Objective
Jammming

Jammming

Jammming
Connected to Codecademy
In this project, you will build a React web application called Jammming. You will use your knowledge of React components, passing state, and requests with the Spotify API to build a website that allows users to search the Spotify library, create a custom playlist, then save it to their Spotify account.

We've broken the Jammming project into 13 sections, called *assessments*. Each assessment contains a descriptive header with an introductory step, followed by a set of steps that guide you to the outcome. The first step of each assessment will explain the assessment's goal and provide a brief overview of how we'll accomplish it. Before you start the second step, try to plan how you would complete the assessment. As you finish the rest of the steps, reflect on how your solution compares to ours.

Drag the edges to resize the window.
Jammming
99/99 Complete
Up Next
Get Help

Although the project may seem daunting, we'll be with you every step of the way. Whether you're completing assessments without the additional steps or banging your head against the wall trying to understand a hint, always use best practices and reflect on your growth.

Good luck!

Tasks
99/99 Complete
Mark the tasks as complete by checking them off
Create a React Application
1.

By the end of this assessment, you will be ready to start building your website. This section walks you through the process of setting up the directory structure and adding CSS presets.

To achieve this, you will create a boilerplate react app, remove unused files, and add **reset.css**, Google font links, and an updated favicon.

- Google fonts — Poppins and Work Sans
- Updated favicon

2.

Create a new React application in a directory called **Jammming**.

Stuck? Get a hint

Use `create-react-app` with the name of the folder you want to create.

3.

In **index.html**, update the `<title>` value to `Jammming`.

4.

Remove **App.test.js** and **logo.svg** from the **src/** folder, as you will not use them in this project.

5.

Add reset.css to the **public/** directory and link to it in **index.html**.

6.

Link to the following Google fonts in **index.html**:

- Poppins
- Work Sans

Stuck? Get a hint

Link the Google fonts using the `<link>` tag.

```
<link href="https://fonts.googleapis.com/css?family=Poppins:600" rel="stylesheet">
<link href="https://fonts.googleapis.com/css?family=Work+Sans:300,500" rel="stylesheet">
```

7.

Update **favicon.ico** with this image.

Create Static Components
8.

In this assessment, you will create a JavaScript file and a CSS file for each of six components in the Jammming app. In the steps below, we will link to the raw HTML and CSS to help you write the JSX for each component.

In the HTML, we use comments to indicate where the JSX for one component renders another component.

The HTML and CSS for each of the six components are listed below:

- **App** — HTML and CSS
- **SearchBar** — HTML and CSS
- **SearchResults** — HTML and CSS
- **Playlist** — HTML and CSS
- **TrackList** — HTML and CSS
- **Track** — HTML and CSS

9.

Create a **src/Components** directory to hold the components.

10.

Create a directory called **App/** in the **Components/** directory.

Add **App.js** and **App.css** to the **App/** folder and update the path in **index.js** accordingly.

Additionally add this background image to the directory as well — it is used by the CSS file.

11.

Inside of the **App.js** `.render()` method, add a return statement with JSX that renders this HTML .

Follow the guidelines below when you write the HTML (linked above) as JSX:

- Change all class attributes to className.
- Do not change the class values, as we will use them in a later step to add style.

12.

Use the comments in the HTML document from the last step to determine the components you need to import into **App.js**.

Note, you will create a folder for each component. The JavaScript file and CSS files for each component will live in the component's folder. The folder, JavaScript file, and CSS file will all have the same name.

Stuck? Get a hint

The comments indicate you need to import the following three components:

- **Playlist**
- **SearchBar**
- **SearchResults**

13.

Add this CSS to the **App.css** file.

Import **App.css** into **App.js**.

14.

Create a **SearchBar/** directory in the **Components/** directory.

Inside of **SearchBar/**, add **SearchBar.js** and **SearchBar.css**.

15.

Inside of **SearchBar.js** create a component called `SearchBar` with a `.render()` method that returns this HTML.

Follow the guidelines below when you write the HTML (linked above) as JSX:

- Change all class attributes to className.
- Do not change the class values, as we will use them in a later step to add style.

Use the comments in the HTML document to determine if you need to import any components.

Export the `SearchBar` component.

Stuck? Get a hint

Follow these steps to create the `SearchBar` component:

- Import `React`
- Create a `SearchBar` class that extends `React.Component`
- Create a render method that returns the HTML linked above
- You do not need to import any other components
- Export `SearchBar`

16.

