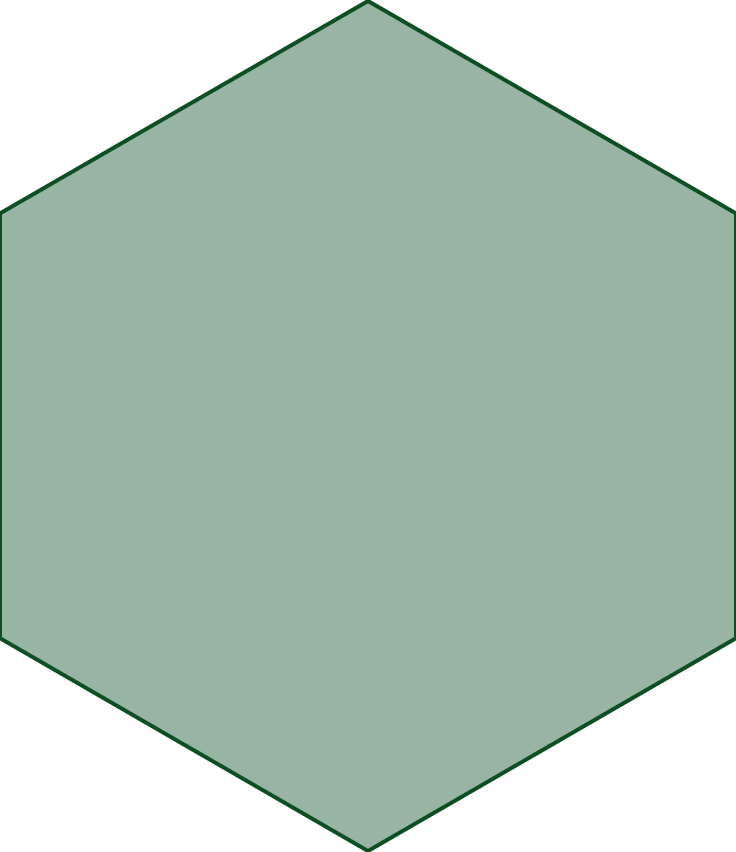
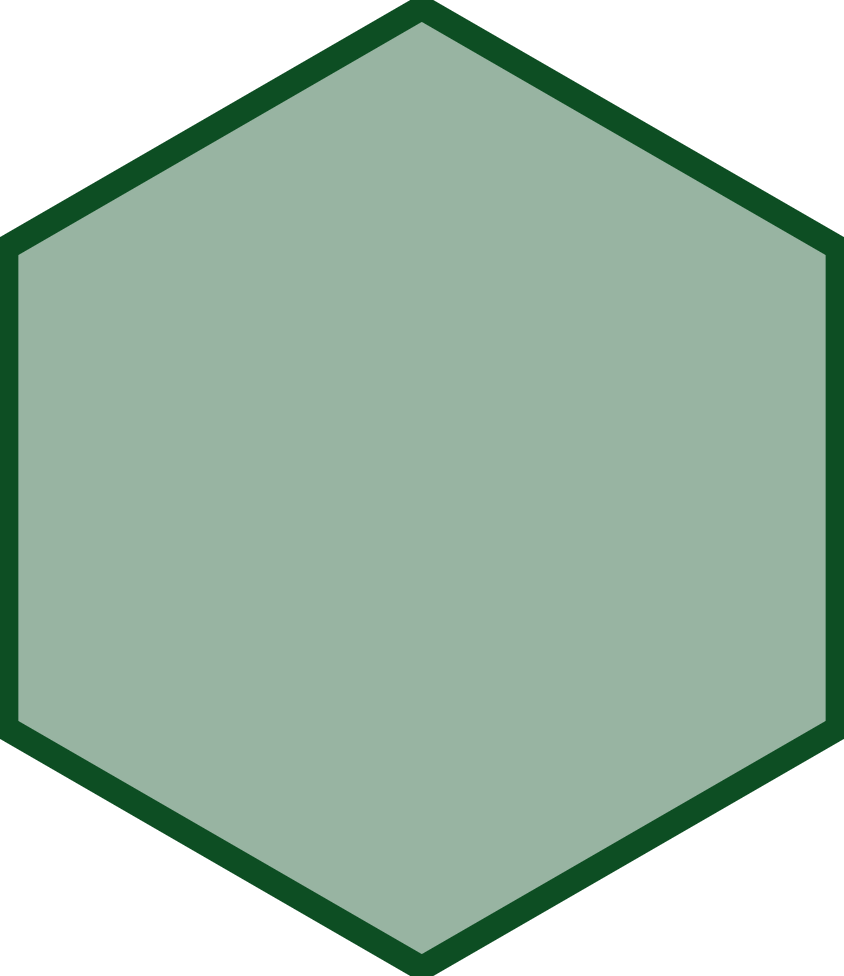
