Spotify Data Visualizer

A COE817 Final Project

Section 1: Introduction

The purpose of this project is to demonstrate OAuth 2.0 and how it can be used to safely access a user's data in different applications. Specifically, the program aims to interact with Spotify and is able to use OAuth to extract user data such as profile information, top tracks and even some recommended songs. The application does this without the knowledge of a user's credentials and links directly to Spotify, meaning that there is no risk in the credentials being known by systems other than Spotify.

OAuth stands for Open Authorization. OAuth can be employed in a multitude of different applications with the goal of safely accessing a user's data. To do this, it makes use of authorization tokens. These tokens can be granted to a user and can have different levels of scope, meaning that they'll be used to allow access to different pieces of user data depending on the scope of a given token. OAuth is not an API or a service, it is an open standard that anyone can implement. There are two versions of OAuth, OAuth 1.0 and OAuth 2.0, these two versions are not compatible with each other. In this project, OAuth 2.0 is used. A major advantage to OAuth is that it allows the app makers to offload security measures to the larger applications, in this case Spotify. Since the application never knows the users credentials, it allows the third-party application to rely on Spotify's strong security infrastructure to safely handle data.

The scope of the project comes down to a few aspects. The first being the UI of the application, giving the user the ability to interact with the application with buttons to grab their profile information, as well as the top and recommended tracks if they so please. The scope also includes the OAuth implementation and Spotify interaction. These two aspects go hand in hand. OAuth is used to authorize the project application to interact with the Spotify Web API and read user data based on the user authorized scope.

In regards to limitations, restraints can be categorized into time, resources, and experience. With regards to time, the implementation of this project and report needed to be completed in a small number of weeks. This led to many meetings at the beginning of the implementation phase of the project to be able to get an idea of what needed to be done as well as who was working on what. When it came to resources and experience, none of the members in the group had much experience with Javascript, HTML/CSS and OAuth to begin with, so time needed to be allocated to learn how all of them can be used together to create the application.

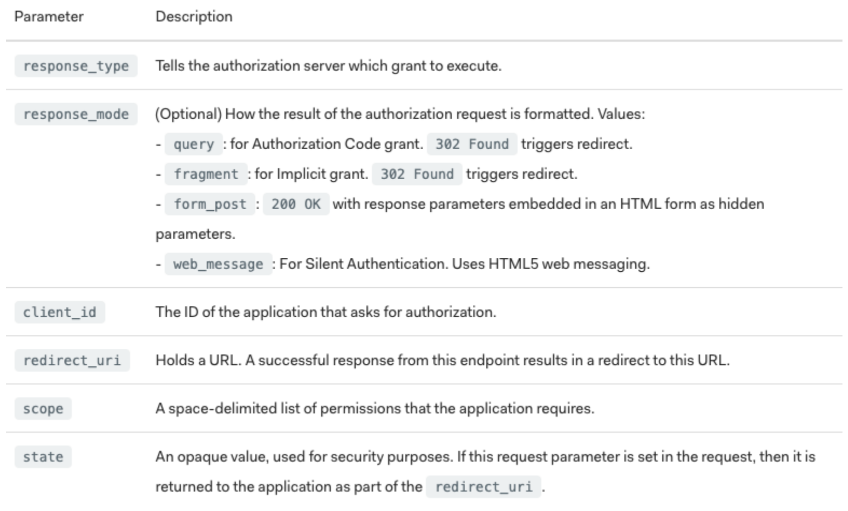
Section 2: Design

Personal Access Tokens

Tokens, generally, can be split into two types: (1) OAuth Tokens and (2) Personal Access Tokens. The distinctive difference between the two is which entity has the ability to dictate how much user information is given to the third-party application (also known as the calling application). With OAuth tokens, the user grants the authorization server - in this case, Spotify - the ability to act on the user's behalf when interacting with the third party application - this project, The Spotify Data Visualizer. Personal access tokens, by virtue of its name, gives the user the ability to decide the level of access the third-party application can have into their personal account. Personal access tokens are linked on a one-to-one basis between the users' accounts on the server and those on the third party application. With regards to security, the most attractive feature about these tokens is that the server must defer to the user to determine what permissions the calling application has into the user's account, allowing the user more control over access of their data. Tokens will never contain credentials or sensitive information; instead, it contains data on how to navigate the Application Programming Interface (API).

Getting Access

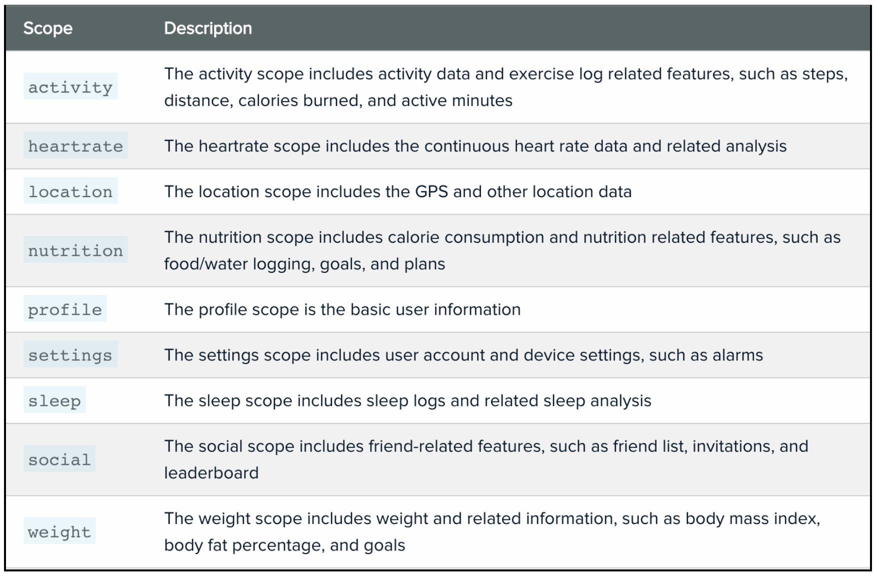
Gaining access into personal accounts can be broken down into three major components: the request, the scopes, and the response. These components are a subset of the server API, which enables communication between software systems. The request is composed of two API endpoints of note `/authorize` and `/oauth/token`. `/authorize` is used by the server to interface with the user when confirming permissions. The parameters needed within an authorization request are outlined by Figure 2.1 below.



