

Audrix - React Native Music Streaming App Template

Created: 20/05/19

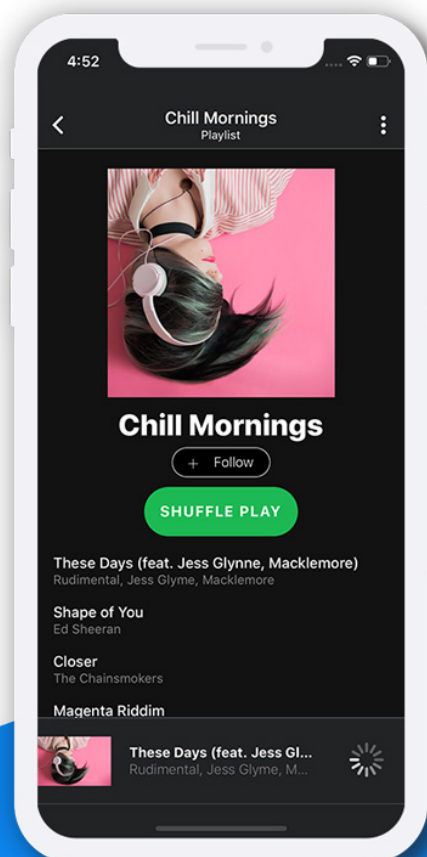
By: Enappd (enappdAdmin)

Email: admin@enappd.com

Thank you for purchasing our theme. If you have any questions that are beyond the scope of this help file, please feel free to email via our user page contact form [here](#). Thanks so much!

AUDIO STREAMING

REACT NATIVE APP TEMPLATE



FEATURES

- Complete UI
- Audio file playback
- Tested on iOS & Android
- iPhoneX compatible

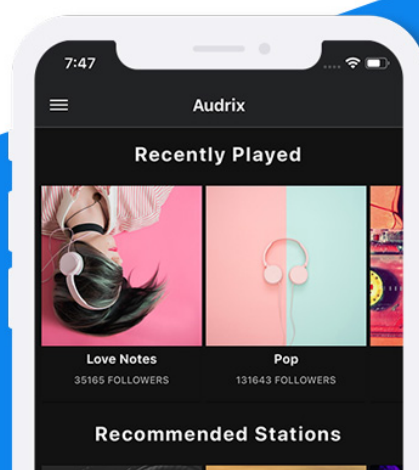


Table of Contents

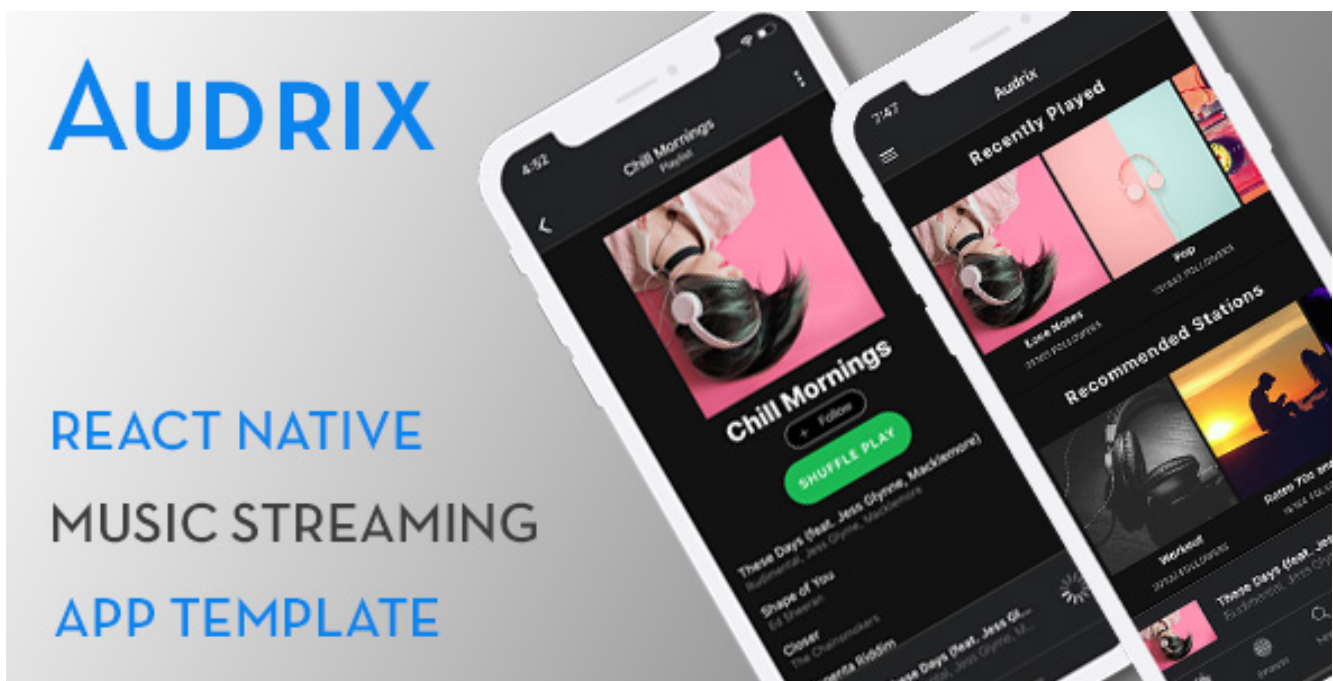
1. Introduction - What is this template
2. Features of this template
 1. Screens
 2. Features
 3. Device compatibility
3. Setup and Installation
 1. Introduction
 2. Installation
 3. Running your app
 4. Run your app with changes
 5. Creating icon and splash
 6. Generating Release builds
4. Modifying code
 1. Screen code structure
 2. Styling structure
 3. Logic structure
 4. Navigation structure
5. Feature Explanation
 1. User Authentication
 2. Adding songs, playlists, artists etc.
 3. Recommendations
 4. Search
 5. Radio
 6. Local Data storage
 7. Offline Data
 8. Music Player
 9. Settings
 10. Nested Navigation
 11. Premium feature addition
6. Integrating Back-end Data
 1. Data structure
 2. Back-end options

-
7. Troubleshooting
 8. Third-party libraries, sources and Credits

Introduction - What is this template

Audrix - React Native Music Streaming App Template is a mobile app theme / template. With this template you can create a Music streaming app like Spotify, Gaana, Wynk music, jio music etc.

This template is the front-end part of the music app, that means the screens and user interfaces (screens) are ready. You can easily change the screens, styles and logics to suit your requirements. To make it a live app like Spotify, Gaana, Wynk music, Jio music etc. you just need to add a back-end to the app, and load your data in the back-end.



This template is made in REACT NATIVE language.

With REACT NATIVE, you can

- Create an iOS and an Android app with single source code
- Cut your project cost and time in half
- Ensure exact same UI in iOS and Android, while following the standard UI norms
- Saves separate updates for each app in future

Features of this template

1. Screens

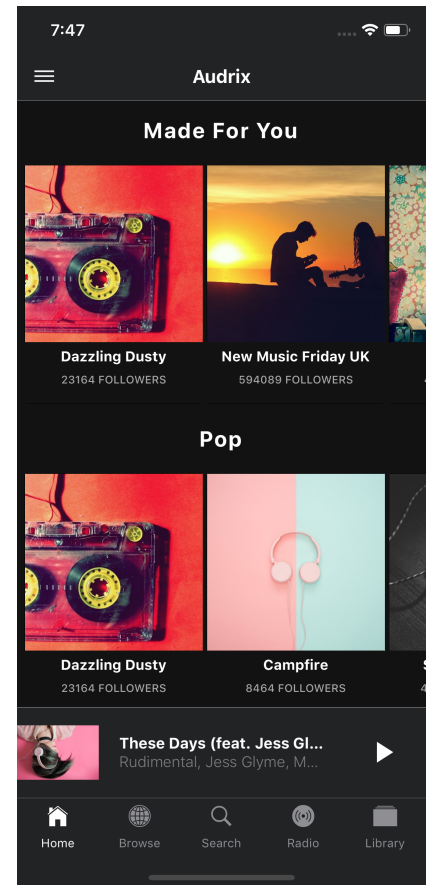
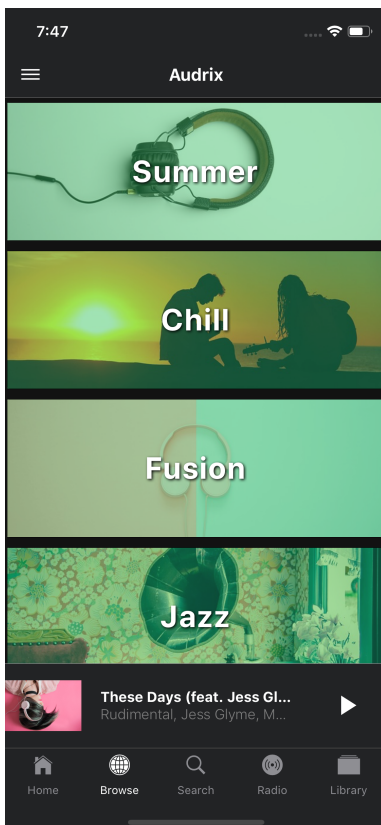
Home

This is first tab of the application.

Home page contains recommended music and other categories for a user.

The songs / albums are arranged in a horizontal scroll for each category

A vertical scroll allows addition of any number of categories on homepage, each category being arranged in horizontal scroll.



Browse

This is the second tab of the application.

Browse page contains music categories of all available music.

The categories are arranged in a vertical scroll. You can add as many categories you want. You can also implement an infinite scroll feature to fetch more categories on scroll.

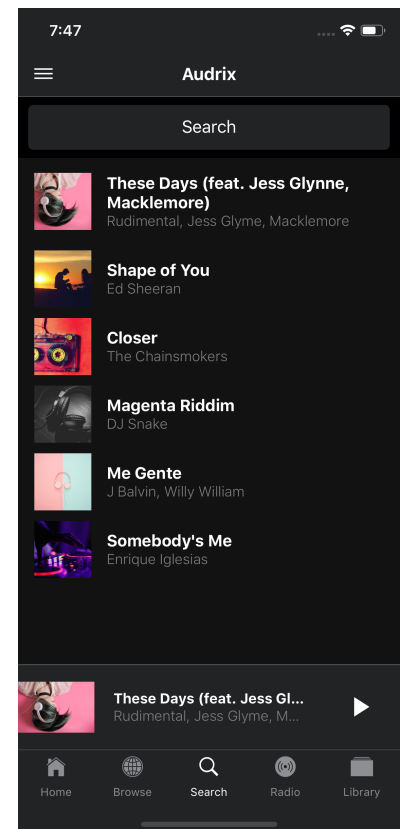
Search

This is the third tab of the application.

Search page contains the UI for searching songs, albums, artists etc.

Just type any word in the input box and the results can be displayed in the content as a list

For demo purpose, when you type in the search box, you see preset results in the result.



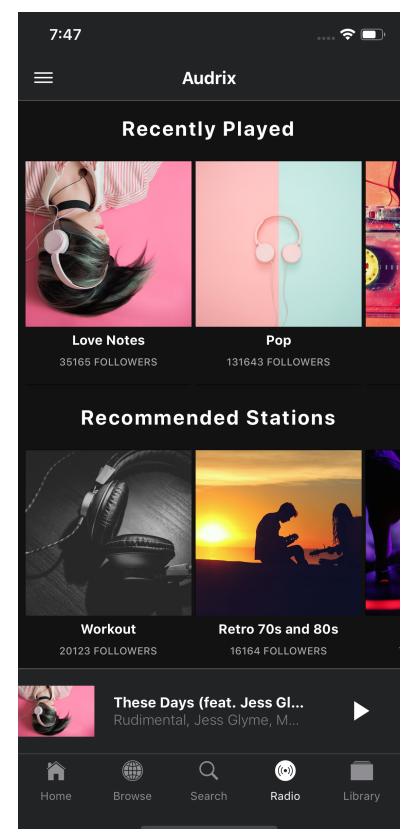
Radio

This is the fourth tab of the application.

Radio page contains options to run one of may radio stations available.

The radio stations are arranged in a horizontal scroll for different category.

A vertical scroll allows addition of any number of categories on homepage, each category being arranged in horizontal scroll.

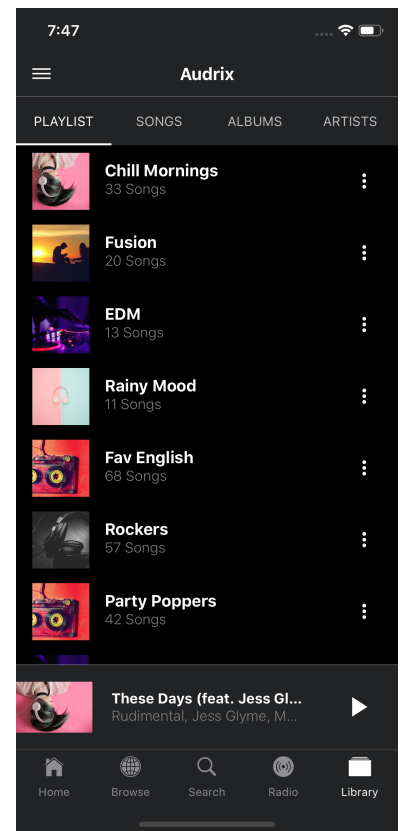
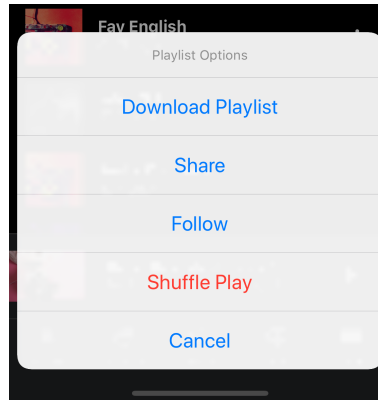


Library - Playlist

This is the fifth tab, first sub-tab of the application.

