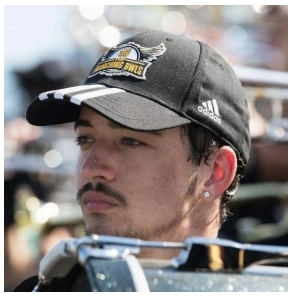
SP-27-Blue - Spotify App - Final Report

CS 4850 - Section 02 - Spring 2025 - 4/20/25 - Sharon Perry

Project Team

 Trey Morris	 Chris Brown	 David Clarke	 Ethan Sutton

Website: https://p-coder258.github.io/bluespotify.github.io/

Github: https://github.com/SP-27-BlueSpotify

STATS AND STATUS	
LOC	11,395
Components/Tools	6 - Github, Spotify, Spotify for developers, Discord, Expo, React Native
Hours Estimate	189
Hours Actual	156
Status	Project is 100% complete and working as designed

Table of Contents

1.0 Project Overview / Abstract

Spotify has been an influential app in the music industry in recent years, and has paved the way for modern music streaming services. Taking this inspiration, we want to be able to expand upon and create our own custom take on this popular streaming service. As always, and expected with Spotify, our application will be mobile friendly and cross platform on all devices such as Android, iOS, and Web.

2.0 Requirements

2.1 Overview

This defines the requirements for the Spotify-like app that is being developed within React Native while building upon the current Spotify API. The purpose of this overview is to represent how the underlying functionalities of the app work and understand the systems behind them.

2.2 Project Goals

The main goals of this app are:

1. Provide a unique experience built upon from the original, by providing different features or mechanics.
2. Provide high quality media systems with no issues.

2.3 Assumptions

It's assumed that users will have a general understanding of how to use mobile and web applications, while also understanding how digital music services function. It's also assumed that users will have access to Spotify in some way.

2.4 Functional Requirements

- Login
    - Login with Spotify Credentials
- User Authentication
    - Authenticate with Spotify using OAuth 2.0
- Display Home Page
    - Options from Home Page
    - Search
    - Playlists
    - Create Playlists
    - Add or remove songs from playlists
    - Now Playing

    - Music player
    - Display current song playing
    - User
    - Showcase user's information
- Search
    - Search for songs, artists, albums, and playlists
    - Display search results
    - Provide filters for search
- Playlists
    - Retrieve and display user playlists using Spotify API
- Music Playback Control
    - Play, pause, and skip songs using Spotify API
    - Adjust volume
    - Support queue management

3.0 Design Constraints

3.1 Environment

The Spotify app will be built upon within React Native. It will get calls from the RESTful API to connect systems and UI elements. RESTful API has built-in support where programs can retrieve and manage Spotify data.

3.2 User Characteristics

Users will be able to access their personal Spotify accounts and search music, create and manage playlist, and control music playback.

3.3 System

- Hardware Requirements
    - The application should be compatible with standard desktop and mobile devices.
    - A stable internet connection is required for real-time Spotify API interactions.
- Software Requirements
    - The system must be able to run on modern web browsers (Chrome, Firefox, Edge, Safari).
    - Support for iOS is required.
- Scalability and Performance
    - The system should handle multiple concurrent users without performance degradation.

- API requests should be optimized using caching mechanisms where applicable.

4.0 Design Considerations

4.1 Assumptions and Dependencies

The assumptions that we have identified for our project would include being an iOS or Android user, OR a web user. React Native is a strong cross platform framework that allows users to be on Web, iOS, or Android without having to write native code for each platform. One code base, one application, multiple platforms of usability. We also assume that the user has a general understanding of modern mobile applications, and how to use the platform that they are accessing our project on. Lastly, it is good to assume that our users have a general understanding of Spotify or digital music streaming. We assume that users will have a registered spotify account to access our project as well.

4.2 General Constraints

Users will be required to have a Spotify account to access our app and also have access to the internet. User authentication will be handled within Spotify by OAuth 2.0. The app will be built using TypeScript with React Native and will get calls from the RESTful API. It will be able to run on mobile devices and on web applications.

4.3 Development Methods

The development methods that we will use for this project will mainly center around an agile-like SDLC. We will also be focused on feature driven development, which is based on agile. Being agile, or agile-like, will allow us to collaborate on our work and be able to write and test our code more efficiently than in a waterfall model. Collaboration is important in this project, and will allow us to work together to work more efficiently in our project.

5.0 Architectural Strategies

For the project's UI, we will focus on designing reusable components to maintain a clean, efficient, and maintainable codebase. Reusable components not only reduce redundancy but also allow us to adhere to DRY principles, which will help improve consistency across the front end.

