# Brand guidelines | Queen&#39;s University Belfast

Name: Philip Sheerin

Email: [psheerin04@qub.ac.uk](mailto:psheerin04@qub.ac.uk)

Student number: 40314018

*‘Synopwiki*

A dissertation submitted in partial fulfilment of

The requirement for the degree of

MASTER OF SCIENCE in Software Development

In

The Queen’s University Belfast

By

*Philip Sheerin*

*09/05/2022*

**SCHOOL OF ELECTRONICS, ELECTRICAL ENGINEERING and COMPUTER SCIENCE**

**CSC7058 – INDIVIDUAL SOFTWARE DEVELOPMENT PROJECT**

A signed and completed cover sheet must accompany the submission of the Individual Software Development dissertation submitted for assessment.

Work submitted without a cover sheet will **NOT** be marked.

**Declaration of Academic Integrity**

Before signing the declaration below please check that the submission:

1. Has a full bibliography attached laid out according to the guidelines specified in the Student Project Handbook
2. Contains full acknowledgement of all secondary sources used (paper-based and electronic)
3. Does not exceed the specified page limit
4. Is clearly presented and proof-read
5. Is submitted on, or before, the specified or agreed due date. Late submissions will only be accepted in exceptional circumstances or where a deferment has been granted in advance.
6. Software and files are submitted via Canvas.

**I declare that I have read both the University and the School of Electronics, Electrical Engineering and Computer Science guidelines on plagiarism - http://www.qub.ac.uk/schools/eeecs/Education/StudentStudyInformation/Plagiarism/ - and that the attached submission is my own original work. No part of it has been submitted for any other assignment and I have acknowledged in my notes and bibliography all written and electronic sources used.**

**I am aware of the disciplinary consequences of failing to abide and follow the School and Queen’s University Regulations on Plagiarism.**

|  |  |  |  |
| --- | --- | --- | --- |
| **Name: (BLOCK CAPITALS)** | **PHILIP SHEERIN** | | |
| **Student Number:** | 40314018 | | |
|  |  |  |  |
| *Student’s signature* | **P.A. Sheerin** | *Date of submission* | **09/05/2022** |

# Acknowledgements

I would like to thank my mentor John Bustard for helping with this Masters’ research project in Software Development, for his time taken during weekly meeting and commitment to impart his knowledge and guidance.

Thank you to Queen’s University Belfast for the opportunity to take on and complete this degree, it has helped alter my career path.

# Abstract

The aim of this project was to design and implement an overall system in three components: a labelled Dataset of movie plots; an open API to query the labelled dataset; a user interface for users to traverse and add to dataset of labelled movie plots. Movies and TV shows are an ever-growing industry and in conjunction with AI and ML have shown many use cases. One such subset is collaborative filtering, used from technology giants such as Amazon and Netflix, as a recommendation system to get the best tailored data to their users. Collaborative filtering formed a core design feature in terms of the current project in terms of experimenting using this as a labelling tool. There are several factors to acknowledge when designing a system encompassing ML. The primary concern is creating an appropriate dataset and with this comes its own challenges: Collating relevant text datasets; Cleansing datasets; Combining datasets; Creating a labelled dataset. An experimental approach was used in terms of the technology stack in an aim optimise reusability and to reduce cost of the project to zero. The database layer is composed of two models, SQLite and GoogleDrive storage. Python’s Flask module is responsible for the backend of the project hosting the open APIs and connecting with the database. SQLAlchemy and Flask-Marshmallow allow for versatile querying of the database layer and due to their versatility can allow for scaling if required later in the systems lifecycle. The frontend user interface uses React allowing for a modular approach. PythonAnywhere enables the project to be deployed online and grants access to the API for end users. The system utilises three key data sets: Movie Lens, Wiki Plots and Imdb. Users can browse the collated dataset, navigate the collaborative filters, and continue to contribute to the dataset to enrich the data. The API allows the data to be accessed and used in future use cases. Although many core features were achieved there are areas that can be streamlined as well as boundless possibilities for future work to build off.

Contents

[0](#_Toc103117913)

[Acknowledgements 2](#_Toc103117914)

[Abstract 3](#_Toc103117915)

[Introduction 5](#_Toc103117916)

[Understanding the problem 6](#_Toc103117917)

[Technology Stack 7](#_Toc103117918)

[User interface design 10](#_Toc103117919)

[Architecture design and algorithm explanation 20](#_Toc103117920)

[Dataset/Database creation 21](#_Toc103117921)

[Backend framework and APIs 27](#_Toc103117922)

[Frontend user interface 34](#_Toc103117923)

[Experimentation 38](#_Toc103117924)

[Testing 41](#_Toc103117925)

[Evaluation and Conclusion 45](#_Toc103117926)

[References 47](#_Toc103117927)

# Introduction

Below is an outline of the structure of this dissertation and key components of each section:

* Understanding the Problem
* User interface design
* Architecture design and algorithm explanation
* Testing
* Evaluation and Conclusion

The first section, Understanding the problem, outlines the rationale for the current project, exploring current projects and any gaps in current software development. Next outlined are the key software technologies researched in this project, including machine learning techniques and new languages. Later discussed is the technology stack used (Python, SQLite, Google Drive Storage, GoogleDriveAPI, Javascript React framework) and the justification for these.

Following is the User interface section, here tried and tested user interface design decisions were researched and analysed for good practice in terms of user interface design and to discover the most suitable design for the project. A rationale, based on research and user feedback, is provided regarding the selected user interface design for the current project. Finally, a high level overview of the current projects user interface is displayed.

Next is the architecture design and algorithm explanation section. Here a high-level overview is detailed of the project’s solution files. Notably the data storage methods, the documentation for external users to utilise the open API and how the system frontend to backend is connected.

Following on from this section is the Testing section. Discussed here is how Test-driven development played a vital role throughout the development lifecycle. Examples of manual testing and API testing is discussed and how the entire project was approached using TDD approaches to reduce impediments.

The final section is the Evaluation and Conclusion section. Here we discuss how the project met our outlined requirements and which areas could require further work to streamline and develop.

# Understanding the problem

The consumer entertainment industry is an ever-growing business. Media consumption is at an all time high and continues to grow, it is a large space with innovative and exciting projects continuing to develop. Increases in media consumption drives users to seek content they most enjoy, to achieve the users’ needs many companies have enlisted the aid of Artificial intelligence (AI) and Machine Learning (ML) practices to target their users with tailored content. AI is a branch of computer science that focuses on developing machines that can perform intelligent tasks, tasks typically performed by humans (Rahwan et al. 2019). ML is a branch of AI – a method in which a system can learn from data, identify patterns within this data and make decisions based on this information. Combining these technologies has proved a powerful and transformative tool when solving many software problems. Netflix is one such company that uses ML to display media items to users based on their previous search history and watch history. This recommendation system applies a ML technique known as Collaborative filtering to achieve content tailored user experiences.

