Design API structure

Status	Chosen for development
Assign	Ⓐ Aaron Li
Figma	
Completed Phases	

1. Introduction

Spotify has support for searching songs based on specific criteria, however, these features are not available for customers to use. The listening customer is immediately affected by the inability to manually filter songs. This doc will provide the structure on handle api calls. This will include client authorization with Spotify's API as well as reading and writing data.

2. Glossary

Client secret key	A piece of information that is used to encrypt and decrypt messages or data, and is kept secret to ensure that only authorized individuals can access the encrypted information.
*	This symbol means required

3. Problem

- Need a method to store users' secret key so that other users do not have access to it but we can still retrieve it for the user
- Functions need to be organized into files/folders

5. Solutions

5.1 Solution 1

This solution uses Firebase to help store user information. Client ID and client secret can be stored using enviornment variables (see Appendex 6.2 for pros and cons). organizes api calls into two main files (see Appendix 6.1). One for authentication and the other for making calls to the Spotify API. Most of these functions have already been generated in a previous project which will be recycled to be used in this project. Functions will be separated into 2 files. One for handling authentication (see Appendix 6.2) and one for handling Spotify API calls (see Appendix 6.2). Estimated 2-4 days to complete.

6. Appendix

6.1 Storing on Cloud

- https://www.youtube.com/watch?v=Pk5xgifoLYI&ab_channel=SmallBatchDevs

6.2 Pros and Cons of Enviornment Variables

Pros	Cons
Convenient and easy to manage, as environment variables can be set and changed outside of the application code.	Environment variables can be compromised if the server or deployment environment is compromised.
Compatible with a wide range of deployment environments, including containers, virtual machines, and serverless functions.	Environment variables may not provide sufficient security for highly sensitive information, such as cryptographic keys or personal identifiable information.
Can be used with many programming languages and frameworks, not just TypeScript and JavaScript.	Environment variables can be difficult to manage in complex or distributed environments, as they may need to be configured differently for each environment.
Can be encrypted or hashed to provide additional security.	Environment variables can be hard to troubleshoot if they are misconfigured or not set correctly.
Often included as a standard feature in many deployment tools and platforms.	

6.2 Authentication Functions

Function	Purpose
void login()	connect the user to our Spotify app to obtain credentials and permissions
void logout()	disconnect the user from Spotify
void exchangeToken(body)	retrieves access token using provided code from authentication
void handleExchangeToken()	stores token needed to get refresh token
void requestRefreshToken(body)	retrieves new token
void handleRefreshToken()	stores new token
boolean isLoggedIn()	allows us to know if a user is connected
generateRandomString(length)	generates a string to be used as state identifier
generateCodeVerifier(length)	generates a code verifer to be used to create a code challenge
generateCodeChallenge(codeVerifier)	generates a code challenge given a code verifier
base64URL(buffer)	converts to base64url format

6.3 Spotify API Functions

For reference: https://developer.spotify.com/documentation/web-api/reference/#/

All calls will be made to endpoint: https://api.spotify.com/v1/

Function	Purpose	Scope(s)	Parameters

Function	Purpose	Scope(s)	Parameters
void getSongs(params, callback, body)	retrieves song recommendations	none needed	/recommendations *seed_artists: String[] of artist ids *seed_tracks: String[] of track ids sum of lengths of arrays above ≤ 5 limit: Int of max tracks wanted following can have max, min, or target • acousticness 0≤ x ≤ 1 • danceability 0≤ x ≤ 1 • duration_ms • energy 0≤ x ≤ 1 • instrumentalness 0≤ x ≤ 1 • key 0≤ x ≤ 11 • liveness 0≤ x ≤ 1 • loudness 0≤ x ≤ 1 • mode 0≤ x ≤ 1 • popularity 0≤ x ≤ 100 • speechiness 0≤ x ≤ 1 • tempo 0≤ x ≤ 1 • time_signature • valence 0≤ x ≤ 1
void search(params, body)	search by keyword	none needed	/search *q: String query *type: String “artist” or “track” limit: Int of max tracks wanted
void getUserPlaylists(params)	retrieves current user’s playlists	playlist-read-private, playlist-read-collaborative	/me/playlists limit: Int of max tracks wanted offset: Int of offset playlist wanted
void addSong(id, params, body)	adds song(s) to a playlist	playlist-modify-public, playlist-modify-private	/playlists/{playlist_id}/tracks *uris: String of comma separated uris ≤ 100 OR in body: *uris: JSON String[] track uris
void removeSong(id, params, body)	removes song(s) from a playlist	playlist-modify-public, playlist-modify-private	/playlists/{playlist_id}/tracks in body: *uris: Object[] of String uri of track