Technical Documentation

Introduction

This documentation is a piece of work designed solely for the purpose of a project for respective coursework. It elaborates the technical aspect and the know-how of the project from a first principle approach to a user who has absolutely limited knowledge about using APIs, integrated web-design, OAuth and its extended fundamentals being used in this project. This documentation is meant for a brief understanding for the user to understand, analyze, evaluate and implement this in future for their own benefits and their growth.

What is this Project?

This project evolves using Spotify's API to display a user's statistics based on their listening history with detailed overview of the artists, genres, playlists etc and hence proves to be a "Spotify Wrapped" but accessible anytime and anywhere. We further extended the scope of our project to use this website to add more features like a song recommendation generator (back then Spotify didn't have their Spotify AI suggestion beta bot). We use this using the same statistics which we have received and to come up with these suggestions.

To be added-

- **Why?**
- **Who is this documentation intended for?**
- **Division of codeflow**
- **Enriching Experience**
- **Further improvements and prospective possibilities**

Division of the codeflow-

- ***Generator***
  - *Generator.js*
  - *index.html*
  - *main.css*
- ***OAuth and API fetching***
  - *index.html*
  - *app.js*

1. Generator

Introduction to be added

Generator.js

This is a JavaScript module pattern that defines two modules: APIController and UIController.
The APIController module handles API interactions with the Spotify Web API. It has private methods for getting an access token, getting genres, getting playlists by genre, getting tracks for a playlist, and getting a single track. The public methods provide access to these private methods.
The UIController module handles the user interface. It has public methods for getting the input fields, creating genre and playlist

select list options, creating a track list group item, creating a song detail, resetting the track detail, tracks, and playlist, storing and getting the access token.
These two modules can be used together to create a web application that interacts with the Spotify Web API and displays the results to the user in a user-friendly way.

This is a JavaScript code for a website that uses Spotify's API. The code defines two modules: APIController and UIController.
APIController is an IIFE (Immediately Invoked Function Expression) that returns an object with five methods:

- getToken: a method that sends a POST request to Spotify's API to get an access token using the client_id and client_secret as authentication. The method returns a Promise that resolves to the access token.
- getGenres: a method that sends a GET request to Spotify's API to get the available music genres. The method receives an access token as a parameter and returns a Promise that resolves to an array of music genres.
- getPlaylistByGenre: a method that sends a GET request to Spotify's API to get playlists based on a music genre. The method receives an access token and a genre ID as parameters, and returns a Promise that resolves to an array of playlists.
- getTracks: a method that sends a GET request to Spotify's API to get a list of tracks for a given playlist. The method receives an access token and a tracks endpoint as parameters, and returns a Promise that resolves to an array of tracks.
- getTrack: a method that sends a GET request to Spotify's API to get a specific track. The method receives an access token and a track endpoint as parameters, and returns a Promise that resolves to the track data.

Index.html

This HTML code represents a web page that provides a user interface for a Spotify Web API application. Let's break down the code and explain each section:

- The HTML code represents a web page designed for a technical documentation of a Spotify Web API application.
- The page is divided into different sections, each serving a specific purpose.
- The code utilizes various HTML elements, attributes, and external resources to create a functional and visually appealing interface.

Document Structure and its purpose:

<!DOCTYPE html> declares the document type as HTML5.
<html lang="en"> indicates that the document's language is English.
The document is divided into two main sections: <head> and <body>.

Head Section:

Using <meta charset="UTF-8">, we specify the character encoding of the document as UTF-8.
We then set the viewport settings for responsive design and provide the title of the web page displayed in the browser tab.
<link rel="stylesheet" href="..."> includes an external CSS file from the Bootstrap framework to style the page.
<link rel="stylesheet" type="text/css" href="main.css"> references an additional CSS file named "main.css" for custom styles specific to this web page.

