Recommend a Movie - Find that Friday night movie with ease

Anton Golubev

B.Sc.(Hons) in Software Development

April 10, 2022

Final Year Project

Advised by: Dr Dominic Carr

Department of Computer Science and Applied Physics
Galway-Mayo Institute of Technology (GMIT)

Contents

Chapter 1

Introduction

In the beginning of my final year of college, we were told that we must undertake a final year project. This project would be worth 25% of our entire degree. We were told that we have the choice of carrying it out either as a group or individually. After much thought, I decided to take on this task by myself. While I have enjoyed working on team projects throughout my degree, I wanted to test my personal limits and examine my ability to complete a project of this scale solely on my own. I felt that I had grown significantly in my academic and practical skills throughout my degree to date, and decided I was ready to test myself and assess the extent of my personal development in completing this project alone. Another reason for this choice is that I had deferred a year of college in 2020 due to COVID-19 and due to this I didn't know any students in my course when I rejoined the degree in 2021. Hence, I didn't know what the abilities of the students were. To me it seemed that the talented people would partner up because they knew each others' abilities from working together the previous 3 years of college, and the rest would be left without a group. Due to this reasoning, I decided to complete this project by myself. I didn't want to chance any issues of miscommunication and under performance in such an important project. My initial idea was to create an e-commerce web application with CRUD functionality. However, after much thought I decided to scrap the e-commerce idea and create a web application that recommends movies to a user.

1.1 The Idea

Consider the following situation: It is a Friday evening and you have decided to stay in, have a quiet night and relax and enjoy a movie. However you

simply cannot decide what to watch. You scroll through IMDB lists looking for a movie that will catch your attention. However, no movie seems to be popping out at you. You end up wasting your time by scrolling for an uncomfortable amount of time just to end up landing on an average movie. Even worse, you might end up giving up and pulling the plug on movie night. I have had this experience many times, and each time is just as disappointing as the last. This is what gave me the idea to create an application that could recommend you a movie to watch based on movies that you have already seen.

1.2 Project Standard

We, as a year, were informed that this project needed to be of a Level 8 standard. This meant that it must use a combination of programming languages, frameworks and technologies. This includes ones that are unfamiliar because one of the key skills that a software developer must possess is the ability to learn new technologies quickly, be it a language, framework, ORM (Object Relational Mapper) etc.

1.2.1 Technology Stack

In order to adhere to the above requirements, I decided after much research, to use the React library for the front end of the application. The use of React also implies the use of JavaScript alongside HTML and CSS because React is a JavaScript library. In order to challenge myself and learn something that was not covered in coursework in my time at GMIT, I decided to learn how to use functional React components as they are quite lightweight and powerful compared to class-based components. This is because functional components use functions called ”hooks” that let you achieve the same functionality as class-based components with a lot less code.

I decided to use Python and the Flask library to design the back-end. Python is a language that was covered very briefly in college. Through my research it was clear that Python is one of the most used languages in the industry and because of this I wanted to challenge myself to improve my knowledge of it. When deciding on a framework for the back-end, I had a choice between Django and Flask. In the end I decided to go with Flask as is it a micro framework which is lightweight, just like functional React components. Also, the recommender website will be a single page application. A single page application is a web application that is rendered and updated piece by piece. Instead of loading an entire new page from a server each time

something changes in the web app, a single page application only changes the specific component that needs to be changed. According to research that I carried out, Flask is perfect for designing an application such as this.

In order to keep track of users and movies that the users like, a database is needed. I decided to use PostgreSQL partly due to the fact that I had not used it before in my college course work, but also because it is most suited for applications that handle data analysis, and where write speeds are crucial. Since I plan to use a large dataset of movies, PostgreSQL seemed like a good fit.

1.3 Objectives

The primary goal of this project is to develop a single page web application that recommends movies to a user. In order to achieve this end goal the following functionality must be present in the final application:

- The web application must display a vast selection of movies. This is necessary because the application would not be of much use if the user has a limited choice of movies that they can get recommendations to.
- A user must be able to register an account.
- A user must be able to log in to that account.
- When a user is logged in, they should be able to like or dislike a movie. Liking a movie will affect the recommendations a user receives, while disliking should remove it from the user's recommendations.
- A user should be able to remove a movie from their likes. This should remove it from their likes and remove recommendations for that movie.
- The user must be able to search for any movie.
- Movies that the application displays must show details and a description of these movies in order to simplify the decision making process for the user.
- A user must be able to filter movies by genre, again to simplify the decision making process of picking a movie.
- Finally, once the user adds a movie to their likes, then the web application should recommend similar movies to that user.

Another goal is to deploy this project onto a cloud hosting service, so that it can be accessed by anyone at anytime. The service I picked for this purpose is Heroku. This is due to the fact that Heroku offers a ready-to-use environment which makes pushing code easier compared to a service such as AWS where the deployment is quite difficult.

