

НАВЧАЛЬНО–НАУКОВИЙ КОМПЛЕКС  
“ІНСТИТУТ ПРИКЛАДНОГО СИСТЕМНОГО АНАЛІЗУ”  
НАЦІОНАЛЬНОГО ТЕХНІЧНОГО УНІВЕРСИТЕТУ УКРАЇНИ  
“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ІМЕНІ ІГОРЯ  
СІКОРСЬКОГО”  
Кафедра Математичних методів системного аналізу

КУРСОВА РОБОТА  
з дисципліни  
Системи баз даних  
на тему: “Стрімінговий сервіс музики”

Студента III курсу групи КА–76

Спеціальність: 122 Комп’ютерні науки

Лисова Богдана Сергійовича

Керівник: асистент Афанасьєва І. В.

Національна оцінка: \_\_\_\_\_

Кількість балів: \_\_\_\_\_

Члени комісії: \_\_\_\_\_

(підпис) (вчене звання, науковий ступінь, прізвище та ініціали)

\_\_\_\_\_  
(підпис) (вчене звання, науковий ступінь, прізвище та ініціали)

\_\_\_\_\_  
(підпис) (вчене звання, науковий ступінь, прізвище та ініціали)

Київ – 2019 рік

## АНОТАЦІЯ

У цій роботі представлена розробка стрімінгового сервісу музики “HeadPhones”.

Широка аудиторія в сучасному світі користується навушниками: в транспорті, вдома, на роботі. Люди слухають музику на різних сервісах, де кожен може створити свої плейлисти, додати туди будь-які пісні, дізнатись інформацію про виконавців та їх альбоми.

Проект зроблено так, щоб користувач насолоджувався зручним інтерфейсом та не витрачав час на зайву інформацію чи не розбирався, що роблять незрозумілі кнопки.

Дана робота буде корисна всім людям, хто полюбляє слухати музику. Простота інтерфейсу робить користування сервісом простим та зрозумілим.

## ABSTRACT

This work presents the development of the streaming service "HeadPhones".

A wide audience in today's world uses headphones: in transport, at home, at work. People listen to music on different services, where anyone can create their own playlists, add any songs there, learn about artists and their albums.

The work is designed so that the user enjoys a user-friendly interface and does not waste time on unnecessary information or understanding what these buttons do.

This work will be useful for all people who enjoy listening to music. The simplicity of the interface makes the use of the service simple and clear.

АНОТАЦІЯ.....	2
ABSTRACT.....	2
Вступ.....	4
РОЗДІЛ 1. ПОСТАНОВКА ЗАДАЧІ РОЗРОБКИ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ДЛЯ Стрімінгового сервісу музики.....	6
Обґрунтування доцільності розробки .....	6
Порівняння аналогів, вибір базового варіанту для розробки .....	7
Обґрунтування вибору інструментів розробки .....	8
Постановка задачі проектування .....	10
РОЗДІЛ 2 ПРОЕКТУВАННЯ БАЗИ ДАНИХ .....	12
Концептуальна модель інформаційної системи. Діаграми потоків даних.....	12
Інфологічна модель бази даних. Діаграми “сутність – зв’язок” .....	13
РОЗДІЛ 3. ОПИС РОЗРОБЛЕНОГО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ .....	15
Реалізація взаємодії прикладної програми зі спроектованою базою даних .....	15
Інструкція користувача .....	16
Результати роботи програми .....	21
ВИСНОВКИ .....	22
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ .....	24
ДОДАТОК А. ЛІСТИНГ НАЛАШТУВАННЯ ЗАПИТІВ ДО БАЗИ ДАНИХ.....	26
ДОДАТОК Б. DDL (DATA DEFINITION LANGUAGE) КОМАНДИ .....	30

## ВСТУП

У цій роботі представлена розробка стрімінгового сервісу музики “HeadPhones”.

Вибір був обумовлений можливістю інтенсивного вивчення веб розробки, в якості фронтенд, бекенд розробника та, що важливіше в даний момент, розробника баз даних. Це дає змогу опанувати “Full stack” розробку (розробка програмного забезпечення клієнт–серверної архітектури, як із клієнтської сторони, так і зі сторони серверу).

Найвагомішою частиною при вивченні предмету “Системи баз даних” є розробка структури і відношень між базами даних на сайті, та визначення алгоритмів операцій над ними.

При цьому розумно розділити розробку на такі складові: зберігання/перегляд/редагування/видалення (CRUD – “Create, read, update and delete”) даних, їх обробка і їх відображення клієнту.

Проект зроблено так, щоб користувач насолоджувався зручним інтерфейсом та не витрачав час на зайву інформацію чи не розбирався, що роблять незрозумілі кнопки. Хоча на стороні програми виконується багато операцій, що залежать від дій користувача.

В цій роботі використовується доволі високорівневі, навіть з точки зору сучасного світу програмістів, технології. Такий вибір був зроблений навмисне, задля слідування підходу до вивчення будь–якої системи абстракцій, а саме підходом “зверху–вниз”. Тобто в цій роботі приводиться практичний вступ до веб розробки з точки зору абстракції від транспортного і тим паче фізичного рівня мереж.

Використовуються такі технології:

- JavaScript programming language;
- Redux js;
- React js;
- HTML (HyperText Markup Language);

- CSS (Cascading Style Sheets);
- phpMyAdmin devtools;
- Express library;
- Webpack;
- Axios library;
- Node js;
- MySql;
- Visual Studio Code IDE (Integrated development environment);

## РОЗДІЛ 1. ПОСТАНОВКА ЗАДАЧІ РОЗРОБКИ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ДЛЯ СТРІМІНГОВОГО СЕРВІСУ МУЗИКИ

### **Обґрунтування доцільності розробки**

З приходом нових технологій, а саме мережі Інтернет закономірно змінилися і способи донесення і використання інформації. В наш час більшість інформації ми сприймаємо через універсальний інтерфейс, який є невід’ємною складовою, цеглиною, вебу – це, сайт.

Сайт розглядається, як універсальний інтерфейс, адже і на мобільному телефоні і на персональному комп’ютері будь-яких співвідношень екрану він відображається і функціонує майже однаково. Треба зауважити, що браузер уже давно став платформою, навіть віртуальною машиною, яка підтримує відповідне програмне забезпечення (HTML, CSS, JS (JavaScript)) на “будь-якому” обладнанні. Саме тому вигідно обрати браузер, як цільову платформу для розробки.

Слухання музики – це той процес, яким займається багато людей в наш час. Зараз існує безліч сервісів, які дають такий функціонал як ця робота, навіть набагато ширше, адже компанії мають на це ресурси: сервери, великі бази, контракти зі співаками та дозволи на користування їх піснями.

Дуже цікаво розібратись в тому, як це насправді працює зсередини. Надати користувачу можливість просто слухати музику без зайвих опитів та інформації – справжній виклик для сервісу. Цей сайт виконує цю задачу.

## **Порівняння аналогів, вибір базового варіанту для розробки**

Перед тим як приступити до розробки самого продукту, необхідно переглянути варіанти схожих програмних рішень.

Існує багато схожих сервісів, але ця робота базується на знаннях таких гігантів: Youtube music [10], Google play music [11], Spotify [12].

1. Визначимо, які функції даних сайтів є бажаними для нашого проекту:
2. Сайти мають чудовий дружній до користувача інтерфейс.
3. Їх особливість – надати користувачу релевантні конкретно для нього пісні. Дізнатись, що слухає користувач та пропонувати це.
4. Також важливою частиною їхньої роботи є реєстрація і авторизація користувачів.
5. Необхідна річ – унікальність кожного користувача, а саме його плейлисти та підбірка музики.

Даний проект, через недостатньо гарне володіння технологією нейронних мереж та малу команду (1 людина), не дає змогу індивідуальної підбірки пісень. Але також надає змогу пошуку доступних треків.

Хотілось би зазначити, що все не обмежується функціоналом аналогів. Щоб проект вигідно виділявся необхідно додати власного бачення в його задумку. Уведемо додатковий авторський функціонал:

1. Більш зручний інтерфейс. Мінімум кнопок.
2. Візуалізація частот пісень, що досі в розробці.

