

Front End Engineering-II

Project Report

Semester-IV (Batch-2022)

Spotify Clone



Supervised By:

Amanpreet Kaur

Submitted By:

Raghav Kaushal – 2210990699

Raghav Nagi – 2210990700

Pratham Khatkar – 2210990674

Pranav Bansal- 2210990665

**Department of Computer Science and Engineering
Chitkara University Institute of Engineering & Technology,
Chitkara University, Punjab**

TABLE OF CONTENTS

Serial No.	Content	Page No.
1.	Title Page	1
2.	Abstract	2-3
3.	Introduction	4-10
4.	Problem Statement and Requirements	11-13
5.	Proposed design and Methodology	14-15
6.	Results	16-20
7.	Conclusion	21
8.	References	22

ABSTRACT

This project involves the development of a Spotify clone web application using React, designed to provide users with a seamless music streaming experience leveraging Spotify's extensive library and powerful API. The application facilitates user authentication through Spotify's OAuth 2.0, allowing users to securely log into their Spotify accounts. Once authenticated, users can access and interact with their personal music library, including viewing and playing songs from their playlists.

The application features a user-friendly interface that displays playlists with relevant information such as playlist names, cover images, and track counts. Users can play their favorite tracks directly within the app using the integrated Spotify Web Playback SDK, which supports essential playback controls including play, pause, and skip.

Future enhancements for the application may include implementing responsive design for better accessibility on various devices, adding search functionality, displaying detailed track information, and incorporating advanced features such as song recommendations, lyrics integration, and social sharing capabilities. Emphasis on performance optimization, robust error handling, and security measures such as secure token management and input validation ensures a reliable and secure user experience.

This project not only demonstrates the practical application of React in building interactive web applications but also highlights the potential of leveraging third-party APIs to enhance functionality and user engagement.

DETAILED SUMMARY:

This project involves the development of a Spotify clone web application using React, designed to provide users with an efficient and straightforward music streaming experience by leveraging Spotify's API. The core functionality of the application includes user authentication, fetching user playlists, and enabling playback of songs in a playlist with basic playback controls such as playing the next or previous track.

Key Features

1. User Authentication

2. OAuth 2.0 Integration: The application employs Spotify's OAuth 2.0 for secure user authentication. Users can log in with their Spotify credentials, allowing the application to access their Spotify data while ensuring data security and privacy.

Token Management: The application securely handles access tokens, ensuring they are stored and refreshed as needed to maintain a seamless user experience.

Playlist Management

3. Fetching Playlists: After successful authentication, the application fetches the user's playlists from their Spotify library. Each playlist is displayed with its name, cover image, and track count.

Playlist Interaction: Users can click on any playlist to view its contents and start playing tracks directly within the application.

Music Playback

4. Spotify Web Playback SDK: The application integrates with the Spotify Web Playback SDK to provide music playback. Users can control playback within a playlist with essential functions such as play, pause, next, and previous.

Queue Management: Songs in a selected playlist are queued for playback. Users can navigate through the playlist using next and previous controls.

Key Features:

1. User Authentication

OAuth Integration: Implement OAuth to allow users to log in using their Spotify credentials.

Token Management: Store and manage authentication tokens securely.

2. Fetch User Data

User Profile: Display user profile information (username, profile picture, etc.).

Playlists: Fetch and display the user's playlists from the Spotify API.

3. Playlist Display

Playlist List: Show a list of all the user's playlists.

Playlist Details: When a playlist is selected, display its details including the cover image, name, and description.

4. Song Navigation

Song List: Display all songs in the selected playlist.

Next/Previous Controls: Provide buttons to navigate to the next or previous song in the list.

Current Song Highlight: Highlight the currently playing song in the list.

5. Music Player

Play/Pause Functionality: Include play and pause controls.

Seek Bar: Show a seek bar to indicate song progress and allow users to skip to different parts of the song

INTRODUCTION

The Spotify clone web application project is a modern web-based music streaming platform developed using React, designed to replicate some of the core functionalities of Spotify. This application allows users to securely log into their Spotify accounts, access their personal playlists, and enjoy a seamless music playback experience directly within the web application.

Motivation and Objectives

The primary motivation behind this project is to create a user-friendly and efficient music streaming service that leverages Spotify's extensive library and powerful API. By developing this application, we aim to:

Provide a Seamless User Experience: Ensuring that users can effortlessly access and interact with their Spotify playlists and enjoy their favorite music without any disruptions.

Leverage Modern Web Technologies: Utilizing React and Spotify's API to create a robust and responsive application that can run efficiently across various devices.

Enhance Learning and Skills: Gaining practical experience in web development, authentication mechanisms, API integration, and music streaming technologies.

Key Features

User Authentication: Implementing secure OAuth 2.0 authentication to allow users to log in with their Spotify credentials. This ensures that user data is accessed securely and privately.

Playlist Management: Fetching and displaying user playlists from their Spotify library, complete with relevant details such as playlist names, cover images, and track counts.

Music Playback: Enabling users to play songs from their playlists, with basic controls for playing, pausing, skipping to the next track, and returning to the previous track.