Playlist page contains your playlists arranged in a vertical list. The option button for each playlist gives you more options to operate on your playlist.

You can add as many playlists you want. You can also implement an infinite scroll feature to fetch more playlists on scroll.

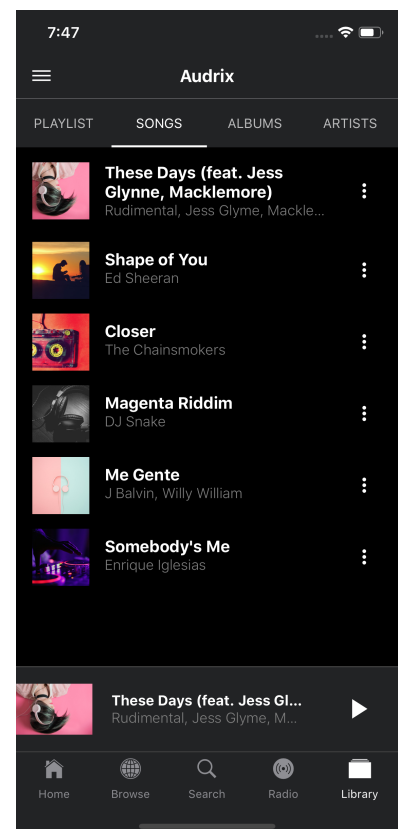
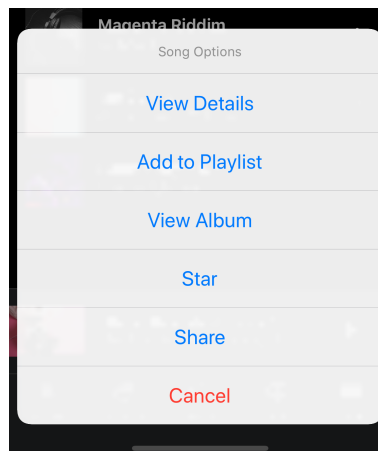


Library - Songs

This is the fifth tab, second sub-tab of the application.

Songs page contains your favorite songs arranged in a vertical list. The option button for each song gives you more options to operate on your song.

You can add as many songs you want. You can also implement an infinite scroll feature to fetch more songs on scroll.

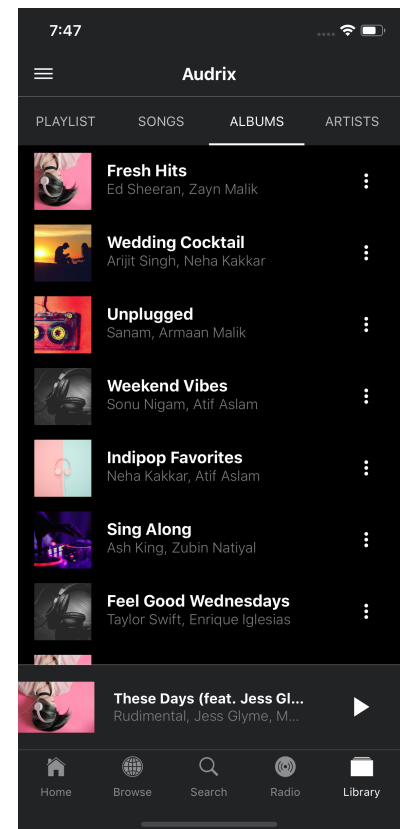
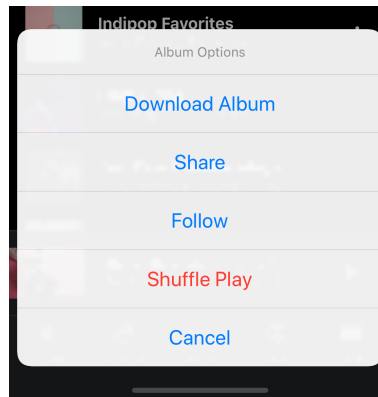


Library - Albums

This is the fifth tab, third sub-tab of the application.

Albums page contains your favorite albums arranged in a vertical list. The option button for each album gives you more options to operate on your album.

You can add as many albums you want. You can also implement an infinite scroll feature to fetch more albums on scroll.

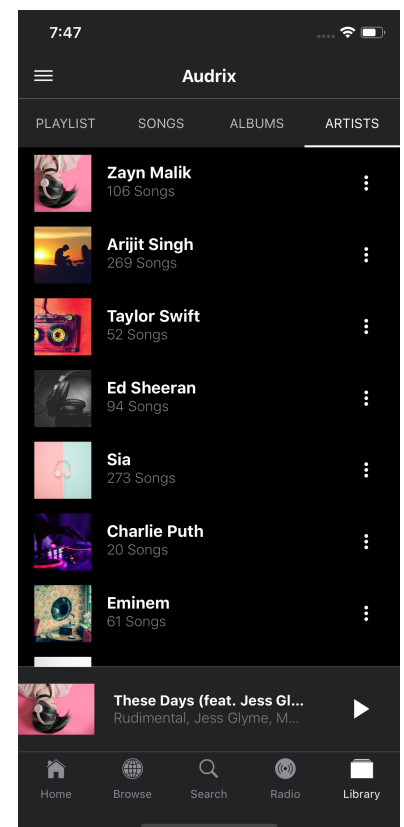
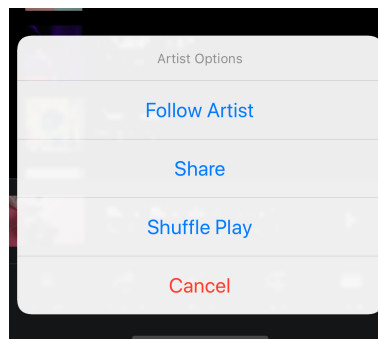


Library - Artists

This is the fifth tab, fourth sub-tab of the application.

Artists page contains your favorite artists arranged in a vertical list. The option button for each artist gives you more options to operate on your artist.

You can add as many artists you want. You can also implement an infinite scroll feature to fetch more artists on scroll.



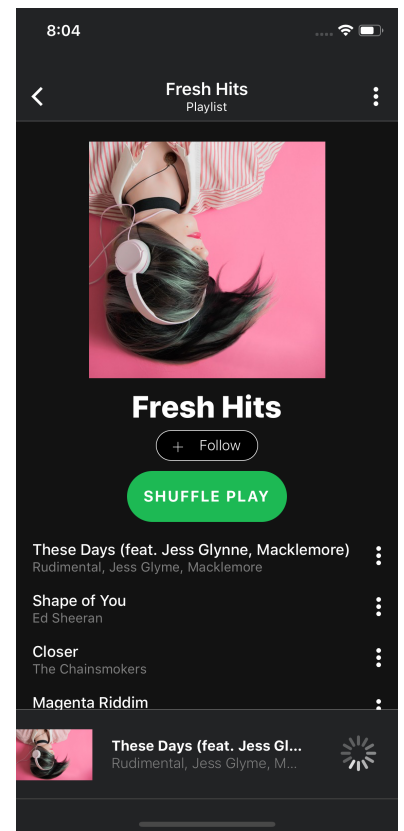
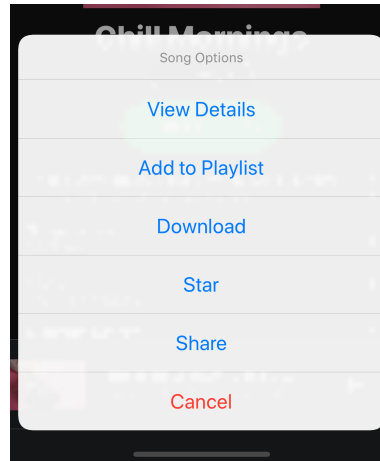
Playlist (General)

This page can be accessed from many places, by clicking on a particular playlist item.

Playlist page contains the songs of the playlist arranged in a list format in the bottom half of the page.

There is option to “follow” the playlist, and “Shuffle” the songs order.

Clicking on the Option button of a song gives you additional options.



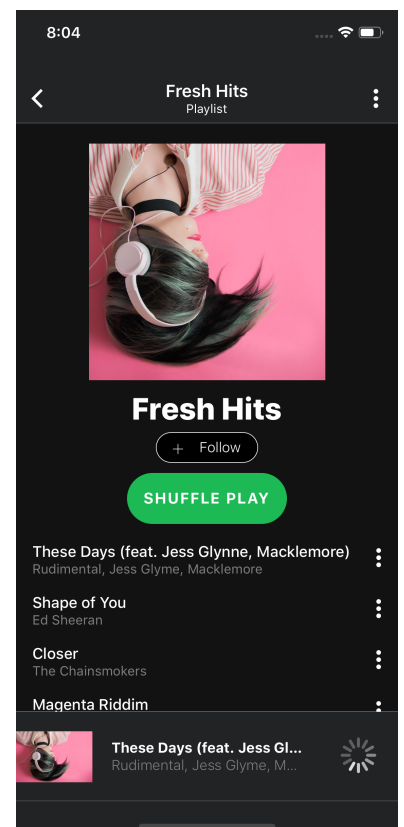
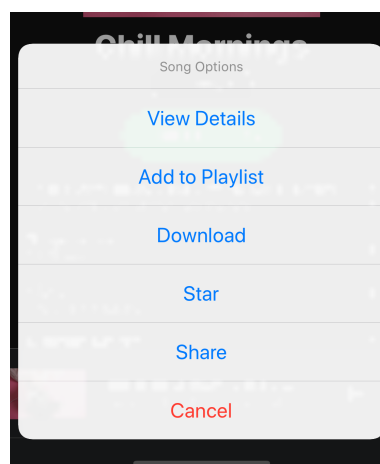
Album

This is actually the Playlist page itself, because each Album can also be a playlist.

Album page contains the songs of the album arranged in a list format in the bottom half of the page.

There is option to “follow” the album, and “Shuffle” play all the songs.

Clicking on the Option button of a song gives you additional options.

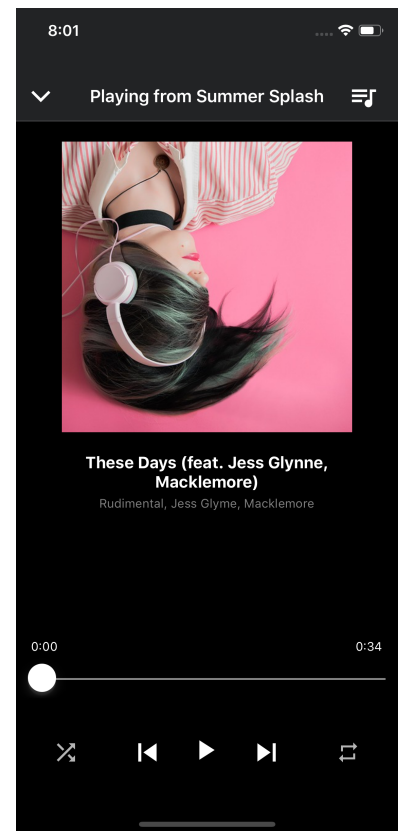


Music Player

This page can be accessed from many places, by clicking on a particular song item or by clicking the Mini-player on the bottom of any page.

Music player page contains the currently playing playlist, with songs of the playlist arranged in an image slider on the top half of the page. At the bottom, there is a music control bar with play, pause, next etc. buttons.

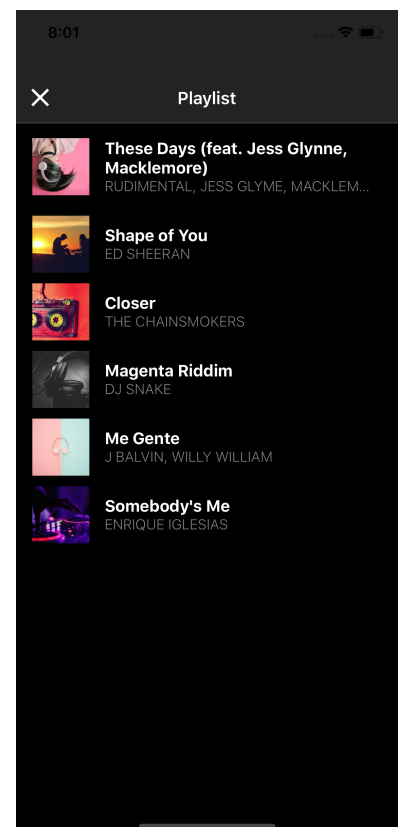
On clicking the Playlist button in top-right of the header, the current playlist opens.



