.uid, email: email, username:
username, dateCreated: serverTimestamp(), };
      localStorage.setItem('uid', user.uid);
      await addUser(uid);
      await setDoc(userDocRef, userDocData);
      return true;
    } catch (ex) {
      console.error(`registerFunction() failed with:
${ex.message}`);
      setAuthErrorMessages([ex.message, 'Did you enable the
Firestore Database in your Firebase project?',]);
      return false;
    }};
  const addUser = async (uid) => {
    try {
      await axios.post(`http://localhost:8081/createUser`, { uid
});
    } catch (error) {
      console.error('Error adding user to another database:',
error);
    }
  };
  const loginFunction = async (email, password) => {
    try {
      let userCredential = await
signInWithEmailAndPassword(myAuth, email, password);
```

```
      let user = userCredential.user;
      if (!user?.uid) {
        let msg = `No UID found after signIn!`;
        console.error(msg);}
      if (user) { localStorage.setItem('uid', user.uid) }
      setUser(user);
      return true;
    } catch (ex) {
      let msg = `Login failure for email(${email}:
${ex.message})`;
      console.error(msg);
      setAuthErrorMessages([ex.message]);
      return false;}};
  const logoutFunction = async () => {
    try {
      setUser(null);
      await signOut(myAuth);
      console.log('Signed Out');
      localStorage.removeItem('uid', user.uid);
      window.location.reload();
      return true;
    } catch (ex) {
      console.error(ex);
      setAuthErrorMessages([ex.message]);
      return false;}};
  if (authLoading) {
    return <h1>Loading</h1>;}
  const loadUserAvatar = async (uid) => {
    try {
      const response = await
axios.post(`http://localhost:8081/loadAvatar`, { uid });
      const { uphoto } = response.data;
      if (uphoto == "") {
setUserPhoto('https://static.vecteezy.com/system/resources/preview
```

```
s/019/879/186/non_2x/user-icon-on-transparent-background-free-
png.png');} else {
        setUserPhoto(`./public/images/${uphoto}`);}
    } catch (error) {
      console.error("Error fetching user avatar", error);}};
  const getPlaylists = async (uid) => {
    try {
      const response = await
axios.post('http://localhost:8081/getPlaylists', { uid });
      setUserPlaylists(response.data);
    } catch (error) {
      console.error('Error fetching playlists:', error);}};
  const updateUserPlaylists = async (playlists) => {
    setUserPlaylists(playlists);};
  const value = {
    authErrorMessages,
    authLoading,
    profile,
    user,
    userPhoto,
    userPlaylists,
    getPlaylists,
    updateUserPlaylists,
    loadUserAvatar,
    login: loginFunction,
    logout: logoutFunction,
    register: registerFunction,};
  const children = props.children;
  return (
    <AuthContext.Provider
value={value}>{children}</AuthContext.Provider>);};
const useAuthContext = () => {
  const context = useContext(AuthContext);
  return context;};
export { AuthProvider, useAuthContext };
```

Додаток Д

Програмний код головної сторінки

```
import User from '../components/user';
import Menu from "../components/menu";
import Search from '../components/search';
import TrackList from "../track/trackList";
import { Navigate } from "react-router-dom";
import Playbar from "../components/playbar";
import Favorite from '../components/favorite';
import AlbumPage from "../components/albumPage";
import AuthorPage from "../components/authorPage";
import UserPlaylist from '../components/userPlaylist';
import { AudioContext } from "../context/AudioContext";
import { useContext, useState, useEffect } from "react";
import { useAuthContext } from '../firebase/authProvider';
import DefaultPage from "../components/defaultPage";
import '../index.css';

const MainPage = () => {
    const { menuItem } = useContext(AudioContext);
    const [content, setContent] = useState('golovna')
    const { profile } = useAuthContext();
    if (!profile) {
        return (<Navigate to="/" />);
    }
    useEffect(() => {
        switch (menuItem) {
            case 'profile':
                setContent(<User />);
                break;
            case 'favorite':
                setContent(<Favorite />);
                break;
            case 'golovna':
```

```
                setContent(<TrackList />);
                break;
            case 'search':
                setContent(<Search />);
                break;
            case 'userPlaylist':
                setContent(<UserPlaylist />);
                break;
            case 'album':
                setContent(<AlbumPage />);
                break;
            case 'author':
                setContent(<AuthorPage />);
                break;
            case 'def':
                setContent(<DefaultPage />);
                break;
            default:
                setContent(<TrackList />);
                break;
        }
    }, [menuItem])
    return (
        <div className="flex flex-row h-screen">
            <Menu />
            <div className="flex flex-col w-full">
                <div className="flex-1 overflow-auto">
                    {content}
                </div>
              <Playbar className="fixed bottom-0 left-0 w-full" />
            </div>
        </div>
    )
};
export default MainPage;
```

Додаток Е

Програмний код компоненти меню

```
import PlaylistAddIcon from '@mui/icons-material/PlaylistAdd';
import SearchIcon from '@mui/icons-material/Search';
import HouseIcon from '@mui/icons-material/House';
import FavoriteIcon from '@mui/icons-material/Favorite';
import { useAuthContext } from '../firebase/authProvider';
import { useContext, useEffect } from "react";
import { AudioContext } from "../context/AudioContext";
import axios from 'axios';
const Menu = () => {
  const { onChangeMenuItem, onChangePlaylist } =
useContext(AudioContext);
  const { userPhoto, loadUserAvatar, userPlaylists, getPlaylists }
= useAuthContext();
  const uid = localStorage.getItem('uid');
  useEffect(() => {
    if (uid) {
      loadUserAvatar(uid);
      getPlaylists(uid);}
  }, [uid, loadUserAvatar, getPlaylists]);
  const handleCreatePlaylist = async () => {
    try {
      await axios.post('http://localhost:8081/createPlaylist', {
uid });
    } catch (error) {
      console.error('Error creating playlist:', error);}};
  return (
    <div className='flex flex-col h-screen w-60 bg-[#212121] text-
white'>
      <div className='h-32 w-auto bg-[#292929] rounded-2xl flex
items-center justify-center m-2 p-2'>
```