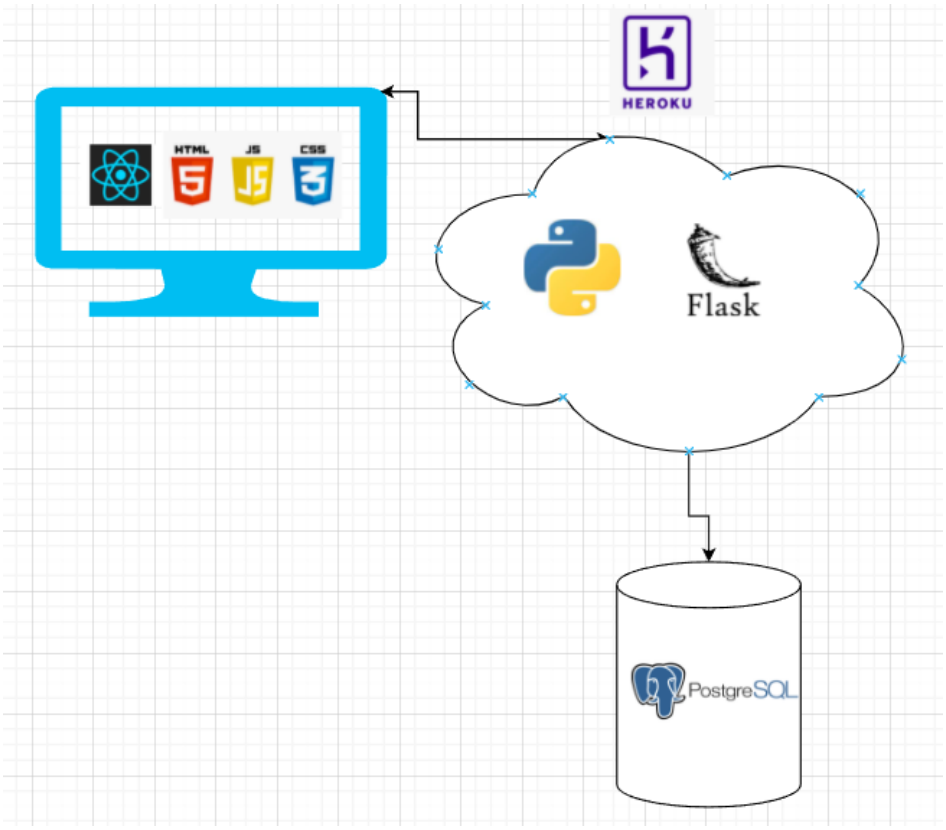

Figure 1.1: A diagram of the planned architecture of the system

1.3.1 Secondary Objectives

The secondary objective of this project is to develop and improve my skills in technologies that I haven't already learned but also in technologies that are used widely in industry. The React library that I will be using is the second most used JavaScript library in industry currently, [1]. while in America, Python is the second most desirable language looked for when hiring new developers. [2]

1.4 Overview

This dissertation is organised into relevant chapters that explain how and why the project was completed. This section contains a brief description of each chapter.

1.4.1 Methodology

This chapter details the steps that were taken in the development stage to guarantee the project's success. It will describe my approach to development, testing and version control.

1.4.2 Technology Review

This chapter will in-depthly examine the technology that was used in the development of this application. Every piece of technology is described at a conceptual level. The process of setting up each piece of the technology is also described.

1.4.3 System Design

This chapter will describe the architecture of the entire application. It will describe the different components of the system in detail and how they interact with one another. It will also contain diagrams of the system's final architecture and code snippets.

1.4.4 System Evaluation

This chapter will discuss the performance of the system as a whole. This entails the discussion of robustness, scalability and code/structure quality.

1.4.5 Conclusion

This chapter reflects on how well the objectives given in the introduction were met. Difficulties in experience during the development of the project will also be discussed.

1.4.6 Project Repository

The repository for this project can be found by clicking this link.

Chapter 2

Methodology

As mentioned previously, this chapter focuses on the steps taken to guarantee the success of this project.

2.1 Planning

2.1.1 Requirements

I determined the requirements for this project by analysing the each of the goals I had established in the introduction. I thought about each one individually and brainstormed ideas on how I would go about implementing each one.

2.1.2 Methodology

Once I had finalised the idea for the project, I started to think about how I would handle the development of both a back-end and a front-end as a single person team. Since I was working by myself, there is no clear development methodology that fit me perfectly. However, I decided that a Feature Driven Development style with an incremental approach would work the best.

Feature driven development(FDD) implements an incremental approach, meaning that the development team does not try to complete the whole system at once. The goal of it is to add some new feature to the system in every iteration.[3] This allowed me to make changes and improvements to the project during its development. This aspect of FDD along with its measure of success being the measure of the ability of actual software to work, made it the best choice.

After the methodology was decided, I researched the technology stack that

I wanted to use. As I mentioned in the introduction, I wanted to challenge myself to learn new technologies and so I picked JavaScript with the React library for the front-end. My research showed me that in order to run my React application I needed to use Node.js. For the back-end, I picked Python with the Flask framework including SQLalchemy with is an Object Relational Mapper that I decided to use to interact with my PostgreSQL database in the back-end. However, that is not the only reason why I picked this technology stack. I conducted research that showed that this technology stack is cohesive and would actually be viable to use.

2.1.3 Tools

After deciding what my technology stack should be, I had to consider which programming tools I should use to implement the project. I considered using PyCharm to develop the back-end because PyCharm was made specifically for Python. However after some thought, I decided to use Visual Studio Code as it allowed me to develop both React and Python code in one program.

I also decided to use pgAdmin, an Open Source administration/development platform for PostgreSQL, to configure and test my database. I chose pgAdmin as it is the most used program for PostgreSQL.

2.2 Development

In order to apply FDD to my project, I began by identifying a list of features that needed to be implemented i.e. A user needs to be able to log in. Once a feature was identified, I thought about the tasks needed to implement this feature. For example, in order for a user to log in, they must first be able to register for an account. In order for registration to work, I must save their details to a database. In essence, this meant dividing large features into relevant sub-features. I then identified how these sub-features were related, and identified features on the front-end that were coupled with features on the back-end. For example, a register form on the front-end is coupled with the back-end route that saves the user's details to the database. Once the sub-features were identified, I created a Kanban style board in order to help me keep track of everything that I needed to do. Visualising the workflow really helped me to remain organised and gave me a sense of achievement as I put items into the "finished" column of the board. When undertaking an item on the board, I conducted a sprint until I completed that feature. During the sprint, I used the incremental approach, to work coupled features on the front-end and the back-end. This made the most sense to me in the

context of my application and allowed me to develop both ends in a timely manner. Once I had progress to show, I met with Dr. Dominic Carr who gave me feedback, guidance and advice on my code and gave me useful ideas that I incorporated into the project.