Current Playlist

This page can be accessed from the Music Player page's playlist button.

This page contains the songs of the currently playing playlist arranged in a vertical list. Any song can be clicked and played in the music player.



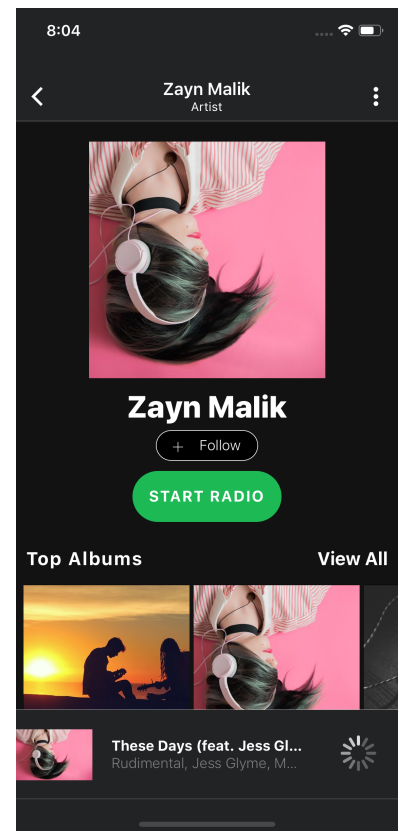
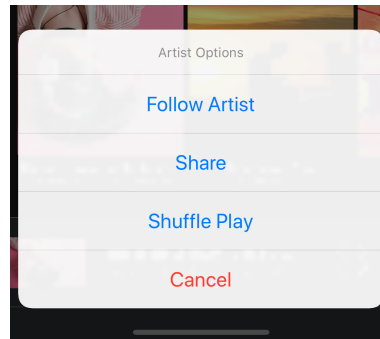
Artist

This page can be accessed from many places, by clicking on a particular artist name.

Artist page contains the artist's image on top, with options to "Follow" and "Start Radio" for that artist's songs

The page also contains the artist's media categorized into many horizontal scrolls - Top albums, Top songs etc.

Option button in the top-right of the header gives more options related to the Artist.



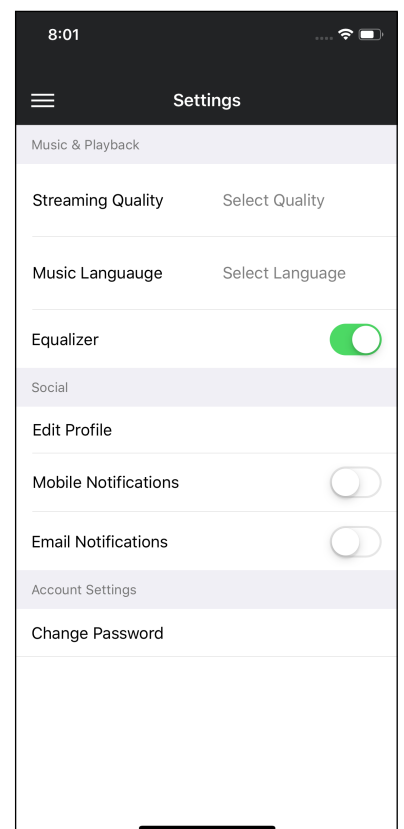
Settings

Settings page can be accessed from the Sidemenu's Settings option.

This page contains the settings / options for the app like

- Streaming quality
- Language
- Equalizer
- Edit Profile
- Notifications
- Change Password

Most of these settings options navigate to other pages. The toggle options can be read directly from this page itself.

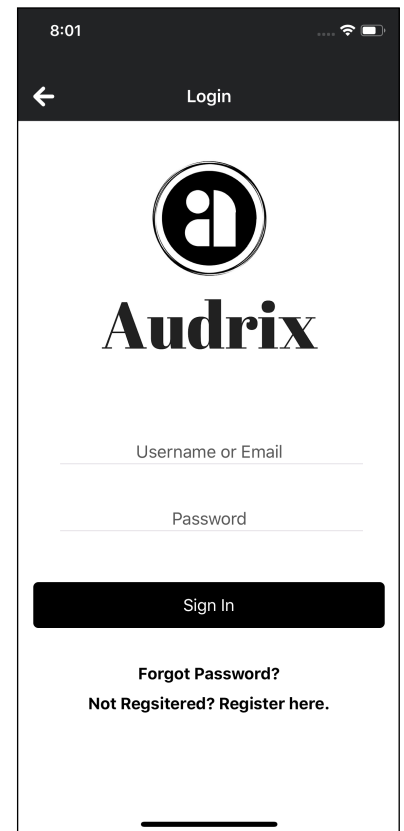


Login

This page can be accessed from Sidemenu's "Login" option.

Login page can be used to perform login action with username / email and password. Currently, the login is only for demo purpose.

Login page also contains "Forgot Password" and "Register" page navigations.



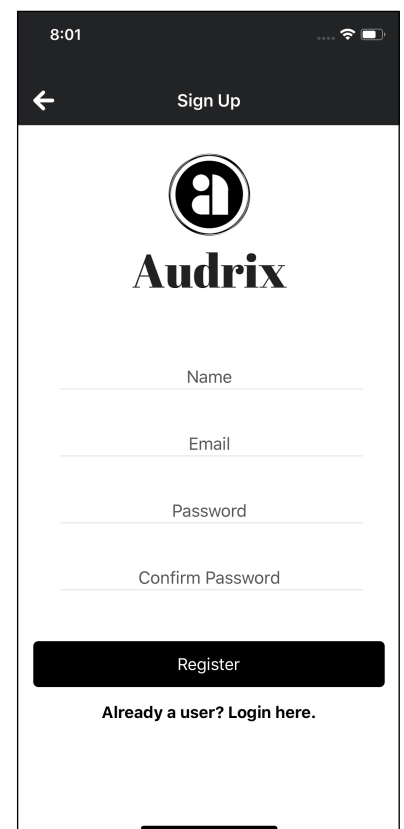
The screenshot shows the Audrix Login page on a mobile device. At the top, the status bar displays the time 8:01, signal strength, Wi-Fi, and battery icons. Below the status bar is a dark header with a back arrow and the title "Login". The main content area features the Audrix logo (a stylized 'a' in a circle) and the brand name "Audrix" in a bold, serif font. Below the logo are two input fields: "Username or Email" and "Password". A black button labeled "Sign In" is positioned below the input fields. At the bottom, there are two links: "Forgot Password?" and "Not Registered? Register here."

Register

This page can be accessed from "Login" page.

Register page can be used to perform signup action with name, username / email and password. Currently, the signup is only for demo purpose.

Register page also contains "Login" page navigation.



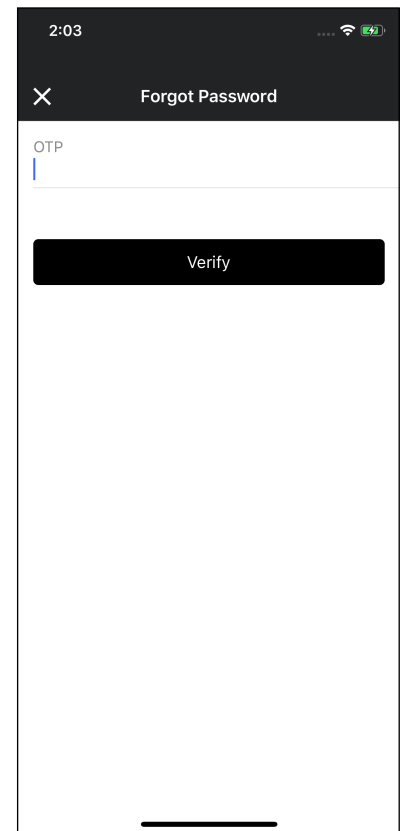
The screenshot shows the Audrix Sign Up page on a mobile device. At the top, the status bar displays the time 8:01, signal strength, Wi-Fi, and battery icons. Below the status bar is a dark header with a back arrow and the title "Sign Up". The main content area features the Audrix logo (a stylized 'a' in a circle) and the brand name "Audrix" in a bold, serif font. Below the logo are four input fields: "Name", "Email", "Password", and "Confirm Password". A black button labeled "Register" is positioned below the input fields. At the bottom, there is a link: "Already a user? Login here."

Forgot Password

This page can be accessed from “Login” page.

Forgot password page can be used to send a reset password url or OTP to user’s email or any other way you want to use. Currently, this page is only UI.

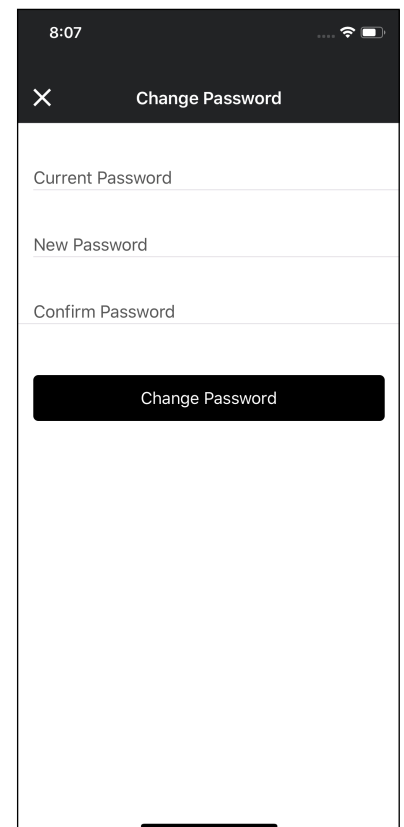
The OTP sent to user can be verified here and user can be asked to provide a new password



Change Password

This page can be accessed from “Settings” page.

Change Password page can be used to change your current password. This requires user to enter their current and new password.

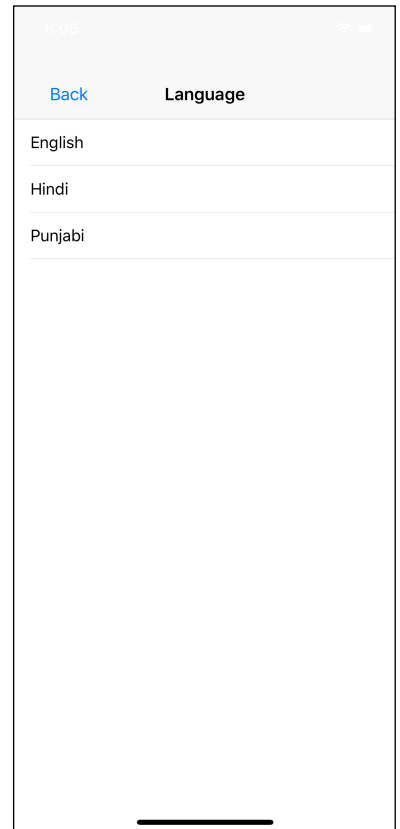


Language

This page can be accessed from “Settings” page.

The options available can be chosen to change the language of the app / music, as per your requirement.

“Back” button takes you back to “Settings” page

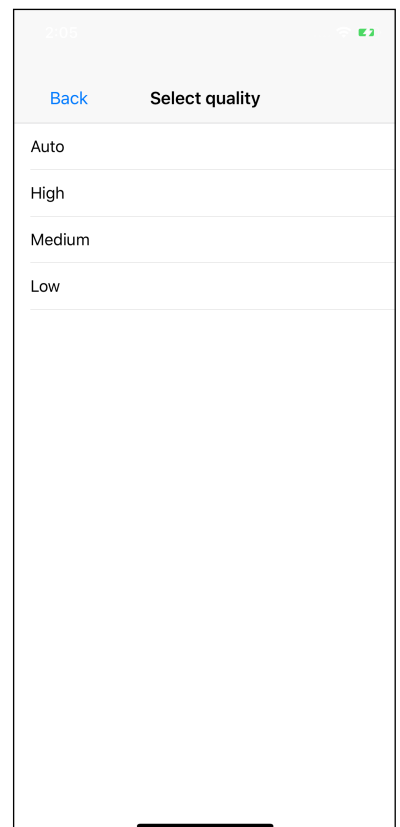


Music Quality

This page can be accessed from “Settings” page.

The options available can be chosen to change the quality of the music, as per your requirement.

“Back” button takes you back to “Settings” page

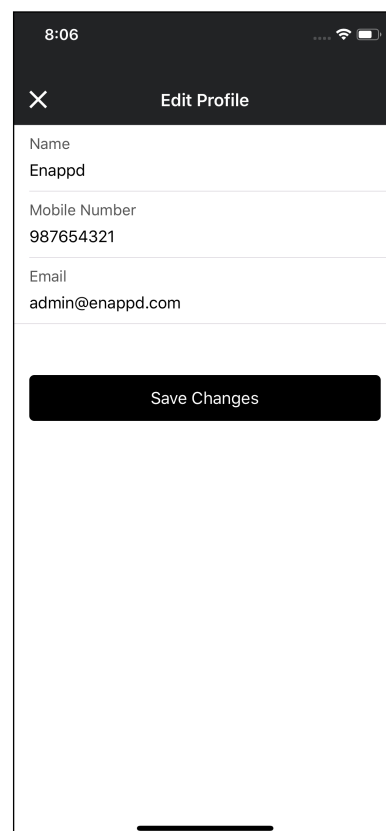


Edit Profile

This page can be accessed from “Settings” page.