Отож, визначивши цільові функції потенційних конкурентів можна приступити до обрання відповідних інструментів.

## Обґрунтування вибору інструментів розробки

Необхідно обрати інструменти для створення сайту. Оскільки є бізнес-логіка сайту, то можна обрати популярні інструменти веб розробки та обґрунтувати їх вибір.

В якості фронт-енд частини було обрано JavaScript бібліотеку React, розробниками якого є компанія Facebook. Вони регулярно підтримують свій продукт та випускають нові оновлення. Дуже багато сайтів написано з використанням React js. Його переваги полягають в тому, що звичайні HTML теги можна писати всередині JavaScript коду, робити маніпуляції, керувати даними.

Також React робить маршрутизацію сайту дуже простою та пропагує SPA (single page application) підхід [4], коли сторінка не перезавантажується при запитах до серверу, бази даних, та при переходах між url.

В допомогу React js існує бібліотека Redux js [7-9], яка надає функціонал додаткового сховища, своєрідної бази даних, всередині програми. Таким чином можна керувати будь-якими даними та відображати їх динамічно без перезавантаження сторінки.

Для роботи з бек-ендом було обрано бібліотеку Express js [1-3], яка надає можливість дуже легко поводитись з серверною частиною. Ще однією перевагою являє собою використання тієї ж мови програмування на бек частині, що і на фронт. Це значно пришвидшує розробку (Додаток А).

Без цієї бібліотеки треба було би написати набагато більше коду, а так файл серверу виглядає дуже просто: підключення бази даних та обробка GET, PUT, DELETE, POST запитів. Це дозволяє відправляти в базу даних SQL (structured query language) команди, які прописуються безпосередньо в JavaScript коді.

Express просто описує запити в чотири кроки:

1. Що за запит?
2. За якою адресою?



3. Яка SQL команда?

4. Відповідь у форматі JSON.

В якості бази даних була обрана СУБД (Система управління базами даних) MySQL, яка адмініструється за допомогою сервісу phpmyadmin. Це надало змогу легко керувати даними, виконувати SQL команди, робити потрібні зв'язки між базами та зберігати програму в консистентному стані.

## Постановка задачі проектування

Проектування сайту – закономірний крок після визначення з інструментами розробки. Треба, щоб сайт був багатосторінковим, містив можливості описані в пункті “Порівняння програм-аналогів, вибір базового варіанту для розробки”. Залишається лише перевести інтерфейс визначений вище в його імплементацію, описану нижче.

Нижче викладений приблизний план розробки програмного продукту. Слід зауважити, що розробка – комплексний процес і що порядок слідування пунктів плану може переплітатись, це – асинхронний процес.

1. Написати основу фронт частини до сайту, структурувати його задля максимізації перевикористання розмітки і стилів.
2. Визначити стандартні для операції (“User stories”).
3. Визначити структуру баз даних і їх поведінку в різних сценаріях.
4. Написати сценарії поведінки для сторінок (бекенд).
5. Підключити віддалену БД (базу даних).
6. “Деплой” (викладення сайту на хостинг, налаштування, деактивація режиму відладки).

Загальний план описує розробку загалом, на високому рівні абстракції. Щоб зрозуміти проект більш детально розіб’ємо пункти, по можливості, на підпункти. Таким чином отримаємо алгоритм створення програмного продукту:

1. Написати основу фронтенду до сайту, структурувати його задля максимізації перевикористання розмітки і стилів.
  - 1.1. Визначити палітру кольорів.
  - 1.2. Зробити мокапи дизайну сайту.
  - 1.3. Власне написати HTML і CSS до основної сторінки.
  - 1.4. Можливо до декількох допоміжних сторінок.
  - 1.5. Наслідувати всі інші сторінки від базової, по можливості.
2. Визначити стандартні операції (“User stories”):

- 2.1. Переглянути аналоги і співставити зі своїм уявленням про проект.
- 2.2. Провести опитування у цільовій аудиторії: чого б вони бажали від такого сайту.
- 2.3. Власне саме створення сценаріїв.
3. Визначити структуру баз даних і їх поведінку в різних сценаріях.
4. Написати сценарії поведінки для сторінок (бекенд).
5. Підключити віддалену БД.
6. Деплой.

## РОЗДІЛ 2 ПРОЕКТУВАННЯ БАЗИ ДАНИХ

### Концептуальна модель інформаційної системи. Діаграми потоків даних

Відповідно до обраної теми курсового проекту, було проведено аналіз і створена діаграма потоків даних (рис. 2.1.), на якій зображені можливості проекту та його дані.

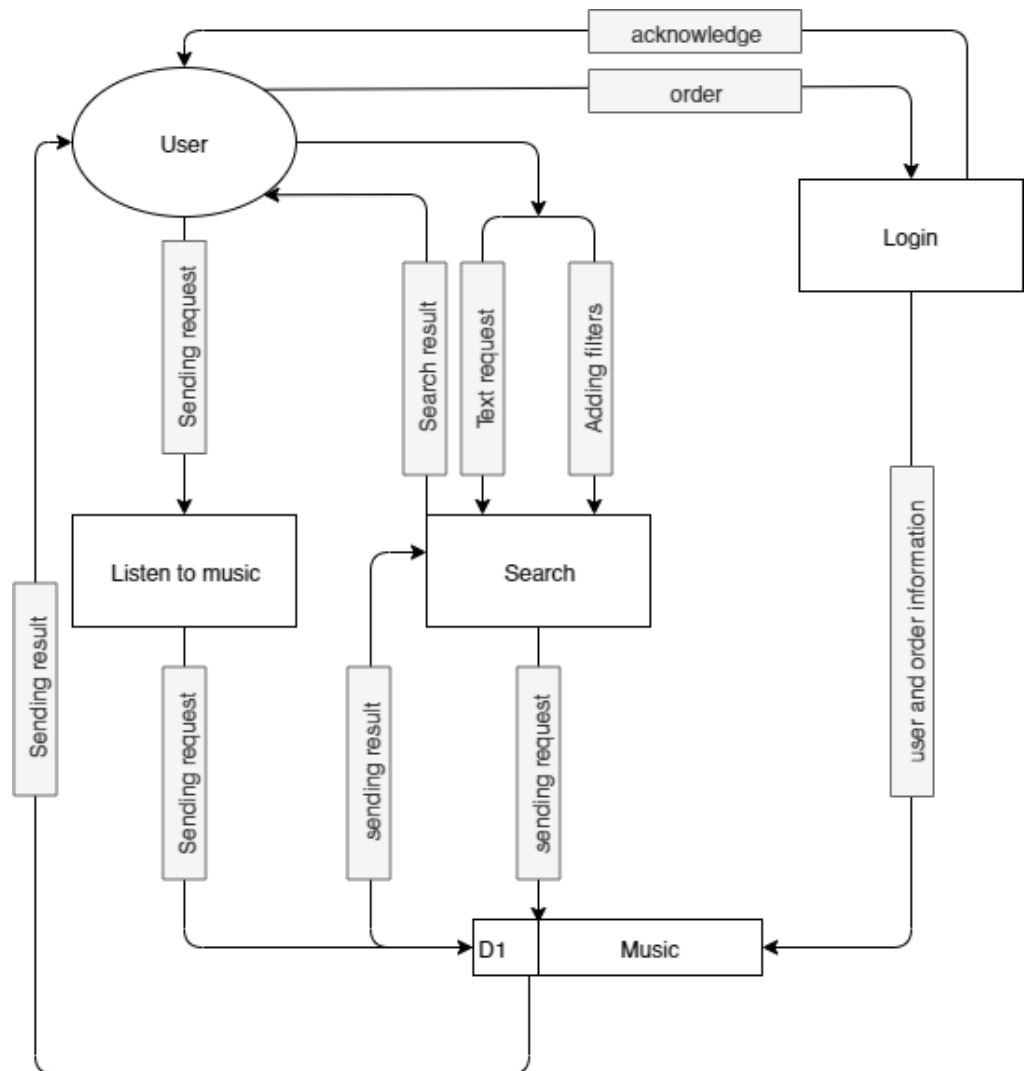


Рис. 2.1. Діаграма потоків даних для "HeadPhones"

## Інфологічна модель бази даних. Діаграми “сутність – зв’язок”

На основі діаграми потоків даних побудуємо діаграму сутність–зв’язок (рис. 2.4.).