Body Section:

Using the , we wrap the entire content of the page within a container, providing a centered layout with responsive behavior.
We set a heading "Discover New Songs!" along with a star symbol and define a form for user input.
<input type="hidden" id='hidden_token'> represents a hidden input field used for storing a token.
<select> elements create dropdown menus for selecting a genre and playlist. Furthermore, we define a submit button for triggering a search action. We then have two containers to display the contents of the song and the details about the song respectively. This is achieved using the following lines -

-
-

Script Tags:

The provided script tags include various JavaScript libraries and custom script files necessary for the page's functionality and interaction.
These scripts handle actions such as interacting with the jQuery library, managing Bootstrap components, and executing custom code from "generator.js".

Main.css

The CSS code enclosed in main.css contains style rules for different elements used in the technical documentation web page. Here's an explanation of each rule and its purpose:

.btn.dropdown-toggle:

- Sets the width of the dropdown toggle button to 150 pixels.
- Aligns the text inside the button to the left.

span.caret:

- Positions the caret (downward arrow) used in dropdown buttons.
- Sets its position to be 90% from the left and 45% from the top of its containing element.

label:

- Adds padding of 5 pixels to the top of all label elements.

.form-label:

- Overrides the default padding on form labels, setting it to 0 pixels.

img:

- Sets the width and height of all img elements to 140 pixels, creating a square shape.

.btn-success:

- Modifies the background color of buttons with the class "btn-success" to a specific shade of green (#3b7d4a).
- We also ensure that this style is applied even if there are conflicting styles from other sources using the !important keyword.

.btn-success:hover:

- Changes the background color of the "btn-success" buttons when hovered to a lighter shade of green (#4daf64).

img.track:

- Adds a 1-pixel solid border around images with the class "track".

h2:

- Adds padding of 15 pixels to the top of all h2 elements.

2. OAuth and API Fetching

App.js

The provided JavaScript code sets up a server-side application using the Express framework and makes use of several libraries to implement authentication and authorization functionality for the Spotify Web API. Here's a detailed explanation of each part:

var express = require("express"); // Express web server framework: Imports the Express module to create a web server for handling HTTP requests.

var request = require("request"); // "Request" library: Imports the Request library, which simplifies making HTTP requests to external APIs.

var cors = require("cors"); : Imports the Cross-Origin Resource Sharing (CORS) library, which allows cross-origin requests to be handled.

var querystring = require("querystring");: Imports the Querystring module, which provides methods for working with query strings.

var cookieParser = require("cookie-parser"); : Imports the Cookie Parser library, which parses cookies attached to incoming requests.

var client_id = "d8ec3c80570b46ffb917d983f43ae9be";
var client_secret = "5626c023e88349ed90f0ff104798ad4b";*:* Defines the client ID for your Spotify Web API application and the client secret for your Spotify Web API application.

var redirect_uri = "http://localhost:8888/callback";: Defines the redirect URI where the Spotify API will send the authorization code after the user authenticates.

var generateRandomString = function(length) { ... }: Defines a function that generates a random string of a specified length. This function is used to generate a state value for security purposes during the authentication process.

var stateKey = "spotify_auth_state"; : Defines the key name for storing the state value in a cookie.

*var app = express(); :*Creates a new instance of the Express application.

app.use(express.static(__dirname + "/public")).use(cors()).use(cookieParser()); *:* Configures the Express application to serve static files from the "public" directory and adds middleware to enable CORS support and parse cookies in incoming requests.

app.get("/login", function(req, res) { ... });: Defines a route handler for the "/login" endpoint.Then it generates a random state value and stores it in a cookie. Following this, it redirects the user to the Spotify authorization page to grant permission to the application.

app.get("/callback", function(req, res) { ... }); : Defines a route handler for the "/callback" endpoint and handles the callback from the Spotify authorization page. Then, it verifies the state value to prevent cross-site request forgery attacks. It exchanges the authorization code for access and refresh tokens and uses the access token to make a request to the Spotify Web API to retrieve the user's profile information. Lastly, it redirects the user back to the client-side application with the access and refresh tokens as URL parameters.

app.get("/refresh_token", function(req, res) { ... });: Defines a route handler for the "/refresh_token" endpoint and retrieves the refresh token from the request From that, it uses the refresh token to request a new access token from the Spotify API.

console.log("Listening on 8888");: Outputs a message to the console indicating that the server is listening on port 8888.

app.listen(8888);: Starts the Express server and listens for the incoming requests on port 8888.