This page allows user to change their profile options. Many more options can be added here, as per your requirement.

“X” (close) button takes you back to “Settings” page



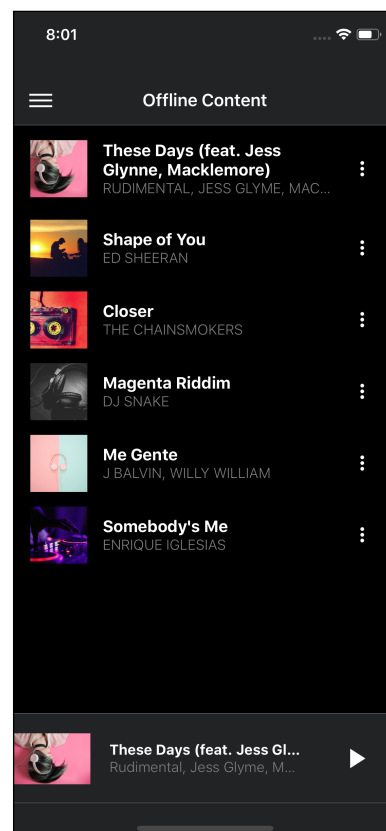
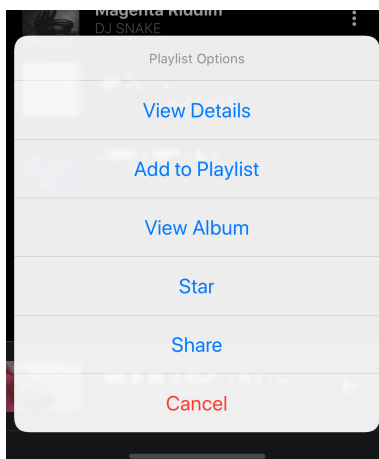
Offline Content

This page can be accessed from Sidemenu’s “Offline Content” option.

This page contains your offline downloaded songs in a vertical list arrangement. You can show as many songs you want in the list, or you can

Currently the offline page is visible to all users, but ideally it should only be visible to logged in users. If there is no offline data, the page should show “No data downloaded”

Clicking on the Option button of a song gives you additional options.



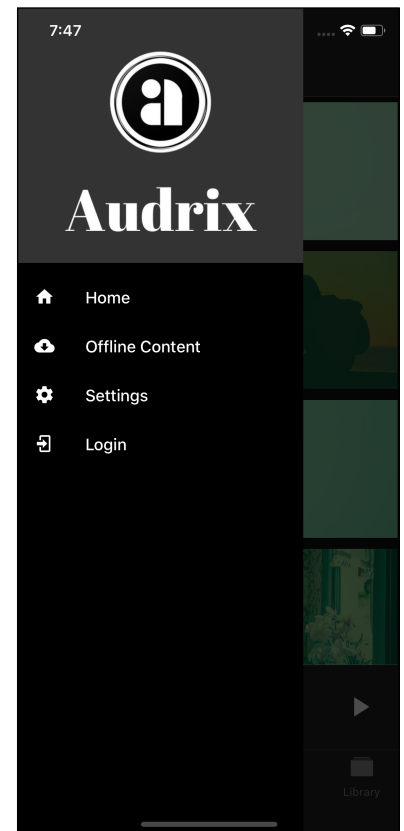
2. Features

Sidemenu

Sidemenu can be accessed from any page that shows a “Menu” button on the top-left in the header. The Sidemenu contains options to navigate to

- Home page
- Offline content
- Settings
- Login

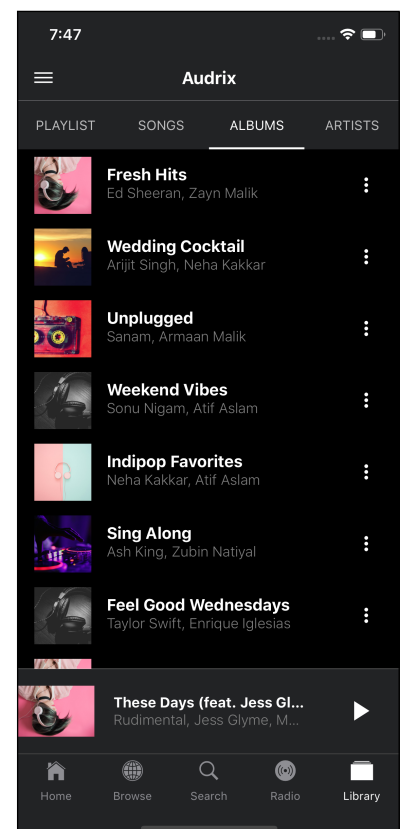
Sidemenu also shows the app’s name and logo, which you can change as per your liking.



Mini Music Player

Mini Music Player is present in almost all screens on the bottom of the screen. It shows the currently playing song, and a play / pause button depending on whether the song is playing or paused.

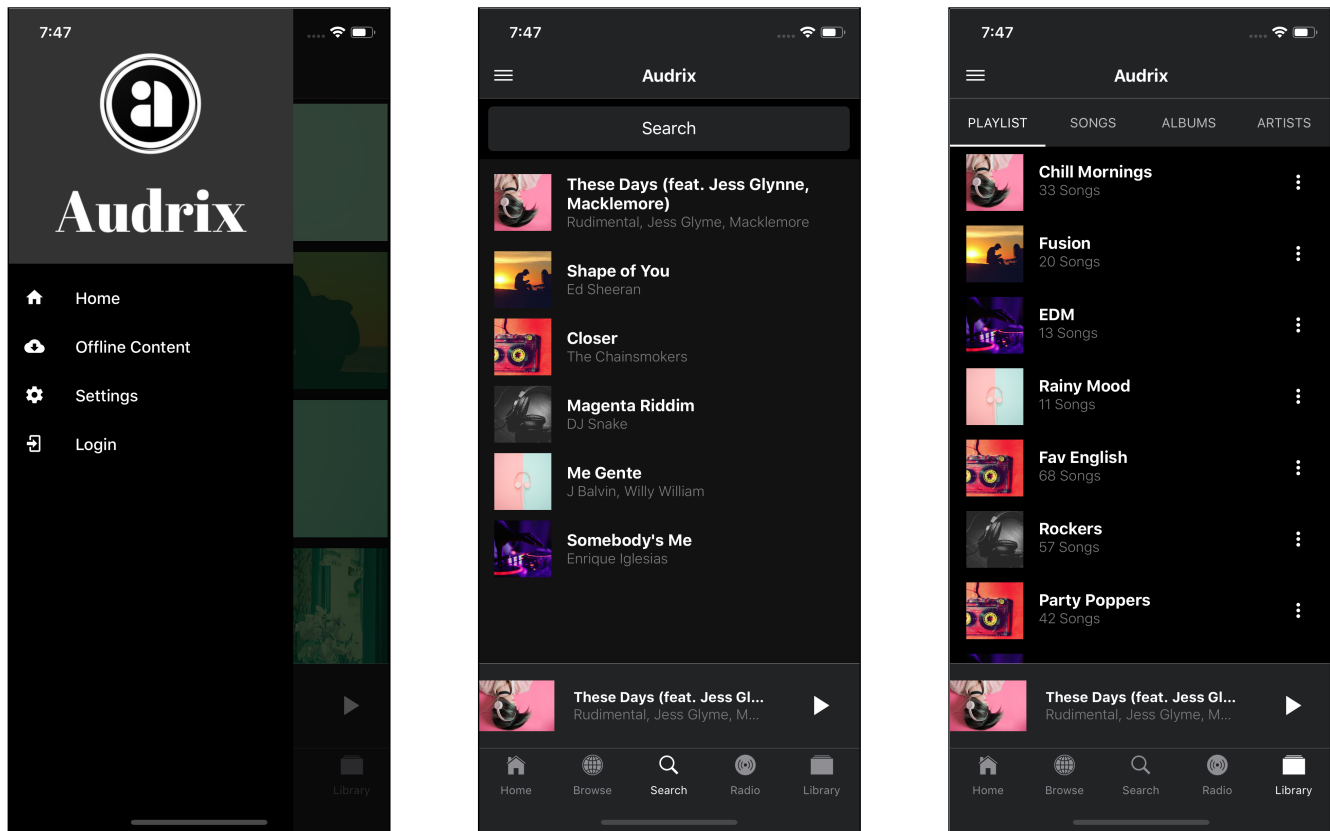
Clicking on Mini Music Player takes you to Music player page.



Nested Navigation

The app contains a nested navigation of Sidemenu and multi-level Tabs.

The app contains a Sidemnu which is the parent.



Five tabs inside “Home Page” is the first level of nesting

Four tabs inside “Library” page is the second level of nesting

3. Device Compatibility

As mentioned earlier , the app is written in React Native language. This allows a single source code to create both iOS and Android apps.

This template has been tested in all latest Android versions and major devices.

This template has also been tested in all latest versions of iPhones, including iPhone X.

Setups and Installation

1. Introduction

Audrix is a music streaming application made with React Native (v0.59) and NativeBase (v2.12.1). For navigation in application, we have used React Navigation (v3.8.1).

What is React Native

React Native is an app development framework developed by Facebook which can create iOS and Android native apps with single source code. Ideally, iOS apps are coded in Objective-C / swift and Android apps are coded in Java / Kotlin. React Native removes the double language barrier and allows creation of both iOS and android apps with Javascript source code.

How can I use this template

This template is a ready-made front-end template for a **Music Streaming App**. You can use the screens coded in this app, navigation between the screens and specific functions applied to buttons etc. This saves your 100% cost and time in front-end UI development. You can simply add a back-end / server to this app template, and replace the demo data with data coming from your server. This way, you only spend time and money on back-end development.

2. Installation

Setting Up Environment and Installing Dependencies

Refer this [Getting Started](#) guide in React Native Documentation to set up your development environment as per your system (Mac / Windows / Linux). Make sure you are referring to Building Projects with **React Native CLI** guide (and not Expo guide).

Ideally, there are three parts of Setup

- Install node
- Install React-Native CLI
- Install supporting SDK for Android and/or iOS.

For iOS, XCode is required to be installed, and iOS apps can only be made on a Mac. For Android, a recent JAVA SDK also needs to be installed, along with Android Studio.

Installing Required Node Modules

Unzip the source code you received with this product. Open terminal in the project directory and run following command.

```
$ npm install
```

This will install all the dependencies of the project. You will see a node modules_ folder in your project root once this step is complete.

Linking Dependencies to Platforms

To integrate already implemented module into your application or module which completes your react native's module functionality, run following command in your project directory.

```
$ react-native link
```

Re-generating Platform folders

By default, the source code contains ios and android platform folders with their respective files. If you ever had to delete your platform folders, you can run following command inside your directory to add android and ios platform again

```
$ react-native eject
```

3. Running your app

React Native apps are built and served on local network for testing. That means, you will need an emulator in your PC / Mac or a device attached to your system to run the app.

Note: The build system for the app should have latest XCode and Latest Android Studio (SDK) for correct compilation of apps.

For iOS, execute following command.

```
$ react-native run-ios
```

For android, execute following command.

```
$ react-native run-android
```

Please note, in both iOS and Android build cases, the React Native files are packaged by a packager/bundler named **Metro Bundler**. It starts automatically with the above commands in a separate terminal window, and looks like the following

A screenshot of a terminal window with a dark background and light gray text. The text shows the Metro Bundler starting on port 8081, providing instructions to keep it running, and a link to the GitHub repository. Below this, it shows the bundler looking for JS files in a specific directory and loading the dependency graph.

```
Running Metro Bundler on port 8081.
```

```
Keep Metro running while developing on any JS projects. Feel free to  
close this tab and run your own Metro instance if you prefer.
```

```
https://github.com/facebook/react-native
```

```
Looking for JS files in  
  /Users/abhijeetrathore/Audrix-new
```

```
Loading dependency graph, done.
```



If the metro bundler does not run automatically, open terminal in your project root and run

```
$ react-native start
```

This will start the Metro bundler fresh.

Note: Don't run two Metro bundlers at the same time

4. Run your app with changes

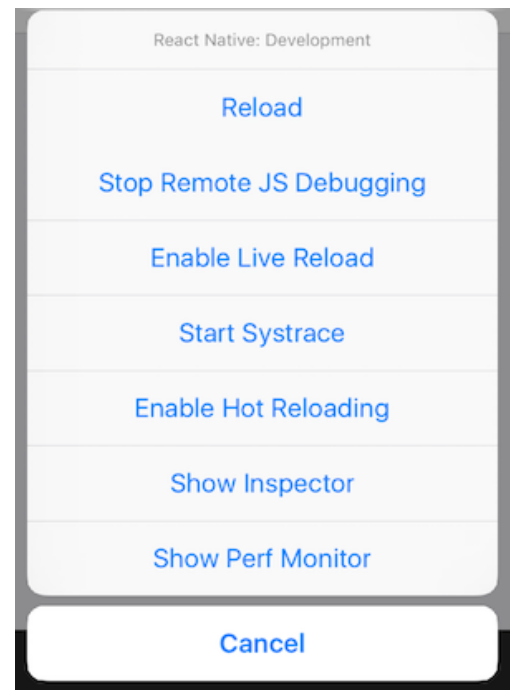
Modifying App Code

The app is laid down in a very standard fashion. Each page's code is written in a separate folder with 3 files in it, mostly, as shown in the image below.