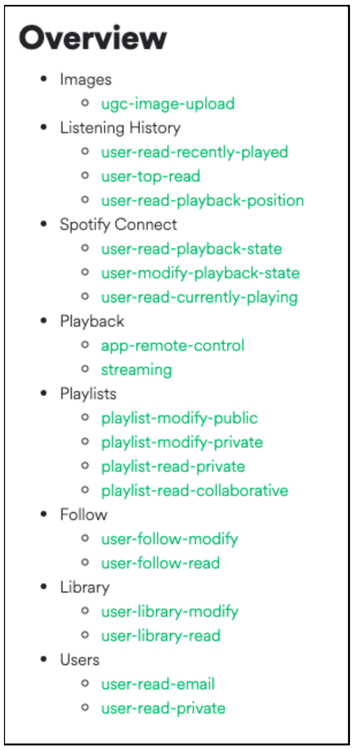
Parameter	Description
`response_type`	Tells the authorization server which grant to execute.
`response_mode`	(Optional) How the result of the authorization request is formatted. Values: - `query` : for Authorization Code grant. `302 Found` triggers redirect. - `fragment` : for Implicit grant. `302 Found` triggers redirect. - `form_post` : `200 OK` with response parameters embedded in an HTML form as hidden parameters. - `web_message` : For Silent Authentication. Uses HTML5 web messaging.
`client_id`	The ID of the application that asks for authorization.
`redirect_uri`	Holds a URL. A successful response from this endpoint results in a redirect to this URL.
`scope`	A space-delimited list of permissions that the application requires.
`state`	An opaque value, used for security purposes. If this request parameter is set in the request, then it is returned to the application as part of the `redirect_uri` .

Figure 2.1: Authorization Request Parameters

Within the request, scopes are key phrases used to uniquely identify the type of information the calling application is interested in. The phrases are commonly listed by the server for interested developers and will be correlated to the category of data the server collects from their users. Figure 2.2 shows the kind of information Fitbit can provide to third-party applications as compared to that of Spotify.



Scope	Description
`activity`	The activity scope includes activity data and exercise log related features, such as steps, distance, calories burned, and active minutes
`heartrate`	The heartrate scope includes the continuous heart rate data and related analysis
`location`	The location scope includes the GPS and other location data
`nutrition`	The nutrition scope includes calorie consumption and nutrition related features, such as food/water logging, goals, and plans
`profile`	The profile scope is the basic user information
`settings`	The settings scope includes user account and device settings, such as alarms
`sleep`	The sleep scope includes sleep logs and related sleep analysis
`social`	The social scope includes friend-related features, such as friend list, invitations, and leaderboard
`weight`	The weight scope includes weight and related information, such as body mass index, body fat percentage, and goals

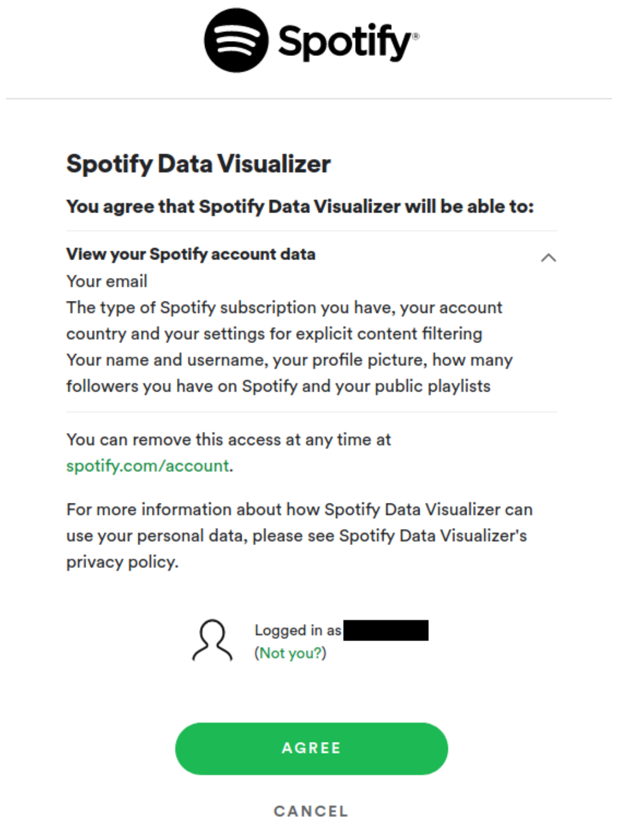


Overview

- Images
  - ugc-image-upload
- Listening History
  - user-read-recently-played
  - user-top-read
  - user-read-playback-position
- Spotify Connect
  - user-read-playback-state
  - user-modify-playback-state
  - user-read-currently-playing
- Playback
  - app-remote-control
  - streaming
- Playlists
  - playlist-modify-public
  - playlist-modify-private
  - playlist-read-private
  - playlist-read-collaborative
- Follow
  - user-follow-modify
  - user-follow-read
- Library
  - user-library-modify
  - user-library-read
- Users
  - user-read-email
  - user-read-private

Figure 2.2: Fitbit scopes (top), Spotify Scopes (bottom)

When a calling application sends the authorization request, the server will verify the user's credentials to authenticate them, and allow the calling application access to their data. If this is the user's first time using the application, Spotify will first redirect the user to a page where they are shown a list of the permissions that the calling application is requesting (as defined within the scopes sent in the request).