Collaborative filtering is a technique of ML where items that a user is predicted to enjoy such as products or media are shown, based on data from similar users. It traverses a group of known users aiming to locate a smaller subset of users that have similar preferences to the current user. One such project that uses a similar recommendation system is a personality quiz created by Open-Source Psychometrics project. They have developed an interactive personality quiz that attempts to match the user to a fictional character from a large database. This database was created through crowd-sourcing ratings of characters. The backend of this project uses a recommendation system engine to compare the users’ inputted data with their database of 1400+ fictional characters.

ML is multi-faceted, there are several models that operate in different ways. Unsupervised learning is one of these models. Unsupervised learning can be computed on non-labelled data sets and produce meaningful data, however since the data is not labelled it can be difficult to predict outcomes, making this process less suitable for text generation and recommendation systems. Contrasted to this is supervised learning, a model that is used on labelled datasets. This process is suited to predicting outcomes as it is based off previous labelled data. Collaborative filtering is a supervised learning model, using user rated data to make predictions. Additionally, the advantage of using supervised learning model is that data without a label can potentially be accurately predicted based on previous labelled data.

MovieLens is a non-commercial project that aims to provide personalised movie recommendations based on data primarily gathered from IMDB and internal user rating and labels. For educational and development purposes MovieLens provides access to their rich dataset of movies. These labelled datasets are crucial in terms of ML development.

Although there are projects that have labelled datasets of movies, there is untapped potential in unlabelled datasets available. WikiPlots is an unlabelled dataset that contains 120,000 entries, a combination of movie and written literature, with data pertaining to the plot title and a brief synopsis of the story. Beginning to label such datasets and collate with existing labelled datasets means that powerful labelled datasets can be created at scale only extending the possibilities of AI and ML.

With the application of ML growing and the untapped potential of unlabelled datasets to be transformed into powerful labelled datasets it provides a clear rational for this project. There is a multifaceted aim of the current project: Collate a foundational dataset from labelled and unlabelled data that can be utilised by supervised learning ML algorithms; Create an open API where this data is available to external users; Create a user interface that allows users to traverse this dataset and feedback labelled data into the system.

## Technology Stack

Before deciding which technology and architecture design components to employ, individual research was carried out to explore which was the most suitable. Primarily backend code was handled by Python to collate data, host the APIs, and perform the collaborative filtering machine learning analysis on the dataset. Data storage was handled by two components, SQLite and Google Drive API. SQLite data storage was utilised to store most of the web function data, while Google Drvie API storage stored most of the plot data set. Pythons SQLAlchemy and Flask-Marshmallow Object oriented mapper (ORM) allowed for seamless querying. PythonAnywhere web hosting was used to host the API online as this can also be tailored to host static react pages for minimal cost. Next, we will explore the justification for the chosen technology stack.

The following key architecture aspects were identified and explored:

* Programming language
* Dataset exploration and creation
* Dataset storage and querying
* Web development and hosting

**Programming language**

The core programming language for any project is a foundational factor that must be carefully considered. Due to the nature of the project, when researching the appropriate programming language there three key factors taken into consideration:

* Machine learning and dataset capabilities
* Web development capabilities
* Ease of development for prototyping and testing

Machine learning and dataset capabilities

In terms of ML there are numerous programming languages that are useful when developing and working with datasets. Research suggests that Python, Javascript and Java are languages that are largely utilised when developing ML specific projects.

Java is a well-supported language in terms of AI development and its compatibility across many platforms is advantageous, however the complex syntax can lead to slower development time.

JavaScript is another choice when developing an AI system – it has similar advantages to Java, it is well supported and is a desired option due to the combability with web development. However due to the complex syntax it can generate complexity and become time consuming during development.

There appears consensus that the programming language Python is highly valuable when developing AI and ML software (Voskoglou, 2017; Codeacademy, 2021), due to a multitude of factors: Diverse Libraries, including AI specific libraries; intuitive syntax; interpretive runtime, allowing for rapid prototyping, particularly ML techniques; Frontend and backend capabilities. Additionally, are the numerous powerful modules available in python when working with datasets. Namely the Pandas module that allows for efficient analysis of data via dataframes.

Web development capabilities

When developing frontend web development projects, JavaScript is a frontrunner, providing versatile development scope and a range of frameworks such as Angualr.js, Node.js and React. For the current project React was the chosen framework for the frontend. Primarily due to the modular and reusable components architecture and its ease of use when developing and testing, in contrast to the complexity of Angular.js and Vue.js. React is one of the most used web frameworks in web development.

In terms of implementing the backend of a web project, Python is evidently a language that is powerful and versatile. One of Python modules is Flask. Flask is micro web framework that caters for ease of user when developing and quick deployment.

Ease of development for prototyping and testing

Python is a high-level language and can be reasoned that the syntax is simpler to understand especially for novice programmers. There are many online resources for setting up Python projects that are quick and easy to follow. This along with an extensive and ever-growing list of libraries allows for use cases in a large range of projects, allows for rapid prototyping capabilities.

React is a highly utilised frontend framework due to a multitude of factors namely the time to development efficiency and reusability in terms of internal web components. For example, hot reload is a highly desirable function of React that actively updates the local hosted web page when the user makes a code change and saves.

**Dataset exploration**

An integral component when developing an ML useable dataset is high quality relevant data, with this come several challenges:

* Exploring relevant text – data that is specific to the problem domain
* Cleansing – removing errors and creating a normative data
* Combining – joining multiple data sources
* Labelling – correctly labelling data so that it can be processed

Criteria for relevant data is critical, for the current project focus was placed on data associated with literature and movie plots. Several datasets were identified, these included: WikiPlots, MovieNet, MovieLens, IMDB and TMDB.

WikiPlots is an unlabelled dataset that contains 120,000 entries, a combination of movie and written literature, with data pertaining to the plot title and a brief synopsis of the story.

MovieLens offers a range of datasets, recommend for educational and development use cases is the dataset that contains 1700 movie entries. This includes 100,000 ratings, 3600 tags completed by 600 users.

MovieNet dataset contains 1100 movie with diverse data within including trailers, photos, text (subtitle, synopsis, and script), Character IDs, Scene segmentation, Action/Place tags, Shot cinematic style. Although advertised as such not all data is available due to copyright restrictions.