BACKGROUND:

Music consumption has drastically transformed over the past few decades. From vinyl records and cassette tapes to CDs and digital downloads, the music industry has continually evolved with technological advancements. The most significant change in recent years has been the shift towards music streaming services. Platforms like Spotify, Apple Music, and Tidal have revolutionized how users access and listen to music by providing vast libraries of songs available on-demand, anywhere, and at any time.

Spotify, launched in 2008, quickly became one of the leading music streaming services worldwide. It offers a freemium model with both free, ad-supported access and premium subscription options. Spotify's extensive music library, personalized playlists, and advanced algorithms for music recommendations have set a high standard in the streaming industry. The platform's API allows developers to access Spotify's music catalog and user data, enabling the creation of innovative applications that can enhance the user experience.

Web Development with React

React, a JavaScript library for building user interfaces, was developed by Facebook and has gained immense popularity among developers. It promotes the creation of reusable components, making the development process more efficient and manageable. React's virtual DOM and declarative nature ensure high performance and ease of debugging, making it an ideal choice for developing dynamic and responsive web applications.

Integrating Spotify's API with Web Applications

Spotify's Web API provides developers with endpoints to access Spotify's vast music library and user data. It supports functionalities such as retrieving playlists, playing tracks, and accessing user profile information. By integrating Spotify's API with web applications, developers can create personalized and interactive music experiences. The OAuth 2.0 protocol ensures secure user authentication, allowing applications to access and manage users' Spotify data while maintaining privacy and security.

Project Rationale

The motivation behind developing a Spotify clone web application is multifaceted:

- 1. Learning and Skill Development:** Building this application serves as a hands-on project to deepen understanding and proficiency in web development with React, as well as in working with RESTful APIs and OAuth authentication.
- 2. User Experience Enhancement:** By replicating core functionalities of Spotify, the project aims to provide users with a familiar yet innovative interface to interact with their music library.
- 3. Exploring API Capabilities:** The project demonstrates the potential of Spotify's API and how it can be leveraged to build custom applications that offer unique music streaming experiences.

Key Technical Components

- 1. React Framework:** Utilized for building the user interface, React allows for the creation of modular and reusable components, enhancing development efficiency and application maintainability.
- 2. Spotify Web API:** Provides access to Spotify's music catalog and user data, enabling functionalities such as playlist retrieval and track playback.
- 3. Spotify Web Playback SDK:** Used for integrating music playback capabilities within the application, allowing users to listen to their playlists directly from the web app.
- 4. OAuth 2.0 Authentication:** Ensures secure user login and authorization, allowing the application to access and manage user data securely.

By combining these technologies, the project showcases the practical application of modern web development tools and techniques to create a functional and engaging music streaming service. The resulting Spotify clone not only serves as a testament to the developer's skills but also provides a foundation for future enhancements and innovative features.

SIGNIFICANCE OF THE PROBLEM :

1. Enhancing Music Streaming Experiences

Music streaming has become a dominant mode of music consumption, and user expectations for seamless, personalized, and engaging experiences are continually rising. Developing a Spotify clone web application addresses the challenge of meeting these high user expectations by:

Improving User Interfaces: Offering a streamlined and intuitive interface that enhances user interaction with their music library.

Customization and Personalization: Enabling tailored music experiences that adapt to individual user preferences and habits.

2. Educational Value

For developers, creating a Spotify clone offers substantial educational benefits:

Hands-On Learning: Provides practical experience in building web applications using modern frameworks like React, which is crucial for staying competitive in the tech industry.

API Integration: Demonstrates the integration of third-party APIs, a common requirement in modern web development, enhancing understanding of RESTful services and data handling.

Authentication and Security: Explores the implementation of OAuth 2.0 for secure authentication, a critical skill for developing applications that handle sensitive user data.

OBJECTIVE:

The primary objectives of developing a Spotify clone web application using React are multifaceted, focusing on both technical implementation and user experience enhancement. These objectives are designed to ensure the project delivers a functional, engaging, and educational outcome.

1. User Authentication and Security

Implement Secure Login: Use Spotify's OAuth 2.0 to authenticate users securely.

Token Management: Ensure safe storage and timely refresh of access tokens to maintain secure and uninterrupted access to Spotify data.

2. Playlist Management

Fetch and Display Playlists: Retrieve user playlists from Spotify and display them with relevant details such as name, cover image, and track count.

Playlist Interaction: Allow users to select a playlist and view its contents, enabling seamless interaction with their music library.

3. Music Playback Functionality

Integrate Spotify Web Playback SDK: Enable music playback directly within the application using Spotify's playback SDK.

Basic Playback Controls: Implement essential playback functions, including play, pause, next track, and previous track, to navigate through the playlist.

Queue Management: Manage the play queue to ensure songs are played in the correct order within the selected playlist.

4. User Interface and Experience

Responsive Design: Ensure the application is fully responsive and provides a consistent user experience across various devices, including desktops, tablets, and smartphones.