Add this CSS to the **SearchBar.css** file.

Import **SearchBar.css** into **SearchBar.js**.

17.

Create a **SearchResults/** directory in the **Components/** directory.

Inside of **SearchResults/**, add **SearchResults.js** and **SearchResults.css**.

18.

Inside of **SearchResults.js** create a component called `SearchResults` with a `.render()` method that returns this HTML.

Follow the guidelines below when you write the HTML (linked above) as JSX:

- Change all class attributes to className.

- Do not change the class values, as we will use them in a later step to add style.

Use the comments in the HTML document to determine if you need to import any components.

Export the `SearchResults` component.

Stuck? Get a hint

Follow these steps to create the `SearchResults` component:

- Import `React`
- Create a `SearchResults` class that extends `React.Component`
- Create a render method that returns the HTML linked above
- Import `TrackList`
- Export `SearchResults`

19.

Add this CSS to the **SearchResults.css** file.

Import **SearchResults.css** into **SearchResults.js**.

20.

Create a **Playlist/** directory in the **Components/** directory.

Inside of **Playlist/**, add **Playlist.js** and **Playlist.css**.

21.

Inside of **Playlist.js** create a component called `Playlist` with a `.render()` method that returns this HTML.

Follow the guidelines below when you write the HTML (linked above) as JSX:

- Change all class attributes to `className`.
- Do not change the class values, as we will use them in a later step to add style.
- Change the `value` property to `defaultValue` and set it equal to `{'New Playlist'}`
- Comment out `<TrackList />` since it doesn't work without any props.

Use the comments in the HTML document to determine if you need to import any components.

Export the `Playlist` component.

Stuck? Get a hint

Follow these steps to create the `Playlist` component:

- Import `React`
- Create a `Playlist` class that extends `React.Component`
- Create a render method that returns the HTML linked above
- Import `TrackList`
- Export `Playlist`

22.

Add this CSS to the **Playlist.css** file.

Import **Playlist.css** into **Playlist.js**.

23.

Create a **TrackList/** directory in the **Components/** directory.

Inside of **TrackList/**, add **TrackList.js** and **TrackList.css**.

24.

Inside of **TrackList.js** create a component called `TrackList` with a `.render()` method that returns this HTML.

Follow the guidelines below when you write the HTML (linked above) as JSX:

- Change all class attributes to `className`.
- Do not modify the class values, as we will use them in a later step to add style.
- For now, you will hard code three tracks. In a later assessment, we will replace the hard-coded values with tracks from Spotify.

Use the comments in the HTML document to determine if you need to import any components.

Export the `TrackList` component.

Stuck? Get a hint

Follow these steps to create the `TrackList` component:

- Import `React`
- Create a `TrackList` class that extends `React.Component`
- Create a render method that returns the HTML linked above
- Import `Track`
- Export `TrackList`

25.

Add this CSS to the **TrackList.css** file.

Import **TrackList.css** into **TrackList.js**.

26.

Create a **Track/** directory in the **Components/** directory.

Inside of **Track/**, add **Track.js** and **Track.css**.

27.

Inside of **Track.js** create a component called `Track` with a `.render()` method that returns this HTML.

Follow the guidelines below when you write the HTML (linked above) as JSX:

- Change all class attributes to `className`.
- Do not change the class values, as we will use them in a later step to add style.
- Create a method called `renderAction` that displays a `-` anchor tag if the `isRemoval` property is true, and a `+` anchor tag if the `isRemoval` property is false. Set the class name to `Track-action`.

Use the comments in the HTML document to determine if you need to import any components.

Export the `Track` component.

Stuck? Get a hint

Follow these steps to create the `Track` component:

- Import `React`

- Create a `Track` class that extends `React.Component`
- Create a render method that returns the HTML linked above
- You do not need to import any other components
- Export `Track`

28.

Add this CSS to the **Track.css** file.

Import **Track.css** into **Track.js**.

Pass Down Search Result and Render Result List
29.

In this assessment, you will pass the state of a search results parameter through a series of components to render an array of tracks.

When a user requests data from Spotify, the JSON response will include a set of song tracks. Each `track` will contain a field for `name`, `artist`, and `album`. For each track in the results list, your Jammming web app will display the song name, artist, and album.

In a later assessment, you will build a method that sets the state of the search results parameter to a response from the Spotify API.

30.