IMDB dataset contains a wealth information of that is accessible using the IMDB API. Data that can be accessed via the IMDB website can be returned using the API (e.g. Movie title, Movie synopsis, rating, cast)

It was concluded that MovieLens, WikiPlots and IMDB would be the most suitable datasets to combine for the current project for the following reason: MovieLens provided a labelled dataset suitable for ML experimentation; IMDB allowed access to an API to return desired results and WikiPlots provided an extensive list of unlabelled plots that could be string matched with MovieLens.

**Data storage**

There are varying solutions when storing data depending on specification of each system. Structured data has a predefined structure, and it can be easier to navigate. However, the storage can be rigid and be a limiting factor in scalability. Unstructured data can be difficult to analyse however it allows for flexibility to be added to, cheaper storage and data to be stored in native formats. JSON documents act as a bridge between structured and unstructured data, they are a flexible document type, additional information can be added and is more suitable when using metadata.

Several database solutions were investigated, and the below key factors were core requirements:

* Expense
* Ease of use
* Applicability

MongoDB is a paid service, it proves a flexible database solution, it is suited to the storage of unstructured, simplistic with informative documentation and is highly performative due to storing data in RAM for quick access. However, with better performance comes greater memory usage – due to the nature of the project in the future the data set will continue to grow this could be a limiting factor in scalability later in the software development cycle.

SQLite is a free open-source tool that allows for fast prototyping due to its simplistic nature. It is suited to development and testing. Scalability is an issue with SQLite, it is not suited for large scale application. However ever due to specifications of the current project it may prove useful during development and have use cases such as being used as a cache before writing data to a database.

Google drive is a cloud-based storage system which has extensive flexibility. There is a learning curve to implementing Google Drive as a database layer, however, it allows access to edit and read files and storage cost is relatively cheap from a development perspective.

**Web hosting**

PythonAnywhere and Heroku appear to be two viable methods for web hosting. Both offer a free tier system to host on their platform. Heroku is a fully customisable web hosting platform that allows users to deploy using a range of language such as Python, Node.js and Java, this is advantageous but may add complexity to the current project as a higher degree of architectural set up may be required. In contrast to this PythonAnywhere primarily supports Python projects, therefore many of the python system modules are already installed in the cloud service to their latest versions. This is advantageous for the current project as it allows for quicker deployment if necessary.

**Aim:**

With the problem domain explored and the aim of the current project outlined: Collate a foundational dataset from labelled and unlabelled data that can be utilised by supervised learning ML algorithms; Create an open API where this data is available to external users; Create a user interface that allows users to traverse this dataset and feedback labelled data into the system. The following functional requirements and desired functionality of the project were defined.

**Key functional requirements:**

* Collate a dataset of labelled and unlabelled data
* Create an Open API that users can query the created dataset to return JSON results
* User interface that users can use to search and navigate the created dataset
* Use collaborative filtering model scores so that users can view related movies
* Dataset labelling functionality

**Desired functionality**

* As user interacts with the data tailored data is displayed to the user
* Full CRUD functionality with Google Drive API
* Automated labelling - Improve labelling by updating current labels based on user feedback (Loop)

# User interface design

User interface (UI) design is a crucial aspect of any front facing user product. As the global internet has developed, content is ever growing at an exponential rate, the average user is consuming more content. However, holding the attention of users is becoming more challenging. Excellent UI design can increase traffic and keep users engaged. There are several key factors to be aware of when designing UI: easily navigable, simple design, useability, and consistency across the product. When these factors are considered and implemented correctly it creates a better overall user experience.

**Tried and tested UI design:**

**Netflix**

Graphical user interface, application

Description automatically generated

*Figure 1. Netflix website user interface design*

Netflix is a substantial consumer product that must engage customers and keep them on the platform. One such design tactic that Netflix employs is a card-based system provides graphically rich tangible elements that draw the users attention. This design not only engages the user but provides a pleasurable experience when navigating the various genres and movie titles (Pal, N, 2020). Below is an example of Netflix’s UI, notice the easily accessible navigation along the top of the page and the decluttered nature of the available content. This is an excellent example of the key features discussed above when developing a seamless UI. However, with this simplified design comes drawbacks, simple settings and features can be hidden within layers of menus and can be difficult for users to access. At times needing multiple user interactions to get to the desired setting. This can lead to frustrated users and is an important balance between simplicity and useability.

**Amazon**

Timeline

Description automatically generated*Figure 2. Amazon website user interface*

Similarly, Amazon is another example of a major user consumer that employs a card based graphically rich UI which helps to engage customers. Amazon’s UI is designed to provide the customer quick access to all their needs on a single view. Differing from Netflix they have many settings quickly accessible within one click and always have a search bar available in the navigation bar enabling the user to navigate seamlessly. Although this quick access design gives customers better overall control it can make the experience cluttered and confusing, especially for newer users to the site. Below is an example of Amazon’s main site, notice the numerous options available to the user.

**YouTube**

Graphical user interface, website

Description automatically generated*Figure 3. YouTube website user interface*

Like the Netflix and Amazon, YouTube has opted for the media card design. However, the UI achieves a hybrid approach, the landing page appears decluttered but also has a vertical side menu bar that allows the end user greater control over their experience.

**Prototyping:**

When prototyping the User interface design for Synopwiki site, simulating, and iterating on the tried and tested UI designs were a key aspect. The aim was to create a simplified card-based UI with the following functionality:

* Search functionality for all current procured data
* Simple card based paginated scrolling
* Navigate between the different collaborative filtering scores (Fast AI scores)
* CRUD operations for user fronted plot data
* Allow user to query the API to retrieve data

**Final Design:**

Material UI css framework was used primarily to develop the aesthetic and design of the final UI. The reasoning for using this framework is that it integrates well with React, additionally due to its modular and simple nature it allows for rapid development to create a professional UI.

*Graphical user interface, website

Description automatically generatedFigure 4. Main Synopwiki landing page*

Synopwiki Landing page:

The first page that the user lands on it the Synopwiki landing page (Figure 4.), this is a basic information page to catch the users attention and lead them to the about page where they can learn about the functionality of the site. The Naviagtion bar at the top of each page is persistant to give the users ease of navigation at all times.

Graphical user interface, text, application

Description automatically generated*Figure 5. Synopwiki about page which details key information to navigate the site.*

Synopwiki About Page:

The about page is an information page to guide the user on how to use the site and answer any questions they may have (Figure 5.). It has an API documentation section so that the users have clear instructions when querying the API endpoints.