```
        <img onClick={() => onChangeMenuItem('profile')}
src={userPhoto} className='m-auto max-h-full max-w-full object-
contain' />
      </div>
      <div className='h-full w-auto bg-[#292929] rounded-2xl flex
flex-col m-2 p-2 gap-y-1'>
        <p onClick={() => onChangeMenuItem('golovna')}>Головна
сторінка<HouseIcon /></p>
        <p onClick={() =>
onChangeMenuItem('search')}>Пошук<SearchIcon /></p>
        <div onClick={() => onChangeMenuItem('favorite')}>
          <p>Улюблений список <FavoriteIcon /></p>
        </div>
        <div onClick={handleCreatePlaylist}>
          <p>Створити плейлист <PlaylistAddIcon /></p>
        </div>
        <div className='h-72 overflow-auto'>
          {userPlaylists.map((playlist) => (
            <div className='flex flex-row items-center space-x-2'
key={playlist.playlist_id}>
              <div className='flex-shrink-0'>
                <img src={playlist.playlist_image} onClick={() =>
{ onChangeMenuItem('userPlaylist'); onChangePlaylist(playlist); }}
className='w-12 h-12' />
              </div>
              <p onClick={() => {
onChangeMenuItem('userPlaylist'); onChangePlaylist(playlist); }}
className='break-words whitespace-normal'>
                {playlist.playlist_name}
              </p>
            </div>))}
        </div>
      </div>
    </div >);};
export default Menu;
```

Додаток Є

Програмний код компоненти стрічки мультимедіа

```
import { PlayArrow, Pause, SkipPrevious, SkipNext, VolumeUp,
VolumeDown, Repeat, RepeatOne, Shuffle } from "@mui/icons-
material";
import { useContext, useEffect, useState } from 'react';
import { AudioContext } from '../context/AudioContext';
import ShuffleOnIcon from '@mui/icons-material/ShuffleOn';
import { IconButton, Slider } from '@mui/material';
import secondsToMMSS from '../utils/secondsToMMSS';
const TimeControls = () => {
    const { audio, currentTrack } = useContext(AudioContext);
    const { song_duration } = currentTrack;
    const [currentTime, setCurrentTime] = useState(0);
    const formattedDuration = secondsToMMSS(currentTime);
    const sliderCurrentTime = song_duration ?
Math.round((currentTime / song_duration) * 100) : 0;
    const songDuration = secondsToMMSS(song_duration);

    const handleChangeCurrentTime = (_, value) => {
        const time = Math.round((value / 100) * song_duration);
        setCurrentTime(time);
        audio.currentTime = time;};
    useEffect(() => {
        const timeInterval = setInterval(() => {
setCurrentTime(audio.currentTime) }, 1000);
        return () => { clearInterval(timeInterval); };
    }, []);
    return (
        <div className='flex flex-row items-center space-x-2'>
            <p className='pr-2'>{formattedDuration}</p>
            <Slider step={1} min={0} max={100}
value={sliderCurrentTime} onChange={handleChangeCurrentTime}
className='flex-grow' />
```

```
            <p className='pl-2'>{songDuration}</p></div>);};
const LoudControls = () => {
    const { audio } = useContext(AudioContext);
    const [valueLoud, setValueLoud] = useState(25);
    const loudnessChange = (event, newValue) =>
{setValueLoud(newValue);};
    useEffect(() => {audio.volume = valueLoud / 1000;});
    return (
    <div className='flex flex-row items-center space-x-2 mr-2'>
<VolumeDown />
            <Slider value={valueLoud} onChange={loudnessChange}
className='w-64' /><VolumeUp /></div>);};
const Playbar = () => {
    const { currentTrack, isPlaying, handleToggleAudio,
findAuthorById, onChangeMenuItem, playNextSong, playPreviousSong,
toggleShuffle, shuffle, repeatOne, toggleRepeatOne } =
useContext(AudioContext);
    const { song_id, song_name, song_author, author_id, song_image
} = currentTrack;
    const isCurrentTrack = currentTrack.song_id === song_id;
    const handleAuthorClick = (author_id) => {
        findAuthorById(author_id);
        onChangeMenuItem('author');};
return (
  div className='bg-[#212121] p-3 text-white'>
     <div className='w-full flex flex-row items-center justify-
between bg-[#292929] rounded-2xl'>
        <div className='flex flex-row items-center space-x-4 p-2'>
                    <img src={song_image} alt={song_name}
className='ml-2 w-16 h-16 object-cover' />
                    <div className='flex flex-col'>
                        <p className='whitespace-nowrap overflow-
hidden overflow-ellipsis w-52'>{song_name}</p>
                        <p onClick={() =>
handleAuthorClick(author_id)} className='text-white hover:text-
```

