IT 309 SOFTWARE ENGINEERING

PROJECT DOCUMENTATION

SoundStream

Prepared by:
Adnan Kuljančić
Tarik Suljić

Proposed to:
Nermina Durmić, Assist. Prof. Dr.
Aldin Kovačević, Teaching Assistant

21/06/2023

TABLE OF CONTENTS

1. Introduction

1.1. About the Project

The project/application we have been working on is called SoundStream. It is a web application designed to function similarly to Spotify, offering users an online music streaming platform. SoundStream allows users to access a vast library of songs and playlists from various genres and artists. Users can enjoy their favorite music and discover new tracks within the application.

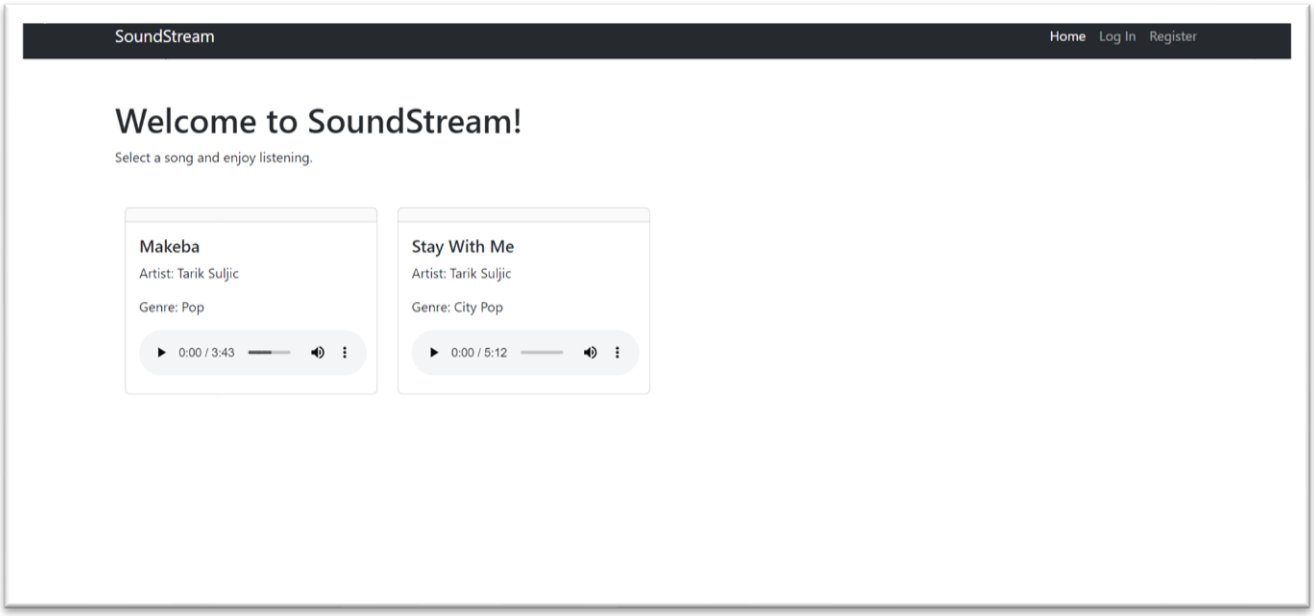
1.2. Project Functionalities and Screenshots

SoundStream incorporates the following main features:

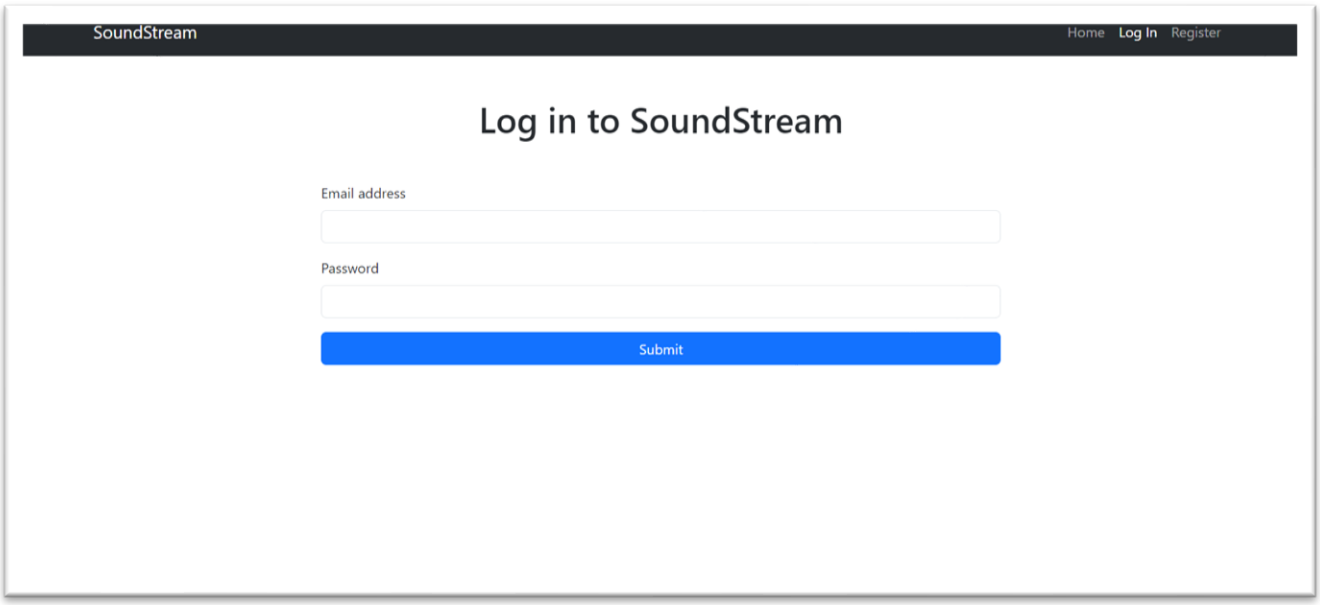
1. Home Page: The application includes a home page where users can browse and explore a collection of songs. This page provides an overview of the available music library.
2. Login and Register: SoundStream offers a login and register page, allowing users to create accounts or log in to existing ones. This functionality ensures personalized experiences for each user.
3. MySongs Page: Once logged in, users gain access to the MySongs page. Here, they can manage their own songs, including the ability to upload new songs or remove existing ones. This feature enables users to maintain their personalized music collection.

Screenshots:

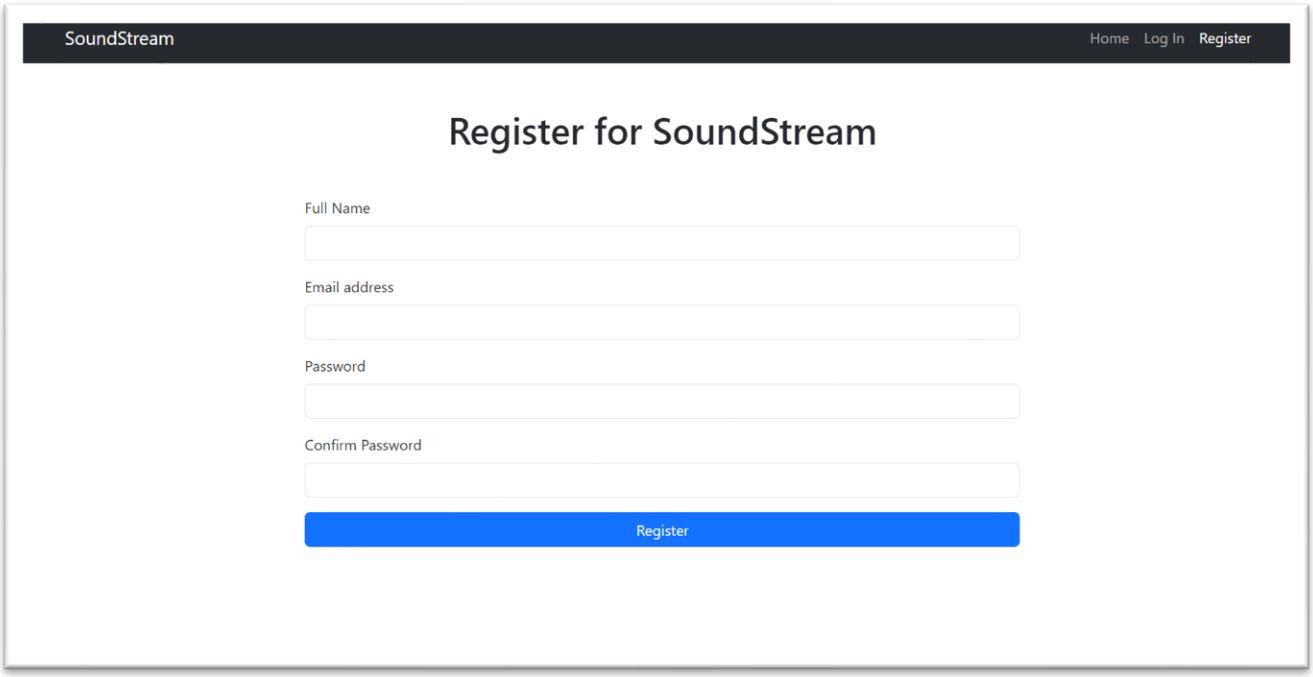
- Homepage



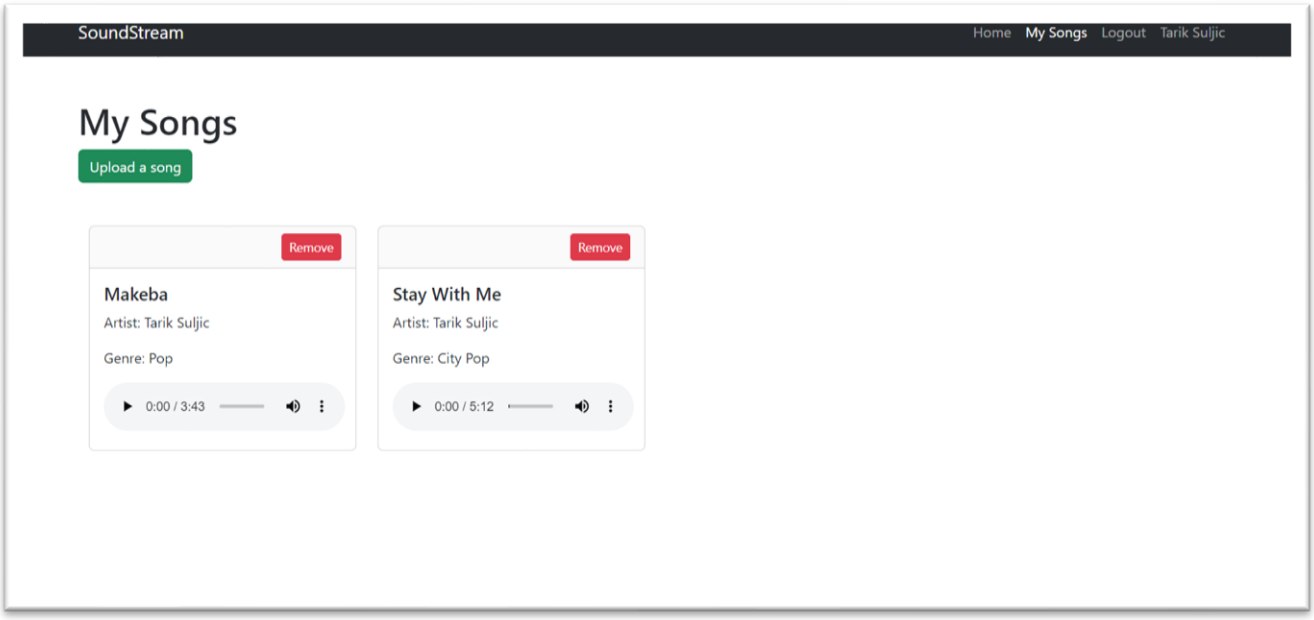
- Log in page:



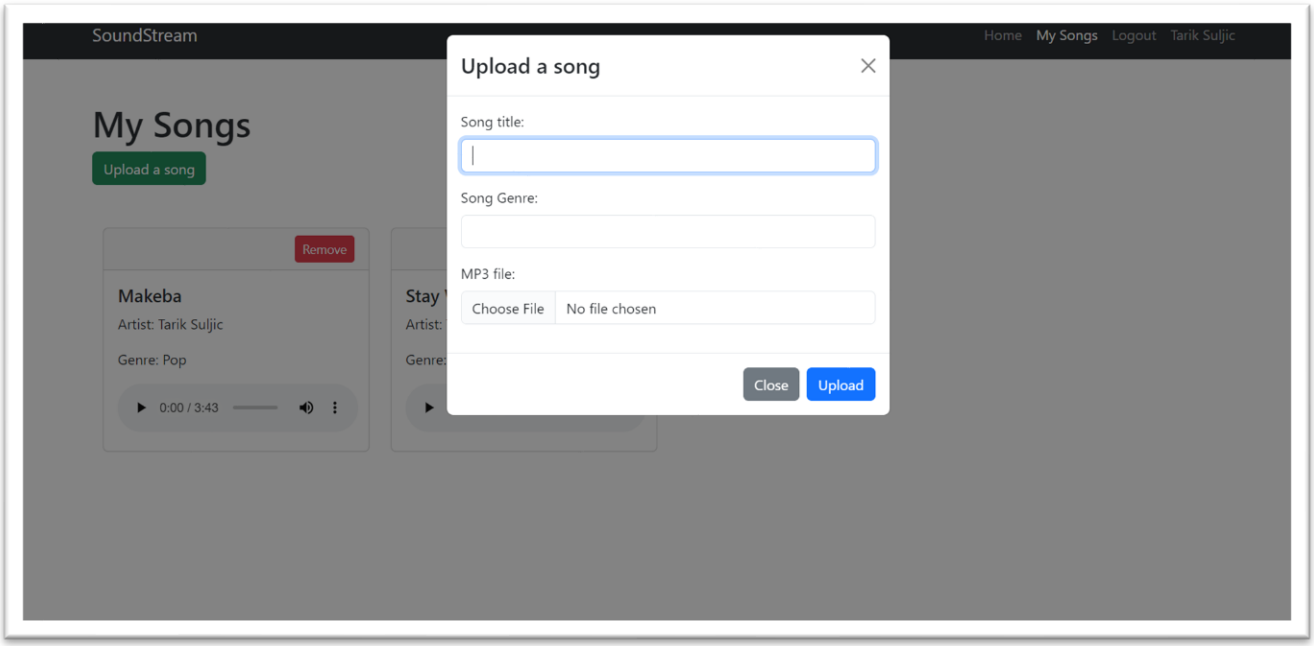
- Register page:



- “My Songs” page



- Upload a song modal:



2. Project Structure

2.1. Technologies

For our project, we utilized the MERN technology stack, which stands for:

- MongoDB: A NoSQL database used for storing application data.
- Express.js: A web application framework used for building the backend server.
- React: A JavaScript library used for building the frontend user interface.
- Node.js: A runtime environment used for running JavaScript on the server-side.

Additionally, we incorporated the following technologies and third-party tools:

- Firebase Storage: We leveraged Firebase Storage, a cloud storage service provided by Google, to securely store the MP3 files for our application.
- Postman: Used for testing and documenting API endpoints

Here are the coding standards used in the project:

- Backend (Node.js): we have followed the Node.js Style Guide or the Airbnb JavaScript Style Guide with specific considerations for Node.js development. This coding standard ensures consistent and maintainable code by providing guidelines for naming conventions, code organization, error handling, asynchronous programming, and other best practices. It is applied to the server-side code, including routes, controllers, services, and database interactions.
- Frontend (React.js): we have followed the Airbnb JavaScript Style Guide with additional guidelines for React-specific components and patterns. This coding standard focuses on best practices for writing clean and readable code in React.js. It covers guidelines for component naming, JSX syntax, props and state usage, event handling, component lifecycle methods, hooks, styling, and component documentation. It is applied to the client-side code, including React components, hooks, and UI-related logic.

2.2. Database Entities

The tables or entities in our database are:

- Users: This table stores user information, including their full name, email, password hash, creation timestamp, and update timestamp.
- Songs: This table stores song details, such as the title, artist, genre, audio URL, and a reference to the user who uploaded the song.

2.3. Design Patterns

The design patterns used in the project:

- **Singleton pattern**: used in the backend, in the file backend/index.js. The app object is created using the Express.js framework and is exported using module.exports. This indicates that it follows the Singleton pattern, as there is only one instance of the app object throughout the application.
- **Factory pattern**: used in the backend, in the file backend/middleware/upload.js. The upload.js module acts as a factory by encapsulating the creation and configuration of a multer instance for file uploads. It sets up the desired storage mechanism and other options, abstracting the complexity of multer configuration. The exported upload object allows other parts of the application to easily handle file uploads without directly dealing with multer implementation details.

Some other design patterns used:

1. Context Pattern: The UserContext is used to manage and share user-related data across components, demonstrating the Context pattern.
2. Observer Pattern: The useEffect hook is utilized for subscribing to changes in the userData state, representing the Observer pattern, where the component reacts to changes in the observed state
3. State Pattern: React's state management system, specifically the useState hook, is used to manage and update component state in a predictable manner.
4. Conditional Rendering Pattern: Conditional rendering is employed throughout the code, displaying different UI components based on conditions such as the user's authentication status or the presence of uploaded songs.
5. Component Pattern: React components, such as Modal, Form, and Button from the react-bootstrap library, follow the Component pattern, allowing for reusable and modular UI elements.

2.4. Tests

Although tests are not yet implemented in the project, there are plans to include both unit tests and Selenium tests to ensure the application's quality and stability.

These tests will likely be located in separate test files within the project, following a naming convention like ComponentName.test.js or ModuleName.test.js.

On the other hand, Selenium tests will be used to perform end-to-end testing by simulating user interactions and verifying the application's behavior across different browsers.

These tests will likely be located in a dedicated test directory or folder, organized based on different test scenarios or functionalities.

3. Conclusion

We are proud of our achievement in developing a functional web application using the MERN stack. As students, we see this project as a valuable learning experience that has provided us with insights into project organization, code structure, and the implementation of design patterns.

While we are satisfied with the overall implementation, we acknowledge that there is room for improvement. Our future plans involve enhancing the application by addressing any existing issues or bugs, implementing unit tests and Selenium tests to ensure its quality, and further refining the user experience.

In the future, we want to make the user experience even better by adding loaders and toaster messages (since the project covers mostly the “happy path”). These will let users know if there are any issues with their login or registration credentials, or if something went wrong. We believe these improvements will make the application more user-friendly and help us grow as developers.

This project has been an excellent opportunity for us to apply our knowledge and gain practical skills, and we are excited about the potential for growth and improvement.