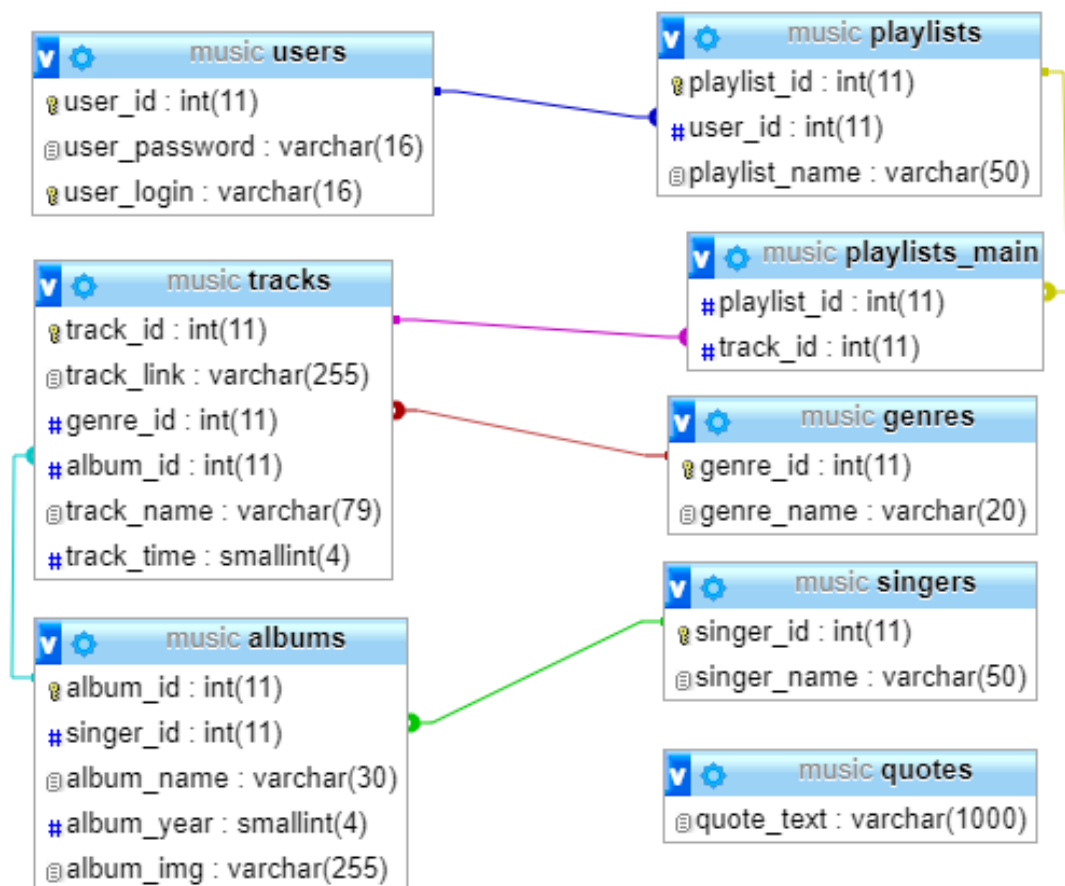


Рис. 2.4. Діаграма “Сутність – зв’язок” бази даних music

Оскільки у даній базі даних всі атрибути є простими, всі домени містять скалярні значення і немає повторів кортежів у відношенні, то відношення знаходиться у першій нормальній формі (1НФ).

Так як кожен неключовий атрибут залежить від первинного ключа, то відношення знаходяться у 2НФ.

Оскільки всі неключові поля, вміст яких може бути віднесений до декількох записів таблиці винесені у окремі таблиці, то усі відношення знаходяться у 3НФ.

Так як всі багатозначні функціональні залежності фактично являються функціональними залежностями від ключів, то БД знаходиться у 4НФ.

Оскільки відсутні складні залежні з'єднання між атрибутами, то відношення знаходиться у 5НФ.

База даних не містить жодних інших обмежень крім обмежень доменів і обмежень ключів, тобто БД знаходиться у доменно–ключовій нормальній формі.

## РОЗДІЛ 3. ОПИС РОЗРОБЛЕНОГО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

### Реалізація взаємодії прикладної програми зі спроектованою базою даних

Для роботи з БД була використана СУБД MySQL 5.7. Створені таблиці та зв'язки між ними.

Програма взаємодіє з базою даних завдяки бек частині. У файлі `server.js` йде підключення бази (рис. 3.1), а потім прописується логіка кожного запиту, який робиться з фронт частини (рис. 3.2).

```
const pool = mysql.createPool({
  host: "zanner.org.ua",
  port: "33321",
  user: "ka7608",
  password: "123123",
  database: "music"
});
```

Рис. 3.1. Підключення бази даних.

```
app.get("/playlists", function(req, res){
  const user_id = req.query.user_id;
  pool.query(
    `SELECT DISTINCT playlist_id, playlist_name
    FROM playlists
    WHERE user_id = ${user_id}`, user_id,
    function(err, data) {
      if(err) {
        return console.log(err);
      }
      res.json(data);
    }
  );
});
```

Рис. 3.2. Приклад реалізації логіки запиту.

## Інструкція користувача

Користувача зустрічає форма логіну або створення акаунту (рис. 3.3. та рис. 3.4.).

Можна зайти з існуючого акаунту або зареєструватись. Якщо користувач введе неправильний логін або пароль, чи неправильно введе дані, йому покажуть відповідне повідомлення (рис. 3.5. та рис. 3.6.).

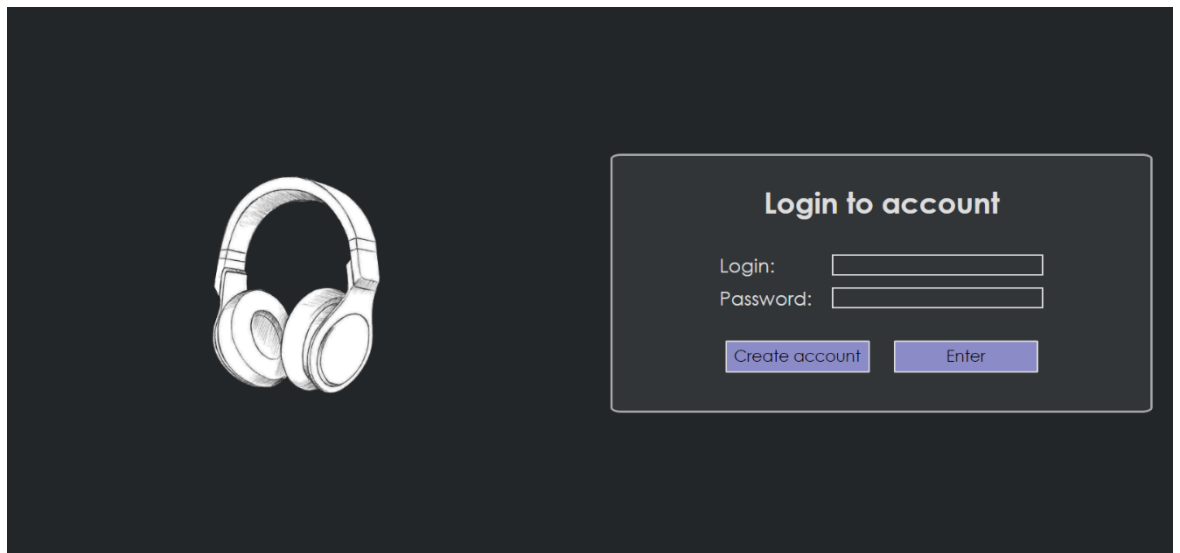
The image shows a dark-themed login interface. On the left is a white line-art icon of a pair of headphones. On the right is a light gray box titled "Login to account". Inside the box, there are two input fields: "Login:" and "Password:". Below the "Login:" field is a blue button labeled "Create account". Below the "Password:" field is a blue button labeled "Enter".