```
blue-600 cursor-pointer whitespace-nowrap overflow-hidden
overflow-ellipsis w-32'>{song_author}</p>
                    </div>
                </div>
        <div className='flex flex-col items-center space-y-2 p-2'>
           <div className='flex flex-row items-center space-x-2'>
                  <IconButton style={{ color: 'white' }}
onClick={toggleShuffle} >
                           {shuffle ? <ShuffleOnIcon /> :
<Shuffle />}</IconButton>
                       <IconButton style={{ color: 'white' }}
onClick={playPreviousSong}>
                           <SkipPrevious />
                       </IconButton>
                       <IconButton style={{ color: 'white' }}
onClick={() => handleToggleAudio(currentTrack)}>
                           {isCurrentTrack && isPlaying ? <Pause
/> : <PlayArrow />}
                       </IconButton>
                       <IconButton style={{ color: 'white' }}
onClick={playNextSong}>
                           <SkipNext />
                       </IconButton>
                       <IconButton style={{ color: 'white' }}
onClick={toggleRepeatOne}>
                        {repeatOne ? <RepeatOne /> : <Repeat/>}
                       </IconButton>
                   </div>
                   <div className='w-96'>
                       <TimeControls />
                   </div>
               </div>
               <div className='w-40 p-2'>
                   <LoudControls /></div></div></div>);}
export default Playbar;
```

Додаток Ж

Програмний код файлу AudioContext.jsx

```
import { createContext, useState, useEffect } from "react";
import axios from "axios";

export const AudioContext = createContext({});
const audio = new Audio();

const AudioProvider = ({ children }) => {
  const [defaultPlaylists, setDefaultPlaylists] = useState([]);
  const [openPlaylistEdit, setOpenPlaylistEdit] = useState(false);
  const [menuItem, setMenuItem] = useState();
  const [activePlaylist, setActivePlaylist] = useState();
  const [activeAlbum, setActiveAlbum] = useState();
  const [activeDefPlaylist, setActDefPlaylist] = useState();
  const [activeAuthor, setActiveAuthor] = useState();
  const [playlistName, setPlaylistName] = useState();
  const [albums, setAlbums] = useState([]);
  const [authors, setAuthors] = useState([]);
  const [playlistCountAndDuration, setPlaylistCountAndDuration] =
useState();

  const [albumsSongs, setAlbumsSongs] = useState([]);
  const [authorsSongs, setAuthorsSongs] = useState([]);
  const [defaultPlaylistSongs, setDefaultPlaylistSongs] =
useState([]);
  const [songsPlaylist, setSongsPlaylist] = useState([]);
  const [favoriteSongs, setFavoriteSongs] = useState([]);
  const [songs, setSongs] = useState([]);

  const [listType, setListType] = useState();
  const [activeListSong, setActiveListSong] = useState([]);
  const [shuffle, setShuffle] = useState(false);
  const [defaultTrack, setDefaulTrack] = useState([]);
```

```
  const [currentTrack, setCurrentTrack] = useState(defaultTrack);
  const [isPlaying, setPlaying] = useState(false);
  const [playedSongsHistory, setPlayedSongsHistory] =
useState([]);
  const [repeatOne, setRepeatOne] = useState(false);

  useEffect(() => {
    fetch('http://localhost:8081/songsAndAuthors')
      .then(response => response.json())
      .then(data => {
        setSongs(data);
        setDefaulTrack(data[0]);
      });
  }, []);

  const songListPlaying = (list) => {
    switch (list) {
      case 'standartList':
        setShuffle(false);
        setActiveListSong(songs);
        break;
      case 'defaultPlaylist':
        setShuffle(false);
        setActiveListSong(defaultPlaylistSongs);
        break;
      case 'albumType':
        setShuffle(false);
        setActiveListSong(albumsSongs);
        break;
      case 'authorType':
        setShuffle(false);
        setActiveListSong(authorsSongs);
        break;
      case 'userPlaylist':
        setShuffle(false);
```

```
        setActiveListSong(songsPlaylist);
        break;
      case 'favoriteList':
        setShuffle(false);
        setActiveListSong(favoriteSongs);
        break;
      default:
        setShuffle(false);
        setActiveListSong([]);
        break;
    }
  }

  useEffect(() => {
    if (currentTrack) { setPlayedSongsHistory(prevHistory =>
[...prevHistory, currentTrack.song_id]); }
  }, [currentTrack]);

  useEffect(() => {
    if (repeatOne && shuffle) {
      setRepeatOne(false);
      setShuffle(false);
    } else if (repeatOne) {
      setShuffle(false);
    } else if (shuffle) {
      setRepeatOne(false);
    }
  }, [repeatOne, shuffle, setShuffle, setRepeatOne]);

  const handleToggleAudio = (track, type) => {
    if (currentTrack.song_id !== track.song_id) {
      setCurrentTrack(track);
      audio.src = track.song_path;
      audio.currentTime = 0;
      audio.play();
```

```
      setPlaying(true);
      setListType(type);
      songListPlaying(type);
    } else {
      if (isPlaying) {
        audio.pause();
        setPlaying(false);
      } else {
        audio.play();
        setPlaying(true);
      }
    }
  };

  const playRandomSong = () => {
    let remainingSongs = activeListSong.filter(song =>
!playedSongsHistory.includes(song.song_id));
    if (remainingSongs.length === 0) {
      setPlayedSongsHistory([]);
      remainingSongs = activeListSong;
    }
    const randomIndex = Math.floor(Math.random() *
remainingSongs.length);
    const nextTrack = remainingSongs[randomIndex];
    setCurrentTrack(nextTrack);
    audio.src = nextTrack.song_path;
    audio.currentTime = 0;
    audio.play();
  };

  const playNextSong = () => {
    const remainingSongs = activeListSong.filter(song =>
!playedSongsHistory.includes(song.song_id));
    if (shuffle && remainingSongs.length === 0) {
      setPlayedSongsHistory([]);
```