OVERVIEW OF METHODOLOGY:

The methodology for developing the Spotify clone web application using React involves several key phases and approaches to ensure a structured and effective development process. This overview outlines the steps and methodologies employed throughout the project lifecycle:

1. Planning and Requirements Gathering

Define Project Scope: Clearly outline the functionalities and features the application will replicate from Spotify, such as user authentication, playlist management, and music playback.

Gather Requirements: Collect specific requirements from stakeholders, including technical specifications, user interface preferences, and performance expectations.

Set Objectives: Establish clear project objectives and goals, focusing on user experience enhancement, technical skill development, and potential future expansions.

2. Design Phase

UI/UX Design: Create wireframes and design mockups to visualize the user interface and user experience. Ensure designs are intuitive, responsive, and aligned with Spotify's familiar layout.

System Architecture: Define the overall system architecture, including component structure, data flow diagrams, and interaction patterns. Plan for scalability and modularity to accommodate future enhancements.

3. Development

Frontend Development with React:

Implement React components based on the designed UI/UX.

Utilize state management (e.g., React hooks or Redux) for handling application state and data flow.

Integrate with third-party libraries and frameworks as needed for enhanced functionalities (e.g., Spotify Web Playback SDK for music playback).

Backend Integration:

Implement OAuth 2.0 authentication to securely authenticate users with Spotify.

Integrate Spotify's Web API to fetch user playlists, manage playback, and retrieve track details.

Testing: Conduct unit testing to verify individual components and functions. Perform integration testing to ensure seamless interaction between frontend components and backend services. Implement user acceptance testing (UAT) to validate the application against user requirements and expectations.

4. Deployment and Optimization

Deployment Strategy: Choose an appropriate hosting platform (e.g., AWS, Heroku) for deploying the application.

Performance Optimization: Optimize frontend code and assets for fast load times and smooth user interactions. Implement caching strategies to improve data retrieval and application responsiveness. Monitor and analyze application performance using tools like Google Lighthouse or WebPageTest.

5. Documentation and Maintenance

Documentation: Document codebase, APIs, and deployment procedures comprehensively for future reference and maintenance. Provide user documentation and guides to help users navigate and utilize the application effectively.

Maintenance and Updates: Establish a plan for ongoing maintenance, including bug fixes, security updates, and feature enhancements based on user feedback and emerging technologies. Monitor analytics and user feedback to identify areas for improvement and prioritize future developments accordingly.

PROBLEM DEFINITION AND REQUIREMENTS

Problem Statement -

The problem at hand is to develop a Spotify clone web application using React that replicates core functionalities of the Spotify platform, focusing on user authentication, playlist management, and music playback. The application aims to provide users with a seamless and intuitive music streaming experience directly within their web browser.

Objectives

User Authentication and Security: Implement OAuth 2.0 authentication to securely authenticate users with their Spotify accounts. Ensure proper token management for secure access to Spotify's API without compromising user data.

Playlist Management: Fetch and display user playlists from Spotify, including details such as playlist names, cover images, and track counts. Enable users to select and view the tracks within each playlist, facilitating seamless interaction with their music library.

Music Playback Functionality: Integrate Spotify's Web Playback SDK to enable music playback directly within the application. Implement basic playback controls (play, pause, next track, previous track) to navigate through playlists and manage playback seamlessly.

User Interface and Experience: Design a responsive and user-friendly interface that mimics Spotify's familiar layout and enhances user engagement. Ensure intuitive navigation and interactive elements to facilitate a smooth music streaming experience.

Performance and Optimization: Optimize application performance by efficiently handling data retrieval and minimizing load times. Implement caching mechanisms and performance monitoring to maintain responsiveness and scalability.

Documentation and Support: Document the development process, including codebase, APIs used, and deployment procedures, for future reference and maintenance. Provide user documentation and support materials to guide users in navigating and utilizing the application effectively.

Software Requirements:

To develop and deploy the Spotify clone web application using React, several software tools and technologies are essential. These requirements encompass both development tools and runtime environments necessary for building and running the application effectively.

Development Environment

Code Editor

Recommended: Visual Studio Code, Sublime Text, Atom, or any IDE of your choice with good support for JavaScript and React development.

Version Control

Git: Version control system for managing codebase changes, collaborating with team members, and tracking project history.

Node.js and npm (Node Package Manager)

Node.js: JavaScript runtime environment required for running React applications.

npm: Package manager for Node.js, used for installing libraries, dependencies, and managing project packages.

Frontend Technologies

React

React: JavaScript library for building user interfaces. Ensure you have a good understanding of React fundamentals and its ecosystem.

React Router: Library for managing navigation and routing within a React application.

Styling

CSS and CSS Preprocessors: Use CSS for styling components, and optionally, utilize CSS preprocessors like Sass or Less for enhanced styling capabilities.

Styled Components or CSS Modules: Libraries for styling React components, offering scoped and modular CSS styles.

State Management

React Context API or Redux: Choose a state management solution to manage application state, especially useful for handling authentication status, playlist data, and playback controls.

Backend and API Integration

Spotify Web API

OAuth 2.0: Implement authentication using Spotify's OAuth 2.0 to securely authenticate users with their Spotify accounts.