Graphical user interface, timeline

Description automatically generated

*Figure 6. Example of Synopwiki of main plots page.*

Main plot page:

The main plots page is split into two sections (Figure 6.) Firstly, an alphabetically ordered grid, consisting of plot media cards with paginated navigation. The next section is the search tab, where users can use the search functionality to search for a specific plot.

Graphical user interface, application, Word

Description automatically generated

*Figure 7. Example of individual plot card.*

Individual plot page:

Users can click into each plot card on the main plots page and are brought to a specific page holding information on that plot (Figure 7.). Each specific plot page has 3 keys section. Firstly, the information page which is constructed using four cards: IMDB card (generic plot data); IMDB synopsis card; Wiki Plot card and a Collaborative filtering data table, that stores the dimensional score for each label. Next is the Similar cards page, this page holds the top 10 cards that are closely associated with this plot. Finally, for logged in users is the review tab, on this tab users can input data into various forms that is fed back to the database for analysis later.

Graphical user interface, application

Description automatically generated

*Figure 8. Example section where users can perform CRUD options to the dataset on various pages.*

Data collection and Registered user vs non-registered user:

Synopwiki has two main states, Registered user, and non-registered user. A non-registered user has read only access to the site they can navigate the plots and labelled data as expected, they do not have the permissions to add data to the system. A registered user has the same permissions as a non-registered user, but they also have the functionality to add data to the database via different methods as displayed in figure 8.

On various sections of the site the logged in user is given the opportunity to feedback data to the database, this is stored in various tables that will allow for this to be validated and later analysed to better label data.

Text

Description automatically generated

*Figure 9. Example of open API*

API Access:

All users have access to the open API hosted on the PythonAnywhere site at <https://philipsheerin.pythonanywhere.com/api/>. Users can query multiple endpoints to return the desired data (Figure 9.), a detailed list of these endpoints are outlined in the about page of the Synopwiki site.

# Architecture design and algorithm explanation

The Synopwiki project was designed based on three major components:

* Dataset/Database creation
* Backend framework and APIs
* Frontend user interface

Each of these milestones will be explored below with relevant links to the solution code.

Graphical user interface, text, application

Description automatically generated

*Figure 10. Solution file structure displaying major components of the Synopwiki Project.*

Project design:

Firstly, the overall system design will be explored. Originally the proposed system design centred around a three-component system. A singular data base to hold persistent data, a flask API to interface with the database and finally a frontend that interfaces with the API (Figure 11.).

Diagram

Description automatically generated

*Figure 11. Proposed system design of the Synopwiki project.*

As the project was developed and the requirements scope narrowed it was identified that an extra component would be implemented at the data storage level. As discussed previously Google Drive offers ample free storage and with the use of Google Drive API this allows efficient CRUD functionality to be applied to files stored on google drive. The use of Google Drive storage is a novel approach, and the implementation could prove exciting for future software projects. As the project progressed and additional use case was identified, creating an open API that allowed external users and other software projects to potentially integrate with the current project in the future. Figure 12 displays a high-level overview of the final project design implemented as part of Synopwiki development.

Diagram

Description automatically generated

*Figure 12. Box and Line diagram displaying high level overview of the final project architecture.*

## Dataset/Database creation

Dataset:

Within the Data folder the Dataset.py file was used to perform most of the data formation (collating/cleansing). Excel functions were also used split data and manually inspect data throughout out this first phase. The dataset was created from three keys sources, movieLens (1700 movies), WikiPlots(120K) and IMDB data. The MovieLens data set formed the core list due to viability for collaborative filtering modelling, discussed in the experimentation section.

The first step in the data process was to get an individual lists of movie titles from MovieLens and WikiPlots. At this stage some data cleansing had to be performed due to titles being in various formats. MovieLens has IMDB included in the data set, however some of these IDs were an older IMDB format they could not be used when querying the OMDB API. To address this issue a function was created to reformat the MovieLens IMDB IDs into a correct format (Figure 13.)

Text

Description automatically generated

*Figure 13. Function to correctly format MovieLens IMDB IDs for API querying.*

The next step was creating the file and folder structure to store the Plots data. It was decided that an alphabetical folder structure with a folder per plot name would allow for easier navigation. An example of how plot data was stored is displayed in figure 14. Each plot title had a parent folder, the individual file was stored in JSON format with the file title referencing the source and any identifiable IDs (Figure 14.)

Graphical user interface, text

Description automatically generated

*Figure 14. Example of stored plot data.*

When inserting the plots into the designed folder structure several key functions were required to look up the data, create the file path and then insert these plots into the correct folder. The primary function(add\_movieLens\_imdb\_entries) would read in a text file of IMDB IDs (Figure 15.)

Text

Description automatically generated

*Figure 15. Function to read in the IMDB IDs and add them to the correct folder.*

The next stage in this primary function was feeding into a secondary function (get\_imdb\_data\_by\_movieLens\_number) to query the OMDB API to return JSON data for the plot (Figure 13.) When this JSON data was return the title was extracted from the JSON file and passed through a regex function to format out unwanted characters (Figure 15.). The next stage in this primary function was to pass off to another secondary function(add\_imdb\_to\_plots) where the JSON would be inserted into the correct folder, this function is referenced in figure 15 (highlighted red box 2). This final function created the file name and searched the directory for a matching alphabetical folder and if none existed it created a folder and inserted the JSON file (figure 16.).

A screenshot of a computer

Description automatically generated with medium confidence

*Figure 16. Example function that created the JSON file name and placed the Plot file into the correct directory*

Some error handling was performed at this stage to ensure that failed API results were not inserted into the plots folder. Instead, these were printed in the console to be manually checked. Due to the prior data cleansing. few failed attempts occurred when inserting the movieLens IMDB data.

Now that the core MovieLens dataset had been inserted into the folder structure the attention turned to the WikiPlots data set. The next step was cross referencing the WikiPlots list of movie titles with the movieLens movie titles via string matching (figure 17.)

Text

Description automatically generated

*Figure 17. Function to compare plot title and return a list of matches.*

When the list of WikiPlots had been narrowed down via string matching, a list of 529 matches had been created. This list of titles was then re-joined to match the WikiPlots Title with the corresponding synopsis. This provided an ample opportunity to work with Pandas data frames to string match the cross referenced list with the original WikiPlots list (figure 18.) Finally, the cross referenced WikiPlots data was converted to a JSON format and inserted into the existing Plots folder.