Spotify Data Visualizer

You agree that Spotify Data Visualizer will be able to:

View your Spotify account data

Your email

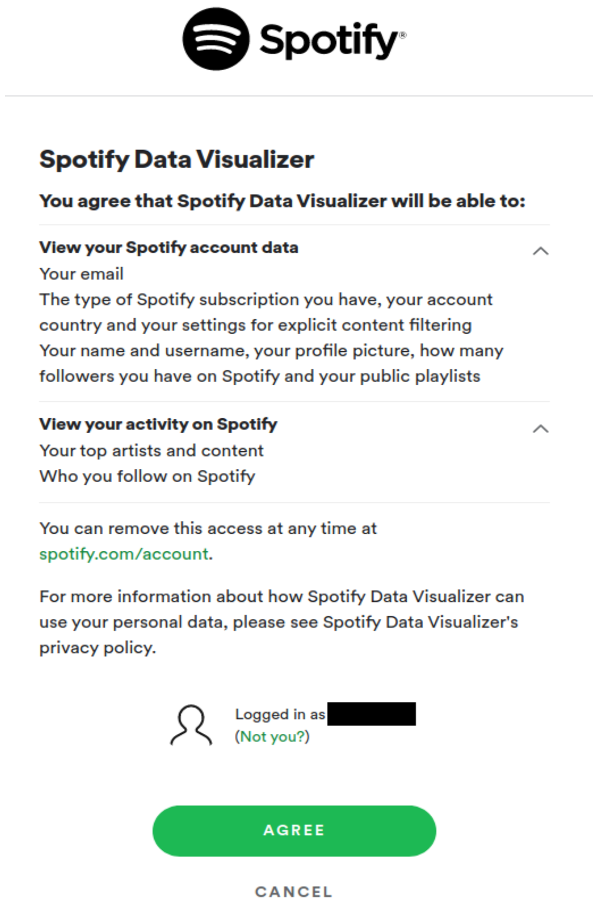
The type of Spotify subscription you have, your account country and your settings for explicit content filtering

Your name and username, your profile picture, how many followers you have on Spotify and your public playlists

You can remove this access at any time at spotify.com/account.

For more information about how Spotify Data Visualizer can use your personal data, please see Spotify Data Visualizer's privacy policy.

Logged in as [redacted]
(Not you?)



Spotify Data Visualizer

You agree that Spotify Data Visualizer will be able to:

View your Spotify account data

Your email

The type of Spotify subscription you have, your account country and your settings for explicit content filtering

Your name and username, your profile picture, how many followers you have on Spotify and your public playlists

View your activity on Spotify

Your top artists and content

Who you follow on Spotify

You can remove this access at any time at spotify.com/account.

For more information about how Spotify Data Visualizer can use your personal data, please see Spotify Data Visualizer's privacy policy.

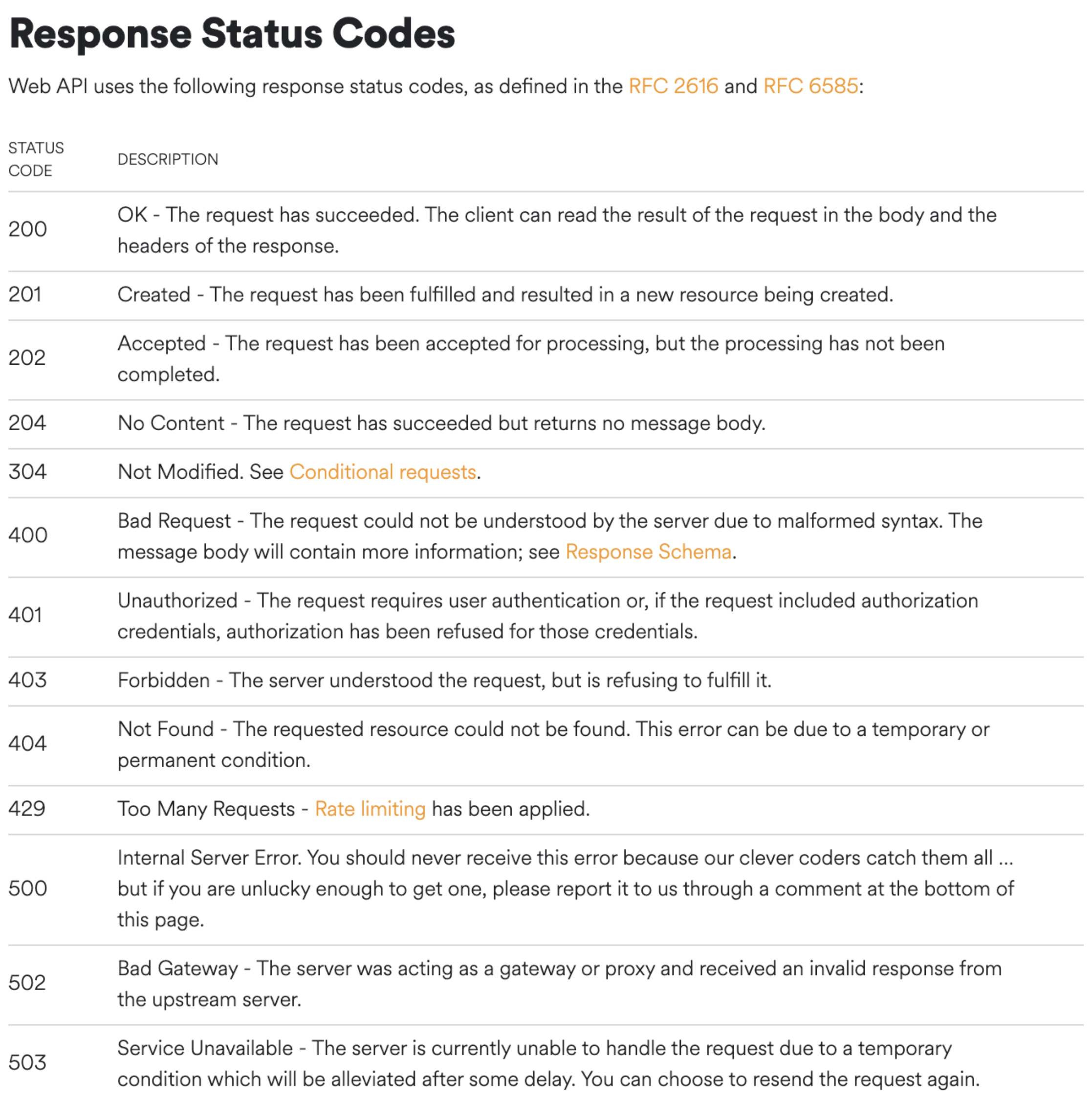
Logged in as [redacted]
(Not you?)