Рис. 3.3. Форма логіна.

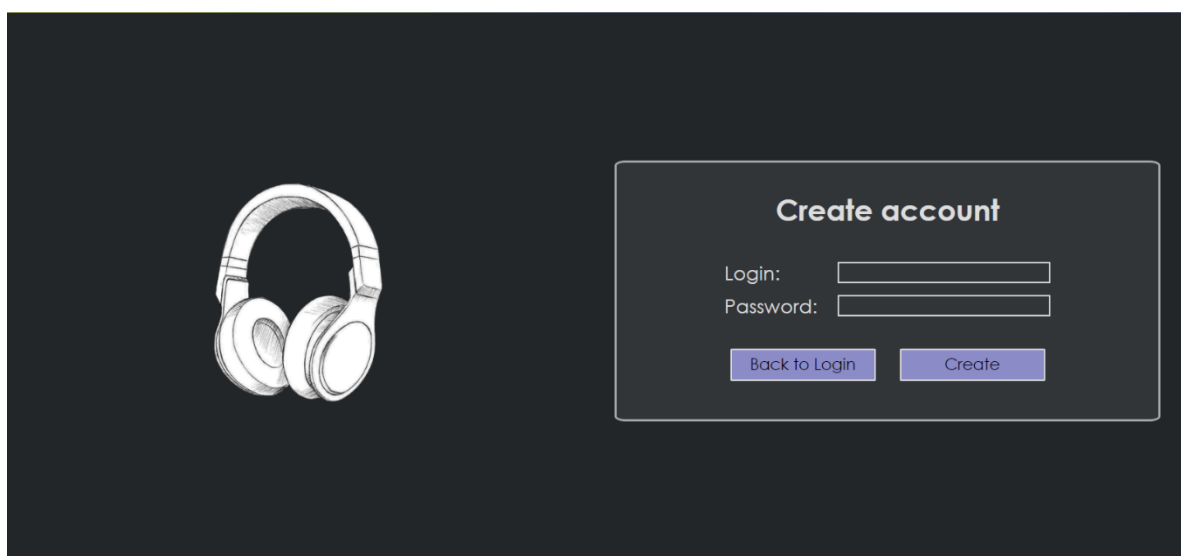
The image shows a dark-themed "Create account" form. On the left is a white line-art icon of a pair of headphones. On the right is a light gray box titled "Create account". Inside the box, there are two input fields: "Login:" and "Password:". Below the "Login:" field is a blue button labeled "Back to Login". Below the "Password:" field is a blue button labeled "Create".

Рис. 3.4. Форма створення акаунту.

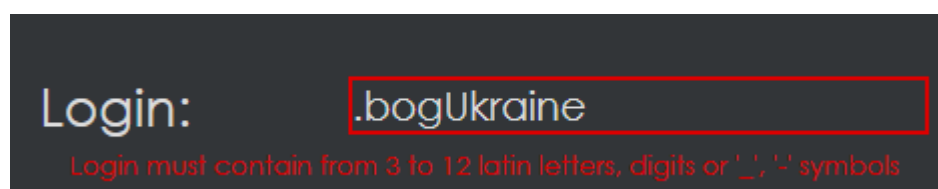
The image shows a close-up of the login form with an error message. The "Login:" label is on the left. The input field contains the text ".bogUkraine" and is highlighted with a red rectangular border. Below the input field, a red error message reads: "Login must contain from 3 to 12 latin letters, digits or '\_', '-' symbols".

Рис. 3.5. Попередження про неправильно введений логін.



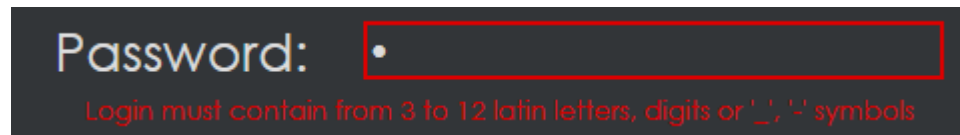


Рис. 3.6. Попередження про неправильно введений пароль.

Відразу після цього користувач потрапляє на домашню сторінку (рис. 3.7.), де йому відображаються 7 пісень, його логін, та випадково підібрана цитата про музику від відомих людей. Користувач має можливість перейти до Library або обрати будь-який свій плейлист (рис. 3.8.).

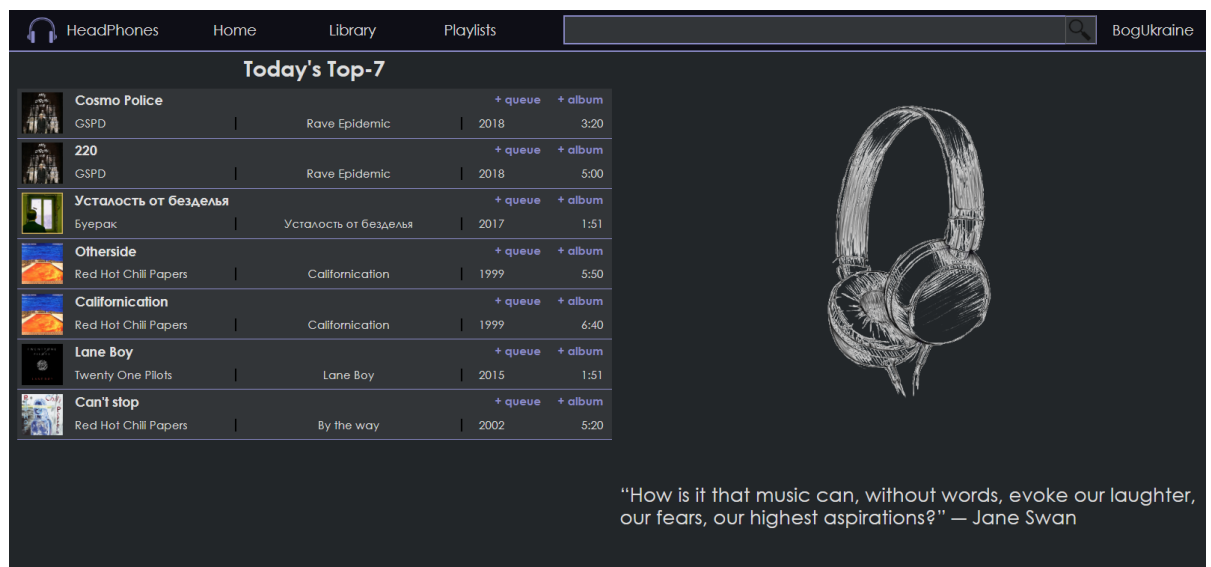


Рис. 3.7. Сторінка Home.

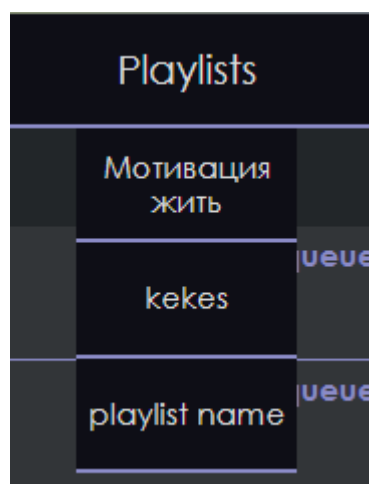


Рис. 3.8. Плейлисти користувача, список яких відкривається при наведенні курсору на Playlists.

Користувач може взаємодіяти з піснями. При натисканні на будь-яку пісню відкриється плеєр (рис. 3.9.).

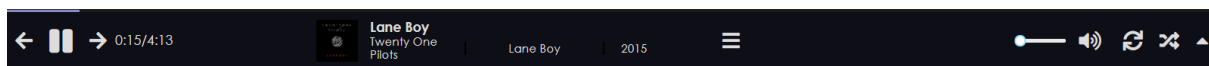


Рис. 3.9. Плеєр.

У ньому можна перемикається на наступну в черзі пісню, повертатись до попередньої, поставити на паузу, зробити гучніше або тихіше. Можна перемішати пісні – наступний трек буде випадковим із черги. Можна поставити пісню на повторення. Є можливість перемотувати пісню: при наведенні на лінію, що заповнюється, вона зробиться ширше і при натисканні на будь-яку її точку, трек почне грати в конкретному обраному місці по часу.