Text

Description automatically generated

*Figure 18. Example of a Pandas function using data frames to merge a cross referenced list of movie titles.*

Once the core dataset structure had been created this was copied to Google Drive so that it could later be accessed via Google Drive API (figure 19). The final stage of the dataset creation was creating a lookup table that corresponded to the Google Drive File ID so that each file could be efficiently queried. To achieve this, a CSV file of the Plots dataset was created, using Google Drive API a list of the Google File IDs were returned in alphabetical order so that they could be added to the CSV lookup table (figure 20.)

A screenshot of a computer

Description automatically generated with medium confidence

*Figure 19. Core plot dataset hosted on Google Drive*

A screenshot of a computer

Description automatically generated with medium confidence

*Figure 20. Example of the plot lookup table containing Google File ID information.*

Database creation:

Now that the core data had been created the next stage was creating a database structure so that the core requirements of the backend API and frontend interface could functions as expected.

Diagram

Description automatically generated

*Figure 21. Synopwiki database structure ERD diagram.*

Illustrated in figure 21 is and entity relationship diagram (ERD) of the database structure of Synopwiki. The database structure was designed based on the core requirements of the project. Focusing primarily on storing plot data, collaborative filter data and user rated data. All tables in the ERD are SQLite database tables, with the exception of the Google Drive storage. This is a separate entity that is queried using the Google Drive API.

The PlotLookUp table as referenced in figure 20, stores a list of accessible plots. The card layout when navigating the main plots page is constructed via data from this table. The PlotLookUp table also has a vital function of storing the google file ID so that the google drive API can be efficiently queried on this.

The User table is designed so that user activity could be tracked across the site and attributed to each user. This has a relationship with may database tables for this reason. Additionally it also has links to the UserTypeTable, this is so that the user permission level can be selected and that differentiation can be achieved between registered and unregistered system users.

The PlotCollabFilteringScores table stored data for the collaborative filtering scores that were obtained using the FastAI.py algorithm detailed in the experimentation section below. This table has a foreign key like with the PlotLookUp table as each plot has associated Collaborative filtering scores.

User collected data was stored across four primary tables. PlotSource, PlotRating, UserPlotLabels and DimensionLabels. PlotSource stored data in relation to any additional sources that a user may add. PlotRating stored any user rating for plots selected by users. UserPlotLabels stored user selection based on labels that they attributed to each plot. Finally, DimensionLabels stored data pertaining to what label name should be attributed to each dimesion. Each of these tables had a foreign key relationship with the user table so that user activity could be attributed and later analysed. UserPlotLabels and DimensionLabels both had a foreign key relationship with the Labels table.

The Labels table is designed as a lookup table to store label names, this provides flexibility if more labels are to be added in the future.

## Backend framework and APIs

The backend code is all stored within the Flask Backend folder (figure 22.) It is composed of four primary files:

* flaskApp.py – central point to configure and run the app
* models.py – interface with the SQLite database
* googleApi.py – Query google Drive storage
* apicontorller.py – Store routes for API requests

FlaskApp.py:

The backend is run using the flaskApp.py file (Figure 23). This file is the centre point when running the backend. There are several key settings to highlight. The app configuration is set here to allow connection to the database. SQLAlchemy and Flask-Marshmallow data handler are initiated here so that the API can effectively query the SQLite database. The Cross-Origin Resource Sharing (CORS) mechanism is also instantiated, and this is required when hosting the app outside of local host. Finally, the Blueprint module handles the API views and any static files that are to be fronted to the user.

Graphical user interface, text, application, chat or text message

Description automatically generated

*Figure 22. Folder and file structure of backend files*

Text

Description automatically generated

*Figure 23. FlaskApp.py code structure.*

Models.py:

The Models.Py file stores the interface structure with the SQLite database. Within this file two key python modules are used. SQLAlchemy is used as an Object Relational Mapper, there are numerous benefits of using an ORM as opposed to raw SQL, including security and quick query building. Marshmallow which is the second used with the models.py file, Marshmallow excels in data serialization. Using these modules in conjunction enable a robust and highly scalable system that can be transferred between different database types such as MySQL or Postgres.

The Model.py file follow a similar structure throughout each database table is mapped using a class that reflects the columns and settings set up at the database level. Marshmallow is then used to create a schema of this database class so that serialised data can be returned (Figure 24.) Text

Description automatically generated

*Figure 24. Example of database models in the Models.py file.*

googleApi.py:

The googleApi.py file houses the functional code so that users can connect to Google Drive storage and perform various CRUD operations. When setting up it requires Oauth authentication and several security files stored with the folder structure (credentials.json/token.json). Google provide documentation for connecting to Google storage using Google Drive API (<https://developers.google.com/drive/api/guides/enable-drive-api>).

When performing operation using Google Drive API each function requires a connection and build function prior. To reduce repeated code these functions were refactored so that they could be called each timer for various CRUD operations (Figure 25.)

Text

Description automatically generated

*Figure 25. Refactoring Google Drive API methods to reduce repeated code.*

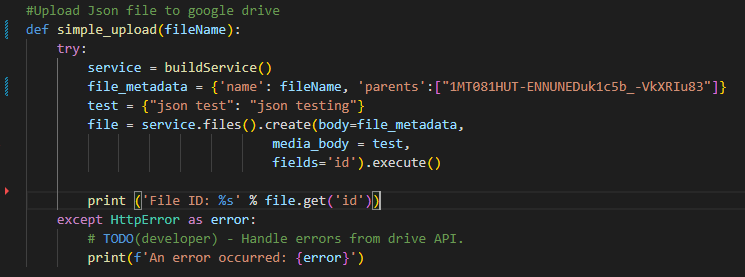
One of the key functions used in the googleAPI.py file is the DownloadJson function, this function accepts a file ID parameter, it then connects to google storage, searches on the file ID, downloads the file and returns the downloaded file (Figure 26.). This function is used in the API to display a JSON result when queried on the correct Google File ID.

Text

Description automatically generated

*Figure 26. Performing a file download using Google Drive API.*

Another key function is simple\_upload (Figure 27.) This function enables the ability to upload a file to google drive storage to a specific parent folder. Unfortunately, due to time constraints this functionality has not be implemented at the user API or user Interface level as additional testing is required.



*Figure 27. Example of a file upload to google drive storage.*

ApiContoller.py:

ApiController.py is the performs the core functionality in terms of combining the above function together. It is responsible for querying the database and returning data based on passed in parameters. There are numerous routes in the API that perform CRUD operations across all database tables and Google Drive Storage.