```
    }
    if (shuffle) {
      playRandomSong();
    } else {
      const currentIndex = activeListSong.findIndex(song =>
song.song_id === currentTrack.song_id);
      const nextIndex = (currentIndex + 1) %
activeListSong.length;
      const nextTrack = activeListSong[nextIndex];
      if (repeatOne) {
        setCurrentTrack(currentTrack);
        audio.src = currentTrack.song_path;
        audio.currentTime = 0;
        audio.play();
      } else {
        setCurrentTrack(nextTrack);
        audio.src = nextTrack.song_path;
        audio.currentTime = 0;
        audio.play();
      }
    }
  };

  const playPreviousSong = () => {
    if (shuffle) {
      const lastPlayedSongIndex = playedSongsHistory.length - 2;
      if (lastPlayedSongIndex >= 0) {
        const lastPlayedSongId =
playedSongsHistory[lastPlayedSongIndex];
        const lastPlayedSong = activeListSong.find(song =>
song.song_id === lastPlayedSongId);
        if (lastPlayedSong) {
          setCurrentTrack(lastPlayedSong);
          audio.src = lastPlayedSong.song_path;
          audio.currentTime = 0;
```

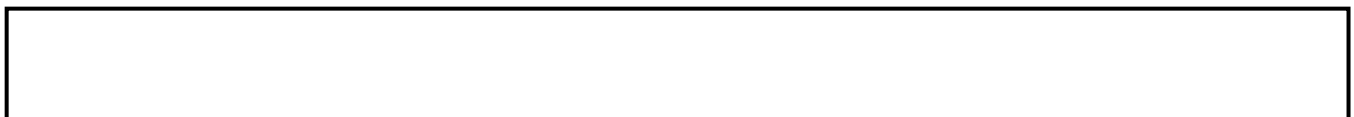
```
          audio.play();
          setPlayedSongsHistory(prevHistory =>
prevHistory.slice(0, -1));
        }
      }
    } else {
      const currentIndex = activeListSong.findIndex(song =>
song.song_id === currentTrack.song_id);
      const prevIndex = (currentIndex - 1 + activeListSong.length)
% activeListSong.length;
      const prevTrack = activeListSong[prevIndex];
      if (repeatOne) {
        setCurrentTrack(currentTrack);
        audio.src = currentTrack.song_path;
        audio.currentTime = 0;
        audio.play();
      } else {
        setCurrentTrack(prevTrack);
        audio.src = prevTrack.song_path;
        audio.currentTime = 0;
        audio.play();
      }
    }
  };

  const toggleShuffle = () => {
    if (!shuffle) { setPlayedSongsHistory([]) }
    setShuffle(!shuffle);
  };

  const toggleRepeatOne = () => {
    setRepeatOne(!repeatOne);
  };

  useEffect(() => {
```

```
    if (shuffle && currentTrack) {
setPlayedSongsHistory(prevHistory => [...prevHistory,
currentTrack.song_id]) }
  }, [currentTrack, shuffle]);

  useEffect(() => {
    audio.addEventListener('ended', playNextSong);
    return () => { audio.removeEventListener('ended',
playNextSong) };
  }, [currentTrack, activeListSong, shuffle]);

  const removeSongFromPlaylist = (songId) => {
    setSongsPlaylist(prevSongs => prevSongs.filter(song =>
song.song_id !== songId));
  };

  const onChangeMenuItem = (item) => {
    return setMenuItem(item);
  }

  const onChangePlaylist = (item) => {
    return setActivePlaylist(item);
  }

  const onChangeAlbum = (item) => {
    return setActiveAlbum(item);
  }

  const onChangeAuthor = (item) => {
    return setActiveAuthor(item);
  }

  const onChangeDefPlaylist = (item) => {
    return setActDefPlaylist(item);
  }
```

```
  const findAlbumById = (albumId) => {
    const foundAlbum = albums.find(album => album.album_id ===
albumId);
    if (foundAlbum) {
      setActiveAlbum(foundAlbum);
    } else {
      console.error(`Album with ID ${albumId} not found.`);
    }
  };

  const findAuthorById = (authorId) => {
    const foundAuthor = authors.find(author => author.author_id
=== authorId);
    if (foundAuthor) {
      setActiveAuthor(foundAuthor);
    } else {
      console.error(`Author with ID ${authorId} not found.`);
    }
  };

  const getPlaylistName = async (id) => {
    try {
      const response = await
axios.post('http://localhost:8081/getPlaylistName', { id });
      setPlaylistName(response.data[0].playlist_name);
    } catch (error) {
      console.error('Error fetching playlist name:', error);
    }
  }

  const newName = (id, inputValue) => {
    axios.post('http://localhost:8081/updatePlaylistName', { id,
inputValue });
    getPlaylistName(id);
  }
```

```
  useEffect(() => {
    const fetchAlbumsAndAuthors = async () => {
      if (!menuItem) {
        try {
          const responseDefault = await
axios.post('http://localhost:8081/defaultPlaylists');
          const responseAlbums = await
axios.post('http://localhost:8081/albums');
          const responseAuthors = await
axios.post('http://localhost:8081/authors');
          setAlbums(responseAlbums.data);
          setAuthors(responseAuthors.data);
          setDefaultPlaylists(responseDefault.data);
        } catch (err) {
          console.error(err);
        }
      }
    };
    fetchAlbumsAndAuthors();
  }, [menuItem]);

  const value = {
    audio,
    albums,
    authors,
    currentTrack,
    isPlaying,
    menuItem,
    activePlaylist,
    openPlaylistEdit,
    playlistName,
    playlistCountAndDuration,
    activeAuthor,
    activeAlbum,
    defaultPlaylists,
```