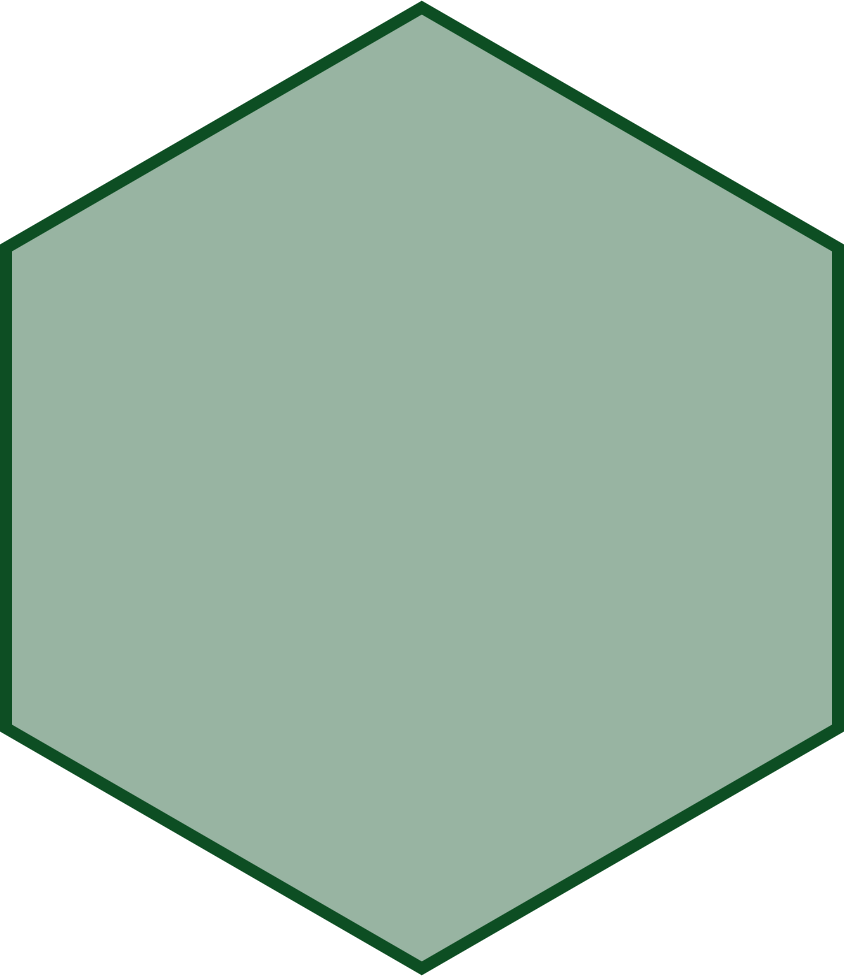
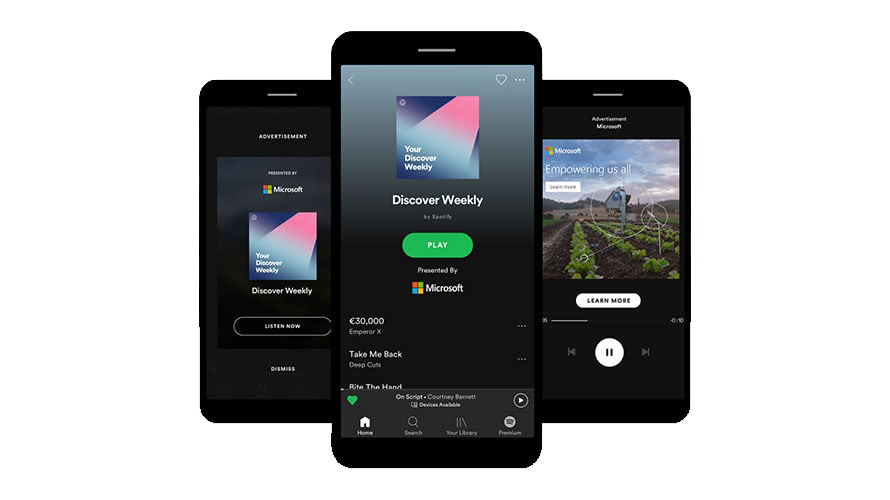