An example of a GET request with query parameters is the getPlotById route (Figure 28.) This function takes an ID query parameter and passes this to the SQLalchemy query that interfaces with the PlotLookUp table, the query result is then passed into a Marshmallow schema to serialise the data. This result is then checked to enure that it is not none and returns the result object in JSON format.

A screenshot of a computer

Description automatically generated with medium confidence

*Figure 28. Example of a GET request function on the Synopwiki API.*

AddPlotRating function is an example of a POST request route on the API. This route has API key authentication, if the API key supplied is not authenricated it returns an Invalid API Key result to the user. If the User provides a valid API Key the POST request is processed. The Posted variables are set and passed into a SQLAlchemy object before being comitted to the UserRatings table.

Text

Description automatically generated

*Figure 29. Example of Post request on the Synopwiki API.*

A final example of an API route is the getPlotByGoogleName route (Figure 30.) This route integrates with the previosuly discussed googleAPI.py file to use the getGoogleFileNameLike and downloadJson functions (Figure 26.). This routes accepts two name parameters, logic determines if one or two names have been supplied in the URL. The Google Drive API is then queried using these files names to return a Google File ID. This File ID is then passed the the DownloadJson fucntion and finally displayed to the user user via the API.  
  
Text

Description automatically generated

*Figure 30. Example API call integration with the Google API to return a file based on the name parameter.*

List of API calls

**GETS:  
Open calls:**<http://philipsheerin.pythonanywhere.com/api/plots>   
<http://philipsheerin.pythonanywhere.com/api/plot?id=1>   
<http://philipsheerin.pythonanywhere.com/api/plotName?title=101>   
<http://philipsheerin.pythonanywhere.com/api/collabFilters>  
<http://philipsheerin.pythonanywhere.com/api/collabFilters?id=1&label=00>   
<http://philipsheerin.pythonanywhere.com/api/collabFilterScale?label=00&order=desc>  
<https://philipsheerin.pythonanywhere.com/api/googlePlot?id=1WW9LXDXB4PeeYGoMW2lr7uM_9sca05Op>  
<https://philipsheerin.pythonanywhere.com/api/googlePlot/search?name1=101>

**Functional calls:**  
<https://philipsheerin.pythonanywhere.com/api/getUser?apiKey=12345&userId=10>  
https://philipsheerin.pythonanywhere.com/api/getCurrentUser

**PUTS:**<https://philipsheerin.pythonanywhere.com/api/updateUser?apiKey=12345>  
<https://philipsheerin.pythonanywhere.com/api/>  
https://philipsheerin.pythonanywhere.com/api/

**POSTS:**<https://philipsheerin.pythonanywhere.com/api/addPlotRating?apiKey=12345>  
<https://philipsheerin.pythonanywhere.com/api/addPlotLabel?apiKey=12345>  
<https://philipsheerin.pythonanywhere.com/api/addDimensionLabel?apiKey=12345>  
<https://philipsheerin.pythonanywhere.com/api/addPlotSource?apiKey=12345>  
<https://philipsheerin.pythonanywhere.com/api/addPlot?apiKey=12345>  
<https://philipsheerin.pythonanywhere.com/api/logout>  
<https://philipsheerin.pythonanywhere.com/api/register>  
<https://philipsheerin.pythonanywhere.com/api/>login

**DELETES:**<https://philipsheerin.pythonanywhere.com/api/deletePlot/1652>  
https://philipsheerin.pythonanywhere.com/api/deleteUser?apiKey=uqwerty&userId=6

The API Controller has a large number of routes there has been some refactoring to reduce the number of routes while being careful not to remove functioanlity. However, reducing these routes further to improve robustness would be advantageous, an example of this would be to combine GET, POSTS, PUTS and DELETE request for a particular database model into a single route and use logic to filter the request type.   
  
A futher area for improvement is the authentication on the API a more robust method could be achieved by implementing futher credential validation. Currently the API has a function that can authenticate based on valid and invalid API keys.

Finally error handling is an aspect of the API that needs further development, each function does have error handling in the event of exceptions however providing the user with specific errors to better indentify issues would be a worthwhile investment.

## Frontend user interface

The user interface was built using React.js the files pertaining to this are stored in the ReactFrontend folder (Figure 31.). The aim of the frontend component was to allow the user to traverse the dataset and give the user an opportunity to add data. The key benefit of using react was the use of reusable components. The app is split into various folders that we will explore.

Graphical user interface, text, application

Description automatically generated

*Figure 31. Folder Structure of the react frontend*

The App.js component takes care of the routing throughout the frontend (Figure 32.). This also contains basic skeleton of the page with the Navigation and Footer components referenced here.

Text

Description automatically generated

Figure 32. App.js component that handles routing throughout the frontend.

Th pages folder will be explored next, this folder stores the components for each page, that same components that are available in the App.js router. Some of these pages hold static HTML data that is displayed to the user with no additional components, such as the Home.js component. However, many of the page components are constructed using various other components stored in the components folder. An example of this is the PlotsPage.js (Figure 33.) The Plots page is the page that displays a list of plot cards in a paginated fashion (Figure 6.). If we delve into the SearchPlotData component, we can see that this component is also constructed using multiple functions. This trend continues throughout the frontend as this is how the react framework is designed to work.

Text

Description automatically generated

Figure 33. Code structure for the PlotsPage.js

Continuing to focus on the SearchPlotData component it should be noted how the data is retrieved and can be searched before returning the result to the user. Using useEffect function and axios the data can be retrieved from the Synopwiki API and stored in the data variable (figure 34). This data variable is then passed into the PlotCards component via the SearchPlots function (figure 35). The PlotCard component uses an array mapping function to map the data to a Card component.

Text

Description automatically generated

*Figure 34. Example of useEffect and Axios returning data from the API.*

Text

Description automatically generated

*Figure 35. Example of data being mapped to a card component.*

When the user updates the search option in TextField on the frontend the searchPlots function is updated, performing an array search on the data variable, and returning the data that matches the user inputted option. During testing it was noticed that this functionality was demonstrating poor performance when loading search results, which could be due to the Array search function not being optimised. A potential update that could improve performance is to move the search functionality to the server side so that the API performs the search and returns expected results.

Next, we will explore a specific plot page and how the user can input a plot rating to send to the API for processing. When the user is submitting a plot rating for an individual plot this is handled in the front-end component components/UserPlotData/AddPlotRating.js (Figure 36.) This component is a form that builds a POST request to send to an API endpoint (http://localhost:5000/api/addPlotRating?apiKey=12345):  
  
Text

Description automatically generated

Figure 36. Example of user rating POST request.