Figure 2.3: Authorization Details

If the user has already granted the calling application the permissions specified by the scopes (i.e. this is not the user's first time logging in), the response will skip the above step.

In exchange for a successful authorization request, the server will redirect to the second endpoint `/oauth/token`, where the calling application is granted an access token. When a token has expired or new resources not covered in the token scope are needed, a refresh token can be requested through the same process.

A response is always generated following a request. In the event that the scope is not reflective of the level of access the calling application is seeking, the server can reject the request on grounds of invalidity. Figure 2.4 shows the list of error codes pertaining to Spotify. Error 403 indicates an invalid request by the calling application. Should the request be granted, the server will redirect the calling application to the appropriate endpoints as permitted by the user.



Response Status Codes

Web API uses the following response status codes, as defined in the RFC 2616 and RFC 6585:

STATUS CODE	DESCRIPTION
200	OK - The request has succeeded. The client can read the result of the request in the body and the headers of the response.
201	Created - The request has been fulfilled and resulted in a new resource being created.
202	Accepted - The request has been accepted for processing, but the processing has not been completed.
204	No Content - The request has succeeded but returns no message body.
304	Not Modified. See Conditional requests.
400	Bad Request - The request could not be understood by the server due to malformed syntax. The message body will contain more information; see Response Schema.
401	Unauthorized - The request requires user authentication or, if the request included authorization credentials, authorization has been refused for those credentials.
403	Forbidden - The server understood the request, but is refusing to fulfill it.
404	Not Found - The requested resource could not be found. This error can be due to a temporary or permanent condition.
429	Too Many Requests - Rate limiting has been applied.
500	Internal Server Error. You should never receive this error because our clever coders catch them all ... but if you are unlucky enough to get one, please report it to us through a comment at the bottom of this page.
502	Bad Gateway - The server was acting as a gateway or proxy and received an invalid response from the upstream server.
503	Service Unavailable - The server is currently unable to handle the request due to a temporary condition which will be alleviated after some delay. You can choose to resend the request again.

Figure 2.4: Spotify Web API error codes

Application Architecture

The project application architecture is found in Figure 2.5. The application implementation as detailed in the next section follows this diagram closely. As shown, the user visits the application and is directed by the calling application to the Spotify Accounts Services as shown in step 1, where they will login and authorize access to the application to obtain their Spotify data. Spotify then sends the user back to the application with the access token that the application can then use to interact and obtain data from the Spotify Web API as shown in step 2.

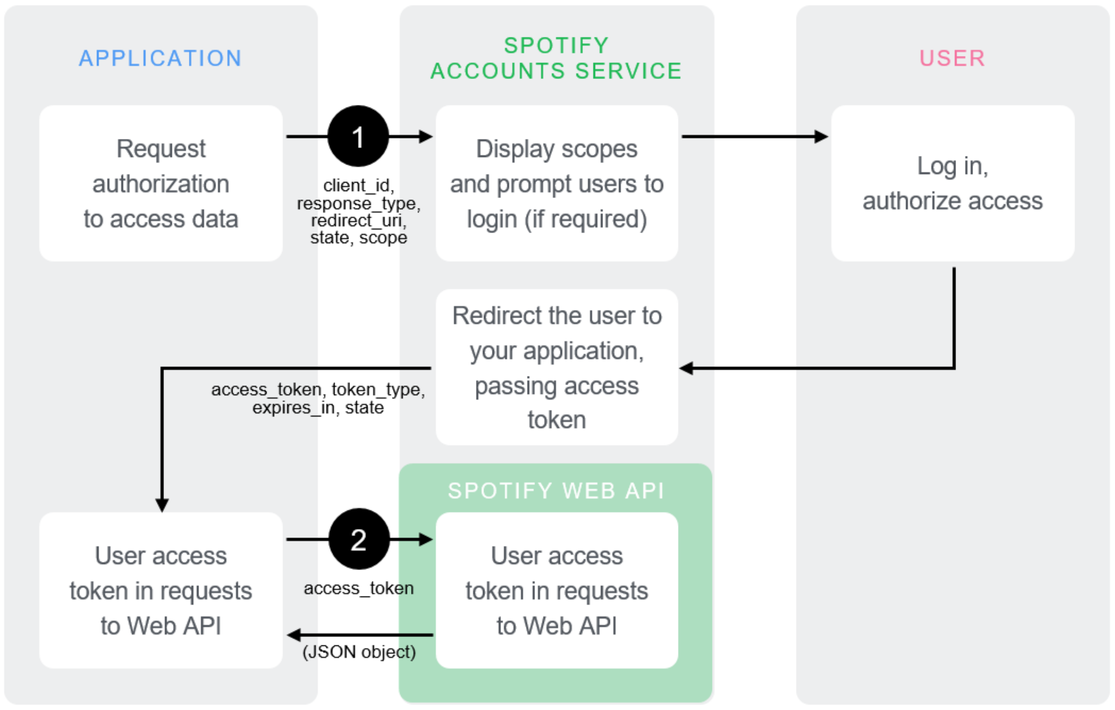

APPLICATION
SPOTIFY ACCOUNTS SERVICE
USER
Request authorization to access data
1
client_id, response_type, redirect_uri, state, scope
Display scopes and prompt users to login (if required)
Log in, authorize access
Redirect the user to your application, passing access token
access_token, token_type, expires_in, state
SPOTIFY WEB API
User access token in requests to Web API
2
access_token
(JSON object)
User access token in requests to Web API

Figure 2.5: Application Architecture

Section 3: Implementation

The following section describes in detail the project source code. The code is open source, and can be found at the following GitHub repository. All explanations of the code will be in reference to this repository. The application has also been deployed at the following location.

