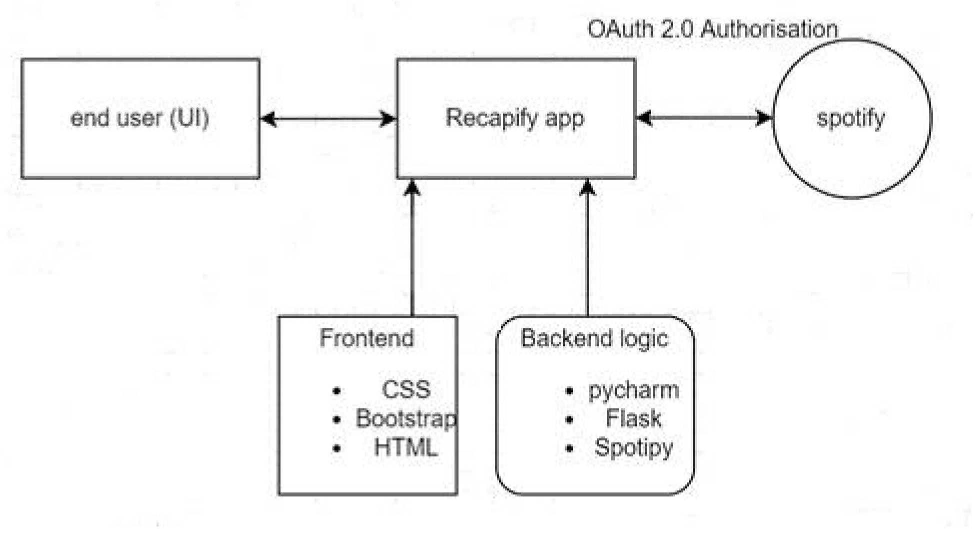
CFGDegree Group Project Report - Recapify

Lydia Duncan, Lucy Whitchurch, Katie Cox, Alicia Martin

Introduction

For our CFGDegree group project, our aim was to create a web application called Recapify. We decided on the name Recapify as it is a portmanteau of recap, API and Spotify and our main objective was to use Flask, PyCharm and Bootstrap to display insightful listening data of Spotify account holders.

We created an architecture diagram to visualise our initial idea:



Background

Our web application calls on the Spotify API to extract and showcase a user's account data. It utilises Spotify API endpoints such as top tracks, top artists, playlists and recently played tracks. It's essentially an extension of Spotify Wrapped – which is a feature that is released at the end of each year by Spotify to give a summary of each user's data.

Recapify enables a user to continually access this data throughout the year on a greater scale, with the potential for more features to be uploaded. Initially, when a user visits the Recapify homepage, they will have to follow a link to authorise access to their Spotify data.

Once achieved, the web application can successfully showcase a user's Spotify information.

Specifications and Design

For our group project we had the following specifications:

- To successfully call upon the Spotify API to extract main data using the following endpoints: recently played, top artists, top tracks, saved tracks and playlists
- To display a user's data using HTML & Bootstrap using a simplistic but elegant layout that implements the main design used in the Spotify app

Here are some links relating to our specification and design process:

- We created a Trello board to showcase our Agile 3-week sprint and requirements for the project: https://trello.com/b/tlx1sUIB/recapify
- We also utilised the Spotify for Web Developers framework: Web API Guides | Spotify for Developers

Implementation and Execution

Initially, Lydia created a GitHub repository for Recapify which we could all use to work on the project at our own pace. We decided to use PyCharm as our main IDE for the project, along with the Spotipy module.

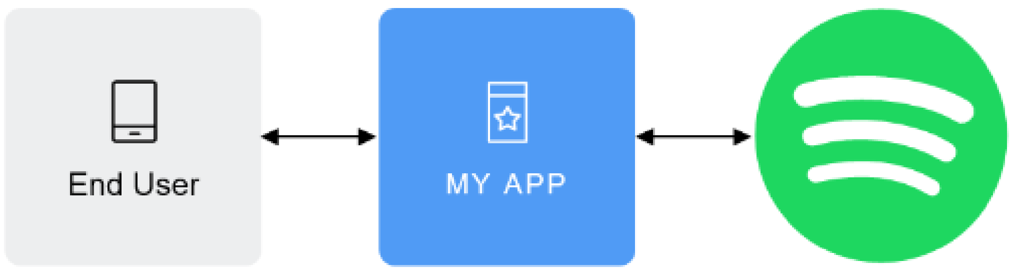
We then imported Flask to create our web application and Bootstrap to implement HTML and CSS. We decided on using Bootstrap Cards to display a user's Spotify data (main image, artist, album/song name, etc). Additionally, our homepage incorporated a navigation bar and utilised Bootstrap Carousel to enable a user to navigate through links to access their Spotify information.