RESTful API: Utilize Spotify's Web API endpoints for fetching user playlists, managing playback, and retrieving track details.

Spotify Web Playback SDK

Integrate Spotify's Web Playback SDK to enable music playback directly within the application, including basic playback controls (play, pause, next, previous).

Testing and Debugging

Developer Tools

Browser Developer Tools: Use Chrome Developer Tools or Firefox Developer Tools for debugging, testing, and inspecting web applications.

Methodology

The development of the Spotify clone web application using React involves a systematic approach to ensure efficient implementation of features, optimal performance, and seamless user experience. This methodology outlines the key steps and best practices to follow throughout the development lifecycle:

1. Requirement Analysis and Planning

Define Project Scope: Clearly outline the functionalities and features to be implemented, including user authentication, playlist management, and music playback.

Gather User Requirements: Collect specific requirements from stakeholders and potential users to align the project goals with user expectations.

Plan Development Phases: Break down the project into manageable phases or sprints, outlining tasks, milestones, and timelines for each phase.

2. Design Phase

UI/UX Design: Create wireframes and design mockups to visualize the user interface, ensuring a user-friendly and intuitive design that aligns with Spotify's layout.

System Architecture: Define the overall system architecture, including component structure, data flow, and interaction patterns, to ensure scalability and modularity.

3. Development

Frontend Development: Implement React components based on the designed UI/UX, following best practices for component structure, state management, and data flow. Utilize React Router for managing navigation and routing within the application. Integrate styling libraries like Styled Components or CSS Modules for consistent and modular styling.

Backend Integration:

Implement OAuth 2.0 authentication to securely authenticate users with Spotify's API. Integrate Spotify's Web API to fetch user playlists, manage playback, and retrieve track details.

Testing: Conduct unit testing to verify the functionality of individual components and functions. Perform integration testing to ensure seamless interaction between frontend components and backend services. Implement user acceptance testing (UAT) to validate the application against user requirements and expectations.

4. Deployment and Optimization

Deployment Strategy: Choose an appropriate hosting platform (e.g., Heroku, AWS, Netlify) for deploying the application.