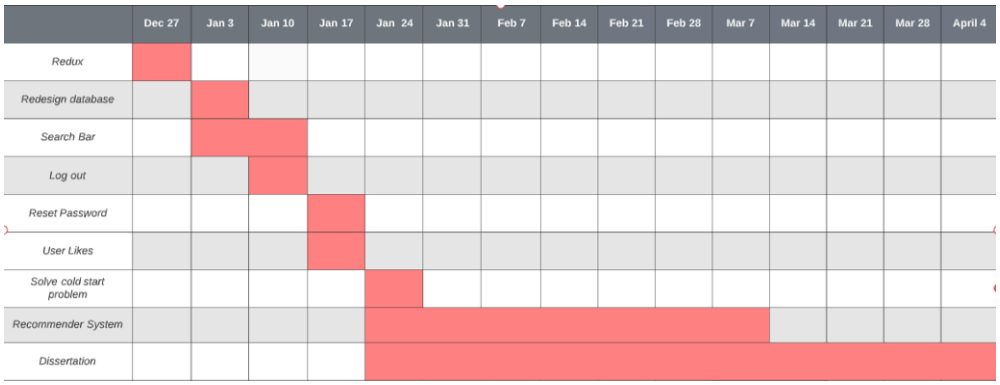

Figure 2.1: Gantt chart of planned features

These meetings would take place either weekly or bi-weekly. I also made Gantt charts that gave me an approximate idea of how long I would have to develop each of the large features I was working on. These were only a rough estimate and some features took longer than expected and vice versa but nonetheless, they enabled me to stay aware of how much time I was taking to complete each feature.

2.3 Testing

Testing is a critical part of any project. It ensures that the project is working as expected and makes sure it satisfies the customer's needs/requirements. After I finished developing any feature, I tested it immediately using White Box Testing. White Box Testing is a technique in which the internal code and structure of the project is available to the tester. I carried out these tests using a software called Postman. Postman is a platform that allows testing of API endpoints. I used Postman to send POST and GET requests to my application. I sent valid data to endpoints that accepted POST requests and examined the output to make sure it was what I was expecting. I also sent invalid data to these endpoints and made sure that some form of an error was reported when I did so. If something unexpected happened, I knew there was a bug somewhere in my code and ensured that I fixed it straight away.

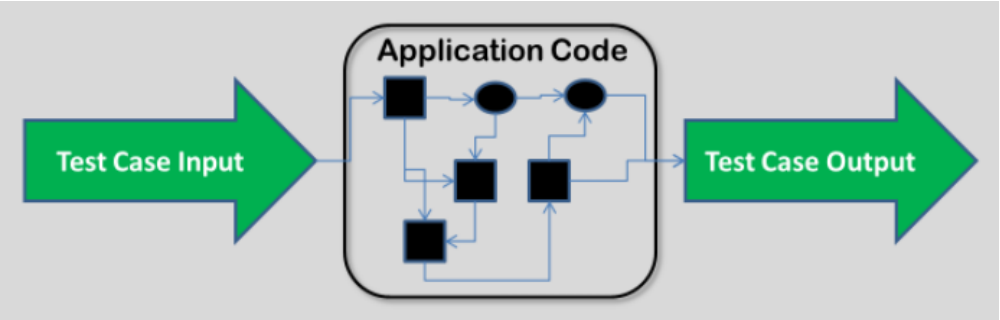

Figure 2.2: White Box Testing

2.4 Version Control

I made sure to make regular commits to GitHub during code sprints. This was a precaution in case any problems arose during development that required me to revert back to a previous version of the program. The reason Github was my version control manager of choice was that I am most familiar with it through my use of it in college coursework. In addition, it is the most used version control manager in industry and, as I mentioned in the introduction, one of my goals for this project was to increase my skills in technologies that are widely used in industry.

Chapter 3

Technology Review

This chapter discusses each of the technologies used in the application's stack at a conceptual level. Firstly, I will discuss the technologies that were used in the **front-end**.

3.1 JavaScript

JavaScript famously originated as a scripting language that was written in 10 days. However, nowadays it is now a technology that affects nearly every single human being. It was developed by a Brendan Eich while he was working in Netscape in 1999. Netscape wanted to make websites dynamic and interactable instead of the plain static pages that they were back then. This made it a very obvious and essential language to include in my technology stack. The main core concepts of Javascript are:

3.1.1 Flexible Multi Paradigm Language

JavaScript is a multi paradigm language. A multi paradigm language is a programming language that allows the developer to use whatever programming paradigm they want. A programming paradigm is just a style of programming. Basically, JavaScript allows the programmer to work in a variety of styles, be it Object-Oriented, Functional, Procedural etc.

Functional programming has some advantages including; side-effect free functions, shorter and less error prone code, and the ability to modify code without breaking the rest of the program. [4]. Functional programming in particular is powerful in JavaScript due to JavaScript's support of First-class functions. The Mozilla Developer Network defines First-class functions as "functions in (a) language are treated like any other variable. For example,

in such a language, a function can be passed as an argument to other functions, can be returned by another function and can be assigned as a value to a variable."[5] This is one reason why I wanted to implement functional components in my front-end.

3.1.2 Dynamic Language

The authors of [6] define a dynamic programming language as a language that "shares a number of common runtime characteristics that are available in static languages only during compilation, if at all." An example of this is that JavaScript allows the developer to change and modify code while the program is running, while in a static programming language such as Java this is not possible.