- JSX file representing the UI part (Similar to HTML)
- JSX file representing the logic part (Similar to JS)
- JSX file representing the Style part (Similar to CSS)

Once your app is running on a device or Emulator, you can modify the code in any of these files and rebuild the application by

- If running on device - Shake the device. The option menu appears. Choose Rebuild/ Reload
- If running on Emulator - Press ctrl + R (Windows) / CMD + R (Mac), OR press ctrl + D (Windows) / CMD + D (Mac) - The option menu appears. Choose Rebuild/Reload
- Note: In either of the above, open the option menu and select Hot reloading and the app will reload on each save after any change. Please note, Hot reloading sometimes does not reflect major changes. Hence, reloading with above two steps is better in those cases.



Deleting a page from the app

You can delete any page you want from the app. Make sure you remove all the reference of that pages from any other page. For this, you can do a global search with that page's name or local url, and delete all references found.

Changing App Name

To change the app name, open **app.json** and replace value of **displayName** key with your required application name.

Example: { ... "displayName": "Audrix" }

Changing Package Name

Package name is an app's identifier for app stores. It is usually in the format *com.something.appname*

For iOS

Change the Bundle ID in Xcode

For Android

Refer this [answer](#) from StackOverflow.

5. Creating Icon and Splash screen

Adding App Icon

We recommend using [Android Asset Studio](#) for generating icon for your application. >Note: Do not change the name `ic_launcher` as it is the default icon file name.

After this, follow the steps provided in this [guide](#), for both Android and iOS platform.

Adding Splash Screen

We recommend using [ApeTools](#) to generate your splash screen for any platform. This step requires overwhelming number of steps, so kindly refer this [article](#) carefully.

6. Generating Release builds

For Apple App Store

Refer this [guide](#) by Christian Engvall.

For Android Play Store

Refer this [guide](#) from React Native docs.

Modifying code

1. Screen / View's Code Structure

Overall Folder Structure

There are broadly 20+ screens in this template. All screens are shown in Section 2 above.

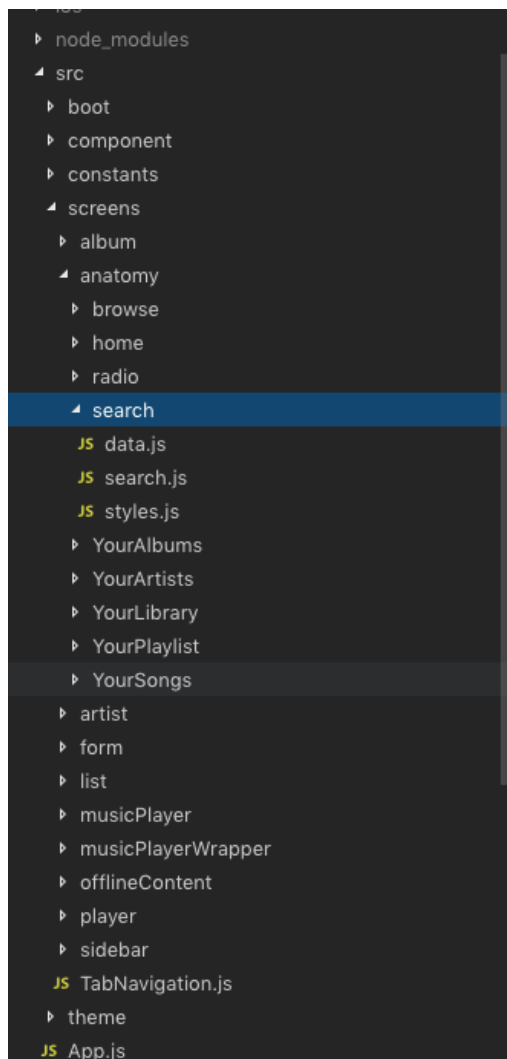
Mostly, each screen's code is kept in a separate folder. Folder names are self-explanatory.

Major screens are inside **Anatomy** Folder,

Most screen folders contain 3 files

- **A screen name JS file** - Contains the layout of the screen
- **Styles.js file** - contains the styling of that screen
- **Data.js file** - Contains the dummy data used in that screen in JSON format

This modular way of coding enables you to add / remove any screen without affecting other screens.



Layout File Code Structure

The layout file has the structure shown in the image.

- **imports** are written on the top
- If the page has any logics associated with navigation or button clicks, that comes next
- Final portion written inside **render** is the actually layout code.

Note: If the logic part becomes very large when you add back-end to the app, it is advisable to take the logic part out of layout file, and into a new file. You can then import the logic in layout file as data is imported in the following screenshot.



```
1 |  
2 | import React, { Component } from 'react';  
3 | import {Container,Content,Text,Input,Icon,Footer,ListItem,Left,Body,Right,Thumbnail} from 'native-base';  
4 | import { View, FlatList, TouchableOpacity } from 'react-native';  
5 | import styles from './styles';  
6 | import MusicPlayerWrapper from '../musicPlayerWrapper/index';  
7 | import Data from '../constants/constantData.js';  
8 | import searchData from './data.js';  
9 |  
10 | const { player } = Data;  
11 | const songs = searchData.data;  
12 |  
13 | class Search extends Component {  
14 |   constructor(props) {  
15 |     super(props);  
16 |     this.state = {  
17 |       searchText: ''  
18 |     };  
19 |   }  
20 |  
21 |   search = text => {  
22 |     this.setState({  
23 |       searchText: text  
24 |     });  
25 |   };  
26 |  
27 |   render() {  
28 |     return (  
29 |       <Container style={styles.container}>  
30 |         <Content>=  
93 |         <TouchableOpacity=  
117 |       </Container>  
118 |     );  
119 |   }  
120 | }  
121 | export default Search;  
122 |
```

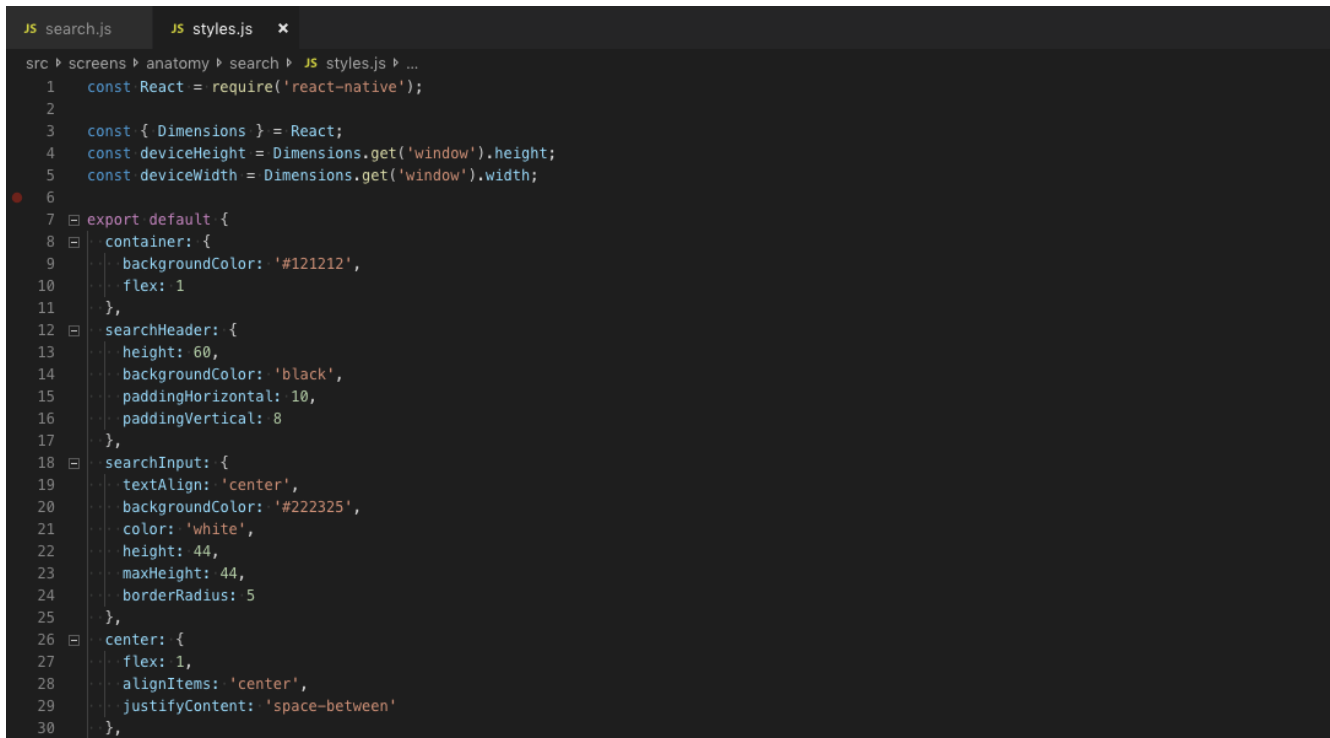
Search View Code

Imports

Page logic and functions

Page layout

2. Styling Code Structure

A screenshot of a code editor with a dark theme. The editor shows a file named 'styles.js' with the following code:

```
1  const React = require('react-native');
2
3  const { Dimensions } = React;
4  const deviceHeight = Dimensions.get('window').height;
5  const deviceWidth = Dimensions.get('window').width;
6
7  export default {
8    container: {
9      backgroundColor: '#121212',
10     flex: 1
11   },
12   searchHeader: {
13     height: 60,
14     backgroundColor: 'black',
15     paddingHorizontal: 10,
16     paddingVertical: 8
17   },
18   searchInput: {
19     textAlign: 'center',
20     backgroundColor: '#222325',
21     color: 'white',
22     height: 44,
23     maxHeight: 44,
24     borderRadius: 5
25   },
26   center: {
27     flex: 1,
28     alignItems: 'center',
29     justifyContent: 'space-between'
30   },
31 }
```

Styles are written separately for each screen, inside the screen folder.

Styles are then imported in the screen layout file using code similar to following

```
import styles from './styles';
```

so all styles are imported into a var **styles**.

As you can see in the screenshot, the styling syntax is very similar to CSS, but not exactly CSS

3. Logic Code Structure

As shown here, the logic part is written in the layout file itself. If the logic part becomes very large when you add back-end to the app, it is advisable to take the logic part out of layout file, and into a new file. You can then import the logic in layout file as data is imported in the following screenshot.



The screenshot displays a code editor with the title "Search View Code". The code is as follows:

```
1 |
2 | import React, { Component } from 'react';
3 | import { Container, Content, Text, Input, Icon, Footer, ListItem, Left, Body, Right, Thumbnail } from 'native-base';
4 | import { View, FlatList, TouchableOpacity } from 'react-native';
5 | import styles from './styles';
6 | import MusicPlayerWrapper from '../../musicPlayerWrapper/index';
7 | import Data from '../../constants/constantData.js';
8 | import searchData from './data.js';
9 |
10 | const { player } = Data;
11 | const songs = searchData.data;
12 |
13 | class Search extends Component {
14 |   constructor(props) {
15 |     super(props);
16 |     this.state = {
17 |       searchText: ''
18 |     };
19 |   }
20 |
21 |   search = text => {
22 |     this.setState({
23 |       searchText: text
24 |     });
25 |   };
26 |
27 |   render() {
28 |     return (
29 |       <Container style={styles.container}>
30 |         <Content>...
93 |         <TouchableOpacity~
117 |       </Container>
118 |     );
119 |   }
120 | }
121 | export default Search;
122 |
```

Annotations in the image include:

- A bracket on the right side of lines 2 through 8, labeled "Imports".
- A bracket on the right side of lines 13 through 25, labeled "Page logic and functions".
- A bracket on the right side of lines 27 through 119, labeled "Page layout".