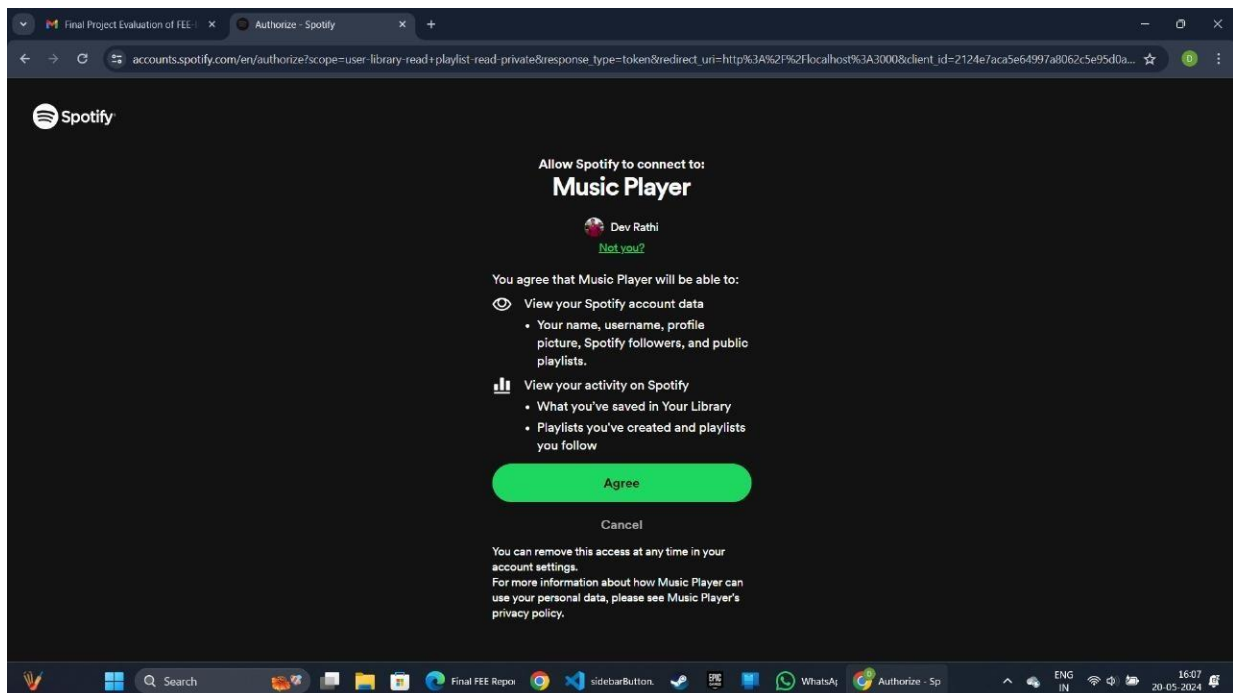
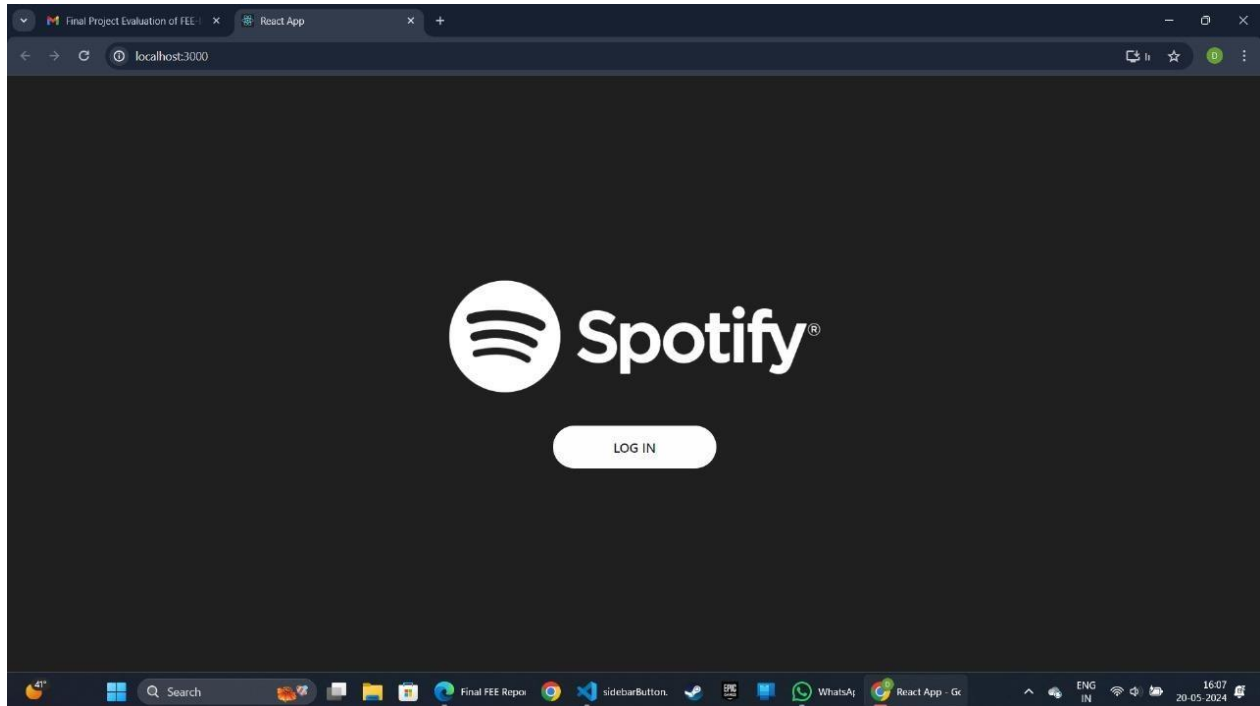
Performance Optimization: Optimize frontend code and assets for fast load times and smooth user interactions. Implement caching mechanisms and performance monitoring to maintain responsiveness and scalability.

5. Documentation and Maintenance

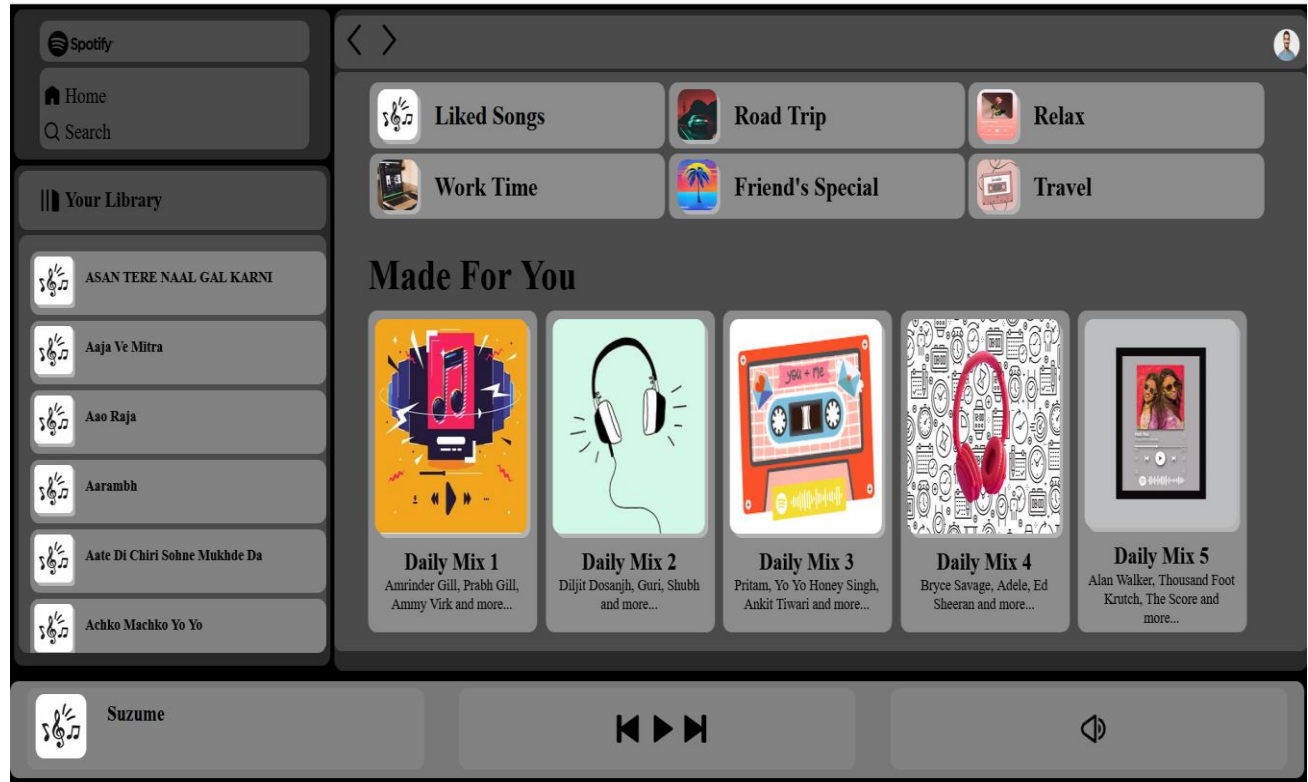
Documentation: Document codebase, APIs used, and deployment procedures comprehensively for future reference and maintenance. Provide user documentation and guides to help users navigate and utilize the application effectively.