The project code is implemented in HTML, CSS, and vanilla JavaScript, with Node.js used to run the web server. The application contains an HTML page which contains all the necessary logic located at `docs/index.html`.

Authentication: Obtaining the Access Token

The page contains the HTML and CSS implementation of the landing page UI. The landing page contains a login button on top of a stylized and formatted static webpage. Upon clicking the login button, a URI is built to the Spotify authorization page at `https://accounts.spotify.com/authorize`. The URI is also appended with the necessary information to complete the OAuth transaction.

The first is the `client_id`, which is the unique identification code assigned to the project application by the Spotify API so that Spotify can track which of its franchised actors is responsible for which requests it is receiving to its API.

Secondly, the `scopes` are appended. As described in the Design section, scopes consist of the Spotify-defined OAuth authentication levels that the application is requesting access to. This application uses the three following scopes to access required data: `user-read-private`, `user-top-read`, and `user-read-email`.

Thirdly, the `redirect_uri` is appended, which is the location where the Spotify authentication server will send the user once authentication is granted and an access token is generated.

Finally, the `state` is appended. The state is a unique, randomly generated string that is passed to the authentication request, and compared with the response. It is used as protection against a security attack. If the response state does not match, it can be inferred that there has been an attack, and the response can be discarded.

Once Spotify authenticates the user, Spotify sends the user back to the application via the `redirect_uri`, appending the access token at the end of the URI.

Interfacing with the Spotify API

From here, the user is shown a page which contains various pieces of user information that has been provided by the Spotify API. Those pieces of information were obtained by sending a request to the `https://api.spotify.com/v1/me` endpoint.

The user is also shown a `Get Top Tracks` button. This button, when clicked, triggers the `getTopTracks()` function in the `docs/index.html` page which sends a request to the Spotify Web API endpoint `https://api.spotify.com/v1/me/top/tracks?limit=10`. This request contains the user's access token within the header, in order for the Spotify API to verify that it can serve the request. The response back from the Spotify Web API is a JSON containing a list of the user's recent top 10 favourite songs. These songs were then displayed to the page using the `displayTopSongs()` function.

Pressing this button also exposes a `Get Recommendations` button. Similar to the above, this button triggers the `getRecs()` function which sends a request to the Spotify recommendations endpoint at `https://api.spotify.com/v1/recommendations?market=US&seed_tracks='+[selected]+'&limit=10`. This request contains the access token in the header, and the URI is further appended with a string of all the user's top tracks as received from the previous function. This parameter is required by the recommendations endpoint in order to generate the response: a JSON consisting of songs and song details that Spotify has determined this user might like. These recommendations were then displayed to the page.

Section 4: Results

Figure 4.1 shows the login page for the Spotify Data Visualizer and clicking 'Login with Spotify' triggers the application to interface with the Spotify API by submitting a request. Upon a successful request, the application redirects the user to the `/authorize` landing page wherein Spotify asks the user how much access they are willing to grant to the calling application.

On the Spotify `/authorize` landing page as shown in Figure 4.2, the server details the kind of information being requested by the calling application and conditions under which the user's information may be used. These details are directly correlated to the scopes defined in the request by the calling application. Spotify dutifully notifies the user of their ability to revoke permissions in the future and indicates the account these changes will have an effect on.

Once authorization is granted by the user, the Spotify Data Visualizer is granted an access token that allows limited access to user information. This application first details the user information - ie. email address, username, display name, and country - as seen in Figure 4.3. With 'Get Top Tracks' the application makes use of the `user-top-read` scope to display the top 10 most listened to tracks. The 'Get Recommendations' option only requires authentication, and does not need a scope to generate 10 song recommendations which can be seen in Figure 4.4 and Figure 4.5.