Dynamic languages have no variable types unlike statically typed languages. This means that variables in a dynamic language can store any type of data, and that the type of the variable is assigned at runtime by an interpreter. This feature has its advantages and disadvantages. One main advantage is that because there is no variable types, it makes the development cycle faster. Dynamic languages are also easier to teach to newcomers and students because they allow students to focus on learning important computer science concepts without getting bogged down by syntax and low-level details. The authors of [7] make a strong point that because dynamic languages such as JavaScript and Python possess a large amount of libraries, they are useful in engaging students and maintaining their interest in programming because it allows them to create useful programs quickly without the burden of learning about variable types, sockets or protocols etc. The authors of [8] express a similar sentiment. They state that in a dynamically typed language, the programmer is much less likely to run into unexpected bugs due to type inference or type systems. This makes dynamic languages in a sense less complex, and lessen the learning curve.

That being said, there are also obvious disadvantages to dynamically typed languages. Clearly, they are not type safe and can and will produce type errors because types do not need to be declared. Also, they produce machine code that will not be optimised. For this reason, dynamic languages are typically slower than statically typed languages.

3.1.3 Prototypal Inheritance

Inheritance in JavaScript is very different from inheritance in other languages. In essence, everything in JavaScript is nothing but an object because of prototype chaining, which is why it is a dynamically typed language. Whenever

you create a JavaScript object, it automatically attaches the object with another object called a prototype. This prototype has its own hidden properties and functions. This prototype also has its own prototype, and this chain continues until the base object "Object" is reached. Object "sits just below null on the top of a prototype chain". [9] Prototypal Inheritance is very powerful and allows the inheritance properties and also of methods and functions because JavaScript has First class functions. This means that when trying to access a property of an object in JavaScript, the browser searches through the object and all of its prototypes until it finds a match.

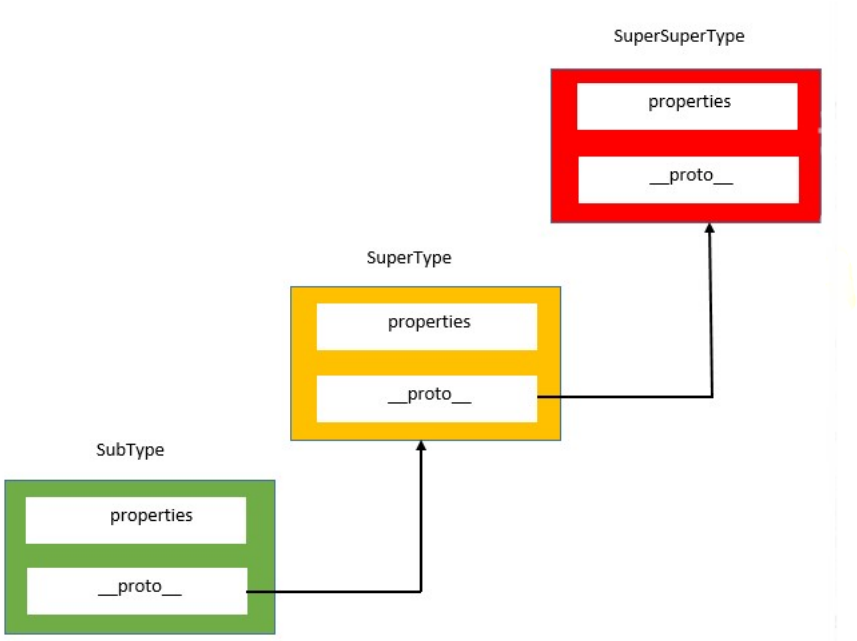

Figure 3.1: Prototypal Inheritance

3.1.4 ECMAScript

ECMAScript is a language that JavaScript derives its bases and standards from. ECMAScript was the language that created de juro standards for JavaScript branches. It was standardized by the ECMA (European Computer Manufacturer's Association)[10]. Languages such as ActionScript, JavaScript and Microsoft's JScript all use ECMAScript at their core. The current version is ECMAScript 6 or ES6

3.2 Node.js

Node.js is an asynchronous runtime that allows you to run JavaScript on a server. Node.js achieved asynchronous operations through the use of an event loop. Node.js is a single threaded process, which means that if a user makes a request that involves a heavy process then the entire application would halt in its tracks until that process is completed. To get around this problem, Node.js has an asynchronous non-blocking event loop. When users make heavy requests that would block the thread, the event loop offloads that operation to the system kernel.[11]

While thread based concurrency models such the one implemented in Java can be lightning fast in their operations, they are also very difficult to use correctly. This is due to a multitude of errors that can occur such as a deadlock, which occurs when two threads become blocked due to both of them locking a resource that the other is reliant on or if both threads are waiting for the other thread to do something. [12] Node.js documentation states that node is more efficient, and also easier to use. It is much easier because since the event loop offloads heavy tasks to the kernel, the developer never has to think about concurrency issues and the main thread never blocks. [13] These concepts of concurrency appealed to me and made Node.js a top choice to run my React front-end on.

3.3 React.js

React is a JavaScript library that was developed at Facebook and was released in 2013. It is used to build components that represent logical, reusable parts of the UI. The beauty of React is the simplicity in which a component can be built. A component in React is just a JavaScript function which returns HTML or UI written in JSX which allows you to combine JavaScript and HTML. Functional components in React really caught my attention because of their simplicity and made me want to implement them in this application, which is one reason why I chose to include it in my technology stack. Another reason it was included is that React has a massive eco system of libraries. Base react does not deal with routing , state management, animation etc. It lets the open source community create libraries for these purposes. This allows React to be lightweight because it lets you select what libraries you need in your application which seemed to be a significant benefit to me.