```
    activeDefPlaylist,
    listType,
    findAuthorById,
    findAlbumById,
    setPlaylistCountAndDuration,
    getPlaylistName,
    setOpenPlaylistEdit,
    removeSongFromPlaylist,
    onChangePlaylist,
    handleToggleAudio,
    onChangeMenuItem,
    onChangeAlbum,
    onChangeAuthor,
    newName,
    onChangeDefPlaylist,
    playNextSong,
    playPreviousSong,
    repeatOne, toggleRepeatOne,
    shuffle, toggleShuffle,
    songs,
    albumsSongs, setAlbumsSongs,
    authorsSongs, setAuthorsSongs,
    songsPlaylist, setSongsPlaylist,
    favoriteSongs, setFavoriteSongs,
    defaultPlaylistSongs, setDefaultPlaylistSongs,
  };
  return (
    <AudioContext.Provider value={value}>
      {children}
    </AudioContext.Provider>
  );
};
export default AudioProvider;
```



Додаток Т

Програмний код компоненти пісні

```
import { useContext, useState, useEffect } from "react";
import { AudioContext } from "../context/AudioContext";
import { PlayArrow, Pause } from "@mui/icons-material";
import { IconButton } from "@mui/material";
import secondsToMMSS from "../utils/secondsToMMSS";
import FavoriteBorderIcon from '@mui/icons-
material/FavoriteBorder';
import FavoriteIcon from '@mui/icons-material/Favorite';
import axios from "axios";
import TrackMenu from './trackMenu';
const Track = (props) => {
    const { song_id, song_name, song_author, song_image,
song_duration, author_id, album_id, playlistType,
removeFromFavorites } = props;
    const { handleToggleAudio, currentTrack, isPlaying,
findAuthorById, onChangeMenuItem, albums, findAlbumById } =
useContext(AudioContext);
    const [isFavorite, setIsFavorite] = useState(false);
    const [albumName, setAlbumName] = useState();
    const formattedDuration = secondsToMMSS(song_duration);
    const isCurrentTrack = currentTrack.song_id === song_id;
    const uid = localStorage.getItem('uid');
    const getAlbumNameFromId = (albumId) => {
        const album = albums.find(album => album.album_id ==
albumId);
        return (album ? setAlbumName(album.album_name) :
setAlbumName(''));}
    useEffect(() => {
        const checkIfFavorite = async () => {
            try {
                const response = await
axios.post('http://localhost:8081/userFavoriteSongs', { uid });
```

```
                const favoriteSongs = response.data;
                const isSongInFavorites = favoriteSongs.some(song
=> song.song_id === song_id);
                setIsFavorite(isSongInFavorites);
            } catch (error) {
                console.error('Error checking if song is
favorite:', error);}};
        getAlbumNameFromId(album_id);
        checkIfFavorite();
    }, [uid, song_id]);
    const handleFavoriteClick = async () => {
        try {
            if (isFavorite) {
                await
axios.post('http://localhost:8081/removeFromFavorites', { uid,
song_id });
                setIsFavorite(false);
                if (playlistType === 'favoriteList') {
                    removeFromFavorites(song_id);}
            } else {
                await
axios.post('http://localhost:8081/addToFavorites', { uid, song_id
});
                setIsFavorite(true);}
        } catch (error) {
            console.error('Error updating favorites:', error);
        }};
    const handleAuthorClick = (author_id) => {
        findAuthorById(author_id);
        onChangeMenuItem('author');
    };
    const handleAlbumClick = (albumId) => {
        findAlbumById(albumId);
        onChangeMenuItem('album');}
    return (
```

```
        <div className="flex items-center p-1">
            <div className="mr-4">
                <IconButton onClick={() =>
handleToggleAudio(props, playlistType)}>
                    {isCurrentTrack && isPlaying ? <Pause
className="text-white" /> : <PlayArrow className="text-white" />}
                </IconButton></div>
            <div className="mr-4">
                <img className="w-16 h-16" src={song_image}
alt={song_name} />
            </div>
            <div className="flex-1 min-w-0 mr-4">
                <p className="text-lg font-semibold
truncate">{song_name}</p>
                <p className="text-sm text-white hover:text-blue-
600 cursor-pointer" onClick={() =>
handleAuthorClick(author_id)}>{song_author}</p></div>
            <div className="text-white hover:text-blue-600 cursor-
pointer mr-4" onClick={() => handleAlbumClick(album_id)}>
                {albumName}</div>
            <div className="mr-4">
                <p className="text-sm text-gray-
500">{formattedDuration}</p>
            </div>
            <div className="mr-4">
                <IconButton onClick={handleFavoriteClick}>
                    {isFavorite ? <FavoriteIcon className="text-
red-500" /> : <FavoriteBorderIcon className="text-white" />}
                </IconButton>
            </div>
            <div>
                <TrackMenu songId={song_id} albumId={album_id} />
            </div>
        </div>);};
export default Track;
```

Додаток С

Програмний код компоненти випадного списку пісні