Challenges and Achievements

Our main challenges during the project involved issues with OAuth and token access.

For the web application to work, it was essential to be able to successfully use the Spotify OAuth 2.0 framework as, without this, data cannot be accessed.

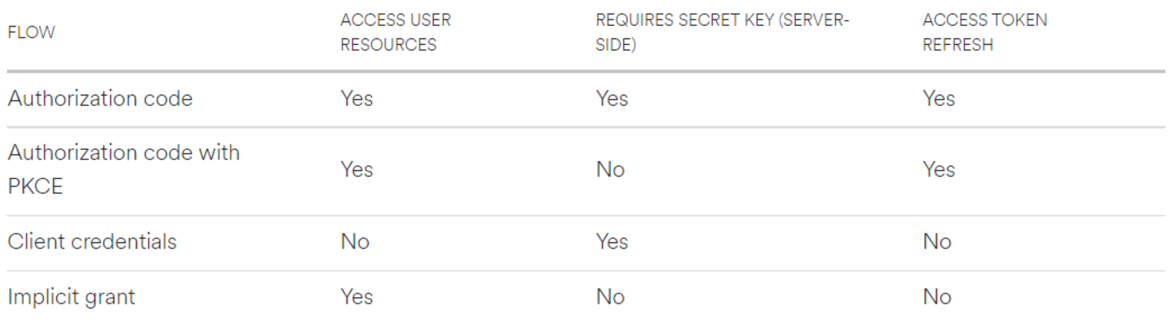
Here is a visualisation of how the OAuth 2.0 framework works:



The framework is implemented through the end user gaining an access token (requested through Recapify) in order to gain permission for their Spotify data to be accessed via the Spotify server scopes.

Initially, we decided to divide project work by assigning each member a file to create in PyCharm for each endpoint. For example, Alice was assigned recently played, Lydia was responsible for top artists, Katie was allocated top tracks and Lucy was allotted playlists. However, due to the tricky nature of OAuth and token access, we were struggling to be able to individually gain token access to implement Python code using our individual Spotify accounts.

There are 4 main grant types available to gain token access using the Spotify API:



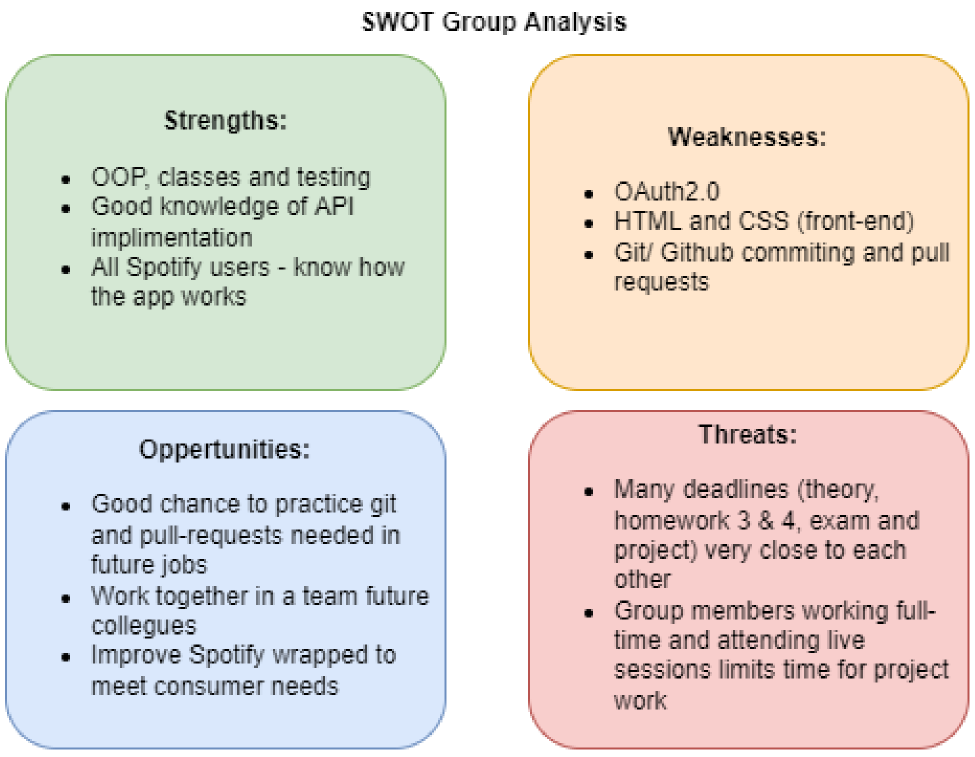
FLOW	ACCESS USER RESOURCES	REQUIRES SECRET KEY (SERVER-SIDE)	ACCESS TOKEN REFRESH
Authorization code	Yes	Yes	Yes
Authorization code with PKCE	Yes	No	Yes
Client credentials	No	Yes	No
Implicit grant	Yes	No	No