3.3.1 JSX

JSX is an ECMAScript feature. It stands for JavaScript XML, and is essentially a syntax extension of HTML. It is used in React to allow the developer to write HTML in JavaScipt code. It allows the conversion of HTML tags into React elements, prevents cross-site scripting(XSS) attacks [14] and is actually faster than regular JavaScipt due to it performing optimization while compiling the source code into JavaScript.[15]

3.3.2 Component Based

React uses components that allow you to split the User Interface into reusable pieces of code. In React, components receive data through props. Props are a way for components to pass data to each other. They work similarly to HTML attributes.

Also, components can be defined as either functions or classes. Functional components are a lot simpler because there is less code to write and they are more concise.

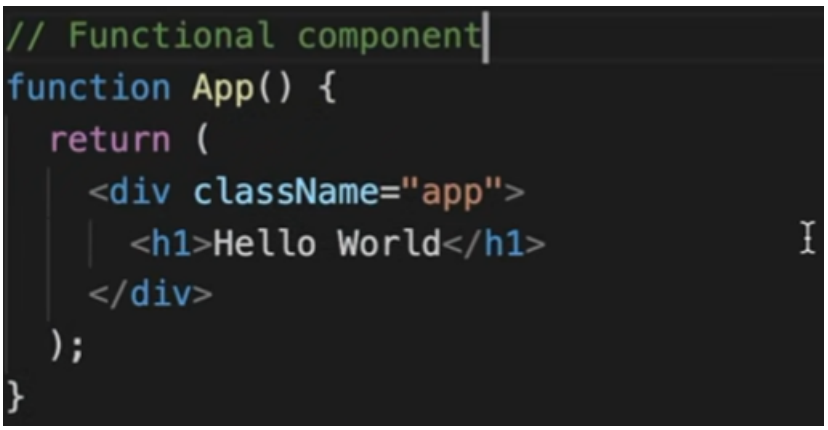

Figure 3.2: Functional Component

In a functional component, the developer is able to just simply return JSX. However, in class based components(as can be seen below in figure 3.3) a render function is required which then has the ability to return JSX. In the past, functional components could not be fully utilized because they could not support state. This meant that if you needed to add state to your component you had to convert it to a class. However the recent addition of Hooks in React allows state to be set with the useState hook. In React, it is difficult to reuse stateful logic without restructuring your component. One benefit of React hooks, as stated in the documentation, is that they ”allow you to reuse

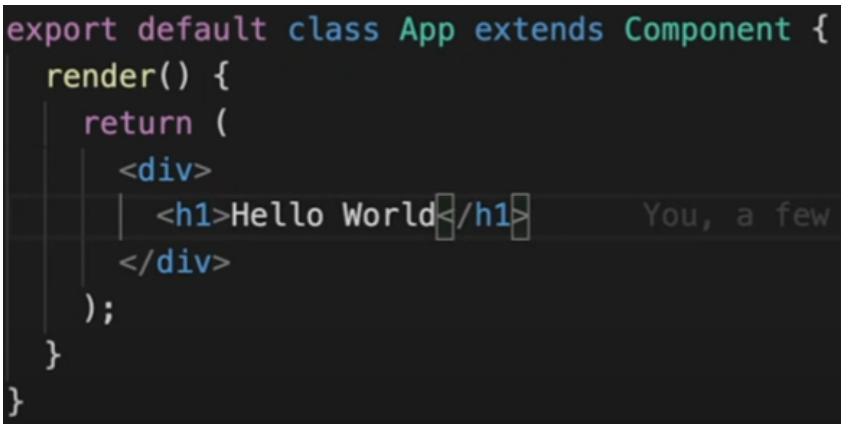

Figure 3.3: Class Based Component

stateful logic without changing your hierarchy".[16] Another Hook introduced by React was the useEffect hook, which carries out the same functions as componentDidMount, componentDidUpdate, and componentWillUnmount in class based components. It simplifies these three methods into one API. Due to how clean and simple component code looks when using function components and Hooks, I decided to learn how to implement them in this project.

3.3.3 Renderer-agnostic

React is a renderer-agnostic user interface library. This means it does not care where your UI will be displayed. For example, your React components could be rendered in ReactDOM, a package that "provides DOM-specific methods". [17] A DOM(Document Object Model) is used to change HTML with JavaScript. The DOM is always the same as the HTML, however it is represented as a JavaScript object. If anything is changed in the DOM, then the HTML automatically gets updated, and vice-versa. In this way the DOM acts as an interface for JavaScript to interact with a HTML page. [18] Basically, ReactDOM allows you to display your UI on a normal browser as a normal web page.

However the React library is not limited to the DOM, and does not care where you render your final components. It can work in a lot of different environments, with some modifications. You could display your components in their React360 library which allows you to create VR applications that run in your browser, or in React Native which is their framework for developing mobile applications.

3.3.4 Virtual DOM

React also has a Virtual DOM, which is a copy of the DOM that is created behind the scenes. Whenever you change something in your code, React changes the virtual DOM instead of the actual DOM. It then checks for differences between each equivalent node in the DOM and the Virtual DOM, in a process called **diffing**, and syncs them with one another through a process called **reconciliation**.[19]