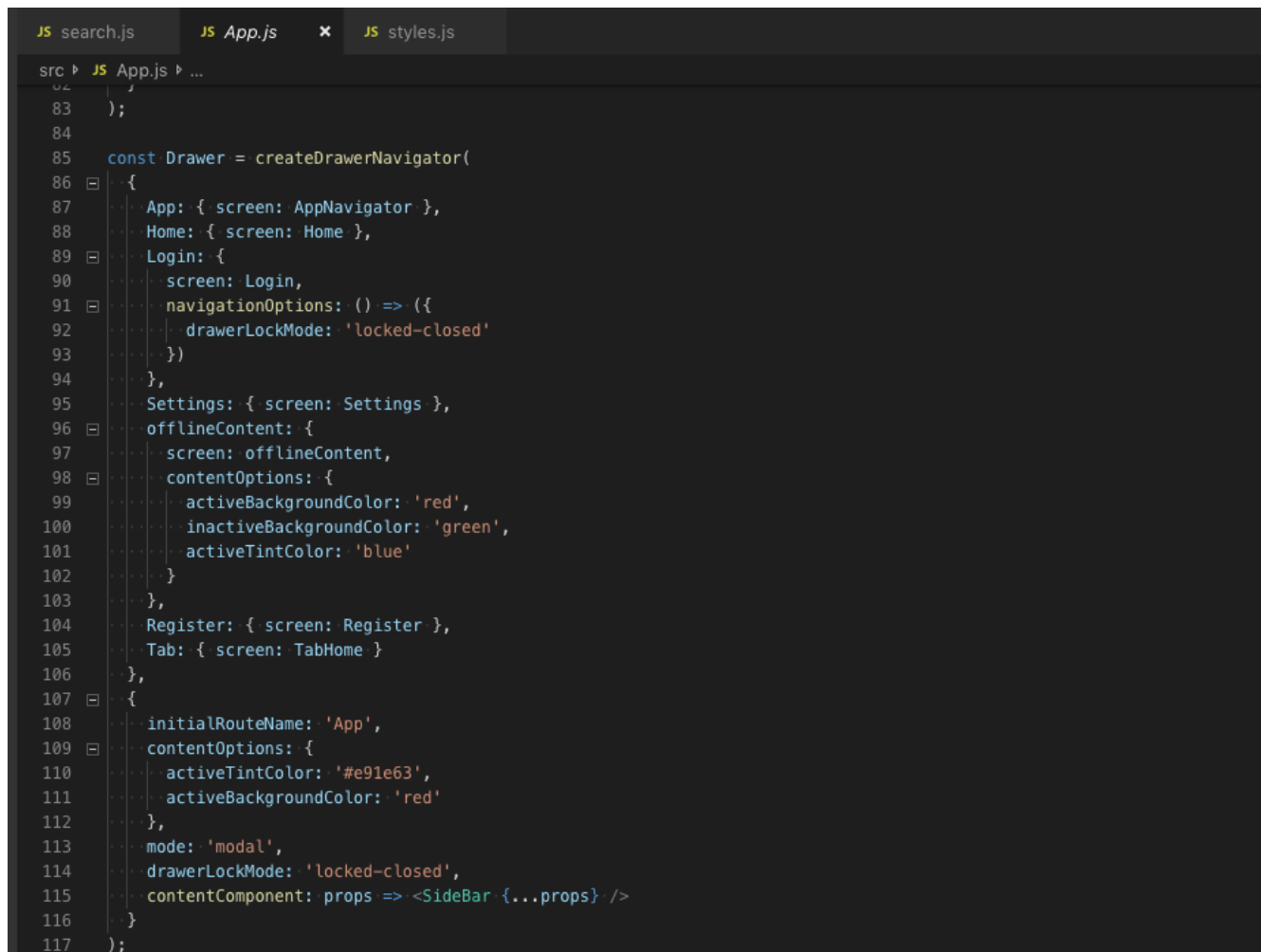
4. Navigation Code Structure

The app contains a variety of navigation - Drawer, nested tabs and stack navigation

Drawer navigation

Controls the navigation of Parent drawer (Sidemenu). This is detailed in **App.js** at the root of source (in **src** folder)

All the settings of Drawer Navigation can be controlled from here.



```
JS search.js JS App.js x JS styles.js
src ▸ JS App.js ▸ ...
82  };
83  );
84
85  const Drawer = createDrawerNavigator(
86    {
87      App: { screen: AppNavigator },
88      Home: { screen: Home },
89      Login: {
90        screen: Login,
91        navigationOptions: () => ({
92          drawerLockMode: 'locked-closed'
93        })
94      },
95      Settings: { screen: Settings },
96      offlineContent: {
97        screen: offlineContent,
98        contentOptions: {
99          activeBackgroundColor: 'red',
100          inactiveBackgroundColor: 'green',
101          activeTintColor: 'blue'
102        }
103      },
104      Register: { screen: Register },
105      Tab: { screen: TabHome }
106    },
107    {
108      initialRouteName: 'App',
109      contentOptions: {
110        activeTintColor: '#e91e63',
111        activeBackgroundColor: 'red'
112      },
113      mode: 'modal',
114      drawerLockMode: 'locked-closed',
115      contentComponent: props => <SideBar {...props} />
116    }
117  );
```

Tab navigation - Parent

Controls the navigation of Parent Tabs (5 tabs in HomePage). This is detailed in *src/screens/TabNavigation.js*

All the settings of Parent Tab Navigation can be controlled from here.

Tab navigation - Child

Controls the navigation of Child Tabs (4 tabs in Library Page). This is detailed in `src/screens/anatomy/YourLibrary/yourLibrary.js`

All the settings of Child Tab Navigation can be controlled from here.

Stack Navigator

Many screens are simply navigated via a simple Stack Navigator. The details are in `src/App.js`



```
32
33  const AppNavigator = createStackNavigator(
34    [{
35      Register: {
36        screen: Register,
37        navigationOptions: () => ({
38          drawerLockMode: 'locked-closed'
39        })
40      },
41      TabHome: { screen: TabHome },
42      Album: { screen: Album },
43      offlineContent: { screen: offlineContent },
44      changePassword: { screen: changePassword },
45      Player: { screen: Player },
46      Login: {
47        screen: Login,
48        navigationOptions: () => ({
49          drawerLockMode: 'locked-closed'
50        })
51      },
52      Artist: { screen: Artist }
53    ],
```

Feature Explanation

1. User Authentication

User authentication is often the first and most important part of the app.

This template has screens for Login, Signup, Forgot password and Change password.

8:01 ← Login	8:01 ← Sign Up	2:03 × Forgot Password	8:07 × Change Password
 Audrix Username or Email Password Sign In Forgot Password? Not Registered? Register here.	 Audrix Name Email Password Confirm Password Register Already a user? Login here.	OTP Verify	Current Password New Password Confirm Password Change Password

As per your requirements you can either

- Allow a user to access the app without login, and ask a login for specific features, OR
- Do not allow any access without login. In this case the Login page will go on top of the *StackNavigator* in *src/App.js*. Only once there is a "login", the user should be allowed access to HomePage.

Authentication can be achieved by your custom back-end or, if you are using Firebase, there are several easy authentication options. You can even choose to integrate social logins in the app. Check details [here](#).

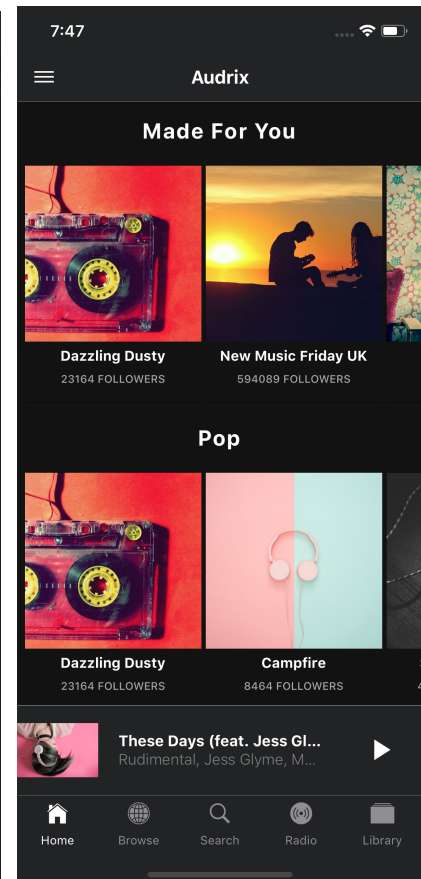
2. Adding songs, playlists, artists etc.

This template takes the dummy data from the JSON files placed in each screen's folder. If you only want to create static data, you can modify the JSON files' data and the data in the page/layout will change.

For example, in Homepage - we see different categories of media.

These categories are imported from the JSON data in the **albums.js** file in **src/screens/anatomy/home** folder

```
JS albums.js x
src > screens > anatomy > home > JS albums.js
1 export default (albumData = {
2   albums: [
3     {
4       name: "New Music Friday UK",
5       followers: "594089 followers",
6       cover: require("../assets/albumCover2.jpg")
7     },
8     {
9       name: "The Pop List",
10      followers: "556245 followers",
11      cover: require("../assets/albumCover1.jpg")
12     },
13     {
14       name: "Summer Splash",
15       followers: "464646 followers",
16       cover: require("../assets/albumCover4.jpg")
17     },
18     {
19       name: "Pop Toppers",
20       followers: "48446 followers",
21       cover: require("../assets/albumCover7.jpg")
22     },
23     {
24       name: "Dazzling Dusty",
25       followers: "23164 followers",
26       cover: require("../assets/albumCover3.jpg")
27     },
28     {
29       name: "Indie Selects",
30       followers: "656564 followers",
31       cover: require("../assets/albumCover6.jpg")
32     },
33     {
34       name: "Campfire",
35       followers: "8464 followers",
36       cover: require("../assets/albumCover5.jpg")
37     }
38   ]
39 });
40
```



Editing this data will edit the view shown in the right hand side.

Once you connect your back-end with the app, this JSON data will be replaced by the back-end data. All the data labels are self-explanatory.

Images

You can choose to load images from URLs (recommended), or keep images in the local storage. If you are loading images from URL, keep a backup image to show in case the actual image takes time to load.

3. Recommendations

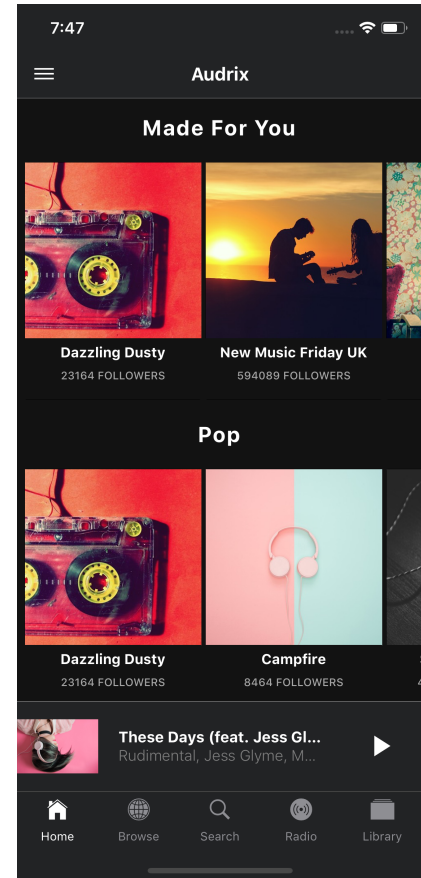
In any modern music streaming app, you generally see that the app recommends music for you. This is generally shown on the HomePage itself.

This template also provides you a similar theme. You can place the recommended music on the HomePage.

Currently this data is coming from a JSON file in the source code. Once you connect your back-end to the app, you can run some algorithms for recommendations in your back-end, and supply the result in JSON format to the app. The app will then display the recommendations in the HomePage.

There are two ways to get access to a recommender engine.

1. **Build it yourself** - This often involves doing extensive research on the several method, learning a new programming language (e.g. Neo4J, etc.) and implementing and hosting this engine (monthly fees can be quite high)
2. Tap into Recommender Algorithms as a Service libraries such as the [Abracadabra Recommender API](#). The setup is very straightforward: you only need to send HTTP calls to the API to train your model and to receive recommendations. [View the docs.](#)



4. Search

Search is undoubtedly the most important feature in a music streaming app.

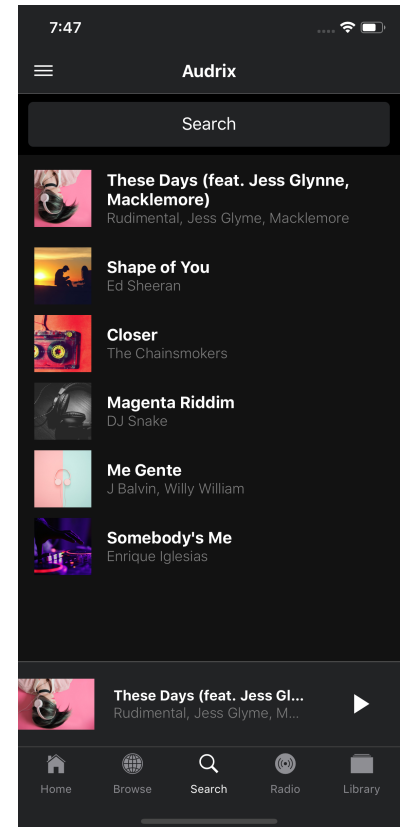
In this template, you can implement full fledged search in the Search Tab

Currently, the Search Tab is only dummy functional.

Once you implement a Search API on the “submit” function of the search input box, it should return a result with the searched data. The app can then display the data as shown here on the right

You can use custom function in your back-end to search through your data, OR use a third-party service like [Algolia](#) to provide efficient search functionality in your app.

To speed up your search results, you might want to fetch only 10-20 search results at a time. You can then implement an [infinite scroll](#) feature to fetch more search results as the user scrolls.



5. Radio

A music streaming app also contains Radio options for seamless listening. This template includes a dedicated tab for Radio stations.

Currently the data is coming from the dummy JSON file in the source folder. But you can actually integrate several Internet Radio APIs without even a need for back-end.

Here are the list of api which will gives your radio stations that can be listen through internet.