Two of these enable the token to repeatedly refresh, whereas the other two ways require a user to repeatedly request a new token after the previous one expires. This ended up taking up a lot of our time to resolve. Therefore, we decided to just use one of our Spotify accounts (Lydia's) to simplify the process. Thus, we then decided to divide the work that we had left in an alternative way. Whereas Lydia and Alicia mainly worked on implementing the main Python code, Lucy also worked on utilising the front-end development of the application and Katie also worked on testing the web application.

Additionally, despite hectic individual schedules, we managed to each use the GitHub repo to work at our own pace using elements of Agile development as shown in the Trello board.

Agile Development Strategies

As a group, we made a SWOT analysis which really helped us decide what we needed to focus on the most and how to delegate the tasks.


SWOT Group Analysis

Strengths:

- OOP, classes and testing
- Good knowledge of API implimentation
- All Spotify users - know how the app works

Weaknesses:

- OAuth2.0
- HTML and CSS (front-end)
- Git/ Github commiting and pull requests

Oppertunities:

- Good chance to practice git and pull-requests needed in future jobs
- Work together in a team future collegues
- Improve Spotify wrapped to meet consumer needs

Threats:

- Many deadlines (theory, homework 3 & 4, exam and project) very close to each other
- Group members working full-time and attending live sessions limits time for project work

Over the course of our project, we successfully focused on maintaining constant collaboration and communication.

We decided to use a three-week sprint. The first week consisted of planning our project, creating an architecture diagram and a Trello board to present our backlog: https://trello.com/b/tlx1sUIB/recapify

Our second week consisted of our team trying to implement the skeleton structure of our Recapify project including attempting to use the OAuth framework along with access tokens. Our final week involved the creation of each page of the web application, code review, front-end development and testing.

This allowed us to implement an iterative approach by successfully using the PDCA cycle (Plan, Do, Check, Act). Consequently, if we came across any major challenges during the check stage (such as token access), we had the flexibility of being able to go back to the design stage to adapt our project.

Testing and Evaluation

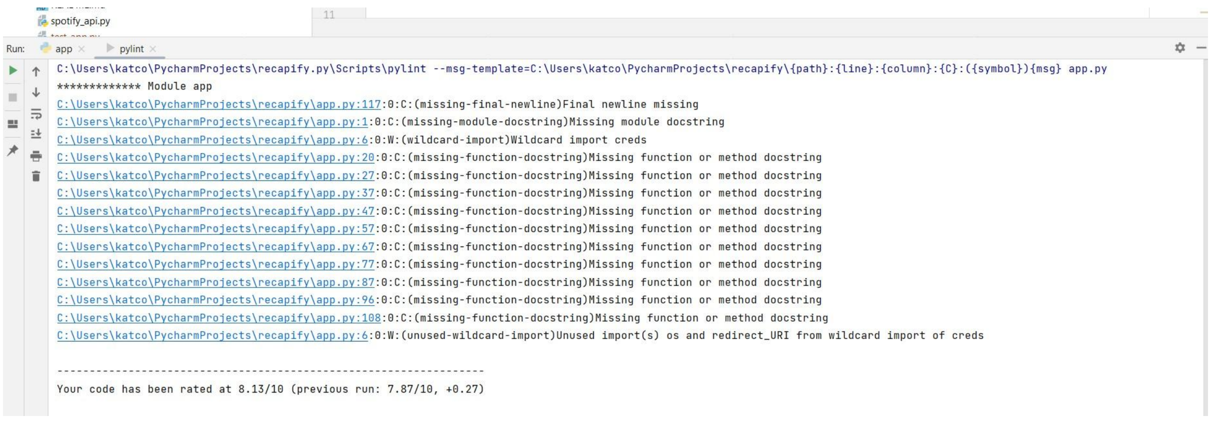
To test that our application worked, we decided to create a separate directory with a file to utilise the unittest module.