Add a constructor function to the `App` component, and pull in `props` from the `React.Component` class.

Stuck? Get a hint

Inside of the `App` component in **App.js**, create a constructor method.

Pass `props` to the constructor function and use `super()` to call the parent constructor.

31.

Inside of the App component, set a hard-coded initial value for `this.state.searchResults`.

Stuck? Get a hint

Inside of the `App` constructor, set `this.state` to an object with a property called `searchResults` set to an array of objects, each containing `name`, `artist`, and `album` properties.

32.

Pass the state of the `App` component's `searchResults` to the `SearchResults` component.

Stuck? Get a hint

Pass `this.state.searchResults` to the `SearchResults` component as an attribute called `searchResults`.

33.

Pass the search results from the `SearchResults` component to the `TrackList` component.

Stuck? Get a hint

Inside the **SearchResults.js** `.render()` method, pass `this.props.searchResults` as an attribute called `tracks` in the `TrackList` component.

34.

In the `TrackList` component, use the `.map()` method to render each track in the `tracks` property.

Set the `key` attribute to `track.id`.

Stuck? Get a hint

Inside the `TrackList` component's render method, use `.map()` on `this.props.tracks` to render each track in the list.

This will require you to pass the current `track` as an attribute called `track` to the `Track` component.

35.

Render the track name, artist, and album.

Stuck? Get a hint

Use the following property calls to access the track's name, artist, and album:

- `this.props.track.name`
- `this.props.track.artist`
- `this.props.track.album`

Pass down Playlist to TrackList
36.

In this assessment, you will pass the state of a user's custom playlist title and tracks from the App component down to components that render them.

When a user adds songs from the search results list to their playlist, a method will update the state of a playlist parameter in **App.js**, and Jammming will render the song in the user's playlist.

In a later assessment, you will write methods that add and remove songs from the playlist. You will also write a method that updates the playlist's title.

37.

Add hard-coded values for `playlistName` and `playlistTracks` to state in **App.js**.

Stuck? Get a hint

You can set `playlistName` to any string.

The `playlistTracks` value should be an array of objects, each containing `name`, `artist`, and `album` properties.

38.

Pass the playlist name and tracks from the `App` component to the `Playlist` component.

Stuck? Get a hint

Inside the **App.js** `.render()` method, pass `this.state.playlistName` and `this.state.playlistTracks` as attributes called `playlistName` and `playlistTracks` in the `Playlist` component.

39.

Pass the playlist tracks from the `Playlist` component to the `TrackList` component.

Stuck? Get a hint

Inside the **Playlist.js** `.render()` method, pass `this.props.playlistTracks` as an attribute called `tracks` in the `TrackList` component.

Add Tracks to a Playlist
40.

In this assessment, you will implement a process for adding a song from the search results track list to the user's custom playlist.

You will add a method to **App.js** called `addTrack` that adds a song to the playlist state. The application passes the method through a series of components to `Track`. The user can trigger the `.addTrack()` method by clicking the `+` sign from the search results list.

41.

In **App.js** create a method called `addTrack` with the following functionality:

- Accepts a `track` argument
- Use the track's `id` property to check if the current song is in the `playlistTracks` state.
- If the `id` is new, add the song to the end of the playlist.
- Set the new state of the playlist

42.

Bind the current value of `this` to `.addTrack()`.

Pass `.addTrack()` to the `SearchResults` component as an `onAdd` attribute.

43.

Pass `onAdd` from the `SearchResults` component to the `TrackList` component.

Stuck? Get a hint

Inside the **SearchResults.js** `.render()` method, pass `this.props.onAdd` as an attribute called `onAdd` to the `TrackList` component.

44.

Pass `onAdd` from the `TrackList` component to the `Track` component.

Stuck? Get a hint

Inside the **TrackList.js** `.render()` method, pass `this.props.onAdd` as an attribute called `onAdd` to the `Track` component.

45.

Create an `.addTrack()` method in the `Track` component. Use it to add `this.props.track` to the playlist.

Stuck? Get a hint

Pass `this.props.track` to `this.props.onAdd`.

46.

Add a constructor to the `Track` component. Call `super(props)` in the constructor method.

Bind `this.addTrack()` to the current value of `this` in the constructor method.

47.

In the **Track.js** `+` element, add an `onClick` property with the value set to `this.addTrack`.

Remove Tracks from a Playlist
48.