Також, є можливість подивитись, яка черга зараз грає, натиснувши на трикутник в правому нижньому кутку (рис. 3.10.)

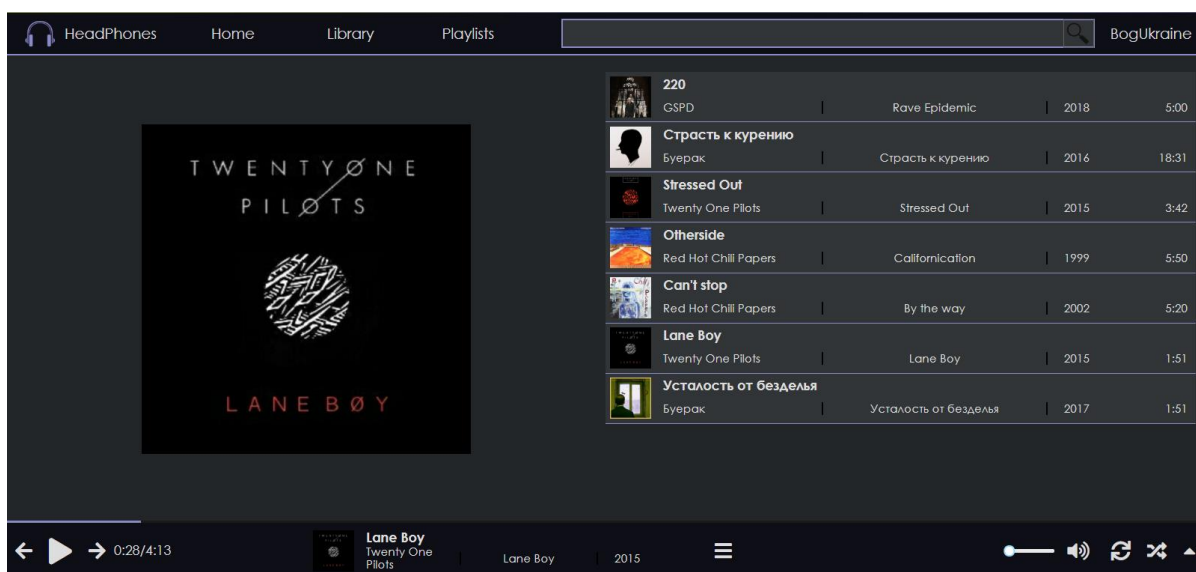


Рис. 3.10. Черга пісень.

У цьому місці можна перемикає треки. Зліва відображається обкладинка альбому обраної пісні.

Важливо, що треки з головного меню можна додавати до черги відтворення та додавати до вибраного альбому (рис. 3.11. та рис. 3.12.) у правому верхньому кутку.

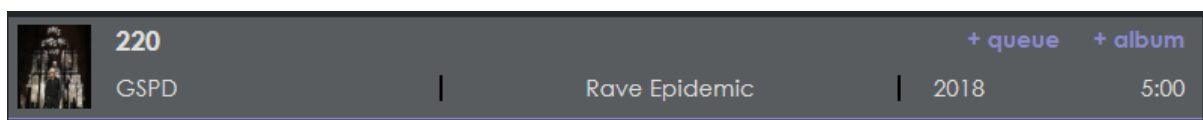


Рис. 3.11. Вигляд треку.

	<b>220</b> GSPD	Rave Epidemic	2018	<a href="#">+ queue</a>	<a href="#">+ album</a>	Мотивация жить
	<b>Страсть к курению</b> Буерак	Страсть к курению	2016	<a href="#">+ queue</a>		kekes
	<b>Stressed Out</b> Twenty One Pilots	Stressed Out	2015	<a href="#">+ queue</a>		playlist name
	<b>Otherside</b> Red Hot Chili Papers	Californication	1999	<a href="#">+ queue</a>	<a href="#">+ album</a>	3:42
	<b>Can't stop</b> Red Hot Chili Papers	By the way	2002	<a href="#">+ queue</a>	<a href="#">+ album</a>	5:20
	<b>Lane Boy</b> Twenty One Pilots	Lane Boy	2015	<a href="#">+ queue</a>	<a href="#">+ album</a>	1:51
	<b>Усталость от безделья</b> Буерак	Усталость от безделья	2017	<a href="#">+ queue</a>	<a href="#">+ album</a>	1:51

Рис. 3.12. Додавання треку до обраного альбому.

Важливе уточнення: у користувача може бути максимум 3 плейлисти з необмеженою кількістю треків у ньому. Якщо на сайт заїде користувач, у якого менше 3х плейлистів або щойно зареєстрований користувач, то він, при наведенні на Playlists, побачить кнопку “Create playlists” (рис. 3.13.). У цьому випадку створиться пустий плейлист без пісень та стандартною назвою playlist name.

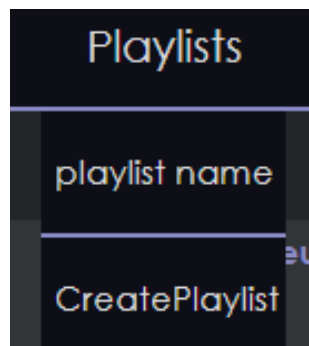


Рис. 3.13. Можна створити плейлист.

У вкладці обраного плейлиста відобразиться його інформація: назва, кількість треків, скільки часу треки продовжуються та всі пісні (рис. 3.14.). Також, можна змінити назву плейлиста (рис. 3.15.). У вкладці library знаходяться випадкові треки (рис. 3.16.).

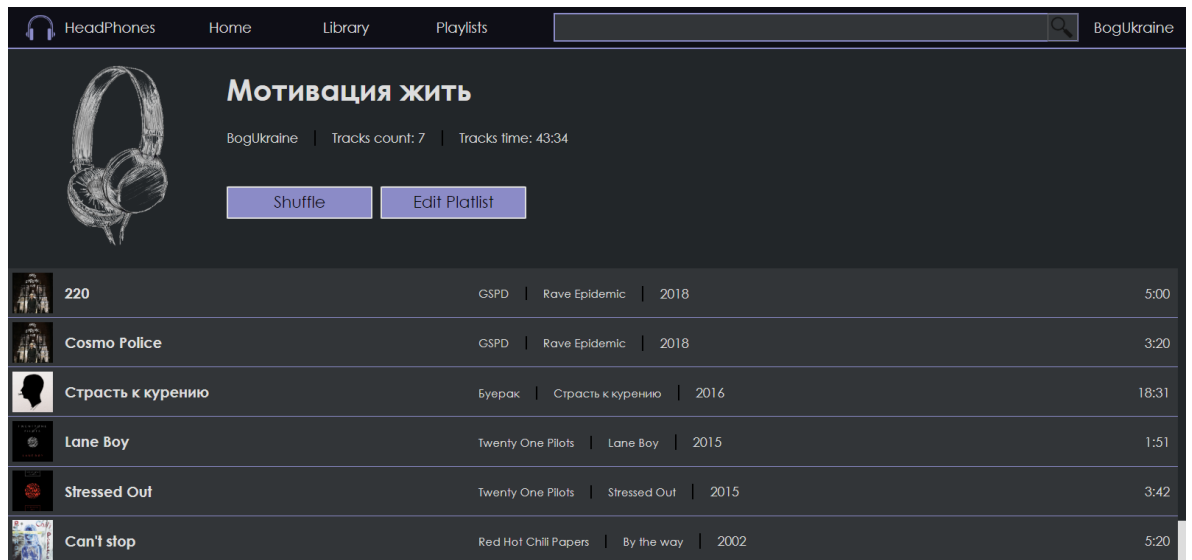


Рис. 3.14. Вигляд вкладки Playlist.