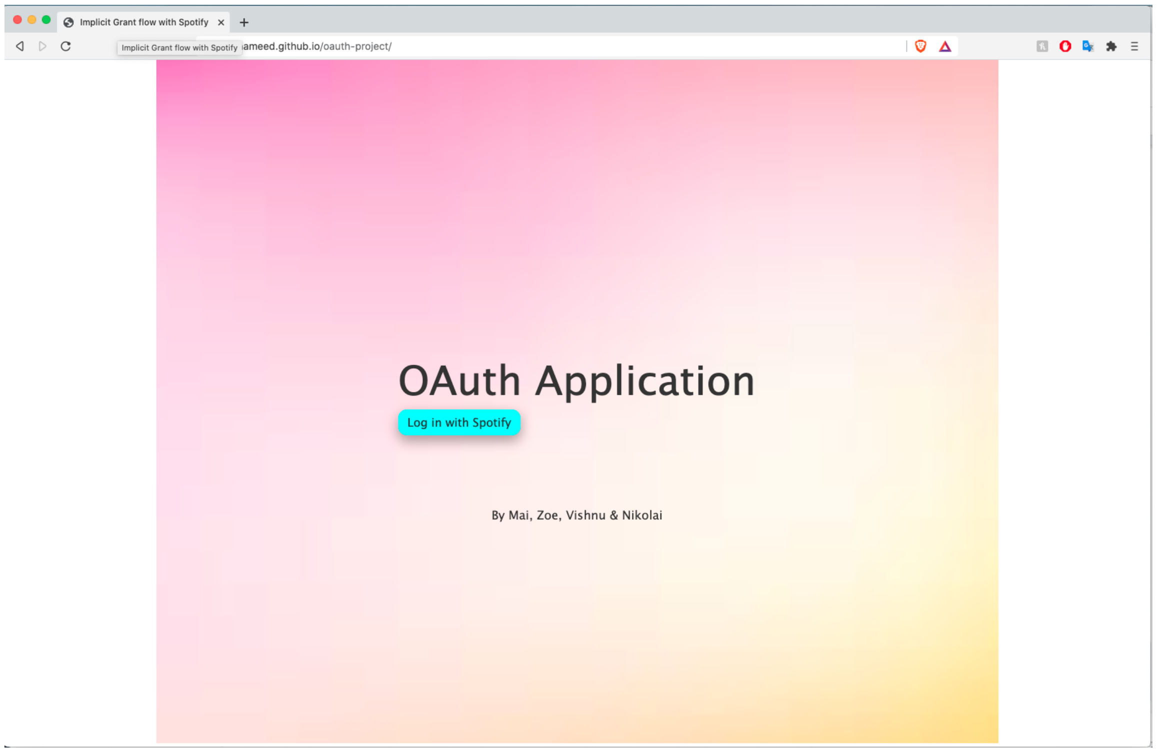
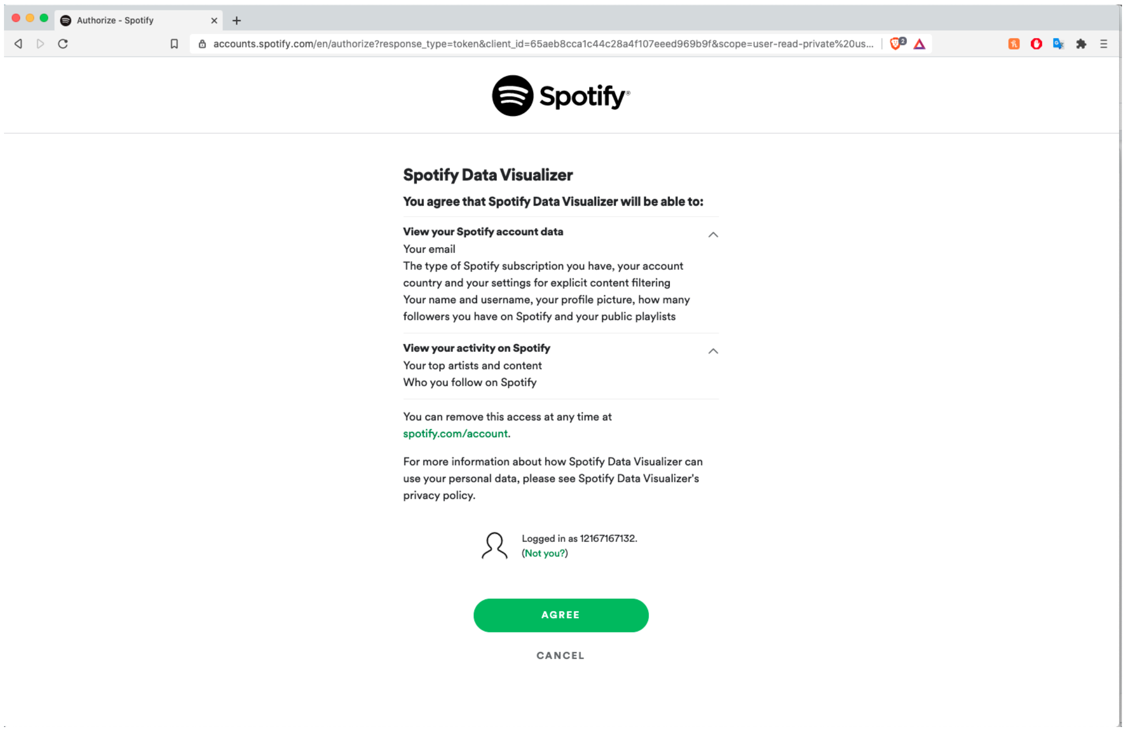

Implicit Grant flow with Spotify
ameed.github.io/oauth-project/
OAuth Application
Log in with Spotify
By Mai, Zoe, Vishnu & Nikolai

Figure 4.1: Login Page


Authorize - Spotify
accounts.spotify.com/en/authorize?response_type=token&client_id=65aeb8cca1c44c28a4f107eeed969b9f&scope=user-read-private%20us...

Spotify Data Visualizer

You agree that Spotify Data Visualizer will be able to:

View your Spotify account data
Your email
The type of Spotify subscription you have, your account country and your settings for explicit content filtering
Your name and username, your profile picture, how many followers you have on Spotify and your public playlists

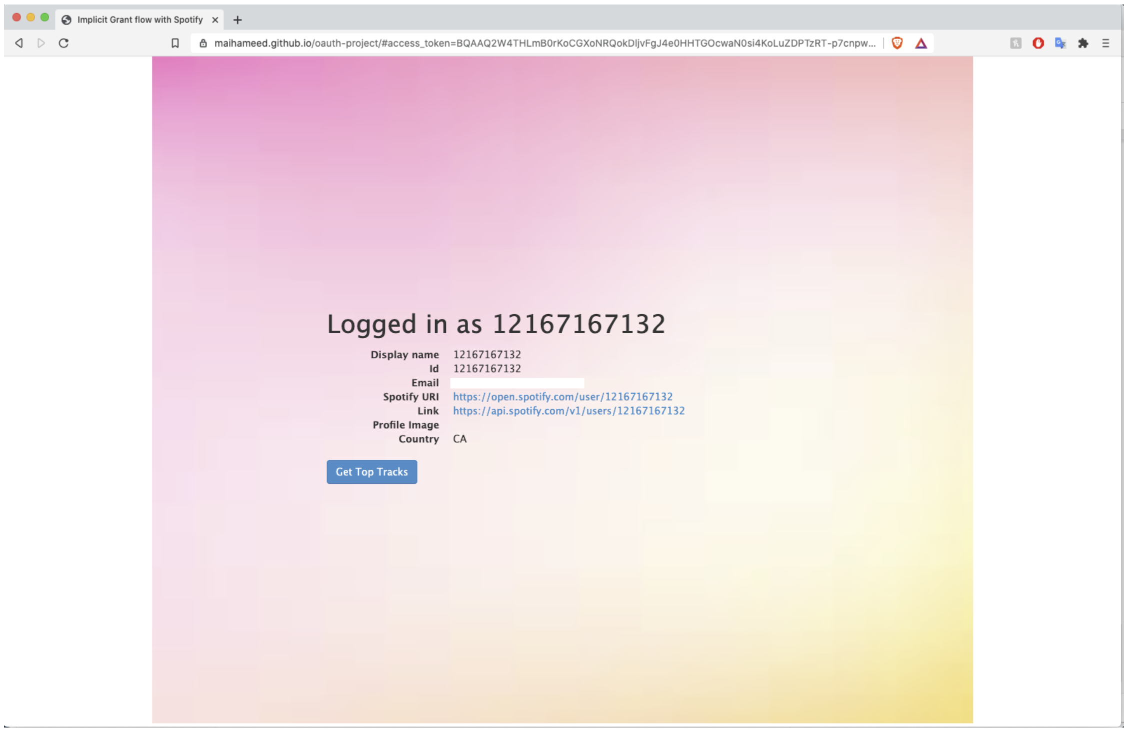
View your activity on Spotify
Your top artists and content
Who you follow on Spotify

You can remove this access at any time at spotify.com/account.