The form sets the rating selected by the user, the plotId and userID are automatically set without user input. The Post request is stored in the plot variable on line 14, when the user submits, the request is sent to the API endpoint for processing and adds this data to the database.

Similar architecture is used throughout the frontend. Each page is built with multiple components and these components pass data between each other and send and receive data with the Synopwiki API. Due to time constraints, there are several features that need continued development regarding frontend functionality.

User authentication and sign in functionality is not yet implemented with the backend server and only functions on the client side.

Frontend validation has not yet been developed to feedback relevant information to the user when POST, PUT or DELETE requests have been sent to the server.

Adaptability of the UI design on smaller devices is an area that needs further development so that mobile user can have a pleasant experience.

Overall, the core functionality of the frontend has been achieved however further development is needed.

# Experimentation

As discussed earlier collaborative filtering data formed the basis of the dataset. An aspect of this project utilised a machine learning component to extract label like data from the MovieLens dataset using a collaborative filtering model to sort data across multiple dimensions (Menon, 2021). MovieLens provides a dataset of 1700 movies, this includes data pertaining to plots rated by users.

First step when creating the collaborative filter model is transforming the data. Pandas a data manipulation model in Python is used to pass the plots and user rating into data frames and merge these together (figure 37).

Text

Description automatically generated

*Figure 37. Passing movieLens data into Pandas data frame.*

Using FastAIs library, a collaborative filtering model can be trained using the inbuilt collab\_Learner function. Next the Movie biases and Movie Weight are calculated on the whole list of plots to account and offset the potential for user bias (figure 38).

Text

Description automatically generated

*Figure 38. Movie biases are calculated on the whole data set and the Movie weights are calculated.*

Finally, Principal Component Analysis (PCA) is used to sort the plots into a reduced number of dimensions. Using PCA allows the data to preserve patterns and trends while making the model computationally efficient. These results were stored and exported to CSV to be used with the database (figure 39, figure 40).  
  
Text

Description automatically generated

*Figure 39. Analysing and sorting the plots across 10 dimensions.*

Table

Description automatically generated

*Figure 40. Example of exported data from FastAI collaborative filtering model.*

For the current project the collaborative filer model sorted the plots into 10 dimensions. It was hypothesized that each dimension would translate to an individual label, each plot had scores ranging from negative 2 to positive 2 on each dimension. The closer the plot appeared to positive 2 the more the plot embodied that label or dimension. The rationale behind this is that many individual users are drawn to enjoy similar movies with similar plot arcs, therefore if a user rates a movie highly it is likely to contain a feature, they enjoyed in previous movies they have encountered.

*Graphical user interface, website

Description automatically generated*

*Figure 41. Example of top plots associated with the first dimension (label 00).*

The current project implements functionality that enables users to suggest labels for the identified dimensions.Displayed in figure 41 are the top movies associated with the first dimension, this gives the user the opportunity to select a predefined label that best describes these plots. This is then stored in the database so that data analysis can later be performed to identify if there are any trends or recurring labels that best describe this dimension.

# Testing

Testing is an essential component of the software development lifecycle. Test driven development (TTD) is a relatively modern process of developing software where writing and testing code is interconnected, benefitting in terms of cutting costs due to lower bug counts and overall improved code quality (Borle et al, 2018; Tosun et al. 2018).

**Testing strategy:**

Due to the supported advantages of TDD, this testing strategy that was employed for the current project. It was decided that manual testing at each stage of writing code would be the optimal process for the current project due to its experimental nature. Manual testing formed the basis of the test-driven development lifecycle of the project. Each method written was manually unit tested until the expected outcomes were achieved, following this the method was refactored to streamline the overall code structure. This cycle was repeated for each part of code used throughout the project from the backend to frontend structure.

As the project began to function closer to the expected requirements, system testing ensured that the project was achieving the expected outcomes. Due to the adherence to TDD this stage of the process allowed issues to be discovered quickly and fixed. An example of this was moving the API from local host hosting to being Hosted on Pythonanywhere, the API failed to function correctly when hosted at first, however it was quickly identified that CORs authentication was missing.

Outlined below are the four major targets of the testing strategy.

**Data gathering**

Gathering, collating, and cleansing data was the foundational stage of the project. The aim of testing during the data gathering stage was ensuring the data gathered and collated was high quality, free from anomalies and in the expected format (figure 43). When gathering data, it was treated that any data was expected to have anomalies. Therefore, many of the dataset functions had error logic included in the code to catch inconsistencies (figure 42). Although error logic was efficient in catching inconsistencies, much of the data gathering testing consisted of manual inspecting the data and manually searching the data tables for anomalies.

Timeline

Description automatically generated

*Figure 42. Example of error logic to catch data inconsistency during data collection.*

Text

Description automatically generated

Figure 43. Example of method testing data console outputs before writing to csv

**Backend Development Testing**

There were several software technologies that were implemented in the backend. As each of these were developed TTD approaches continued and extensive manual testing was performed on each method to ensure that the expected results were produced. One challenge was testing the interface between the developed Flask API with the google drive API. Ensuring that data returned from the google drive API was in the desired JSON format and was fronted from the Synopwiki API in the correct format.

**Frontend Development Testing**

React framework allows for efficient feedback of any changes made to its files using the hot-reload function. This allows for quicker testing time and to ensure the user interface is working as expected. A challenge that came with this was during deployment. Due to the nature of hosting projects on PythonAnywhere the react configuration files required alteration so that the Flask backend hosted static react files. This change created a block when testing frontend changes, not allowing for the hot-reload when usually developing a react project. For this reason, an identical project was created with the original React configuration so that updates to the frontend could be tested at a greater efficiently, without the need to rebuild the solution after each change (figure 44).

Graphical user interface, text, application

Description automatically generated

Figure 44. React file setup for static deployment vs testing

**Deployment**

When deploying to PythonAnywhere it was vital to test that the project was performing to the same expectation as when hosted on a local server. This involved testing previous use cases, ensuring that the endpoints were correct and that the site could function correctly.

**API testing**

The Postman app was used to test all API URL endpoints. This included validation checking and penetration testing to test the integrity of the API security. Successful requests and errors were tested to ensure the API was responding correctly, e.g., ensuring requests that required API Keys did not return data when requested with an invalid key.

API requests were tested in two scenarios, firstly, local host API requests ensuring correct data was returned and sent (Figure 46). Secondly, API requests were tested when hosted on PythonAnywhere. In both scenarios the API performed as expected (Figure 45).