|  |
| --- |
| **SpoPSWI** |
| Tutor: Stuart Warman  Practical: Friday 9AM S520 |
| Alexander Lossberg n10456309 Connor McHugh n10522662 |
|  |



|  |  |  |
| --- | --- | --- |
| Declaration |  | ***Statement of Contribution***  • • • |
| Alexander Lossberg  * User Authentication * API Interactions * Tried to provide sense of sanity * Scrape class that is used to interact with the Spotify API and fetch results   Contribution: 49% Connor McHugh  * Web Server * HTML & CSS * Flask Python Frontend   Contribution: 49% |
|  |
| Alexander Lossberg  Connor McHugh *By completing this form we agree to the following:*   1. We declare that all of the work submitted for this assignment is our own original work except for material that is explicitly referenced and for which we have permission, or which is freely available (and also referenced). 2. We agree that QUT may archive this assignment for an indefinite period of time, and use it in the future for educational purposes including, but not limited to: as an example of previous work; as the basis for assignments, lectures or tutorials; for comparison when scanning for plagiarism, etc. 3. We agree to indemnify QUT and hold it blameless if copyright infringements are found in this work and the copyright owner takes action against QUT that is not covered by the normal terms of Educational Use.   Signed: |

Project Objectives



SpoPSWI (Pronounced spop-swee) would be a lightweight Spotify companion tool written in Python 3.7.3 that enables you to automatically populate a Spotify playlist full of your favourite artist’s releases. Powered by the Spotify API (Accessible in Python through popular package SpotiPy), a small Flask-driven web interface will allow users to take control of extended parameters to edit the playlist to their hearts content. Users can directly manipulate how often it will check for new releases so they can never miss a release from artists they follow on the platform. Users can also choose how far back they would like to feature tracks in their playlist, giving users optimal control over the playlist they create.

Review and Discussion of Technologies Used

High level discussion of the technologies used

## High-Level Programming Language – Python

Python is a general-purpose high-level programming language released in 1991. The high level nature of Python made it attractive to use in this task, with its focus on whitespace syntax and external modules allowing us to rapidly develop a web application for the task.

Some notable other high-level alternatives to Python are Java and C++, however they lack the ability to create flexible web-based interfaces and would be less robust as a result. JavaScript’s popular library ‘React’ could also be used as an alternative, but in the interests of time, Python was seen as a superior choice due to the nature of interacting with JavaScript Object Notation (JSON) files through the Spotify API. There is also significantly increased risk associated with working on node.js when not developing on the environment natively (A private GitHub repository was used to keep the Raspberry Pi version up to date with our updates made in our own Integrated Development Environment of choice).

Python is a flexible language that benefits the development cycle by allowing programs to be written ‘compactly and readably’ (Van Rossum, 1995). It does this by removing unnecessary variable/argument declarations, formulating high-level expressions by passing through parameters to pre-existing libraries, and outlines statements on the basis of indentation and whitespace as opposed to traditional bracket notation (Van Rossum, 1995). Python is also an interpreted language, and does not need to be converted to binary. Within your choice of Integrated Development Environment (IDE), Python ‘converts the source code into an intermediate form called bytecodes and then translates this into the native language of your computer and then runs it’ (Swaroop, 2003). This removes the need for compiling the program and makes Python significantly more portable as opposed to its alternatives.