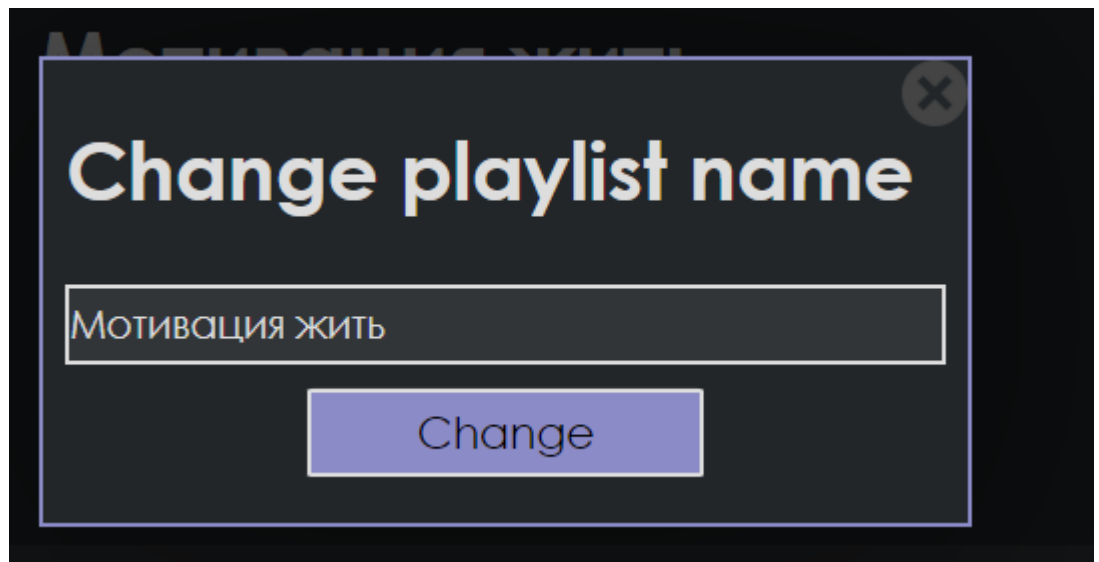


Рис. 3.15. Зміна назви плейлиста.

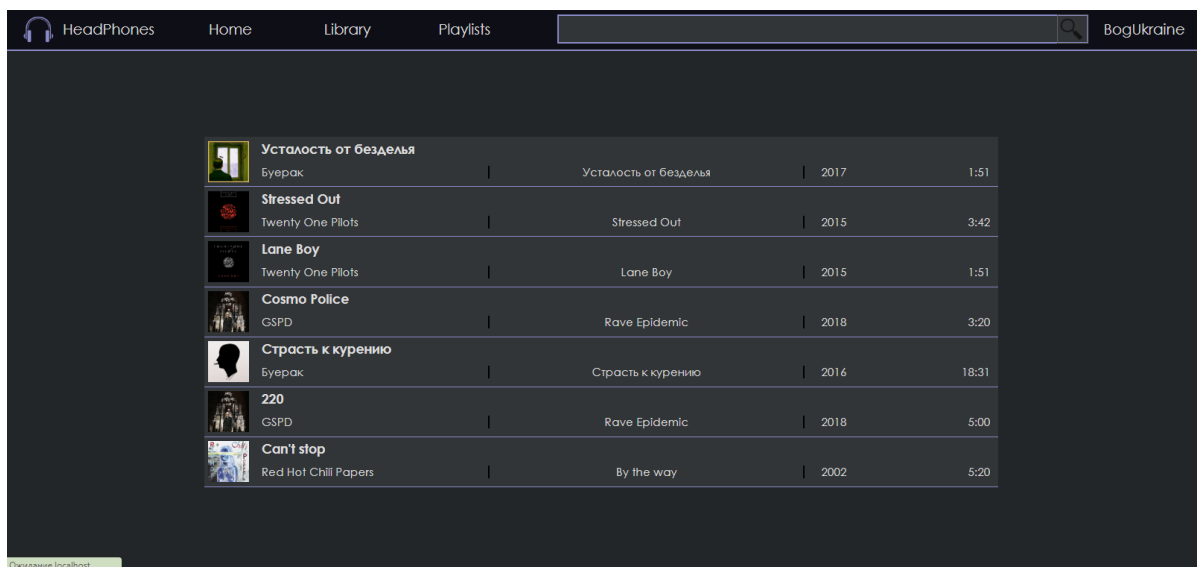


Рис. 3.16. Вкладка Library.

## **Результати роботи програми**

Оскільки даний програмний продукт є сайтом, тобто інструментом для користувачів – не розробників (на відміну від бібліотеки, модуля), то результатами роботи є графічний користувацький інтерфейс – сайт.

Будь-яка людина може насолодитись прослуховуванням музики на цьому сайті, де дружній інтерфейс допомагає не витратити зайвий час на пошуки різних функцій, адже робота ввібрала головні можливості таких сервісів, щоб не перевантажувати користувача зайвим.

Важливим результатом є створена база даних (Додаток Б), яка дає користувачу можливість користуватись інформацією на сайті.

Також результатом роботи програми стали набуті навички:

- React js.
- Redux js.
- HTML.
- CSS.
- Web розробка.
- Робота з базами даних.
- Оптимізація програмного продукту завдяки новим технологіям Redux sagas.
- Expres js.
- Встановлення зв'язку програми з базою даних.
- Створення single page application.

## ВИСНОВКИ

У цій роботі розроблений стримінговий сервіс музики, який, нажаль, є скоріш тренувальним проектом, ніж готовим продуктом. Це пояснюється авторським правом: для того, щоб розміщувати пісні на своєму сервісі – треба підписувати контракти зі співаками або компаніями. У цьому випадку, це піратство.

Незважаючи на це, проект буде розроблятися і надалі, щоб покращити навички веб розробки. Що не зроблено, але буде виконано пізніше:

1. Шифрування даних, щоб зробити користування сайтом безпечним.  
Це стосується в першу чергу JWT (JSON (JavaScript Object Notation) Web Token) – шифрування логіну, паролю та надання користувачу токен, який зберігається у нього локально [5, 6].
2. Графічне зображення частот кожної пісні.
3. Перемикання теми сайту з темної на світлу.
4. Пошук по пісням.
5. Інформація про альбоми.
6. Додати адміністратора, який зможе додавати пісні в базу зручним способом, за допомогою відповідного інтерфейсу сайту, без користування `phpmyadmin`.
7. Надати користувачу обмежені права користування базою даних.

Щоб зробити ці речі – треба не мало часу, але тоді тренування на такому великому проекті можна буде вважати успішним. Але при цьому код був написаний з урахуванням нових технологій, асинхронних запитів та оптимізованих процесів.

Щодо навичок з курсу “Системи баз даних”, то протягом роботи були розроблені:

1. концептуальна модель бази даних: діаграми потоків даних;
2. інфологічна модель бази даних. Діаграма “сутність–зв’язок”;

Оскільки робота є практичною, то в процесі розробки були здобуті навички роботи з різними технологіями та інструментами React, Redux, redux sagas, express.

Було розвинуте вміння створювати SPA – single page application – яке використовує CSR – client side rendering – коли дані підгружаються динамічно без перезавантаження сторінки.

Загалом було створено продукт готовий для використання, але є багато речей, які можна і треба покращити.

## СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Node.js – Express Framework [Електронний ресурс] – Режим доступу до ресурсу:  
[https://www.tutorialspoint.com/nodejs/nodejs\\_express\\_framework.html](https://www.tutorialspoint.com/nodejs/nodejs_express_framework.html)
2. METANIT.COM – Сайт о программировании [Електронний ресурс] – Режим доступу до ресурсу: <https://metanit.com/web/nodejs/8.6.php>
3. Створюємо наш перший API за допомогою Node.js та Express [Електронний ресурс] – Режим доступу до ресурсу:  
<https://code.tutsplus.com/uk/tutorials/code-your-first-api-with-nodejs-and-express-connect-a-database--cms-31699>
4. Creating a Single-Page App in React using React Router [Електронний ресурс] – Режим доступу до ресурсу:  
[https://www.kirupa.com/react/creating\\_single\\_page\\_app\\_react\\_using\\_react\\_router.htm](https://www.kirupa.com/react/creating_single_page_app_react_using_react_router.htm)
5. Introduction to JSON Web Tokens [Електронний ресурс] – Режим доступу до ресурсу: <https://jwt.io/introduction/>
6. Использование JWT в приложении React + Redux для авторизации [Електронний ресурс] – Режим доступу до ресурсу:  
<https://medium.com/freecodecamp-russia-%D1%80%D1%83%D1%81%D1%81%D0%BA%D0%BE%D1%8F%D0%B7%D1%8B%D1%87%D0%BD%D1%8B%D0%B9/%D0%B8%D1%81%D0%BF%D0%BE%D0%BB%D1%8C%D0%B7%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5-jwt-%D0%B2-%D0%BF%D1%80%D0%B8%D0%BB%D0%BE%D0%B6%D0%B5%D0%BD%D0%B8%D0%B8-react-redux-%D0%B4%D0%BB%D1%8F-%D0%B0%D0%B2%D1%82%D0%BE%D1%80%D0%B8%D0%B7%D0%B0%D1%86%D0%B8%D0%B8-585bfe1399b0>
7. Создание и настройка базового проекта ReactJS в связке с Redux с нуля с использованием Webpack [Електронний ресурс] – Режим доступу до ресурсу:



<https://medium.com/@sergey.bakaev/%D1%81%D0%BE%D0%B7%D0%B4%D0%B0%D0%BD%D0%B8%D0%B5-%D0%B8-%D0%BD%D0%B0%D1%81%D1%82%D1%80%D0%BE%D0%B9%D0%BA%D0%B0-%D0%B1%D0%B0%D0%B7%D0%BE%D0%B2%D0%BE%D0%B3%D0%BE-%D0%BF%D1%80%D0%BE%D0%B5%D0%BA%D1%82%D0%B0-reactjs-%D0%B2-%D1%81%D0%B2%D1%8F%D0%B7%D0%BA%D0%B5-%D1%81-redux-%D1%81-%D0%BD%D1%83%D0%BB%D1%8F-%D1%81-%D0%B8%D1%81%D0%BF%D0%BE%D0%BB%D1%8C%D0%B7%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5%D0%BC-webpack-67a8cd3ec7bb>

8. Setting Up Redux DevTools – A Simple Guide [Электронный ресурс] – Режим доступа до ресурсу: <https://medium.com/@samueldinesh/setting-up-redux-devtools-a-simple-guide-3b386a6254fa>
9. A cartoon intro to redux [Электронный ресурс] – Режим доступа до ресурсу: <https://medium.com/russian/a-cartoon-intro-to-redux-e2108896f7e6>
10. Youtube music [Электронный ресурс] – Режим доступа до ресурсу: <https://music.youtube.com/>
11. Google play music [Электронный ресурс] – Режим доступа до ресурсу: <https://play.google.com/>
12. Spotify [Электронный ресурс] – Режим доступа до ресурсу: <https://www.spotify.com/>

## ДОДАТОК А. ЛІСТИНГ НАЛАШТУВАННЯ ЗАПИТІВ ДО БАЗИ ДАНИХ

```
var express = require('express');
var app = express();
const port = process.env.PORT || 3210;
const cors = require('cors');

const bodyParser = require("body-parser");
const urlencodedParser = bodyParser.urlencoded({extended: false});

const mysql = require('mysql2');
const pool = mysql.createPool({
  host: "zanner.org.ua",
  port: "33321",
  user: "ka7608",
  password: "123123",
  database: "music"
});

app.use(cors());
app.use(bodyParser.urlencoded({ extended: false }));
app.use(bodyParser.json());

app.get("/quotes", function(req, res){
  pool.query("SELECT * FROM quotes", function(err, data) {
    if(err) {
      return console.log(err);
    }
    res.json(data);
    //console.log(data);
  });
});

app.get("/checkUser", function(req, res){
  const user = {
    login: req.query.user_login,
    password: req.query.user_password
  }

  pool.query(
    `SELECT *
    FROM users
    WHERE (users.user_login = '${user.login}' AND
    users.user_password = '${user.password}')`, user,
    function(err, data) {
      if(err) {
        return console.log('it is err', err);
      }
      res.json(data);
    });
});

app.get("/playlists", function(req, res){
  const user_id = req.query.user_id;
  pool.query(
    `SELECT DISTINCT playlist_id, playlist_name
```

```

        FROM playlists
        WHERE user_id = ${user_id}`, user_id,
        function(err, data) {
            if(err) {
                return console.log(err);
            }
            res.json(data);
        });
    });
    app.get("/playlists/data", function(req, res){
        const user_id = req.query.user_id;
        pool.query(
            `SELECT
            u.user_id, u.user_login,
            p.playlist_id, p.playlist_name,
            t.track_id,
            a.album_name,
            s.singer_name,
            g.genre_name
            FROM users u JOIN playlists p USING(user_id)
            JOIN tracks t USING(track_id)
            JOIN albums a USING(album_id)
            JOIN singers s USING(singer_id)
            JOIN genres g USING(genre_id)
            WHERE u.user_id = '${user_id}',
            user_id,
            function(err, data) {
                if(err) {
                    return console.log(err);
                }
                res.json(data);
            });
    });

    app.get("/quote", function(req, res){
        pool.query(
            `SELECT quote_text FROM quotes ORDER BY RAND() LIMIT 1`,
            function(err, data) {
                if(err) {
                    return console.log(err);
                }
                res.json(data);
            });
    });

    app.get("/pickedPlaylist", function(req, res){
        const playlist_id = req.query.playlist_id;

        pool.query(
            `SELECT
            p.playlist_id,
            t.track_id, t.track_name, t.track_link, t.track_time,
            a.album_name, a.album_img, a.album_year,
            s.singer_name,
            g.genre_name
            FROM playlists p
            JOIN playlists_main pm USING(playlist_id)

```

```

        JOIN tracks t USING(track_id)
        JOIN albums a USING(album_id)
        JOIN singers s USING(singer_id)
        JOIN genres g USING(genre_id)
        WHERE playlist_id = ${playlist_id}`,
        playlist_id,
        function(err, data) {
            if(err) {
                return console.log(err);
            }
            res.json(data);
        });
    });

app.post("/addUser", function(req, res){
    const user = {
        login: req.body.user_login,
        password: req.body.user_password
    }
    pool.query(
        `INSERT INTO users (user_login, user_password)
        VALUES ('${user.login}', '${user.password}')`,
        user,
        function(err, data) {
            if(err) {
                return console.log(err);
            }
        });
    });

app.post("/addPlaylist", function(req, res){
    const playlist = {
        playlist_name: req.body.playlist_name,
        user_id: req.body.user_id,
    }

    pool.query(`INSERT INTO playlists SET playlist_name =
    "${playlist.playlist_name}" AND user_id = ${playlist.user_id}`, playlist,
    function(err, data) {
        if(err) {
            return console.log(err);
        }
    });
    });

app.put('/changePlaylistName', function(req, res){
    const playlist = {
        playlist_id: req.body.playlist_id,
        playlist_name: req.body.playlist_name
    }
    pool.query(
        `UPDATE playlists
        SET playlist_name = '${playlist.playlist_name}'
        WHERE playlist_id = ${playlist.playlist_id}`,
        playlist,
        function(err, data) {
            if(err) {

```

```

        return console.log(err);
    }
});
});

app.get('/fetchPlaylistName', function(req, res){
    playlist_id = req.query.playlist_id;

    pool.query(
        `SELECT * FROM playlists
        WHERE playlist_id = ${playlist_id}`,
        playlist_id,
        function(err, data) {
            if(err) {
                return console.log(err);
            }
            res.json(data);
        });
});

app.post('/addTrack', function(req, res){
    info = {
        p_id: req.body.p_id,
        t_id: req.body.t_id,
    }

    pool.query(
        `INSERT INTO
        playlists_main (playlist_id, track_id)
        VALUES ('${info.p_id}', '${info.t_id}')`,
        info,
        function(err, data) {
            if(err) {
                return console.log(err);
            }
            res.json(data);
        });
});

app.listen(port);