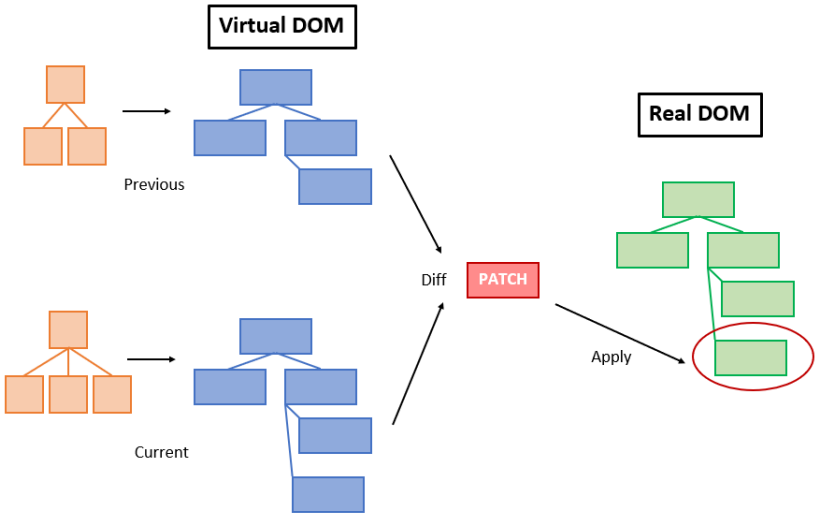

Figure 3.4: Reconciliation

If the root element of nodes in the two DOMS are different, then React destroys that node on the actual DOM and replaces it with the equivalent node from the Virtual DOM. The example below is given in the React documentation. [20]

```
<div>
  <Counter />
</div>

<span>
  <Counter />
</span>
```

Figure 3.5: Diffing nodes with different root elements

However, when the nodes of the DOM have the same root node, react compares the attributes of both and keeps the DOM node and only updates the changed attributes. The examnple below is also from the documentation.

```
<div className="before" title="stuff" />

<div className="after" title="stuff" />
```

Figure 3.6: Diffing nodes with the same root elements

This is one of the reasons why React is so powerful. The use of the virtual DOM and the diffing algorithm gives much better performance than manipulating the actual DOM directly, because to do so takes a lot of resources.

3.3.5 Material UI

I mentioned previously that React has a huge number of useful libraries, and Material UI is one of them. It is an open-source front-end library that contains a huge number of pre-built components. The reason I picked Material UI over other front-end libaries with pre-built components is that Material UI is very well documented and its components have a premium look. This is one of the reasons why several highly successful companies such as Spotify, Amazon, Netflix and even NASA use it. `https://mui.com/`

3.4 Redux

Redux is a JavaScript library that is extremely popular in the React ecosystem. It is a library that is used to store state in JavaScript applications. Basically, Redux just stores variables and their state. This is different than the component state in React because Redux is available globally in the application. Therefore it is used for things like keeping track of user details. In Redux, reducers manage state, and dispatch action push state updates to reducers. Since one of my objectives is to implement a log in system for users, and also for users to be able to like and dislike movies, I needed a state management system. Since Redux is so widely used in industry, it was an easy choice. By implementing Redux, I would complete two objectives outlined in the introduction; implement a log in system and to develop new skills in technologies that I have not used previously.

3.5 JSON

JSON (JavaScript Object Notation) is defined by the documentation as a 'lightweight data-interchange format' that is both easy to read for humans but is also easy for machines to read and generate.[21] It is standardised under ECMA-404. It was derived from ECMAScript, and hence the syntax is very similar to JavaScript objects. [22] However, JSON data is language independent and hence it is extremely popular for exchanging data in web applications and in APIs. Taking this into account, I decided to use JSON to serialise data when exchanging data between the front-end and back-end.

Next I will discuss the **back-end** technologies used in this project.

3.6 JSON Web Token

Since one of my objectives is to give the user the ability to register and log in to an account, I had to research ways in which I could implement this technology. This led me to JSON Web tokens(JWT). JSON tokens are very commonly used for web authentication because they are a secure way of trasmitting information between two parties. A JWT consists of three parts, a header which specifies the type of algorithm used in the JWT, a payload which contains information about the user and a signature which is used to verify that the message was never changed during its delivery. [23]

3.7 Python

Python is a high level interpreted language. It one of the most popular languages in the world because it is easy to learn due its beginner-friendly syntax. However, it is more than practical for serious projects. It is the language of choice for big data analysis and machine learning, which is the reason I wanted to use it for the back-end of this project. In order to get the recommendation system operational, I needed to analyse a dataset and apply some kind of machine learning algorithm to it.

3.7.1 Strongly Typed

Python is dynamically typed just like JavaScript, meaning type annotations are not necessary. However, unlike JavaScript, Python is strongly typed language. This means that in Python attempting to perform operations that are not appropriate to the type of the object will result in an exception. This

results in easier debugging because if you perform an invalid operation, the python interpreter will tell the developer. For example, if you attempted to add a string to a number in JavaScript, it would work and results in a string. However, if this was not the behaviour you intended and you made a mistake, then it would be difficult to realise because no errors were thrown. However, in Python an exception will be thrown if you attempted something like this. [24]

3.7.2 Syntax

The syntax of Python is highly efficient and can be compared to pseudocode that is executable. It allows you to declare multiple variables on a single line and to define tuples list and dictionaries in a literal syntax. Python uses indentation to terminate or determine the scope of a line of code. This eliminates the need for curly braces and semi colons found in other languages. Like JavaScript, Python is also a multi paradigm language meaning you can implement programming paradigms such as Object Orientated Programming or Functional Programming etc. Python has a huge ecosystem for third party libraries such as deep learning frameworks like tensorFlow, and machine learning frameworks such as Scikit-Learn.

3.7.3 Machine Learning

As mentioned above, Python is the language of choice when it comes to data science. The authors of [25] identify a few reasons for why this is the case. One of these is that it allows scientists to make useful programs without getting bogged down in syntax. Also, the fact that it is a multi-paradigm language allows the developer or scientist to pick a paradigm that is suited best for the type of work/analysis being done. The authors mention Python has a huge open source community number of libraries and a large number of these handle plotting, graphing, data processing and data analysis. The authors of [26] mimic these sentiments and also state that because Python is interpreted, it adds to its simplicity as there is no need to spend time ”manipulating a compiler and linker”. These reasons are also mentioned by the author of [27] who presents one other reason: Python code is highly readable and contains a lot less actual code when compared to other languages.