For more information about how Spotify Data Visualizer can use your personal data, please see Spotify Data Visualizer's privacy policy.

Logged in as 12167167132.
(Not you?)

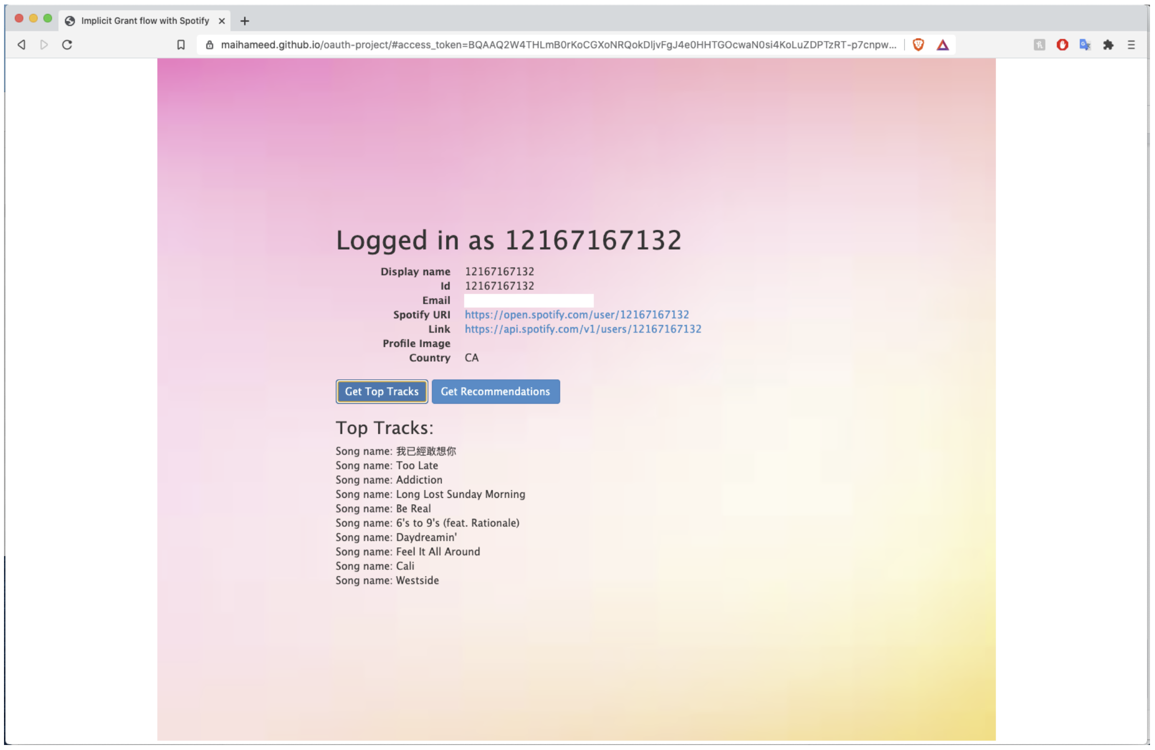
Figure 4.2: `/authorize` endpoint for Spotify server


Implicit Grant flow with Spotify
maihameed.github.io/oauth-project/#access_token=BQAAQ2W4THLmB0rKoCGXoNRQokDljvFgJ4e0HHTGOcwaN0si4KoLuZDPTzRT-p7cnpw...

Logged in as 12167167132

Display name	12167167132
Id	12167167132
Email	[redacted]
Spotify URI	https://open.spotify.com/user/12167167132
Link	https://api.spotify.com/v1/users/12167167132
Profile Image	
Country	CA

Figure 4.3: Permissions granted and implementation of `user-read-private`, `user-read-email` scopes



Implicit Grant flow with Spotify

maihameed.github.io/oauth-project/#access_token=BQAAQ2W4THLmB0rKoCGXoNRQokDljvFgJ4e0HHTGOcwaN0si4KoLuZDPTzRT-p7cnpw...