```
import React, { useEffect, useRef, useState, useContext } from
'react';
import MoreVertIcon from '@mui/icons-material/MoreVert';
import { Dropdown, Space } from 'antd';
import { useAuthContext } from '../firebase/authProvider';
import axios from 'axios';
import { AudioContext } from '../context/AudioContext';

const TrackMenu = ({ songId, albumId }) => {
    const { userPlaylists } = useAuthContext();
    const { activePlaylist, menuItem, removeSongFromPlaylist,
setSongsPlaylist, setPlaylistCountAndDuration, findAlbumById,
onChangeMenuItem } = useContext(AudioContext);
    const [ifPlaylist, setIfPlaylist] = useState(true);
    const dropdownRef = useRef(null);

    const allPlaylists = () => {
        return userPlaylists.map(playlist => ({
            key: `playlist_${playlist.playlist_id}`,
            label: playlist.playlist_name,
            onClick: () =>
addSongToPlaylist(playlist.playlist_id),
        }));
    };

    const addSongToPlaylist = async (playlistId) => {
        try {
            const data = { playlist_id: playlistId, song_id:
songId };
            const response = await
axios.post('http://localhost:8081/addSongToPlaylist', data);
            if (response) {
```

```
                alert(response.data.message)
            }
            const songs = await
axios.post('http://localhost:8081/getPlaylistSongs', {
playlist_id: activePlaylist.playlist_id });
            setSongsPlaylist(songs.data);
        } catch (error) {
            console.error('Error adding song to playlist:',
error);
        }
    };

    const deleteFromPlaylist = async () => {
        try {
            if (!ifPlaylist) {
                const data = { playlist_id:
activePlaylist.playlist_id, song_id: songId };
                await
axios.post('http://localhost:8081/removeSongFromPlaylist', data);
                removeSongFromPlaylist(songId);
                const response = await
axios.post('http://localhost:8081/getPlaylist', { playlist_id:
activePlaylist.playlist_id });
                setPlaylistCountAndDuration(response.data)
            }
        } catch (error) {
            console.error('Error removing song from playlist:',
error);
        }
    };

    const handleAlbumClick = (album_id) => {
        findAlbumById(album_id);
        onChangeMenuItem('album');
    };
```

```
    const items = [
        { key: 'album', label: (<div onClick={() =>
handleAlbumClick(albumId)}>Перейти до альбому</div>), },
        { key: 'add', label: 'Додати до плейлиста', children:
allPlaylists() },
        { key: 'delete', label: (<div
onClick={deleteFromPlaylist}>Видалити з плейлиста</div>),
disabled: ifPlaylist, },
    ];

    useEffect(() => {
        if (menuItem === 'userPlaylist') {
            setIfPlaylist(false);
        } else {
            setIfPlaylist(true);
        }
    }, [menuItem]);

    return (
        <div ref={dropdownRef}>
            <Dropdown getPopupContainer={() =>
dropdownRef.current} menu={{ items }}>
                <a onClick={(e) => e.preventDefault()}>
                    <Space>
                        <MoreVertIcon />
                    </Space>
                </a>
            </Dropdown>
        </div>
    );
};

export default TrackMenu;
```

Додаток І

Програмний код компоненти базового плейлиста

```
import { useContext, useState, useEffect } from "react";
import { AudioContext } from "../context/AudioContext";
import Track from "../track/track";
import axios from "axios";
import secondsToMMSS from '../utils/secondsToTime';
const DefaultPage = () => {
    const { activeDefPlaylist, defaultPlaylistSongs,
setDefaultPlaylistSongs } = useContext(AudioContext);
    useEffect(() => {
        if (!activeDefPlaylist) return;
        const fetchSongs = async (def_playlist_id) => {
            try {
                const responseDefaultSongs = await
axios.post('http://localhost:8081/defaultPlaylistSongs');
                const songs = responseDefaultSongs.data;
                const foundSongs = songs.filter(playlist =>
playlist.default_playlist_id === def_playlist_id);
                setDefaultPlaylistSongs(foundSongs);
            } catch (error) {
                console.error('Error fetching playlist songs:',
error);
            }
        };
        fetchSongs(activeDefPlaylist.default_playlist_id);
    }, [activeDefPlaylist, setDefaultPlaylistSongs]);
    return (
        <div className="bg-[#212121] h-full p-2 text-white">
            <div className='bg-[#292929] rounded-2xl h-full p-2
flex flex-col'>
                <div className="flex flex-row border-b-2 border-
white m-3">
```

```
                    <img src={activeDefPlaylist.photo}
alt={activeDefPlaylist.name} className="w-36" />
                    <div className="flex flex-row items-center
justify-center m-3 gap-4">
                        <p className="text-
4xl">{activeDefPlaylist.name}</p>
                        <div>
                            <p>Кількість треків
{activeDefPlaylist.song_count}</p>
                            <p>Час програвання
{secondsToMMSS(activeDefPlaylist.duration)}</p>
                        </div>
                    </div>
                </div>
                <div className='my-3 overflow-auto'>
                    {defaultPlaylistSongs.map((track, index) => (
                        <Track key={index} {...track}
playlistType="defaultPlaylist" />
                    ))}
                </div>
            </div>
        </div>
    );
};
export default DefaultPage;
```

Додаток Р

Програмний код компоненти плейлиста користувача

```
import React, { useContext, useEffect } from "react";
import { AudioContext } from "../context/AudioContext";
import axios from "axios";
import Track from "../track/track";
import secondsToMMSS from '../utils/secondsToTime';
import ModalEditPlaylist from '../components/modal';
import DeleteIcon from '@mui/icons-material/Delete';

const UserPlaylist = () => {
    const { activePlaylist, onChangeMenuItem, setOpenPlaylistEdit,
playlistName, getPlaylistName, playlistCountAndDuration,
setPlaylistCountAndDuration, songsPlaylist, setSongsPlaylist } =
useContext(AudioContext);

    useEffect(() => {
        if (!activePlaylist) return;
        const fetchSongs = async () => {
            try {
                const response = await
axios.post('http://localhost:8081/getPlaylistSongs', {
playlist_id: activePlaylist.playlist_id });
                setSongsPlaylist(response.data);
            } catch (error) {
                console.error('Error fetching playlist songs:',
error);
            }
        };
        getPlaylistName(activePlaylist.playlist_id);
        fetchSongs();
    }, [activePlaylist, setSongsPlaylist, getPlaylistName]);