On the back end, we are leveraging the Spotify API, which is a RESTful microservice that returns JSON data. To integrate this API, we will implement a Service-Oriented Architecture. The project will be structured into layers to encapsulate business logic and maintain a separation between the front-end and back-end functionality.

Environment variable management is another critical aspect of the project, especially for securely consuming the Spotify API key. Since this API key is required for authentication, it will be stored in environment variables rather than hardcoded in the codebase. This approach protects sensitive information from being exposed, particularly in public repositories like GitHub, where API key crawlers are common. By configuring a *.env* file, we can store the API key and ensure it is accessible locally or in deployment without being included in version control.

Testing, while important, will be constrained due to limited time. Our primary focus will be on unit tests using Jest, which integrates with TypeScript. Additionally, we will use Playwright for end-to-end (e2e) testing to validate our project . While being able to achieve full test coverage may not be feasible within the project timeline, these testing strategies will ensure key functionality is reliable.

6.0 System Architecture

On the UI portion of the project, we will follow component based architecture. We plan on our components being able to be reusable in the application. This will help follow DRY principles and allow us to have a consistent UI across the project. The front-end will focus on reusable components, and also interact with the service layer of our project. To give an overview of our components, we could structure it like a music player component, a home page component, a tab bar component, and a user library component. These components give us an overview of what we might have, and these components can also contain smaller sub components that can be shared and reused over various components.

While the front-end layer will focus on building and managing the UI and its components, the service layer will handle all API-related logic. For example, we will define an interface (*IProjectAPIService)* that will be implemented by *ProjectAPIService*. This approach ensures that our service layer adheres to a defined contract, making our codebase more testable and robust. The *ProjectAPIService* class will encapsulate business logic related to the Spotify API, including API calls. Using interfaces allows for easier testing, such as mocking, though testing may be limited due to time constraints. To integrate the service with React, we will use Context providers to manage dependency injection, ensuring the service can be accessed across components.

7.0 Detailed System Design

7.1 Classification

Our main component will be our music player. This component will have multiple functionalities, and will be dependent upon the functionality of some sub components for functionality. The central component is responsible for handling audio controls, playback, and streaming interactions.

7.2 Definition

The music player component will be responsible for a pleasant music playback experience. The audio data is fetched from the Spotify API, which is decoded and buffered. The user interaction with the playback controls will allow the user to be able to skip, play, scrub, pause, and more. The component will be very important for our UI, and will be the main focal point for development.

7.3 Constraints

Some constraints we might find along the way would be things like state management, like playback, pause, play, resume, etc. Network is also a constraint we could face, depending on the internet connection of a user and our app will probably have limited offline access. The last likely constraint that we will face is synchronization to make sure the UI is able to be updated and synchronous with the backend. Exception handling will need to be important to help deal with some of these issues, like network failure, track loading issues, and more.

7.4 Resources

Resources that will be external will be the Spotify API that we will be heavily integrating with. User preferences will need to be stored for personalized settings. Most of our external resources will be heavily reliant upon the Spotify API. We will need to make sure that we are able to be concurrent with the API, and make sure that multiple user interactions are handled and that deadlocks will not happen while waiting on the API.

7.5 Interface/Exports

Some sample API calls to the Spotify API will be listed below, however you can find more information about these API calls referenced in the bibliography.