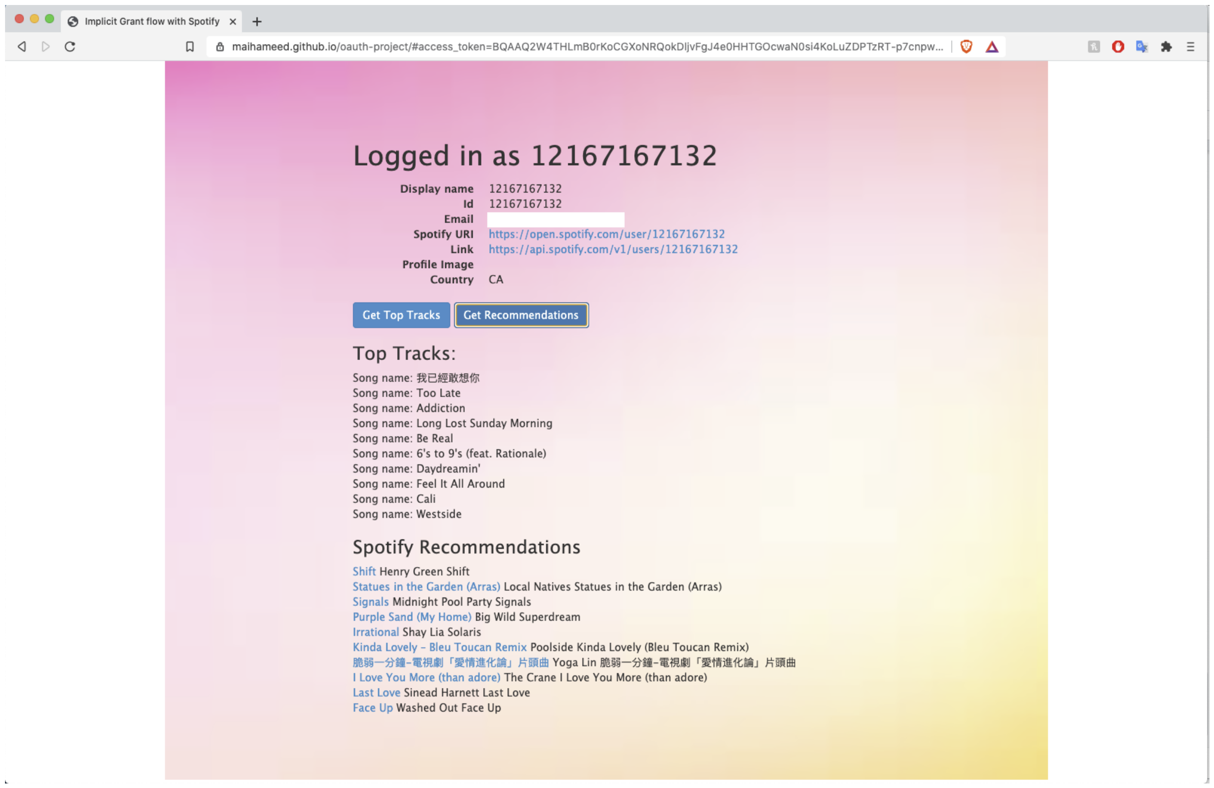
Logged in as 12167167132

Display name	12167167132
Id	12167167132
Email	[redacted]
Spotify URI	https://open.spotify.com/user/12167167132
Link	https://api.spotify.com/v1/users/12167167132
Profile Image	
Country	CA

Top Tracks:

Song name: 我已經敢想你
Song name: Too Late
Song name: Addiction
Song name: Long Lost Sunday Morning
Song name: Be Real
Song name: 6's to 9's (feat. Rationale)
Song name: Daydreamin'
Song name: Feel It All Around
Song name: Cali
Song name: Westside

Figure 4.4: Implementation of `user-top-read` scope for top tracks



Implicit Grant flow with Spotify

maihameed.github.io/oauth-project/#access_token=BQAAQ2W4THLmB0rKoCGXoNRQokDljvFgJ4e0HHTGOcwaN0si4KoLuZDPTzRT-p7cnpw...

Logged in as 12167167132

Display name	12167167132
Id	12167167132
Email	[redacted]
Spotify URI	https://open.spotify.com/user/12167167132
Link	https://api.spotify.com/v1/users/12167167132
Profile Image	
Country	CA

Top Tracks:

Song name: 我已經敢想你
Song name: Too Late
Song name: Addiction
Song name: Long Lost Sunday Morning
Song name: Be Real
Song name: 6's to 9's (feat. Rationale)
Song name: Daydreamin'
Song name: Feel It All Around
Song name: Cali
Song name: Westside

Spotify Recommendations

Shift Henry Green Shift
Statues in the Garden (Arras) Local Natives Statues in the Garden (Arras)
Signals Midnight Pool Party Signals
Purple Sand (My Home) Big Wild Superdream
Irrational Shay Lia Solaris
Kinda Lovely - Bleu Toucan Remix Poolside Kinda Lovely (Bleu Toucan Remix)
脆弱一分鐘-電視劇「愛情進化論」片頭曲 Yoga Lin 脆弱一分鐘-電視劇「愛情進化論」片頭曲
I Love You More (than adore) The Crane I Love You More (than adore)
Last Love Sinead Harnett Last Love
Face Up Washed Out Face Up

Figure 4.5: Get Recommendations option

Section 5: Conclusion

Project Summary

The Spotify Data Visualizer is an application that demonstrates the use of OAuth by allowing users to login to their Spotify account to display their Spotify data. This data includes username, email, and other profile information, as well as the user's top tracks and some recommendation tracks as determined by Spotify. This project allowed the group to get hands-on experience with OAuth, granting them a thorough understanding on the details and intricacies of this well utilized security tool.

Learning, Contributions, and Leadership

Over the course of the project, the team members were exposed to a variety of information about the OAuth process. The research conducted showed that a huge swathe of modern-day applications rely on OAuth to safely pass private user data between applications . This is done by abstracting the user's credentials and the login process behind the main data holder, which is typically the more secure provider (examples include Facebook, Google, or in this case Spotify). The team learned about the implementation of OAuth and were able to translate this information into the development of the application. During this development process, the errors encountered (e.g. 403 errors) helped the team learn about the specifics of scoping in relation to the Spotify API, and this learning was useful in tightening up the project.

Work for the project was well distributed across the group. Mai handled the initial setup of the application. This included creating a Spotify Developer account to receive a client id, creating and hosting the initial project (first iteration of the main index.html page and all the Node.js project dependencies) on GitHub, as well as setting up the OAuth login functionality to the Spotify web service using the client id. Nikolai's work consisted of making the HTML frontend of the website presentable and usable, including changing the splash page, editing the page background, and editing button and div layout and colour scheme. Finally, Zoe and Vishnu's contribution to the project was the additional user-info-based functionality of the app: creating the Top Tracks and Recommendations buttons and storage divs, as well as creating the functions to send the various HTTP requests to the appropriate endpoints and receive this information from the server.

The separation of work into these silos allowed for all members to become familiar with their assigned work and take on a leadership role in order to explain their work to the team. Each team member was able to explain their part to the team in order for the group as a whole to get a complete understanding of the project implementation. Additionally, this gave all team members an opportunity to lead the discussion around creating slides and practicing demo performance for their particular sections.

Appendix

The source code for the project can be found in this GitHub repository. The entire project is deployed and fully functional at the following location.