Programmableweb

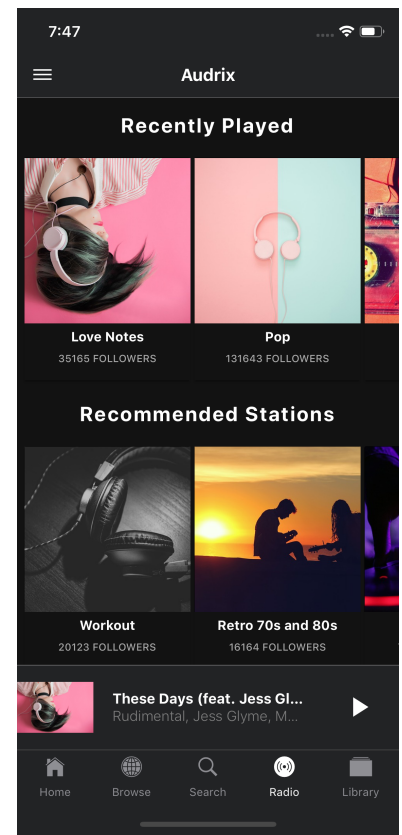
<http://www.programmableweb.com/news/50000-radio-stations-one-api/2012/01/26>

Dribble

<https://dribble.com/developer/api>

Shoutcast

http://wiki.shoutcast.com/wiki/SHOUTcast_Radio_Directory_API



6. Local Data storage

Once your app is fully-functional, you will need to store many types of data

- A user's preferences
 - A user's profile info
 - Current session's data
 - Previous session's data
 - User's playback history
 - User's payment history (if you implement payments)
- etc.

React Native provides methods to persist data. Some are provided out of the box while others are libraries you have to install and use. Below are popular methods of persisting data in React Native

1. **AsyncStorage** - This is the most recommended way to persist data in your React Native applications as it is already built into React Native.
2. **React Native SQLite 2** - This is a plugin which provides a WebSQL API to persist data. It provides SQL-like syntax, for querying your in-app persisted database.

It has support for both Android, iOS, and windows.

3. **Realm** - It's designed as an object-oriented database and this makes it up to 10x faster than SQLite. It's simple to use since it exposes data as objects and it supports relationships among data, mapping classes, tables, foreign keys and so on.

Realm can be combined with server-side databases to allow seamless synchronization of data offline to the cloud / server database. Realm is a fantastic choice if you'll be dealing with large data in your application.

4. **React Native Local MongoDB** - MongoDB is an open-source server-side database built for scalability and complex applications and Big data. It uses key-value stores and a relational database to store data as objects in JSON documents.

MongoDB can read and write JavaScript objects and allows smooth communication between the server and app. It's a right choice if you'll be dealing with large data in your application.

However, MongoDB can be slow for connected models that require joins.

7. Offline Data

Music streaming apps often provide Offline data feature for Premium customers.

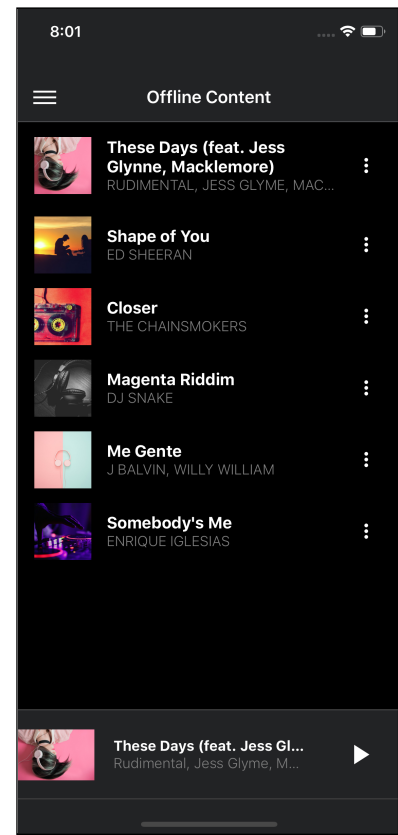
This template also has a page for showing and playing offline data.

You can implement offline data using Authentication methods and encryptions.

Since this is Offline data, you will have to store it in the device storage. But you don't want people to steal the songs once their membership is over or when they uninstall the app. In simple words, you would want that the music you are storing on local device storage, should be readable only by your app.

For this purpose, you will need to implement encryption on the audio files.

You can implement encryption using [react-native-simple-encryption](#) library



8. Music Player

This is the central component of the a music streaming app.

This template has two types of music player - a Mini player at the bottom of each screen and a full-screen player.

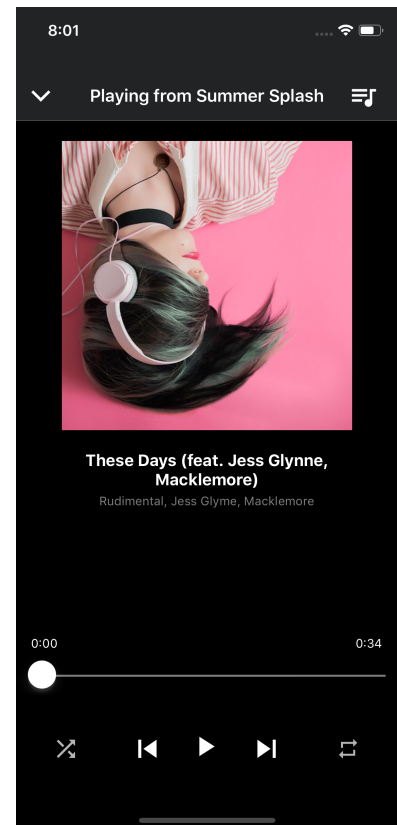
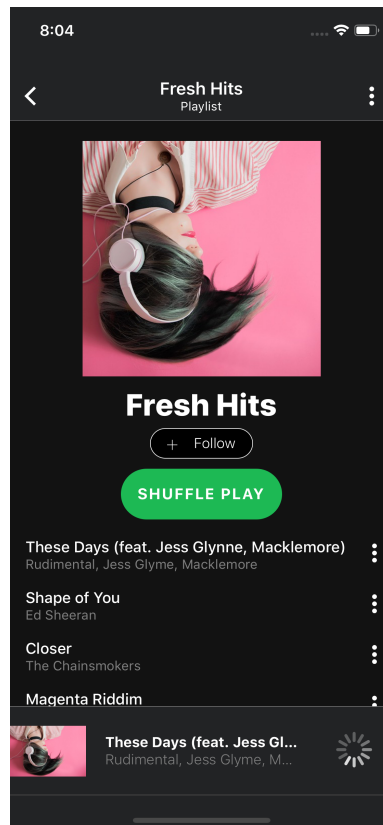
The app uses **React-native-video** library for playing audio. This may seem like a weird choice but it isn't. **React-native-video** is meant mostly for video playback, but as it turns out, a lot of people use it just to control sound.

This project has a bigger community behind it, and there are some functionalities that are not yet available or are not well supported in react-native-sound, like playing sounds in the background or access to events like `onProgress`, `onLoadStart`, `onLoad`, `onPlay` and others.

In this case, authors wanted to create something that is a bit more similar to HTML5 video specification. It works perfectly good with audio as well, since those two APIs are similar by design.

You can also use other libraries like

- [react-native-track-player](#)
- [React-native-sound](#)



9. Settings

Settings are an important part of an app.

This template also has a dedicated and relevant Settings page.

There are options for

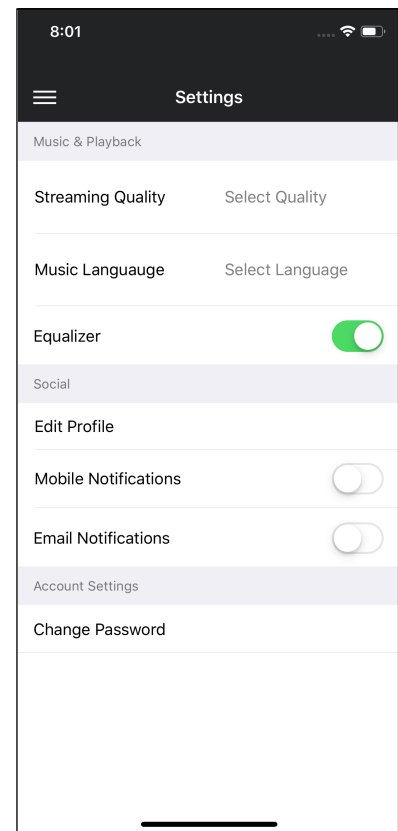
- Changing Quality from dropdown
- Changing Language from dropdown
- Edit Profile
- Change Password
- Toggles for other options

To change quality, you can alter a parameter in your API calls. This should then Fetch a different sound file chunk from your server. This will, of course, require you to setup up a streaming server with Adaptive streaming enabled. You can read more about [Adaptive bitrate streaming here](#).

To change language of the app, you can use libraries like [react-native-internationalization](#). Follow [this blog post](#) to get started in multi-language.

To enable and disable mobile notifications, you will first need a Push notifications system in your back-end. If you are using Firebase - implementing, enabling and disabling push-notification is very easy compared to a custom back-end. Read more about Firebase- React Native push notification [here](#).

Edit Profile and **Change Password** can also be implemented very easily with Firebase, even if you are using Social logins. Custom back-end will require you to make custom APIs for these purposes.



10. Nested Navigation

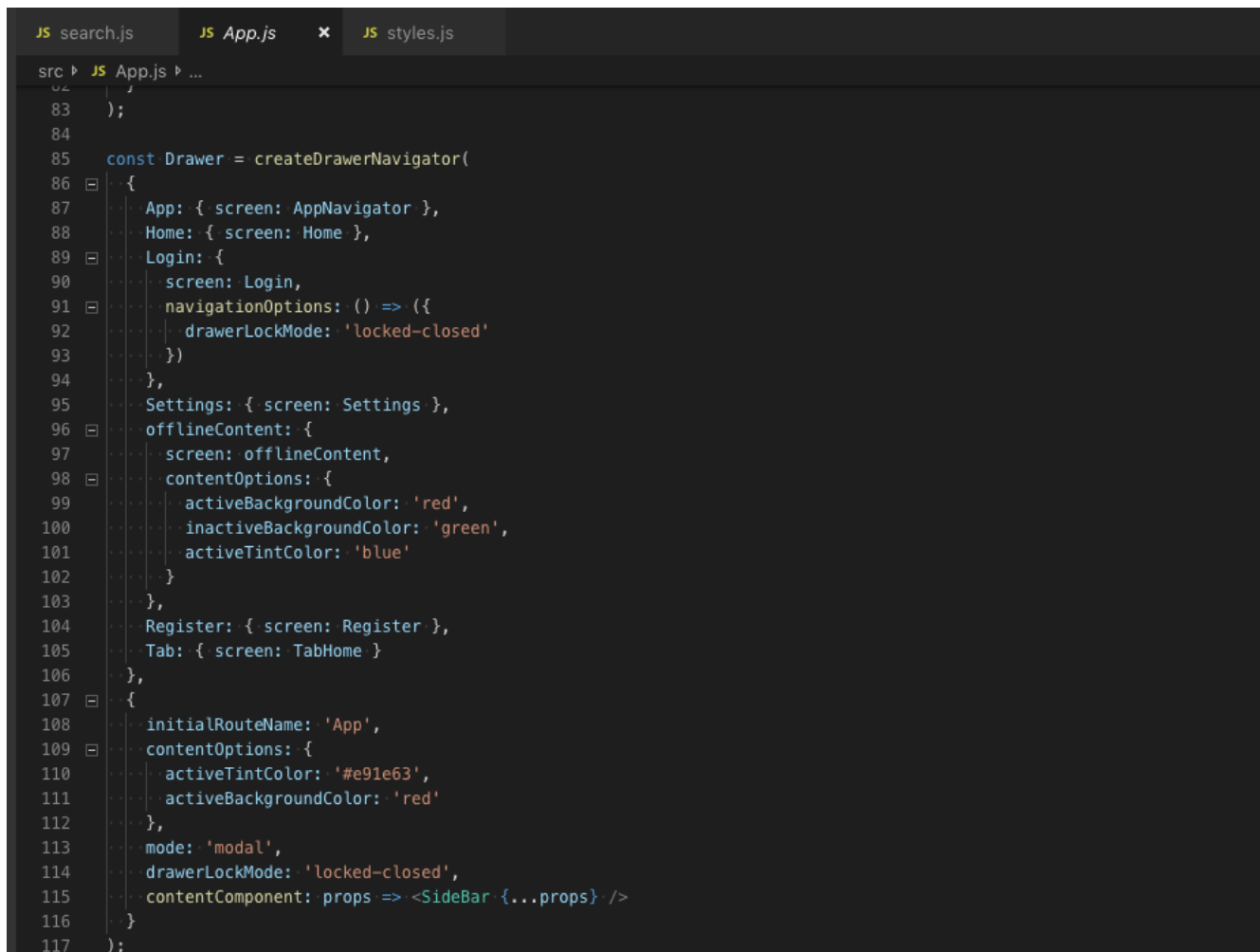
(This section is a duplicate of a previous section)

The app contains a variety of navigation - Drawer, nested tabs and stack navigation. Having a nested Drawer -> Tab -> Tab is pretty awesome !