Get Currently Playing Track - *getCurrentTrack()*

Start playback for a track - *play(trackId: string)*

Pause the current track - *pause()*

Resume the current track - *resume()*

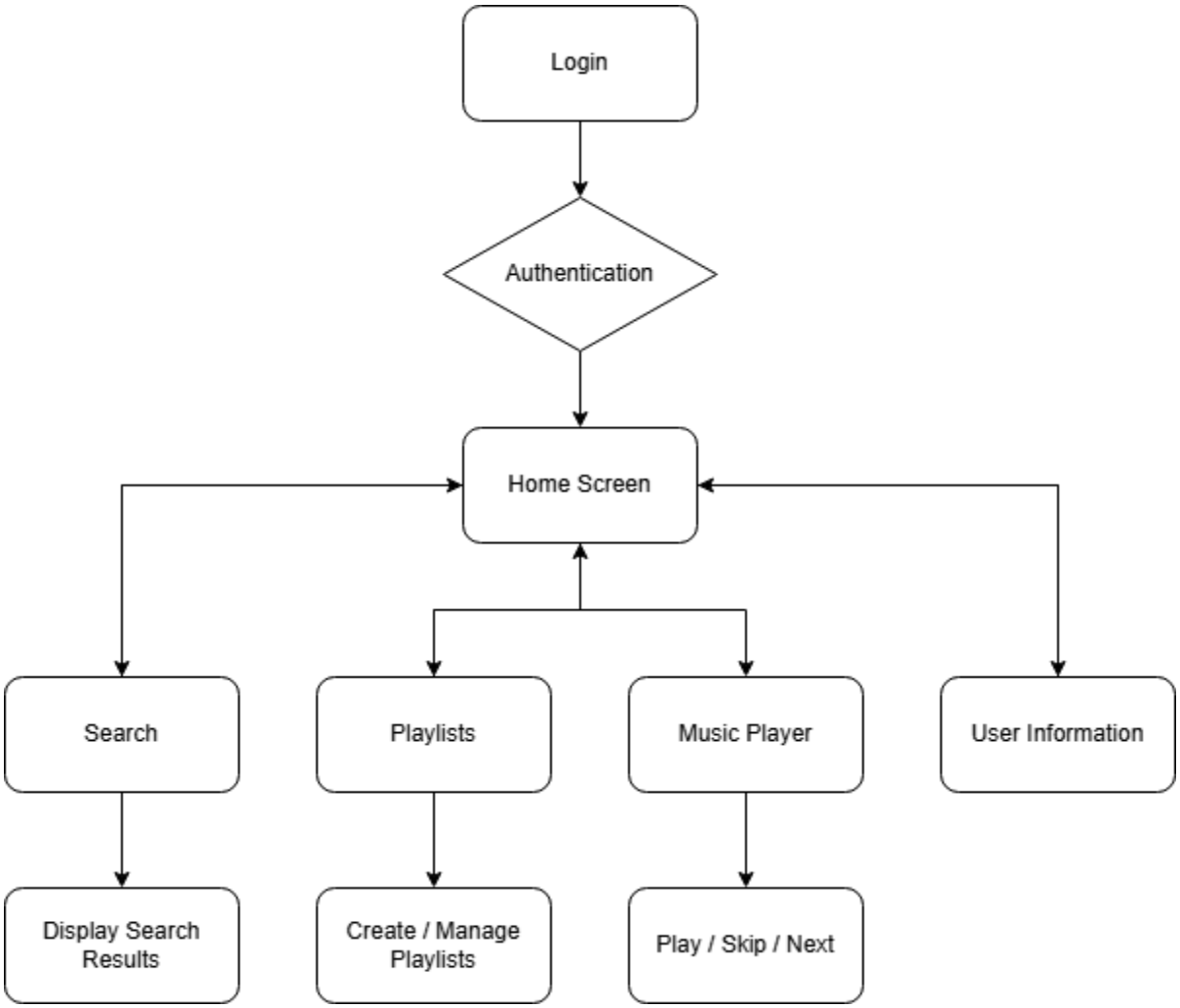
Load track - *loadTrack(trackId: string)*

Fetch Lyrics if available for current track - *fetchLyrics(trackId: string)*

7.6 Process Flow Diagram

- Representation of how the systems will be put together and the overall flow of our app.

- Upon loading our app, the login screen will pop up and the user will login using their Spotify credentials.
- The authentication process will then start, which will grant the user an authentication token upon successful login.
- After logging in, the home screen will pop up with different tabs at the bottom of the screen, these being: home(current) search, playlists, music player, and user information.
- The search tab will allow the user to search songs, playlists, albums, and artists.
- The playlists tab allows the user to create and manage playlists.
- The music player gives the user the ability to control their listening experience by playing, skipping and repeating songs, as well as queue management.
- The user information tab showcases user's information



8.0 Development

Spotify App

- React Native with Expo

- Utilizing Spotify API
    - Login screen/tab
        - Home screen/tab
            - Logout option
        - Playlist screen/tab
            - Playlist details
            - Back button
        - Search screen/tab
            - Search songs, artists, albums, and, playlist
                - Display search results
        - Music playback/control
            - Play, pause, skip songs
            - Adjust volume
            - Queue Management

Upon loading the app, the user will be prompted to login with their spotify credentials. This will send an authorization request to the application endpoint of the Spotify OAuth 2.0 Service. This will request an authorization code which will then grant the user an access token. With this access token, we can use this to generate an authorization header which will be included in API calls or get requests. Upon successful login the user will be redirected to the Home url. From there, the user can switch to different tabs at the bottom of the application. These tabs will include search, playlist, and user. When clicking on these buttons, the user will be redirected to the respective url for each page.

The search page will include a search bar which will take user input and send an API request to the search endpoint, which will send back a response with the filtered search input. This response will display songs, artists, and albums. The playlist page will include the user's playlist. The user can create a playlist by clicking the button in top right. This will bring up a prompt to name the playlist, add a description, and add the first song. Upon clicking on a playlist, the application will proceed to the playlist url. Within the playlist page, the user can play the songs within the playlist, search songs to add to it, or delete songs from it. The search bar will function the same as the search page, except it will only pull tracks. To delete a song from the playlist, the user can hold down on a selected song which will prompt the user to delete the song or cancel said function. The application will be utilizing the Web Playback SDK to play songs. This component will provide information about the track being played and will also include buttons to pause/play the current track, skip track, and previous track. The user page will provide personal information of the user’s account, by sending Get requests and then pulling from specific information depending on what is requested.