Graphical user interface, text, application, email

Description automatically generated

*Figure 45. Example of a GET request to API hosted on PythonAnywhere*

Graphical user interface, text, application, email

Description automatically generated

*Figure 46. Example of a POST request to API testing writing data to the database.*

# Evaluation and Conclusion

The aim of the current project was as follows: Collate a foundational dataset from labelled and unlabelled data that can be utilised by supervised learning ML algorithms; Create an open API where this data is available to external users; Create a user interface that allows users to traverse this dataset and feedback labelled data into the system. Overall, the project can be considered successful each of the Key functional requirements outlined in the beginning have been largely achieved, although it should be noted some components are more streamlined than others and improvements can be made at all aspects of the project. There were a number of challenges throughout the software development process. Namely learning a new technology stack entirely. A personal aim when completing this project was to dive into areas of software development I had little prior knowledge of, to stretch my knowledge and test applying my learned software principles in this environment, this began by learning Python syntax. Developing this project took a full stack approach most aspects of the software lifecycle were attempted during this process including, data collection, data analysis, system architecture, backend development, API integration, frontend development and deployment. Next, we will explore how successful the project was in relation to each of the function requirements.

Firstly, the dataset creation aspect of the project required extensive allocated time to research best practices and ensure that anomalies were minimal. Comparing datasets and using error logic to remove anomalies was a relatively new approach. However, this experience deepened my knowledge of how important datasets can be to the success of a project. Ideally integrating a larger dataset would have been beneficial however the created dataset proved effective. When collating the two primary datasets, a 100% string match algorithm was used, to potentially match more titles between the datasets a percentage string matcher such as Fuzzy Wuzzy could have been used to look for highly similar titles.

The next stage of the project was developing an open API and integrating this with Google Drive API. This was an area of the project were progress excelled. The open API proved successful in performing operations on the database and integrating with Google Drive API. Using SQLALchemy and Marshmallow proved a challenge but as the project continued the benefit of these modules was seen. Specifically, Marshmallow Schemas for serialising data, proving especially effective when passing this data to the end user, reducing the need for added data formatting. It can be contended that the most successful aspect of the project is the use of Google Drive storage and using Google Drives API to perform operations. The prospects to use this for future projects is exciting especially due the free use aspect.

The next outlined requirement was creating a user interface that users could use to search navigate the dataset, this was achieved with the frontend framework allowing users to traverse all plot information. Two key areas were identified that need further development in terms of traversing the data. The first is the performance issue when searching for plots, this appears under performant and shifting this functionality to be handled server side may be a appropriate action to address this. The second area is displaying of the wiki plots data via the frontend when viewing individual plots, the backend functionality is in place however further work will need to be carried out so that this can be fronted to the user. Beyond of these issues, the user interface traversal of the dataset can be considered a success.

Functionality that allows users to view related movies was the next aspect of success criteria. There were two areas on the site where this was to be achieved. Firstly, when viewing specific plots, a list of similar plots should be shown to users. Unfortunately, this functionality was not achieved under the current conditions, further work needs to be completed in the backend to filter related plots based on collaborative filtering scores. The second area for this to be achieved was in the label section on the frontend, so that users could view plots that were largely associated with a particular label. This was achieved and is currently working as expected, when users navigate to the labels tab any plots they view are the top-rated plots associated with that label.

The next key requirement was functionality for users to label the dataset so that the dataset could be enriched. This was achieved in three ways. First the users can rate individual plots on a scale of 1 to 5, these rating are then stored in the database. The second way this was achieved was via a labelling function that allows users to select which label they think best describes the current plot, when the user selects this the data is stored in the database. Finally, this functionality is achieved via allowing users to select from a list of predefined adjectives that best describe the ten plots associated with a particular label. Again, this data is stored in the database so that later data analysis can be performed.

In conclusion the developed project proved successful, the frontend component of this project requires further development to complete functionality and streamline the user experience. However, the dataset and backend components are in consistent and reliable state that will enable future projects to build from. Attention should be drawn to the potential use of Google Drive API in any future software development projects that requires database structure.

**Future research**

Future research can potentially build on the foundational dataset created as part of this project and aim to integrate further synopsis sources to the already 1700 entries. Additionally, Google Drive API offers a greater potential that can be used in future work that offers a free alternative to a paid database model, however caution should be taken to investigate the scalability of this venture.

# References

* Codecademy. (2021, November 21). 7 Top Machine Learning Programming Languages. Codecademy. https://www.codecademy.com/resources/blog/machine-learning-programming-languages/
* Borle, N.C., Feghhi, M., Stroulia, E., Greiner, R. and Hindle, A., 2018. Analyzing the effects of test driven development in GitHub. *Empirical Software Engineering*, *23*(4), pp.1931-1958.
* Menon, R. (2021, December 21). Collaborative Filtering Using fast.ai. Towardsdatascience. <https://towardsdatascience.com/collaborative-filtering-using-fast-ai-f33cfea62812>
* Pal, N. (2020, December 2021). How Amazon’s UI UX Design has made it one of the most glorified online shopping destinations? https://www.cronj.com/blog/usable-features-of-amazons-ui-ux-design/
* Rahwan, I., Cebrian, M., Obradovich, N., Bongard, J., Bonnefon, J.F., Breazeal, C., Crandall, J.W., Christakis, N.A., Couzin, I.D., Jackson, M.O. and Jennings, N.R., 2019. Machine behaviour. Nature, 568(7753), pp.477-486.
* Tosun, A., Ahmed, M., Turhan, B. and Juristo, N., 2018, May. On the effectiveness of unit tests in test-driven development. In *Proceedings of the 2018 International Conference on Software and System Process* (pp. 113-122).
* Voskoglou, C. (2017, November 21). What is the best programming language for Machine Learning?. Towardsdatascience. https://towardsdatascience.com/what-is-the-best-programming-language-for-machine-learning-a745c156d6b7

Documentation reference links:

* Flask (November 2021) Flask Documentation. <https://flask.palletsprojects.com/en/2.0.x/>
* SQLite (December 2021) SQLite Documentation <https://www.sqlite.org/docs.html>
* SQLAlchemy (January 2022) SQLAlchemy Documentation. <https://flask-sqlalchemy.palletsprojects.com/en/2.x/>
* Google Drive API. (December 2021). Google Drive API Documentation <https://developers.google.com/drive/api/v3/quickstart/python>
* React.js (February 2022). React Framework Documentation: <https://reactjs.org/docs/getting-started.html>
* Material UI. (February 2022). Material UI documentation: <https://mui.com/material-ui/getting-started/installation/>
* PythonAnywhere. (December 2021). PythonAnywhere. <https://help.pythonanywhere.com/pages/>