```

## ДОДАТОК Б. DDL (DATA DEFINITION LANGUAGE) КОМАНДИ

```
SET SQL_MODE = "NO_AUTO_VALUE_ON_ZERO";
SET time_zone = "+00:00";

/*!40101 SET @OLD_CHARACTER_SET_CLIENT=@@CHARACTER_SET_CLIENT */;
/*!40101 SET @OLD_CHARACTER_SET_RESULTS=@@CHARACTER_SET_RESULTS */;
/*!40101 SET @OLD_COLLATION_CONNECTION=@@COLLATION_CONNECTION */;
/*!40101 SET NAMES utf8mb4 */;

--
-- База данных: `music`
--

DELIMITER $$
--
-- Процедуры
--

CREATE DEFINER=`ka7608`@`%` PROCEDURE `ADD_TRACK` (IN `VALUE1`
VARCHAR(80), IN `VALUE2` SMALLINT(4) UNSIGNED, IN `VALUE3` INT(11)
UNSIGNED) BEGIN
    INSERT INTO `Track` (`track_name`, `track_time`) VALUES (VALUE1,
VALUE2);
END$$

CREATE DEFINER=`ka7608`@`%` PROCEDURE `CHANGE_TRACK` (IN `VALUE1`
INT(11), IN `VALUE2` INT(11), IN `VALUE3` INT(11), IN `VALUE4` INT(11), IN
`VALUE5` VARCHAR(80), IN `VALUE6` SMALLINT(4), IN `VALUE7` INT(11)) BEGIN
    UPDATE `Track` SET `singer_id` = VALUE2, `album_id` = VALUE3, `genre
id` = VALUE4, `track_name` = VALUE5, `track_year` = VALUE6, `track_time`
= VALUE7
    WHERE `track_id` = VALUE1;
END$$

CREATE DEFINER=`ka7608`@`%` PROCEDURE `DELETE_TRACK` (IN `VALUE`
INT(11)) BEGIN
    DELETE FROM `Track` WHERE `track_id` = VALUE;
END$$

CREATE DEFINER=`ka7608`@`%` PROCEDURE `ToValue` () BEGIN
SELECT COUNT(playlist_id)+1 FROM playlists;
END$$

DELIMITER ;

-- -----
--
-- Структура таблицы `albums`
--

CREATE TABLE `albums` (
  `album_id` int(11) NOT NULL,
  `singer_id` int(11) NOT NULL,
  `album_name` varchar(30) COLLATE utf8_unicode_ci NOT NULL,
```

```

    `album_year` smallint(4) NOT NULL,
    `album_img` varchar(255) COLLATE utf8_unicode_ci NOT NULL
) ENGINE=InnoDB DEFAULT CHARSET=utf8 COLLATE=utf8_unicode_ci;

-- -----

--
-- Структура таблицы `genres`
--

CREATE TABLE `genres` (
  `genre_id` int(11) NOT NULL,
  `genre_name` varchar(20) COLLATE utf8_unicode_ci NOT NULL
) ENGINE=InnoDB DEFAULT CHARSET=utf8 COLLATE=utf8_unicode_ci;

-- -----

--
-- Структура таблицы `playlists`
--

CREATE TABLE `playlists` (
  `playlist_id` int(11) NOT NULL,
  `playlist_name` varchar(50) COLLATE utf8_unicode_ci NOT NULL
) ENGINE=InnoDB DEFAULT CHARSET=utf8 COLLATE=utf8_unicode_ci;

-- -----

--
-- Структура таблицы `playlists_main`
--

CREATE TABLE `playlists_main` (
  `playlist_id` int(11) NOT NULL,
  `user_id` int(11) NOT NULL,
  `track_id` int(11) DEFAULT NULL
) ENGINE=InnoDB DEFAULT CHARSET=utf8 COLLATE=utf8_unicode_ci;

-- -----

--
-- Структура таблицы `quotes`
--

CREATE TABLE `quotes` (
  `quote_text` varchar(1000) COLLATE utf8_unicode_ci NOT NULL
) ENGINE=InnoDB DEFAULT CHARSET=utf8 COLLATE=utf8_unicode_ci;

-- -----

--
-- Структура таблицы `singers`
--

CREATE TABLE `singers` (
  `singer_id` int(11) NOT NULL,
  `singer_name` varchar(50) COLLATE utf8_unicode_ci NOT NULL

```

```

) ENGINE=InnoDB DEFAULT CHARSET=utf8 COLLATE=utf8_unicode_ci;
-- -----

--
-- Структура таблицы `tracks`
--

CREATE TABLE `tracks` (
  `track_id` int(11) NOT NULL,
  `track_link` varchar(255) COLLATE utf8_unicode_ci NOT NULL,
  `genre_id` int(11) NOT NULL,
  `album_id` int(11) NOT NULL,
  `track_name` varchar(79) COLLATE utf8_unicode_ci NOT NULL,
  `track_time` smallint(4) NOT NULL
) ENGINE=InnoDB DEFAULT CHARSET=utf8 COLLATE=utf8_unicode_ci;
--
-- Триггеры `tracks`
--
DELIMITER $$
CREATE TRIGGER `Track_AFTER_INSERT` BEFORE INSERT ON `tracks` FOR EACH
ROW BEGIN
  IF NEW.track_time < 0 AND NEW.track_time > 600 THEN
    SET NEW.track_time = NULL;
  END IF;
END
$$
DELIMITER ;

-- -----

--
-- Структура таблицы `users`
--

CREATE TABLE `users` (
  `user_id` int(11) NOT NULL,
  `user_password` varchar(16) COLLATE utf8_unicode_ci NOT NULL,
  `user_login` varchar(16) COLLATE utf8_unicode_ci NOT NULL
) ENGINE=InnoDB DEFAULT CHARSET=utf8 COLLATE=utf8_unicode_ci;
--
-- Индексы сохранённых таблиц
--

--
-- Индексы таблицы `albums`
--
ALTER TABLE `albums`
  ADD PRIMARY KEY (`album_id`);

--
-- Индексы таблицы `genres`
--
ALTER TABLE `genres`
  ADD PRIMARY KEY (`genre_id`);

--
-- Индексы таблицы `playlists`

```



```

--
ALTER TABLE `playlists`
  ADD PRIMARY KEY (`playlist_id`);

--
-- Индексы таблицы `singers`
--
ALTER TABLE `singers`
  ADD PRIMARY KEY (`singer_id`);

--
-- Индексы таблицы `tracks`
--
ALTER TABLE `tracks`
  ADD PRIMARY KEY (`track_id`);

--
-- Индексы таблицы `users`
--
ALTER TABLE `users`
  ADD PRIMARY KEY (`user_id`),
  ADD UNIQUE KEY `user_login` (`user_login`);

--
-- AUTO_INCREMENT для сохранённых таблиц
--

--
-- AUTO_INCREMENT для таблицы `albums`
--
ALTER TABLE `albums`
  MODIFY `album_id` int(11) NOT NULL AUTO_INCREMENT, AUTO_INCREMENT=9;
--
-- AUTO_INCREMENT для таблицы `genres`
--
ALTER TABLE `genres`
  MODIFY `genre_id` int(11) NOT NULL AUTO_INCREMENT, AUTO_INCREMENT=4;
--
-- AUTO_INCREMENT для таблицы `playlists`
--
ALTER TABLE `playlists`
  MODIFY `playlist_id` int(11) NOT NULL AUTO_INCREMENT,
AUTO_INCREMENT=10;
--
-- AUTO_INCREMENT для таблицы `singers`
--
ALTER TABLE `singers`
  MODIFY `singer_id` int(11) NOT NULL AUTO_INCREMENT,
AUTO_INCREMENT=6;
--
-- AUTO_INCREMENT для таблицы `tracks`
--
ALTER TABLE `tracks`
  MODIFY `track_id` int(11) NOT NULL AUTO_INCREMENT,
AUTO_INCREMENT=10;
--
-- AUTO_INCREMENT для таблицы `users`

```

```
--  
ALTER TABLE `users`  
  MODIFY `user_id` int(11) NOT NULL AUTO_INCREMENT, AUTO_INCREMENT=15;  
/*!40101 SET CHARACTER_SET_CLIENT=@OLD_CHARACTER_SET_CLIENT */;  
/*!40101 SET CHARACTER_SET_RESULTS=@OLD_CHARACTER_SET_RESULTS */;  
/*!40101 SET COLLATION_CONNECTION=@OLD_COLLATION_CONNECTION */;
```