8.1 Setup Process

Login to Spotify Developer Dashboard. Create an app through the dashboard. Fill in the respective information tabs, app name, description, and url. Confirm the information is how you want it displayed, then hit the create button. Go to the dashboard and click on the newly created app. Click on the settings button and then copy down the Client ID and Client Secret. With these

credentials, we can now request an access token. Send a POST request to the token endpoint URL. Add a Content-type header, along with a body with the Client ID and Client Secret with the grant_type parameter set to client_credentials. With these setup, we can now begin development of our application. To use in our application, we will add the Client ID and Client Secret to our .env file. This .env file will be read by our application when it is in use to handle the authentication of the user. To finish, you will need to add a few redirect URIs so that the callbacks are handled correctly for authentication. You will add URIs like the following.

- exp://192.168.10.209:19000
- http://localhost:8081
- exp://192.168.10.209:8081

After adding these URIs, you are ready to run the application. You can test this application using Expo by using the we include design and architectural drawings, which explain how your software project components interact or scanning the QR code in the terminal and accessing the application using Expo Go.

9.0 Software Test Plan

To ensure the application is able to meet high standards of quality and functions correctly, we will perform testing across multiple devices. The tests will focus primarily on functional aspects of the app, but will also include some non functional aspects. Each team member will test these from their respective devices in order to make sure that our product meets expectations.

Our main objective is to validate that all features of the app are functional on multiple devices. During our testing, we can resolve UI issues or any unexpected behavior. Our primary field of testing was over different components. Login is a critical component that handles the user logging in and out of the app. The search feature is also critical to make sure users can search and play songs that they wish to listen to. Playlist creation and management are two large aspects of the app, allowing the user to create playlists and also add and delete songs from these playlists. Finally, we have the music controls. This allows our user to be able to control what songs they want to listen to and to skip songs they don’t want to listen to.

10.0 Software Test Report

Requirement	**Pass**	**Fail**	**Severity**
Login	✓		
Search	✓		

Playlist Creation	✓		
Playlist Management	✓		
Music Controls	✓		

11.0 Version Control

We used Github for version control. We created branches to change and improve our app before pushing it to the main branch. We strived to follow the Gitflow workflow, since it is a basic and simple to understand workflow for GitHub and will adhere to agile principles. We have one default branch “main” and a develop branch. Whenever a developer is working on a new feature, the branch will be a feature branch and follow the naming convention of “feature/NameOfFeature”. Hotfixes follow a similar naming convention. When features are finished, a pull request is opened by the developer and reviewed and merged into the develop branch. When the project is finished and completed, we will do a release from the main branch.

12.0 Conclusion

The SP-27-BlueSpotify App project we have presented demonstrates our successful integration of the Spotify API within a React Native application. Our team has collaborated to work on a responsive, cross platform application that allows users to have a familiar Spotify experience while still providing more features and enhancements for personalization. Our feature-driven development and agile practices with the SDLC allowed us to manage our tasks efficiently and ensure that our critical components were implemented correctly and tested thoroughly.

Although we had some time constraints, we were able to deliver a fully functional application with great performance and usability. This project helped deepen our understanding for mobile software development and API integration, as well as the Software Development Lifecycle. Our team also learned how to manage time constraints and collaborate together on a software project, and provide deliverables.

This project has provided our team the necessary experience for future professional development with specialities in full-stack development, design, and using third-party APIs. We are proud of our final product and how we were able to meet our goals and expectations.

13.0 Appendices

13.1 Appendix A - Glossary

API (Application Programming Interface) – A set of rules and protocols that allow different software applications to communicate with each other.
Spotify API – A web-based service provided by Spotify that allows developers to interact with Spotify's music catalog, playlists, user data, and playback controls.
OAuth 2.0 – A secure authentication protocol used to grant third-party applications access to user accounts without exposing login credentials.
Refresh Token – A token used to obtain a new access token without requiring the user to log in again.
Playlist – A collection of tracks curated by a user or algorithm, which can be accessed and modified through the Spotify API.
RESTful API – An API that follows the principles of REST (Representational State Transfer), allowing interaction using standard HTTP methods like GET, POST, and DELETE.
Scope – A permission level that defines what an application can access on behalf of a user.
SDLC – Refers to the Software Development Lifecycle, which is a structured process that is used to develop high quality software following a methodology that defines the entire procedure.