Drawer navigation

Controls the navigation of Parent drawer (Sidemenu). This is detailed in **App.js** at the root of source (in **src** folder)

All the settings of Drawer Navigation can be controlled from here.



```
JS search.js JS App.js x JS styles.js
src ▸ JS App.js ▸ ...
82
83   });
84
85   const Drawer = createDrawerNavigator(
86     {
87     App: { screen: AppNavigator },
88     Home: { screen: Home },
89     Login: {
90       screen: Login,
91       navigationOptions: () => ({
92         drawerLockMode: 'locked-closed'
93       })
94     },
95     Settings: { screen: Settings },
96     offlineContent: {
97       screen: offlineContent,
98       contentOptions: {
99         activeBackgroundColor: 'red',
100        inactiveBackgroundColor: 'green',
101        activeTintColor: 'blue'
102      }
103    },
104    Register: { screen: Register },
105    Tab: { screen: TabHome }
106  },
107  {
108    initialRouteName: 'App',
109    contentOptions: {
110      activeTintColor: '#e91e63',
111      activeBackgroundColor: 'red'
112    },
113    mode: 'modal',
114    drawerLockMode: 'locked-closed',
115    contentComponent: props => <SideBar {...props} />
116  }
117  );
```

Tab navigation - Parent

Controls the navigation of Parent Tabs (5 tabs in HomePage). This is detailed in *src/screens/TabNavigation.js*

All the settings of Parent Tab Navigation can be controlled from here.

Tab navigation - Child

Controls the navigation of Child Tabs (4 tabs in Library Page). This is detailed in `src/screens/anatomy/YourLibrary/yourLibrary.js`

All the settings of Child Tab Navigation can be controlled from here.

Stack Navigator

Many screens are simply navigated via a simple Stack Navigator. The details are in `src/App.js`

```
32
33 const AppNavigator = createStackNavigator(
34   [{
35     Register: {
36       screen: Register,
37       navigationOptions: () => ({
38         drawerLockMode: 'locked-closed'
39       })
40     },
41     TabHome: { screen: TabHome },
42     Album: { screen: Album },
43     offlineContent: { screen: offlineContent },
44     changePassword: { screen: changePassword },
45     Player: { screen: Player },
46     Login: {
47       screen: Login,
48       navigationOptions: () => ({
49         drawerLockMode: 'locked-closed'
50       })
51     },
52     Artist: { screen: Artist }
53   ]),
```

11. Premium feature addition

Providing premium features is where you can earn using your app.

This template does not have a premium feature page as such. But you can easily integrate it by copying one of the pages like **Change Password**

You can then route each your music streaming request via a checking function which checks for user being premium or non-premium. Accordingly, you can

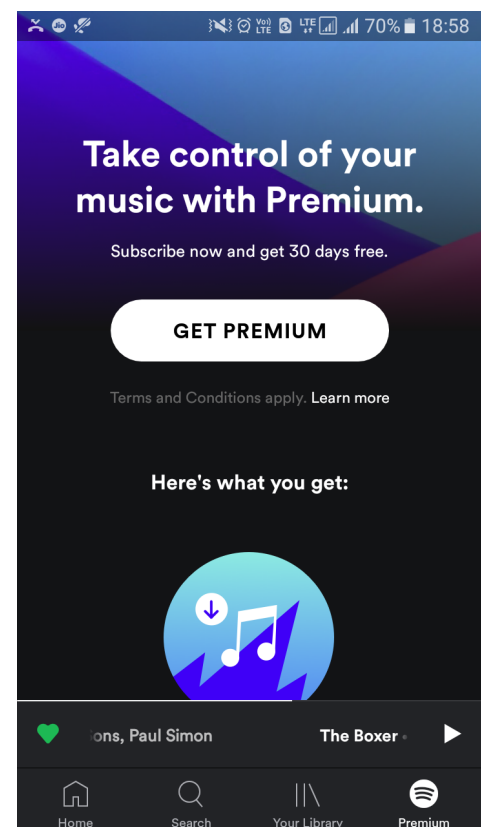
- Serve low quality data to non-premium users, ask them to get premium for better quality
- Don't allow Offline content for non-premium users, ask them to get premium for Offline content
- Don't allow creating more than a fixed number of playlists etc in non-premium
- Show Ads using [react-native-admob](#) for non-premium users

and many similar things.

You can implement the payments by using any one or multiple of the following

- [React Native PayPal](#)
- [React Native Stripe](#)
- [React Native in-app purchase](#)
- [React Native Apple Pay and Android Pay](#)
- [React Native Razorpay](#)

You can check how Spotify asks for Premium membership in its app in the screenshot shown here

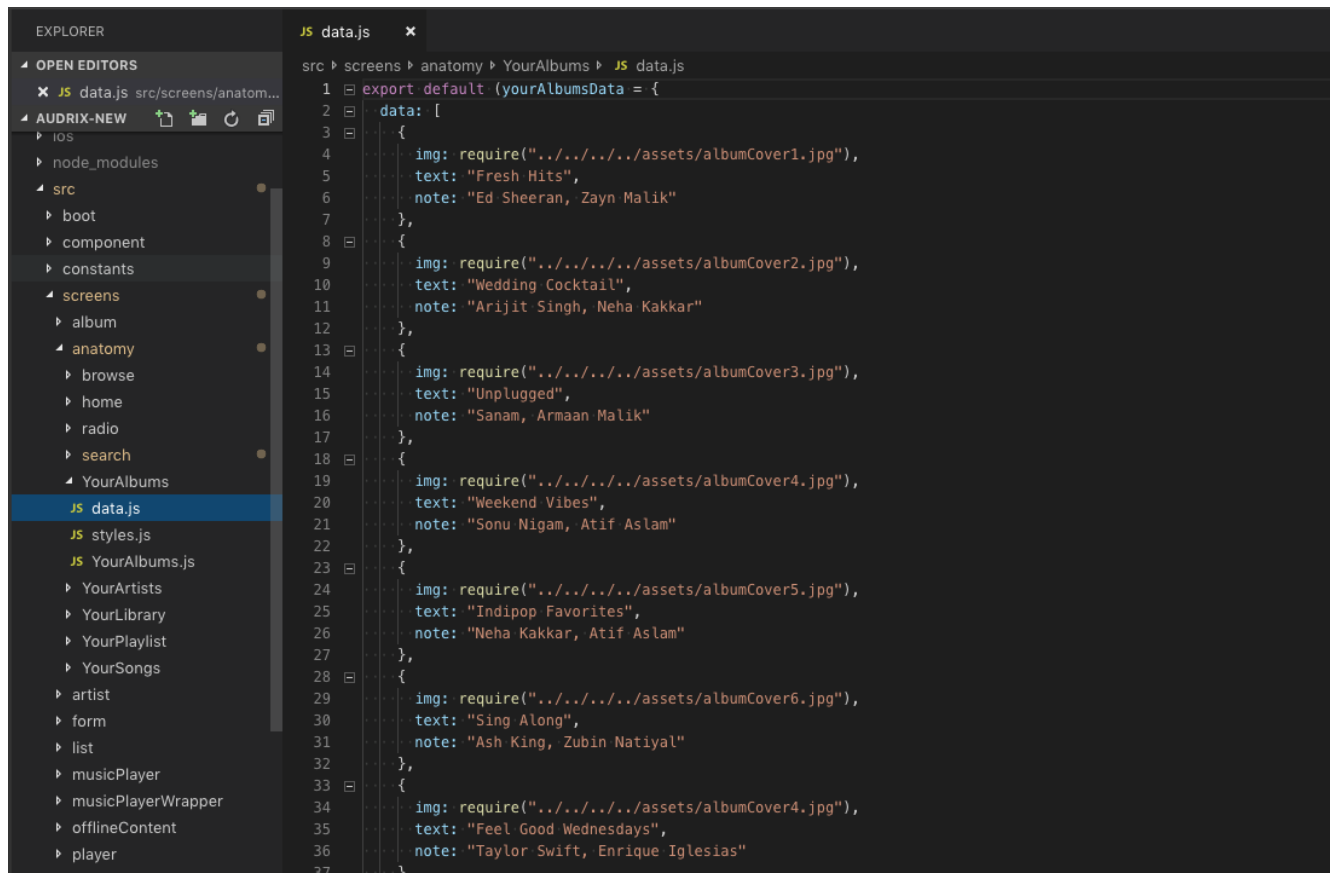


Integrating Back-end Data

1. Data Structure

In each screen's folder, there is a dummy data file placed to mimic the behavior of a back-end.

E.g. Following images shows the data and its structure in **YourAlbums** page. The data is in JSON format.



Data is imported in the layout file as

```
const { player } = Data;
```

This way, all the data is imported inside **player** variable.

This JSON data file can be easily replaced with data coming in from back-end APIs. The data can then be directly connected to the imported variable.

2. Back-end options

To make the app fully operational you will need to attach a back-end to the app. This will require several additions and you will need the assistance of a back-end developer.

We will provide a glimpse into the things you will require

Step 1 - Choose a back-end Technology

There are several options when it comes to back-end. Some of the major ones are

1. Node.js custom back-end
2. **Firebase back-end (node.js)**
3. Django custom back-end
4. Ruby-on-rails custom back-end
5. GO back-end

We recommend starting out with Firebase, if you want quick turn-around of features. Firebase is a BaaS platform and comes with a bunch of ready-made functionality, like

- Storage
- Analytics
- Push notifications
- In-app messages
- Real-time database

and much more.

Step 2 - Connect the back-end to the app

As explained in the section above, you will need to replace the Data coming in form individual JSON files with the data coming in from APIs (from your back-end)

To manage the data efficiently, you will need to arrange all API calls, and data manipulation in a single location. You can manage API calls with [Fetch](#).

You can make use of [Redux](#) package to maintain the functions and data manipulation in a single location. Redux can then be integrated in each screen file so that the data changes reflect in real-time in all the screens. Check this introductory post on [connecting React-Native apps to Redux](#)

Troubleshooting

1. Attribute `application@allowBackup value=(false)` from `AndroidManifest.xml: 11:7-34`

Go to android Folder `android > app > src > AndroidManifest.xml` and search `allowBackup` and make it true `allowBackup="true"`

Example:

```
<application
  android:name=".MainApplication"
  android:label="@string/app_name"
  android:icon="@mipmap/ic_launcher"
  android:allowBackup="true"
  android:theme="@style/AppTheme">
```

2. Android connection error

After successful app build, if the app doesn't connect with device and shows following error

```
[DeviceMonitor]: Unable to open connection to: localhost/xx.xxx.xxx.xx:5037, due to:
java.net.ConnectException: Connection refused: connect [DeviceMonitor]: Connection
attempts: 1 [DeviceMonitor]: Unable to open connection to: localhost/xx.xxx.xxx.xx:5037,
due to: java.net.ConnectException: Connection refused: connect [DeviceMonitor]:
Connection attempts: 2
```

Run this command in your project library to force java to use IPV4 adress by adding a new environment variable :

```
$ _JAVA_OPTIONS=-Djava.net.preferIPv4Stack=true
```

3. Entry, “:CFBundleIdentifier”, Does Not Exist

If so,

Option 1 – Change Xcode to Legacy Build System. More details [here](#)

Option 2 – Open your iOS product in Xcode, give it a proper bundle name like `com.enappd.audrix`, provide a correct provisioning profile, and you should be able to build it properly. After this, even the command `react-native run-android` should build it properly.

4. Android emulator does not run the app

– Run `react-native run-android`. This should run the metro bundler, and open default simulator to run the app. If the metro bundler does not run automatic, go in your project folder, and run `react-native start`. This should start the bundler.

– If you have a device connected, and you face a connection issue via USB, try running `adb reverse tcp:8081 tcp:8081`

And then again run the `react-native run-android` command. Your app should run properly on device.

If not, open the project in Android Studio and run in the device.

Refer [this troubleshooting guide](#) from React Native docs to solve some common issues.

Third-party libraries, sources and Credits

Major third-party libraries used in this app template are

- "color": "^3.1.0"
- "localforage": "^1.7.3",
- "lodash": "^4.17.11",
- "moment": "^2.24.0",
- "native-base": "^2.12.1",
- "prop-types": "^15.7.2",
- "react": "16.8.3",
- "react-native": "0.59.4",
- "react-native-color": "0.0.10",
- "react-native-deck-swiper": "^1.6.7",
- "react-native-gesture-handler": "^1.1.0",
- "react-native-slider": "^0.11.0",
- "react-native-vector-icons": "^6.4.2",
- "react-native-video": "^4.4.1",
- "react-native-view-overflow": "0.0.4",
- "react-navigation": "^3.8.1"

All these are open-source libraries and the license & usage permission of the above are available on their respective URLs.

Assets images are used from unsplash.com. The usage of these images is free.

The demo music file is downloaded from assets-music.icons8.com , which is free for use.

Once again, thank you so much for purchasing this theme. As mentioned at the beginning, we'd be glad to help you if you have any questions relating to this theme. No guarantees, but we'll do our best to assist. Feel free to email via our user page contact form [here](#).

If you have a more general question relating to the themes on ThemeForest, you might consider visiting the forums and asking your question in the "Item Discussion" section.