In this assessment, you will implement a process that removes a song from a user's custom playlist when the user selects the – sign inside of a rendered track.

49.

In **App.js** create a method called `removeTrack` with the following functionality:

- Accepts a track argument
- Uses the track's `id` property to filter it out of `playlistTracks`
- Sets the new state of the playlist

50.

In the `App` constructor method, bind the current value of `this` to `.removeTrack()`.

Pass `.removeTrack()` to the `Playlist` component as an `onRemove` attribute.

51.

Pass `onRemove` from the `Playlist` component to the `TrackList` component.

Stuck? Get a hint

Inside the **Playlist.js** `.render()` method, pass `this.props.onRemove` as an attribute called `onRemove` in the `TrackList` component.

52.

Pass `onRemove` from the `TrackList` component to the `Track` component.

Stuck? Get a hint

Inside the **TrackList.js** `.render()` method, pass `this.props.onRemove` as an attribute called `onRemove` in the `Track` component.

53.

Create a `.removeTrack()` method in the `Track` component. Use it to remove `this.props.track` from the playlist.

Stuck? Get a hint

Pass `this.props.track` to `this.props.onRemove`.

54.

In **Track.js**, bind `this.removeTrack()` to the current value of `this` in the constructor method.

55.

In the **Track.js** – element, add an `onClick` property with the value set to the `this.removeTrack` method.

Change the Name of a Playlist
56.

In this assessment, you will implement code that allows a learner to change the name of their playlist, and save the updated value to the `App` component's state.

57.

In **App.js** create a method called `updatePlaylistName` with the following functionality:

- Accepts a name argument

- Sets the state of the playlist name to the input argument

58.

In the `App` constructor method, bind `this` to `.updatePlaylistName()`.

Pass `updatePlaylistName` to the `Playlist` component as an attribute named `onNameChange`.

59.

In the `Playlist` component, create a method called `handleNameChange`.

The method should accept an event that is triggered by an `onChange` attribute in the `Playlist` component's `<input>` element.

Inside the method, call `.onNameChange()` with the event target's value (from the `<input>` element).

60.

Add a constructor to the `Playlist` component. Call `super(props)` in the constructor method.

Bind the current value of `this` to `.onNameChange()`.

61.

In the `Playlist` render method, pass `.handleNameChange()` to an `onChange` property.

Create a Method that Saves the Playlist to a User's Account
62.

In this assessment, you will create a method that will save a user's playlist to their Spotify account and resets the state of the playlist name and tracks array.

To accomplish the goal of this assessment, you will need to access a `track` property named uri. Spotify uses this field to reference tracks in the Spotify library. You will create an array containing the `uri` of each track in the `playlistTracks` property.

In a later assessment, you will pass the playlist name and the array of `uris` to a Spotify-linked method that writes the tracks in `playlistTracks` to a user's account.

63.

In **App.js** create a method called `savePlaylist` with the following functionality:

- Generates an array of `uri` values called `trackURIs` from the `playlistTracks` property.
- In a later step, you will pass the `trackURIs` array and `playlistName` to a method that will save the user's playlist to their account.

64.

Bind the current value of `this` to `.savePlaylist()`.

Pass `savePlaylist` to the `Playlist` component as an attribute called `onSave`.

65.

In the **Playlist.js** `SAVE TO SPOTIFY` element, add an `onClick` property with the value set to `this.props.onSave`.

Hook up Search Bar to Spotify Search

66.

In this assessment, you will create a method that updates the `searchResults` parameter in the `App` component with a user's search results. You will write the logic that allows a user to enter a search parameter, receives a response from the Spotify API, and updates the `searchResults` state with the results from a Spotify request.

In a later assessment, you will hook the `.search()` method up to the Spotify API.

67.

In **App.js** create a method called `search` with the following functionality:

- Accepts a search term
- Logs the term to the console

In a later assessment, we will hook this method up to the Spotify API.

68.

In the `App` constructor method, bind `this` to `.search()`. In a later assessment, we will use `this` in `.search()`.

Pass `.search()` to the `SearchBar` component as an `onSearch` attribute.

69.

In **SearchBar.js**, create a method called `search` that passes the state of the term to `this.props.onSearch`.

70.

In the `SearchBar` component, create a contstructor method with a call to `super(props)`.

Inside of the constructor, bind the current value of `this` to `.search()`.

71.

In **SearchBar.js** create a method called `handleTermChange` with the following functionality:

- Accepts an event argument
- Sets the state of the search bar's term to the event target's value.