    useEffect(() => {
```

```
        setPlaylistCountAndDuration({
            song_count: activePlaylist.song_count,
            playlist_duration: activePlaylist.playlist_duration
        });
    }, [activePlaylist, setPlaylistCountAndDuration])

    const deletePlaylist = () => {
        onChangeMenuItem('golovna');
        axios.post('http://localhost:8081/deletePlaylist', {
playlistid: activePlaylist.playlist_id });
    }

    if (!activePlaylist) {
        return <div>Loading...</div>;
    }

    return (
        <div className="bg-[#212121] h-full p-2 text-white">
            <div className='bg-[#292929] rounded-2xl h-full p-2
flex flex-col'>
                <div className="flex flex-row border-b-2 border-
white m-3">
                    <img src={activePlaylist.playlist_image}
alt={activePlaylist.playlist_name} className="w-36" />
                    <div className="flex flex-row items-center
justify-between m-3 gap-4">
                        <div onClick={() =>
setOpenPlaylistEdit(true)}>
                            <p className="text-
4xl">{playlistName}</p>
                        </div>
                        {playlistCountAndDuration && (
                            <div>
                                <p>Кількість треків
{playlistCountAndDuration.song_count}</p>
```

```
                                <p>Час програвання
{secondsToMMSS(playlistCountAndDuration.playlist_duration)}</p>
                            </div>
                        )}
                        <div className="ml-auto">
                            <button
onClick={deletePlaylist}><DeleteIcon /></button>
                        </div>
                    </div>
                </div>
                <div className='my-5 overflow-auto'>
                    {songsPlaylist.length === 0 ? (
                        <p className="ml-3 text-3xl text-
[#757575]">Ви ще не добавили ніяких пісень :)</p>
                    ) : (
                        songsPlaylist.map((song) => (
                            <Track key={song.song_id} {...song}
playlistType="userPlaylist" />
                        ))
                    )}
                </div>
            </div>
            <ModalEditPlaylist />
        </div>
    );
};

export default UserPlaylist;
```

Додаток У

Програмний код компоненти альбому

```
import { useContext, useState, useEffect } from "react";
import { AudioContext } from "../context/AudioContext";
import Track from "../track/track";
import axios from "axios";
import secondsToMMSS from '../utils/secondsToTime';
const AlbumPage = () => {
  const { activeAlbum, albumsSongs, setAlbumsSongs } =
useContext(AudioContext);
  const [songCount, setSongCount] = useState(0);
  const [totalDuration, setTotalDuration] = useState(0);
  useEffect(() => {
    if (!activeAlbum) return;
    const fetchSongs = async () => {
      try {
        const response = await
axios.post('http://localhost:8081/songsByAlbum', { album_id:
activeAlbum.album_id });
        const songs = response.data;
        const duration = songs.reduce((total, song) => total +
song.song_duration, 0);
        setAlbumsSongs(songs);
        setSongCount(songs.length);
        setTotalDuration(secondsToMMSS(duration));
      } catch (error) {
        console.error('Error fetching playlist songs:', error);
      }
    };
    fetchSongs();
  }, [activeAlbum, setAlbumsSongs]);
  return (
    <div className="bg-[#212121] h-full p-2 text-white">
```

```
      <div className='bg-[#292929] rounded-2xl h-full p-2 flex
flex-col'>
        <div className="flex flex-row border-b-2 border-white m-
3">
          <img src={activeAlbum.album_photo}
alt={activeAlbum.album_name} className="w-36" />
          <div className="flex flex-row items-center justify-
center m-3 gap-4">
            <p className="text-4xl">{activeAlbum.album_name}</p>
            <div>
              <p>Кількість треків {songCount}</p>
              <p>Час програвання {totalDuration}</p>
            </div>
          </div>
        </div>
        <div className='overflow-auto my-3'>
          {albumsSongs.map((track, index) => (
            <Track key={index} {...track} playlistType="albumType"
/>
          ))}
        </div>
      </div>
    </div>
  );
};
export default AlbumPage;
```

Додаток К

Програмний код компоненти автора

```
import { useContext, useState, useEffect } from "react";
import { AudioContext } from "../context/AudioContext";
import Track from "../track/track";
import axios from "axios";
import Album from '../track/albums';
const AuthorPage = () => {
  const { activeAuthor, albums, onChangeMenuItem, onChangeAlbum,
authorsSongs, setAuthorsSongs } = useContext(AudioContext);
  const [authorAlbums, setAuthorAlbums] = useState([]);
  if (!activeAuthor) {
    return <div>Loading...</div>;
  }
  useEffect(() => {
    if (!activeAuthor) return;
    const fetchSongs = async () => {
      try {
        const response = await
axios.post('http://localhost:8081/songsByAuthor', { author_id:
activeAuthor.author_id });
        setAuthorsSongs(response.data);
        const authorAlbum = albums.filter(album => album.author_id
=== activeAuthor.author_id);
        setAuthorAlbums(authorAlbum);
      } catch (error) {
        console.error('Error fetching playlist songs:', error);
      }
    };
    fetchSongs();
  }, [activeAuthor, setAuthorsSongs]);
  return (
    <div className="bg-[#212121] h-full p-2 text-white">
```