## Application Programming Interface – Spotify

An Application Programming Interface (API) is a wide collection of data inputs or outputs, most typically in a database form, that are designed to be interfaced with directly by third parties.

*An API is software code written to function as a communication bridge between databases, programs and Web applications. They are a set of functions or procedures used by computer programs to access services from other operating systems, software libraries or other external systems. (Duesing, 2016)*

Application Programming Interfaces are designed to enable a steady stream of data to flow both in and out of a third party service or software, by predefining a set of protocols for the interaction. A Web Application Programming Interface is a ‘unique type of interface where the communication takes place using the Internet and Web-specific protocols’ (Pedro, 2017). By creating a universal standard, developers are able to directly interface with another application to pull and receive functions or procedures that can then be implemented into their own system. In the case of Spotify, data is transferred through the popular Representation State Transfer (REST) architecture (Spotify, 2014). Representation State Transfer is a popular agent for implementing interface semantics as its “stateless” form provides the necessary tools to ‘create, retrieve, update and delete’, as opposed to the more arbitrary prior implementations (Booth et al., 2004).

Due to the nature of Application Programming Interfaces, there are no direct alternatives that would meet the functionality requirements of our project. A direct scraping approach may have be taken to return analysis of certain expressions and fetch raw data, but it lacks the required ability to generate playlists and fetch user-specific information, which is only permissible through user authentication. For this reason we had to rely strictly on the Spotify Application Programming Interface. Other music streaming services also provide similar offerings, however with Spotify being the industry leader we felt it was most appropriate to design our application for this platform. Future iterations may allow for cross-implementations with other services such as SoundCloud, Deezer, Tidal etc.

An Application Programming Interface works by allowing a pre-existing software to talk and develop a relation with another piece of software (Farnsworth, 2018).

*An API defines functionalities that are independent of their respective implementations, which allows those implementations and definitions to vary without compromising each other. (Pearlman, 2016)*

One major key feature of using Application Programming Interfaces is they allow for the ‘abstraction of functionality between one system and another’ (Pearlman, 2016). They work by providing several connections known as “endpoints,” which the software may access to send and receive requests for fragments of data (Farnsworth, 2018). Typically, the scope of requests that can be made is quite limited, with the developer of the pre-existing software toolkit outlining what can and can’t be requested (Farnsworth, 2018). However, in the case of Spotify, further data can be accessed by gaining user access permissions so they may have access to potentially private and sensitive data.

## Web Server/Application – Flask

A web server is machine or group of machines that host all of the files needed by a website such as HTML documents, stylesheets, and images. Web servers can be either static or dynamic. Static web servers respond to HTTP requests from users by serving up actual files that are stored on the server, whereas dynamic web servers generate the content as its needed, and have to ability to personalise the content to suit the user (e.g. different information based on location) (Veltman, n.d.).

Flask is a python web framework that is used for building web applications and running them on servers. It is one of the simplest web frameworks for python and focuses on being highly customizable and flexible to suit the needs of its users, without being weighed down by unnecessary features.

It [Flask] is a micro level web framework which is written in Python by Armin Ronacher and several open source contributors. Flask is based on Werkzeug (Utilities and Requests), and Jinja 2 (Templating). (Kurama, 2018)

Flask allows for the hosting of a cross between static and dynamic web servers. This means that some content such as images and stylesheets are stored statically on the server, and other content, such as the HTML, is generated dynamically from a template, using python script to fill in some content, and to execute functions connected to parts of the web application.

A very popular alternative web framework to flask is Django. Django has many more features than flask does, but is also heavier and slower.

Django provides an all-inclusive experience: you get an admin panel, database interfaces, an ORM, and directory structure for your apps and projects out of the box. (Dwyer, 2017)

The main reason that Flask was chosen over Django for this project is that we did not need most of the features that are offered by Django, hence it would not make sense to choose its more bloated platform over a much faster and lighter framework such as Flask.