13.2 Appendix B - Source Code

For our draft, we will provide the GitHub repository link. For the final version, we will have both the repository link and a zip file.

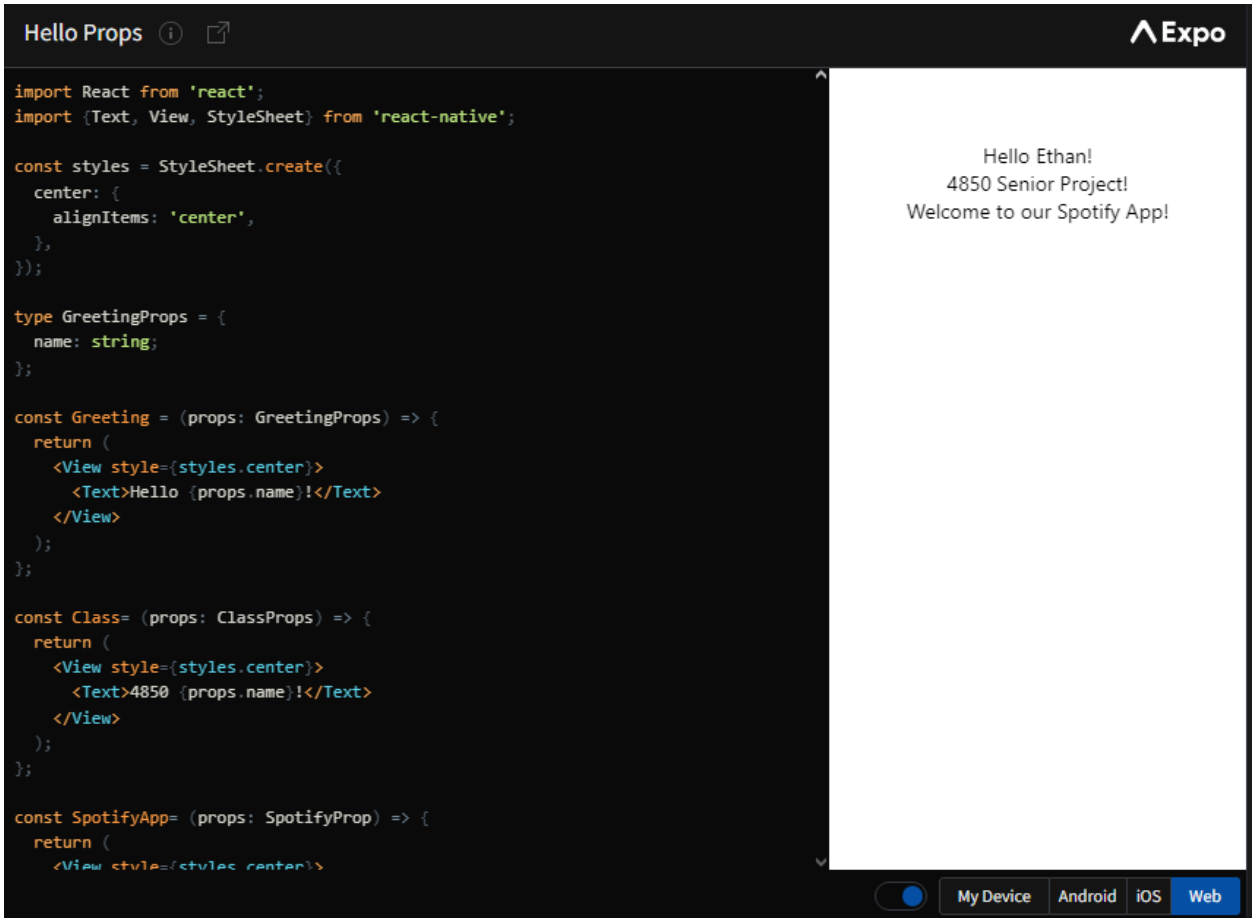
https://github.com/SP-27-BlueSpotify/SP-27-BlueSpotify