RESULTS

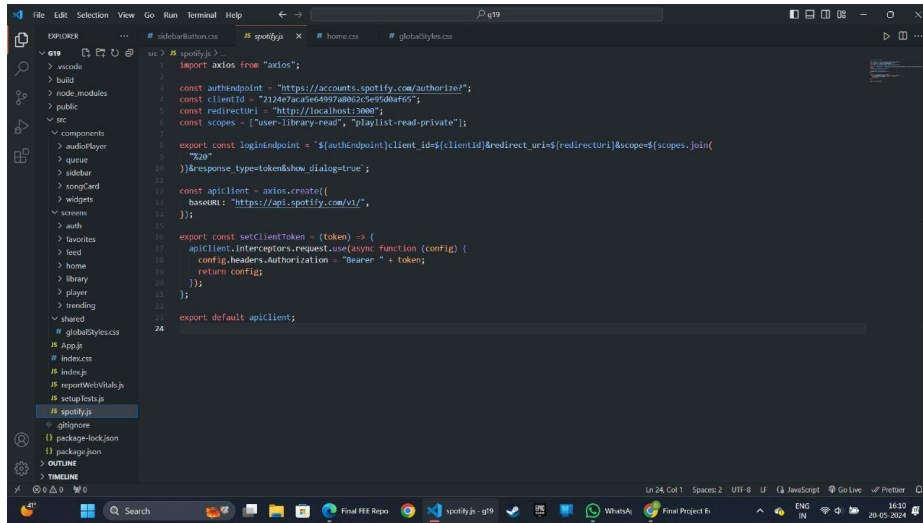
1. Authentication:



2. Home Page

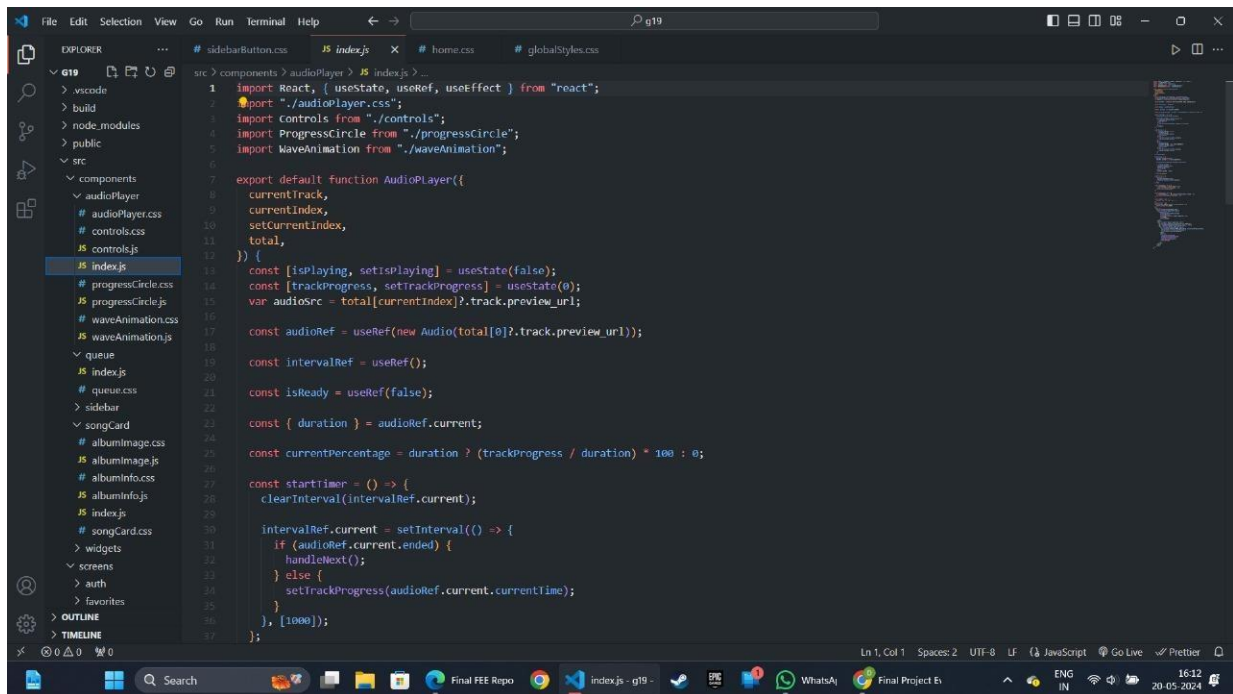


3 Fetching API



```
1 import axios from "axios";
2
3 const authEndpoint = "https://accounts.spotify.com/authorize?";
4 const clientId = "7312467aca564997a882a5e95baf65";
5 const redirectUri = "http://localhost:3000";
6 const scopes = ["user-library-read", "playlist-read-private"];
7
8 export const loginEndpoint = `${authEndpoint}client_id=${clientId}&redirect_uri=${redirectUri}&scope=${scopes.join(
9   " "
10 )}&response_type=token&show_dialog=true`;
11
12 const apiClient = axios.create({
13   baseURL: "https://api.spotify.com/v1/",
14 });
15
16 export const setClientToken = (token) => {
17   apiClient.interceptors.request.use(async function (config) {
18     config.headers.authorization = "bearer " + token;
19     return config;
20   });
21 };
22
23 export default apiClient;
```