It was a challenge at first to be able to test our Recapify code, as most of the tests we initially thought to do would have required OAuth and token access which seemed virtually impossible to do without returning errors/fails.

Eventually, we resolved the issue by creating a dummy token using Lydia's secret token. We were able to access this in the console on the web application in the browser. We then made sure that a user would know how to access a token in the browser in order for a dummy code to be created for testing.

This allowed for various test cases to be created and effectively implemented, such as testing that twenty top tracks could indeed be accessed. In order for a user to be able to test the code themselves, we incorporated docstrings and comments within the testing file.

We also used Pylint for static analysis to check how effective our code was and in the end we were able to modify our code in app.py to obtain a rating of 8/10 which is quite promising.



```
C:\Users\katco\PycharmProjects\recapify.py\Scripts\pylint --msg-template=C:\Users\katco\PycharmProjects\recapify\{path}:{line}:{column}:{C}:({symbol}){msg} app.py
************* Module app
C:\Users\katco\PycharmProjects\recapify\app.py:117:0:C:(missing-final-newline)Final newline missing
C:\Users\katco\PycharmProjects\recapify\app.py:1:0:C:(missing-module-docstring)Missing module docstring
C:\Users\katco\PycharmProjects\recapify\app.py:6:0:W:(wildcard-import)Wildcard import creds
C:\Users\katco\PycharmProjects\recapify\app.py:20:0:C:(missing-function-docstring)Missing function or method docstring
C:\Users\katco\PycharmProjects\recapify\app.py:27:0:C:(missing-function-docstring)Missing function or method docstring
C:\Users\katco\PycharmProjects\recapify\app.py:37:0:C:(missing-function-docstring)Missing function or method docstring
C:\Users\katco\PycharmProjects\recapify\app.py:47:0:C:(missing-function-docstring)Missing function or method docstring
C:\Users\katco\PycharmProjects\recapify\app.py:57:0:C:(missing-function-docstring)Missing function or method docstring
C:\Users\katco\PycharmProjects\recapify\app.py:67:0:C:(missing-function-docstring)Missing function or method docstring
C:\Users\katco\PycharmProjects\recapify\app.py:77:0:C:(missing-function-docstring)Missing function or method docstring
C:\Users\katco\PycharmProjects\recapify\app.py:87:0:C:(missing-function-docstring)Missing function or method docstring
C:\Users\katco\PycharmProjects\recapify\app.py:96:0:C:(missing-function-docstring)Missing function or method docstring
C:\Users\katco\PycharmProjects\recapify\app.py:108:0:C:(missing-function-docstring)Missing function or method docstring
C:\Users\katco\PycharmProjects\recapify\app.py:6:0:W:(unused-wildcard-import)Unused import(s) os and redirect_URI from wildcard import of creds

------------------------------------------------------------------
Your code has been rated at 8.13/10 (previous run: 7.87/10, +0.27)
```

<u>**Conclusion**</u>

In conclusion, we were able to successfully and collaboratively work together in order to create our web application Recapify.
Through implementing agile strategies we were able to work flexibly to adapt our project in order for it to work seamlessly.
During the implementation and testing stages, we faced various challenges mainly regarding OAuth and token access. However, we were able to work together to think of alternative ways to fix these issues. To say that after 13 weeks of the CFGDegree we've been able to create a web application such as Recapify is something to be extremely proud of.