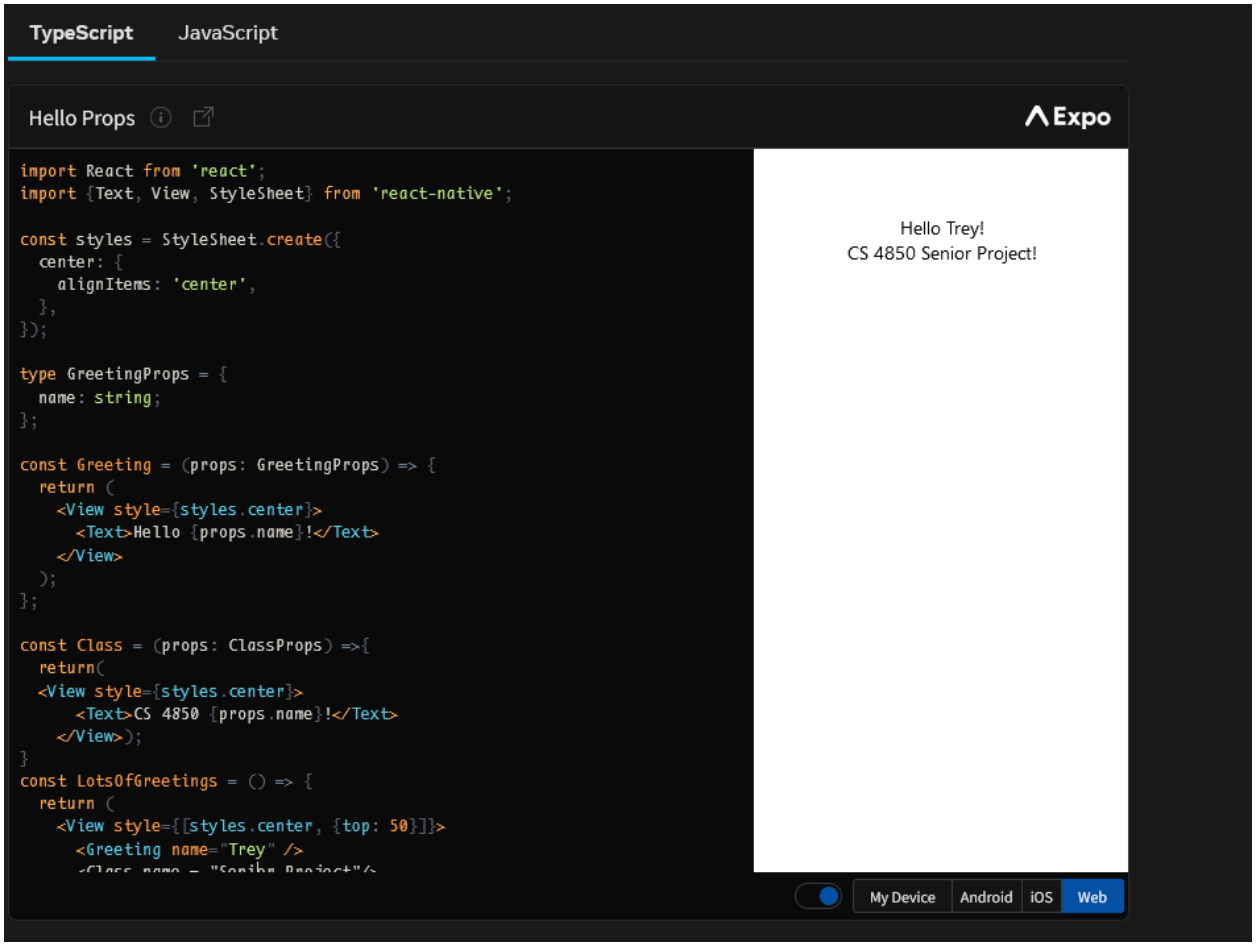
13.3 Appendix C - Bibliography

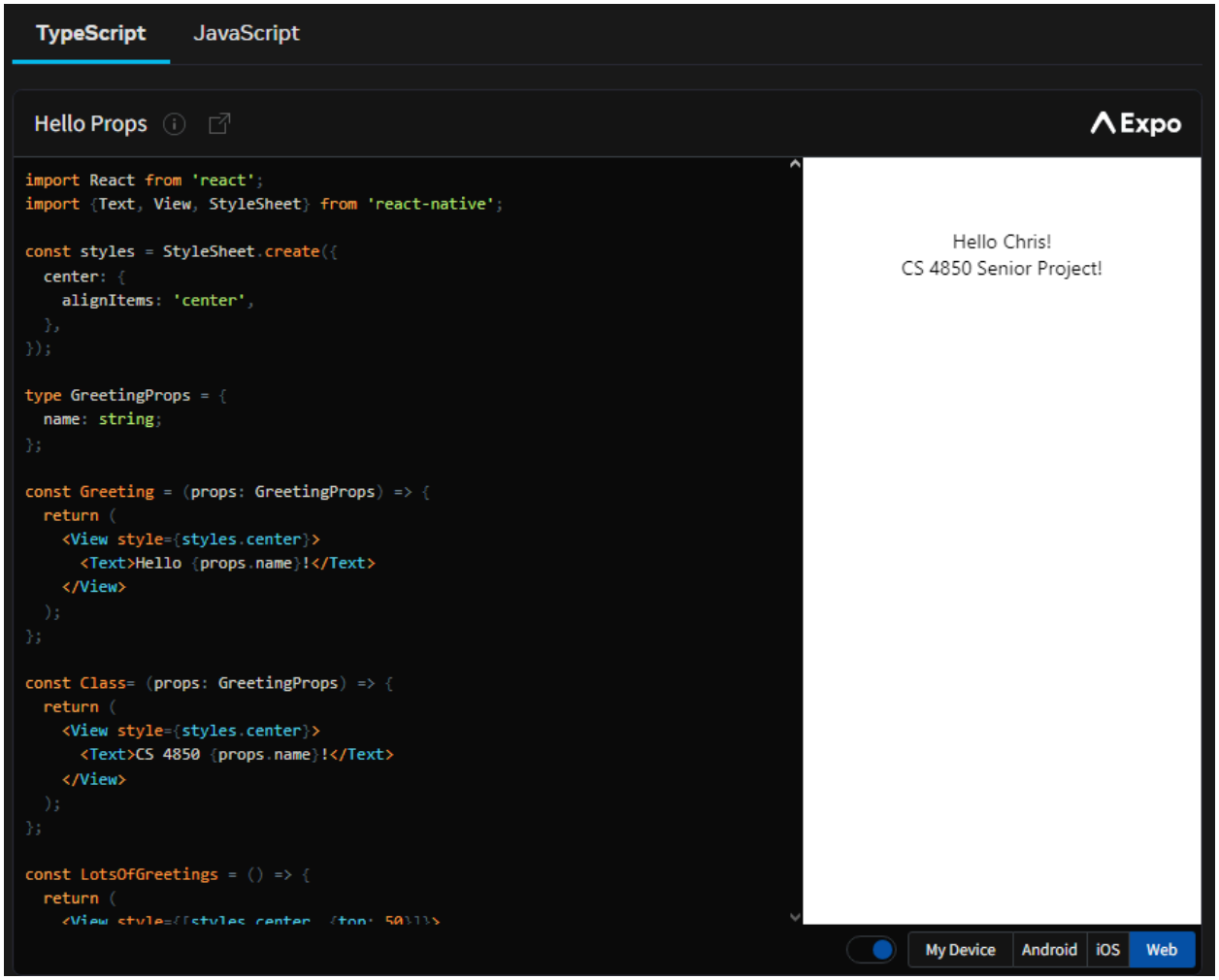
Spotify API Documentation:
https://developer.spotify.com/documentation/web-api/concepts/api-calls