## User Authentication – Spotify & Scopes

Spotify authenticates API calls in two different ways. For developers, calls are made through an API Key and Secret, generated when you register an application (Spotify, 2014). This gives developers the necessary tools to interact with the Spotify platform. User authentication is also made possible through the Spotify Web API, where developers can send a POST request for the user to sign-in with their credentials, and a unique access token is granted to gain access to user information. Whilst this is needed to meet the functionality requirements of our web application, it does also provide a notable sense of risk, as users may unknowingly be sharing sensitive data that may compromise their personal identity. For this reason, Spotify have implemented “scopes” that are outlined in accordance with their endpoints (Spotify, 2014). When users access the sign-in page, they are given a comprehensive list of the specific endpoints that our application will be interfacing with, and what data will be shared as a result.

User-authentication requires a protocol to provide the necessary interaction needed to fetch an access token. The Spotipy package offers three unique methods of user-authentication, but only one was deemed suitable for our task (Lamere, 2019). The *client* module requires users to embed their credentials directly into the source code, which would be impractical for third party users interfacing with our application, as well as those not experienced with Python. The second authenticator offered (And the one used in the initial conceptualization of our application) is the *util* module. Util was much safer as it prompts the user to sign-in to fetch their token, and only relies on embedding the developer-side API keys into the source code. It also would save a cache file to the localhost to avoid unnecessary calls, however this disabled multiple users simultaneously using the software. Instead we relied on the OAuth authorization module that would allow for on-the-fly interfacing and support multiple users. A problem with the util module was that it would also open from the command line, and not from the web server internally. This served problems not only when using command-line interfacing tools such as PuTTY but would also not allow for users to authenticate themselves on non-localhost machines.

Our investigation concluded that OAuth was also the most secure for end users, as it does not share password data into our application (Sobers, 2018). This mitigates a potential worse-case scenario and limits liability if the application was to be further redefined for public use.

Reference List

1. Van Rossum, G. (1995). Python Tutorial. 3rd ed. Centrum voor Wiskunde en Informatica.
2. Swaroop, C. (2003). A Byte of Python.
3. Duesing, S. (2016). The API Revolution. Retrieved from <https://www.cleverbridge.com/corporate/the-api-revolution-infographic/>
4. Pedro, B. (2017). What are Web APIs. Retrieved from <https://hackernoon.com/what-are-web-apis-c74053fa4072?gi=936b834fb37d>
5. Spotify. (2014). Web API. Retrieved from <https://developer.spotify.com/documentation/web-api/>
6. Booth, D., Haas, H., McCabe, F., Newcomer, E., Champion, M., Ferris, C., & Orchard, D. (2004). Web Services Architecture. Retrieved from <https://www.w3.org/TR/2004/NOTE-ws-arch-20040211/>
7. Kurama, V. (2018). Flask Web Programming from Scratch. Retrieved from <https://hackernoon.com/flask-web-programming-from-scratch-9ada8088fde1>
8. Veltman, N. Static vs. Dynamic. Retrieved from <https://noahveltman.com/static-dynamic/>
9. Dwyer, G. (2017). Flask vs. Django: Why Flask Might Be Better. Retrieved from <https://www.codementor.io/garethdwyer/flask-vs-django-why-flask-might-be-better-4xs7mdf8v>
10. Farnsworth, B. (2018). What Is an Api? (And How Do They Actually Work?). [online] iMotions. Available at: https://imotions.com/blog/what-is-an-api/
11. Lamere, P. (2019). Spotipy Documentation. [online] Available at: <https://spotipy.readthedocs.io/en/latest/>
12. Pearlman, S. (2016). What are APIs and how do APIs work?. [online] MuleSoft Blog. Available at: <https://blogs.mulesoft.com/biz/tech-ramblings-biz/what-are-apis-how-do-apis-work/>
13. Sobers, R. (2018). What is OAuth? Definition and How it Works. [online] Inside Out Security. Available at: <https://www.varonis.com/blog/what-is-oauth/>
14. Spotify (2014). Authorization Guide. [online] Available at: https://developer.spotify.com/documentation/general/guides/authorization-guide/ [Accessed 30 May 2019].