72.

In the **SearchBar.js** constructor method, bind the current value of `this` to `this.handleTermChange`.

73.

In the search bar's `<input>` element, add an `onChange` attribute and set it equal to `this.handleTermChange`.

Obtain a Spotify Access Token
74.

In the next few assessments, you will write three methods that accomplish the following:

- Get a Spotify user's access token
- Send a search request to the Spotify API
- Save a user's playlist to their Spotify account.

Before you begin, you will need to create an empty JavaScript module called `Spotify` located in **src/util/Spotify.js**.

In this assessment, you will register a Spotify application and create a method called `getAccessToken` in the `Spotify` module. The method will get a user's access token so that they can make requests to the Spotify API.

Use the Spotify Applications Registration Flow and Spotify Authentication guide to help you write the method.

75.

Create a **src/util** directory and add a file called **Spotify.js**

76.

In **Spotify.js** create a `Spotify` module as an empty object.

At the bottom of **Spotify.js** export `Spotify`.

77.

Above the empty object, declare an empty variable that will hold the user's access token.

78.

Inside the `Spotify` module, create a method called `getAccessToken`.

Check if the user's access token is already set. If it is, return the value saved to access token.

79.

If the access token is not already set, check the URL to see if it has just been obtained.

You will be using the Implicit Grant Flow to setup a user's account and make requests. The implicit grant flow returns a user's access token in the URL.

Use the guide to determine how to parse the URL and set values for your access token and expiration time.

Look at the hint if you help parsing the URL.

Stuck? Get a hint

In the implicit grant flow, values for the access token and expiration time are in the URL parameter after authentication.

Use `window.location.href` and the `.match()` method to retrieve the access token and expiration time from the URL.

Example URL from Spotify API:

```
https://example.com/callback#access_token=NwAExz...BV3O2Tk&token_type=Bearer&expires_in=3600&state=123
```

Use the `.match()` method on the URL string. Provide the regular expressions below as inputs:

```
/access_token=([^&]*)/
/expires_in=([^&]*)/
```

80.

If the access token and expiration time are in the URL, implement the following steps:

- Set the access token value
- Set a variable for expiration time
- Set the access token to expire at the value for expiration time

- Clear the parameters from the URL, so the app doesn't try grabbing the access token after it has expired

The hint below contains the code that wipes the access token and URL parameters.

Stuck? Get a hint

Use the following code to help you wipe the access token and URL parameters

```
window.setTimeout(() => accessToken = '', expiresIn * 1000);
window.history.pushState('Access Token', null, '/');
```

81.

The third condition is that the access token variable is empty and is not in the URL.

Before you write this conditional code block, you need to register your application using the Spotify application registration flow.

Give your application a relevant name and description. Also, add the following Redirect URI:

```
http://localhost:3000/
```

82.

At the top of **Spotify.js** create constant variables for your application's client ID and redirect URI.

Set the client ID variable to the value provided on your application page.

Set the redirect URI to `"http://localhost:3000/"`.

83.

Back in your conditional statement, redirect users to the following URL:

```
https://accounts.spotify.com/authorize?client_id=CLIENT_ID&response_type=token&scope=playlist-modify-public&redirect_uri=REDIRECT_URI
```

Interpolate your client ID and redirect URI variables In place of `CLIENT_ID` and `REDIRECT_URI`.

Stuck? Get a hint

To redirect a user, you must set `window.location` to the URL in the task above.

Implement Spotify Search Request
84.

In this assessment, you will create a method in **Spotify.js** that accepts a search term input, passes the search term value to a Spotify request, then returns the response as a list of tracks in JSON format.

You will need the user's access token to make requests to the Spotify API. You will use the request parameters in step four of the implicit grant flow to make requests. In the following steps, we will use `fetch()` to make our requests, but any method will work.

You should use the `/v1/search?type=TRACK` endpoint when making your request. Use the Spofity Web API Endpoint Reference to help format your request.

85.

In the `Spotify` object, add a method called `search` that accepts a parameter for the user's search term.

`.search()` returns a promise that will eventually resolve to the list of tracks from the search.

86.

Inside `.search()`, start the promise chain by returning a GET request (using `fetch()`) to the following Spotify endpoint:

```
https://api.spotify.com/v1/search?type=track&q=TERM
```

Replace the value of `TERM` with the value saved to the search term argument.