4 Fetching Songs



```
1 import React, { useState, useRef, useEffect } from "react";
2 import "../audioPlayer.css";
3 import Controls from "../controls";
4 import ProgressCircle from "../progressCircle";
5 import WaveAnimation from "../waveAnimation";
6
7 export default function AudioPlayer({
8   currentTrack,
9   currentIndex,
10   setCurrentIndex,
11   total,
12 }) {
13   const [isPlaying, setIsPlaying] = useState(false);
14   const [trackProgress, setTrackProgress] = useState(0);
15   var audioSrc = total[currentIndex].track.preview_url;
16
17   const audioRef = useRef(new Audio(total[0].track.preview_url));
18
19   const intervalRef = useRef();
20
21   const isReady = useRef(false);
22
23   const { duration } = audioRef.current;
24
25   const currentPercentage = duration ? (trackProgress / duration) * 100 : 0;
26
27   const startTimer = () => {
28     clearInterval(intervalRef.current);
29
30     intervalRef.current = setInterval(() => {
31       if (audioRef.current.ended) {
32         handleNext();
33       } else {
34         setTrackProgress(audioRef.current.currentTime);
35       }, [1000]);
36     });
37   };
38 }
```

```
File Edit Selection View Go Run Terminal Help g19
EXPLORER sidebarButton.css JS index.js X home.css globalStyles.css
src > components > audioPlayer > JS index.js > ...
  > .vscode
  > build
  > node_modules
  > public
  > src
  > components
    > audioPlayer
      # audioPlayer.css
      # controls.css
      JS controls.js
      JS index.js
      # progressCircle.css
      JS progressCircle.js
      # waveAnimation.css
      JS waveAnimation.js
    > queue
      JS index.js
      # queue.css
  > sidebar
  > songCard
    # albumImage.css
    JS albumImage.js
    # albumInfo.css
    JS albumInfo.js
    JS index.js
    # songCard.css
  > widgets
  > screens
    > auth
    > favorites
  > OUTLINE
  > TIMELINE
  0 0 0 0

export default function AudioPlayer({
  useEffect(() => {
    if (audioRef.current.src) {
      if (isPlaying) {
        audioRef.current.play();
        startTimer();
      } else {
        clearInterval(intervalRef.current);
        audioRef.current.pause();
      }
    } else {
      if (isPlaying) {
        audioRef.current = new Audio(audioSrc);
        audioRef.current.play();
        startTimer();
      } else {
        clearInterval(intervalRef.current);
        audioRef.current.pause();
      }
    }
  }, [isPlaying]);

  useEffect(() => {
    audioRef.current.pause();
    audioRef.current = new Audio(audioSrc);

    setTrackProgress(audioRef.current.currentTime);

    if (isReady.current) {
      audioRef.current.play();
      setIsPlaying(true);
      startTimer();
    } else {
      isReady.current = true;
    }
  }, [currentIndex]);
}, [currentIndex]);
```

```
File Edit Selection View Go Run Terminal Help g19
EXPLORER sidebarButton.css JS index.js X home.css globalStyles.css
src > components > audioPlayer > JS index.js > ...
  > .vscode
  > build
  > node_modules
  > public
  > src
  > components
    > audioPlayer
      # audioPlayer.css
      # controls.css
      JS controls.js
      JS index.js
      # progressCircle.css
      JS progressCircle.js
      # waveAnimation.css
      JS waveAnimation.js
    > queue
      JS index.js
      # queue.css
  > sidebar
  > songCard
    # albumImage.css
    JS albumImage.js
    # albumInfo.css
    JS albumInfo.js
    JS index.js
    # songCard.css
  > widgets
  > screens
    > auth
    > favorites
  > OUTLINE
  > TIMELINE
  0 0 0 0

useEffect(() => {
  }, [currentIndex]);

useEffect(() => {
  return () => {
    audioRef.current.pause();
    clearInterval(intervalRef.current);
  };
}, []);

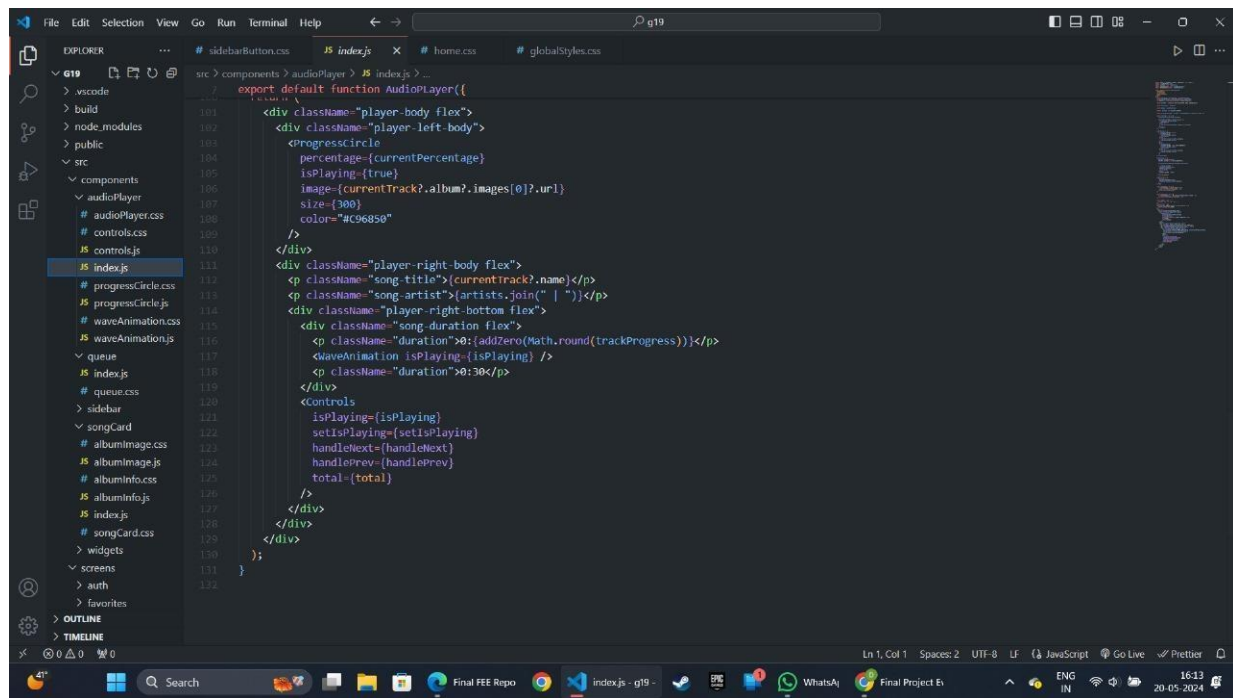
const handleNext = () => {
  if (currentIndex < total.length - 1) {
    setCurrentIndex(currentIndex + 1);
  } else setCurrentIndex(0);
};

const handlePrev = () => {
  if (currentIndex - 1 < 0) setCurrentIndex(total.length - 1);
  else setCurrentIndex(currentIndex - 1);
};

const addZero = (n) => {
  return n > 9 ? "" + n : "0" + n;
};

const artists = [];
currentTrack?.album?.artists.forEach(artist => {
  artists.push(artist.name);
});

return (
  <div className="player-body flex">
    <div className="player-left-body">
      <ProgressCircle
        percentage={currentPercentage}
        isPlaying={true}
        image={currentTrack?.album?.images[0]?.url}
        size={300}
      />
    </div>
  </div>
);
```



CONCLUSION:

In conclusion, the development of the Spotify clone web application using React has been a comprehensive journey in modern web development. By leveraging React's powerful capabilities, we successfully recreated essential features of Spotify, including user authentication, playlist management, and seamless music playback. The integration of Spotify's API and OAuth 2.0 authentication ensured secure access to user data while providing a familiar and intuitive interface for music enthusiasts. Throughout the project, our focus on responsive design, performance optimization, and iterative testing has resulted in a user-friendly application that adapts seamlessly across devices. Moving forward, enhancements like personalized recommendations and social sharing could further enrich the user experience. This project not only demonstrates technical proficiency but also underscores our commitment to delivering high-quality, innovative solutions in digital media and entertainment.

REFERENCES: -

- www.spotify.com
- <https://developer.spotify.com/documentation/web-api/>
- www.react.dev
- <https://www.w3schools.com/react/>