As you can see in the figure above, Python is second to only Ruby in terms of project size.

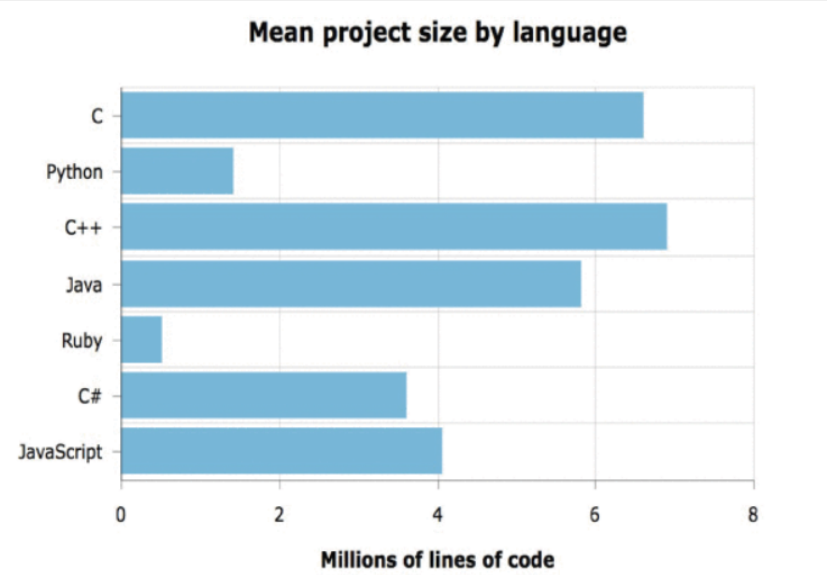

Figure 3.7: Project size by language

3.8 Scikit-Learn

Scikit-Learn is a popular Python machine learning library. From my research of Python, this library was an easy choice to use to build the recommendation system for this project. This is because it includes powerful machine learning algorithms out of the box. It contains tools for classification, regression, clustering and dimensional reduction. It also includes matplotlib which is a very useful library for visualising data. With the use of sci-kit learn, I will be able to implement any machine learning algorithm that fits best.

3.9 Flask

3.9.1 Micro-framework

Flask is a micro-framework that is used to create servers in Python. It is a micro-framework because it keeps the core of the framework very light, and allows you to add whatever extensions that you might need. This means that your project can be as lightweight as you want it to be.

3.9.2 Blueprints

Flask uses blueprints to make modular application components. A blueprint is similar to a Flask application object, the difference being that it isn't an application but a mould of how to construct a section of a web application. Blueprints allow the reuse of modules in an application.[28]

3.9.3 Flask-SQLAlchemy

Flask-SQLAlchemy is an extension of Flask that allows support for SQLAlchemy. SQLAlchemy is an ORM (Object Relational Mapper designed for Python). An ORM is a framework that enables you to map objects in Object Oriented languages to rows in relational databases such as PostgreSQL. The use of an ORM makes inserting, updating and deleting from a database much less tedious. The use of the ORM allows the developer to declare how classes should be mapped to database tables while letting the ORM deal with the actual SQL. It implements a **data mapper pattern**. A data mapper defined as a 'Data Access Layer that performs bi-directional transfer of data between objects in memory and persistent storage.'[29]

3.10 PostreSQL

PostgreSQL is a relational database. It is a powerful and flexible database mostly due to its open-source hard working community which developed a lot of flexibility and solid performance features for it. PostgreSQL is only fourth in terms of popularity according to[30], however StackOverflow's 2021 developer survey [31] puts its popularity in second place. Whatever its current place is, the fact is that a lot of tech companies are switching to PostgreSQL. Apple has been using PostgreSQL for its internal company databases for a long time now, however in 2010 they removed MySQL from the latest version of the Mac OS X server and replaced it with PostgreSQL[32]. Other big tech companies that use it include TripAdvisor, Reddit, Instagram and Spotify.

3.10.1 Open Source

One reason for the rise in PostgreSQL's popularity is the fact that it is open-source. Oracle and IBM's DB2 both charge licensing fees, and mySQL was recently acquired by Oracle, making PostgreSQL the only fully open-source relational database. For this reason I decided to use PostgreSQL as my database of choice. Due to it being open source, it will be the clear choice to

use in a startup if I ever decide to work at one or create one. It also has a lot of dedicated developers who have made useful tools that are open-source and free. Examples include pqsl which is a command line tool that comes with the installation of PostgreSQL. Another example is pgAdmin which is an administration/development platform. Also, it has endless plugin libraries for any language such as psycopsg2 for Python. I decided to use psycopsg2 to build this project because it intergrates with the Flask framework.

3.10.2 Premium Performance

While PostreSQL is not the best performing and fastest database in the world, it has a lot of features that ensure that it is always high performing and useful in production applications.

Multiversion Concurrency Control(MVCC)

This allows the developer to perform parallelization in queries. This means that PostgreSQL is able to take advantage of multiple cores on the machine that is hosting the database to result in faster performance, a feature MySQL does not have. This feature ensures that 'reading never blocks writing and writing never blocks reading'.[33]

Non-blocking Indexes

PostgreSQL allows you to create or update an index while PostgreSQL is in production, being written to or being read from without interfering with those operations.[34] This is unlike a lot of databases which require you to freeze the database, take it offline and stop all operations before you can update an index.

3.10.3 DataTypes

PostgreSQL supports a large range of data types to meet the needs of any program's storage. Users are also able to create their own types. This means that things stored under concise types in PostgreSQL would increase the performance of your application.