Add an Authorization header to the request containing the access token.

Stuck? Get a hint

You will need to pass a second argument to the `fetch` method. The second argument is an object with one field called `headers`. Set `headers` to an object with one `Authorization` property with the user's access token. Use the format in step four of the implicit grant flow.

Pass the following object as the second fetch parameter:

```
{
  headers: {Authorization: `Bearer ${accessToken}`}
}
```

87.

Convert the returned response to JSON.

Then, map the converted JSON to an array of tracks. If the JSON does not contain any tracks, return an empty array.

The mapped array should contain a list of track objects with the following properties:

- ID — returned as `track.id`
- Name — returned as `track.name`
- Artist — returned as `track.artists[0].name`
- Album — returned as `track.album.name`
- URI — returned as `track.uri`

88.

In **App.js**, import `Spotify` and update the `.search()` method with the `Spotify.search()` method.

Update the state of `searchResults` with the value resolved from `Spotify.search()`'s promise.

Save a User's Playlist
89.

In this assessment, you will create a method called `savePlaylist` that writes the learner's custom playlist in Jammming to their Spotify account.

The `.savePlaylist()` method accepts a playlist name and an array of track URIs. It makes the following three requests to the Spotify API:

- GET current user's ID
- POST a new playlist with the input name to the current user's Spotify account. Receive the playlist ID back from the request.

- POST the track URIs to the newly-created playlist, referencing the current user's account (ID) and the new playlist (ID)

You will update the `.savePlaylist()` method in **App.js** to use the new `Spotify.savePlaylist()` method.

90.

Create a method in **Spotify.js** that accepts two arguments. The first argument is the name of the playlist. The second is an array of track URIs.

Inside the function, check if there are values saved to the method's two arguments. If not, return.

91.

Create three default variables:

- An access token variable, set to the current user's access token
- A headers variable, set to an object with an `Authorization` parameter containing the user's access token in the implicit grant flow request format
- An empty variable for the user's ID

92.

Make a request that returns the user's Spotify username.

Convert the response to JSON and save the response `id` parameter to the user's ID variable.

Stuck? Get a hint

Make the request to the following Spotify endpoint:

```
https://api.spotify.com/v1/me
```

You must pass a second argument with an object containing the `headers` object. See below

```
{headers: headers}
```

93.

Use the returned user ID to make a POST request that creates a new playlist in the user's account and returns a playlist ID.

Use the Spotify playlist endpoints to find a request that creates a new playlist.

Set the playlist name to the value passed into the method.

Convert the response to JSON and save the response `id` parameter to a variable called `playlistID`.

Stuck? Get a hint

Make a request to the following Spotify endpoint:

```
/v1/users/{user_id}/playlists
```

You must pass a second argument that contains an object with parameters for `headers`, `method`, and `body`.

94.

Use the returned user ID to make a POST request that creates a new playlist in the user's account and

returns a playlist ID.

Use the Spotify playlist endpoints to find a request that adds tracks to a playlist.

Set the URIs parameter to an array of track URIs passed into the method.

Convert the response to JSON and save the response `id` parameter to a variable called `playlistID`.

Stuck? Get a hint

Make a request to the following Spotify endpoint:

```
/v1/users/{user_id}/playlists/{playlist_id}/tracks
```

You must pass a second argument that contains an object with parameters for `headers`, `method`, and `body`.

95.

In **App.js** update the `.savePlaylist()` method to call `Spotify.savePlaylist()`.

After you call `Spotify.savePlaylist()`, reset the state of `playlistName` to `'New Playlist'` and `searchResults` to an empty array.

Deploy (Optional)
96.

In this assessment, you will use surge to deploy your Jammming project.

You will start by installing surge globally.

In your console, run `npm install --global surge`.

97.

Before you deploy, you need to think of a domain name with the following format:

```
SOME_NAME.surge.sh
```

`SOME_NAME` can be replaced with anything you like.

Next, you need to replace or add this URI to two locations in your project.

- In *_Spotify.js_*, set `redirectUri` to your new domain
- In your Spotify application, add your new domain as a redirect URI

98.

Back in the command line, from the Jammming project's root directory, run:

```
$ npm run build
```

99.

`cd` into the `build` directory and run the command

```
$ surge
```

Follow the steps on the screen. Change the domain value to your new URI.

Congrats! You've just deployed a React App that queries the Spotify API!

Report a Bug