```
      <div className='bg-[#292929] rounded-2xl h-full p-2 flex
flex-col overflow-auto'>
        <div className="flex flex-row border-b-2 border-white m-
3">
          <img src={activeAuthor.author_photo}
alt={activeAuthor.author_name} className="w-36" />
          <div className="flex flex-row items-center justify-
center m-3 gap-4">
            <p className="text-4xl">{activeAuthor.author_name}</p>
          </div>
        </div>
        <div className='my-3'>
          {authorsSongs.map((track, index) => (
            <Track key={index} {...track}
playlistType="authorType" />
          ))}
        </div>
        <div className="m-3">
          {authorAlbums.map((album, index) => (
            <div className="-m-3" onClick={() => {
onChangeMenuItem('album'); onChangeAlbum(album); }}
key={`${album.album_id}-${index}`}>
              <Album {...album} />
            </div>))}
</div></div></div>);};
export default AuthorPage;
```

Додаток Л

Програмний код компоненти улюблених пісень користувача

```
import React, { useEffect, useContext, useState } from 'react';
import axios from 'axios';
import { AudioContext } from "../context/AudioContext";
import Track from "../track/track";
import secondsToMMSS from '../utils/secondsToTime';

const Favorite = () => {
    const { favoriteSongs, setFavoriteSongs } =
useContext(AudioContext);
    const uid = localStorage.getItem('uid');
    useEffect(() => {
        const fetchFavoriteSongs = async () => {
            try {
                const response = await
axios.post('http://localhost:8081/userFavoriteSongs', { uid });
                setFavoriteSongs(response.data);
            } catch (error) {
                console.error('Error fetching favorite songs:',
error);}};
        fetchFavoriteSongs();}, []);

    const removeFromFavorites = (songId) => {
        setFavoriteSongs(favoriteSongs.filter(song => song.song_id
!== songId));
    };
    return (
        <div className="bg-[#212121] h-full p-2 text-white">
            <div className='bg-[#292929] rounded-2xl h-full p-2
flex flex-col'>
                <div className="flex flex-row border-b-2 border-
white m-3">
```

```
                    <p className="text-3xl m-3">Ваш улюблений
список треків</p>
                </div>
                <div className='my-3 overflow-auto'>
                    {
                        favoriteSongs.length === 0 ? (
                            <p className="ml-3 text-3xl text-
[#757575]">Ви ще не добавили ніяких пісень :)</p>
                        ) : (
                            favoriteSongs.map((track, index) => (
                                <Track key={index} {...track}
playlistType="favoriteList"
removeFromFavorites={removeFromFavorites} />))))}
                </div>
            </div>
        </div>);};
export default Favorite;
```

Додаток М

Програмний код модального вікна

```
import { useContext, useState, Fragment, useEffect } from 'react';
import Button from '@mui/joy/Button';
import FormControl from '@mui/joy/FormControl';
import Input from '@mui/joy/Input';
import Modal from '@mui/joy/Modal';
import ModalDialog from '@mui/joy/ModalDialog';
import DialogTitle from '@mui/joy/DialogTitle';
import Stack from '@mui/joy/Stack';
import { AudioContext } from '../context/AudioContext';
const ModalEditPlaylist = () => {
    const { openPlaylistEdit, setOpenPlaylistEdit, activePlaylist,
newName, playlistName } = useContext(AudioContext);
    const [inputValue, setInputValue] = useState('');
    const id = activePlaylist.playlist_id;
    useEffect(() => {
        setInputValue(playlistName);
    }, [playlistName]);
    const handleChange = (event) => {
        if (event.target.value.length <= 28) {
            setInputValue(event.target.value);
        }};
    return (
        <Fragment>
            <Modal open={openPlaylistEdit} onClose={() =>
setOpenPlaylistEdit(false)}>
                <ModalDialog style={{ backgroundColor: '#191919',
color: '#fff' }}>
                    <DialogTitle style={{ backgroundColor:
'#212121', color: '#fff' }}>Введіть нову назву
плейлиста</DialogTitle>
```

```
                    <form onSubmit={(event) => {
event.preventDefault(); setOpenPlaylistEdit(false); newName(id,
inputValue); }}>
                        <Stack spacing={2} style={{ padding:
'20px' }}>
                            <FormControl>
                                <Input inputprops={{ maxLength:
28, style: { color: '#fff' } }} value={inputValue}
onChange={handleChange} required />
                            </FormControl>
                            <Button type="submit" style={{
backgroundColor: '#424242', color: '#fff' }}>Змінити</Button>
                        </Stack>
                    </form>
                </ModalDialog>
            </Modal>
        </Fragment >
    );
}
export default ModalEditPlaylist;
```

Додаток П

Таблиця П.1 Кошторис витрат на проєктування

Найменування статей витрат	Сума, грн	Обґрунтування
Зарплата проєктувальників.	24150	Сума зарплати проєктувальників.
Відрахування на соціальні потреби.	3300	Сума відрахування на соціальні потреби.
Контрагентські роботи і послуги.	2250	Сума для контрагентських послуг та робіт.
Витрати на відрядження.	3000	Сума для відрядження працівників.
Інші прямі витрати.	6750	Сума для інших прямих витрат (програмне забезпечення, обладнання тощо).
Усього прямих витрат.	39450	Сума прямих витрат.
Накладні витрати.	11835	Сума накладних витрат (витрати на комунальні послуги, оренда офісу тощо).
Планові накопичення.	10257	Сума на реалізацію майбутніх ідей проєкту.
Усього, кошторисна вартість проєкту.	61542	Сума планових накопичень, прямих та накладних витрат.
Податок на додану вартість.	12308,5	Сума податку кошторисної вартості.
Загалом, договірна ціна розробки Зп.	174842,5	Загальна вартість розробки проєкту.