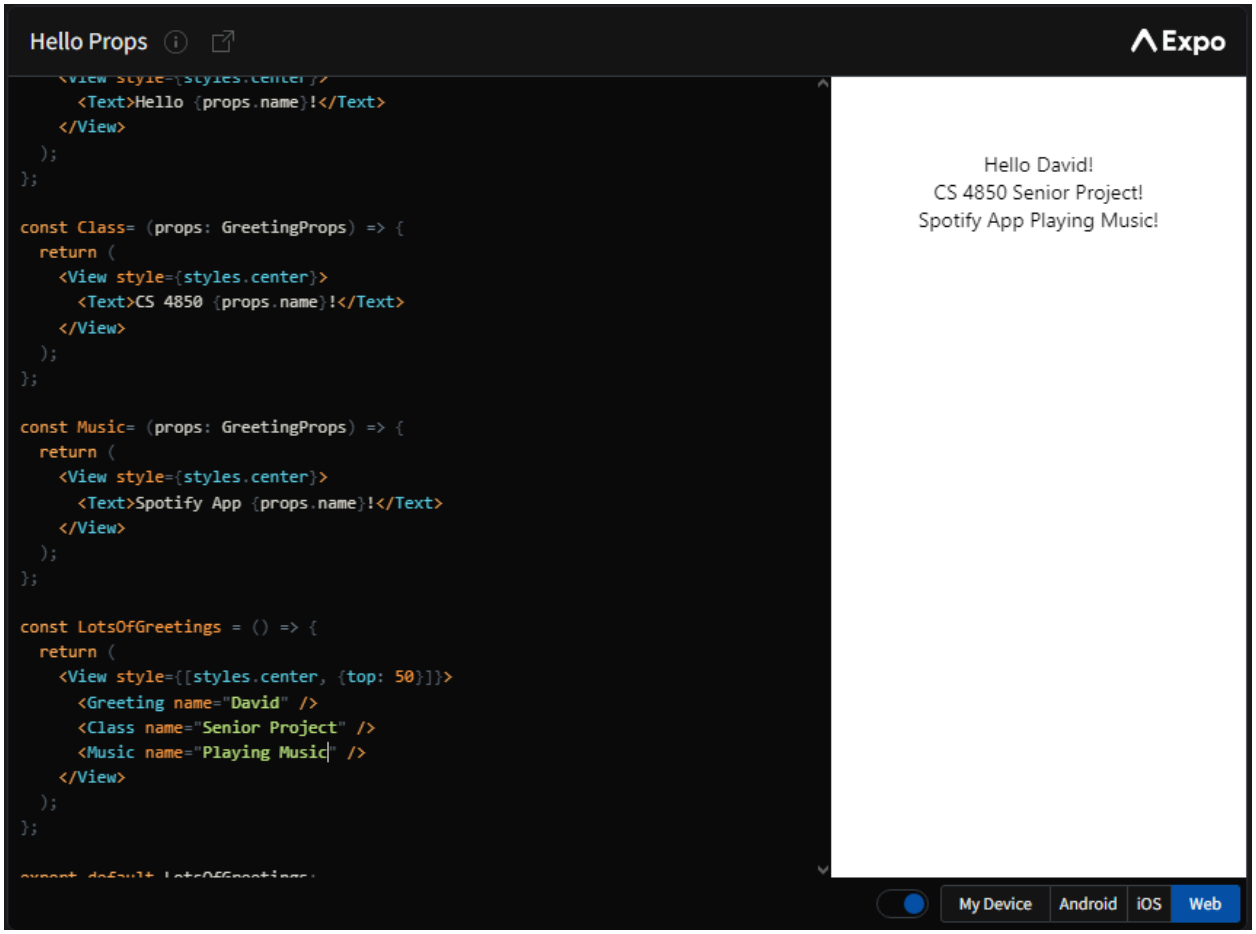
13.4 Appendix D - Tutorial Verification

"Hello Props!" tutorial in React Native:


Hello Props
Expo
import React from 'react';
import {Text, View, StyleSheet} from 'react-native';
const styles = StyleSheet.create({
center: {
alignItems: 'center',
},
});
type GreetingProps = {
name: string;
};
const Greeting = (props: GreetingProps) => {
return (
<View style={styles.center}>
<Text>Hello {props.name}!</Text>
</View>
);
};
const Class= (props: ClassProps) => {
return (
<View style={styles.center}>
<Text>4850 {props.name}!</Text>
</View>
);
};
const SpotifyApp= (props: SpotifyProp) => {
return (
Hello Ethan!
4850 Senior Project!
Welcome to our Spotify App!
My Device
Android
iOS
Web


TypeScript
JavaScript
Hello Props
Expo
import React from 'react';
import {Text, View, StyleSheet} from 'react-native';
const styles = StyleSheet.create({
center: {
alignItems: 'center',
},
});
type GreetingProps = {
name: string;
};
const Greeting = (props: GreetingProps) => {
return (
<View style={styles.center}>
<Text>Hello {props.name}!</Text>
</View>
);
};
const Class = (props: ClassProps) =>{
return(
<View style={styles.center}>
<Text>CS 4850 {props.name}!</Text>
</View>);
}
const LotsOfGreetings = () => {
return (
<View style={[styles.center, {top: 50}]}>
<Greeting name="Trey" />
Hello Trey!
CS 4850 Senior Project!
My Device
Android
iOS
Web


TypeScript
JavaScript
Hello Props
Expo
import React from 'react';
import {Text, View, StyleSheet} from 'react-native';
const styles = StyleSheet.create({
center: {
alignItems: 'center',
},
});
type GreetingProps = {
name: string;
};
const Greeting = (props: GreetingProps) => {
return (
<View style={styles.center}>
<Text>Hello {props.name}!</Text>
</View>
);
};
const Class= (props: GreetingProps) => {
return (
<View style={styles.center}>
<Text>CS 4850 {props.name}!</Text>
</View>
);
};
const LotsOfGreetings = () => {
return (
Hello Chris!
CS 4850 Senior Project!
My Device
Android
iOS
Web


Hello Props
Expo
<View style={styles.center}>
<Text>Hello {props.name}!</Text>
</View>
);
};
const Class= (props: GreetingProps) => {
return (
<View style={styles.center}>
<Text>CS 4850 {props.name}!</Text>
</View>
);
};
const Music= (props: GreetingProps) => {
return (
<View style={styles.center}>
<Text>Spotify App {props.name}!</Text>
</View>
);
};
const LotsOfGreetings = () => {
return (
<View style={[styles.center, {top: 50}]}>
<Greeting name="David" />
<Class name="Senior Project" />
<Music name="Playing Music" />
</View>
);
};
Hello David!
CS 4850 Senior Project!
Spotify App Playing Music!
My Device
Android
iOS
Web