3.11 The Movie Database

The Movie Database (TMDB) is a 'community built movie and TV database'. It is a database that is maintained by its users. It can be found here `http:`

`//themoviedb.org`. In my objectives, I identified one important feature, and that was for the user of this project to be able to find any movie they wanted. Initially to achieve this goal I was going to find a movie dataset online and import it into PostgreSQL, and then display it to the users. However, after searching through datasets I found a lot of them to be lacking in a wide variety of movies. This caused me to look into TMDB. TMDB offers an API that allows access to their huge database of movies. Not only did the selection of movies on their database beat any dataset I could find, their database also contained relevant information that I could use in my recommendation system. For these reasons I decided to use TMDB to display movies to the user on the front-end.

3.12 Heroku

Heroku is a Platform as a Service (PaaS). It allows you to host web applications.

3.12.1 PaaS

Platform as a service is a development environment in the cloud that allows you to produce web applications.

One of my objectives in the introduction was to host this web application on the cloud so that it can be accessed by anyone. Heroku was chosen as it is free and relatively quick to deploy.

Chapter 4

System Design

4.1 Architecture

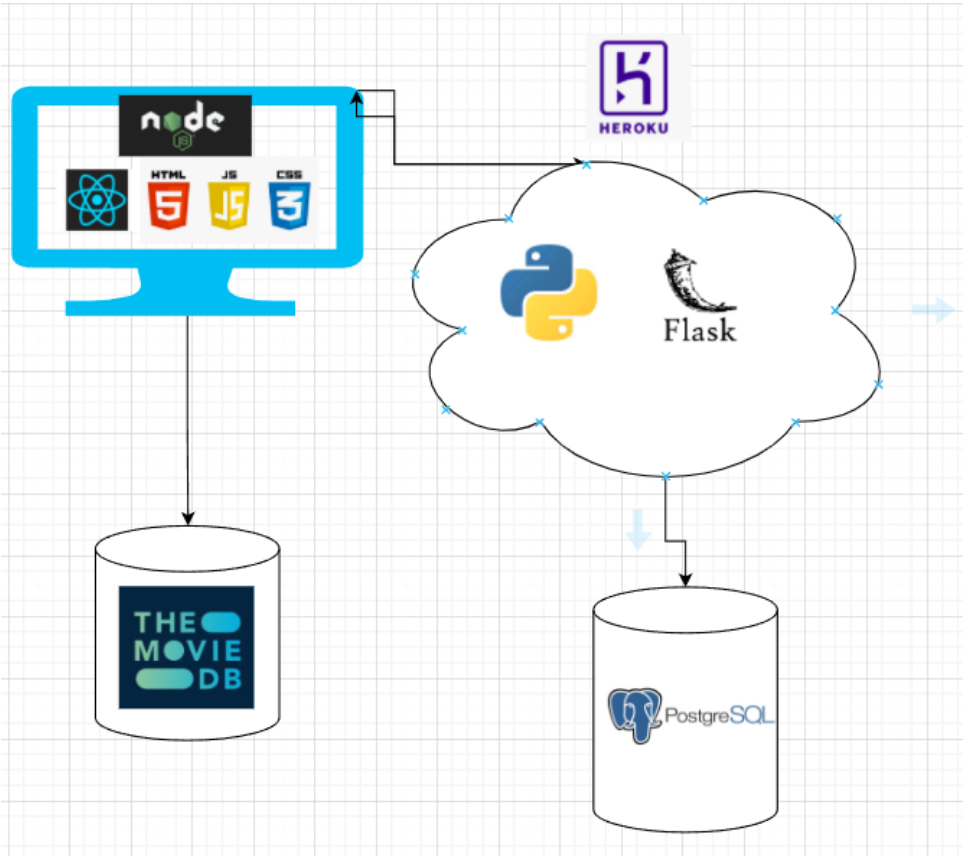

Figure 4.1: High Level System Architecture

Figure 4.1 shows a high level overview of the design of this application. In this chapter, each element of the above figure is discussed.

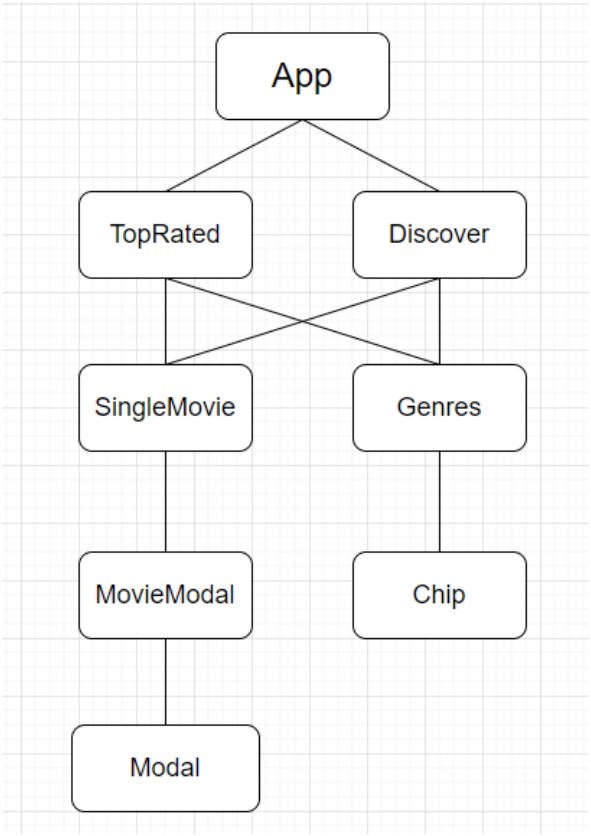

Figure 4.2: React Component Tree

4.2 Front-End

In the front end, I developed purely functional components using the useState Hook in React for any state variables, and also the useEffect hook that replaces the functionality carried out by componentDidMount, componentDidUpdate, and componentWillUnmount in class components, as mentioned previously in the Technology Review.

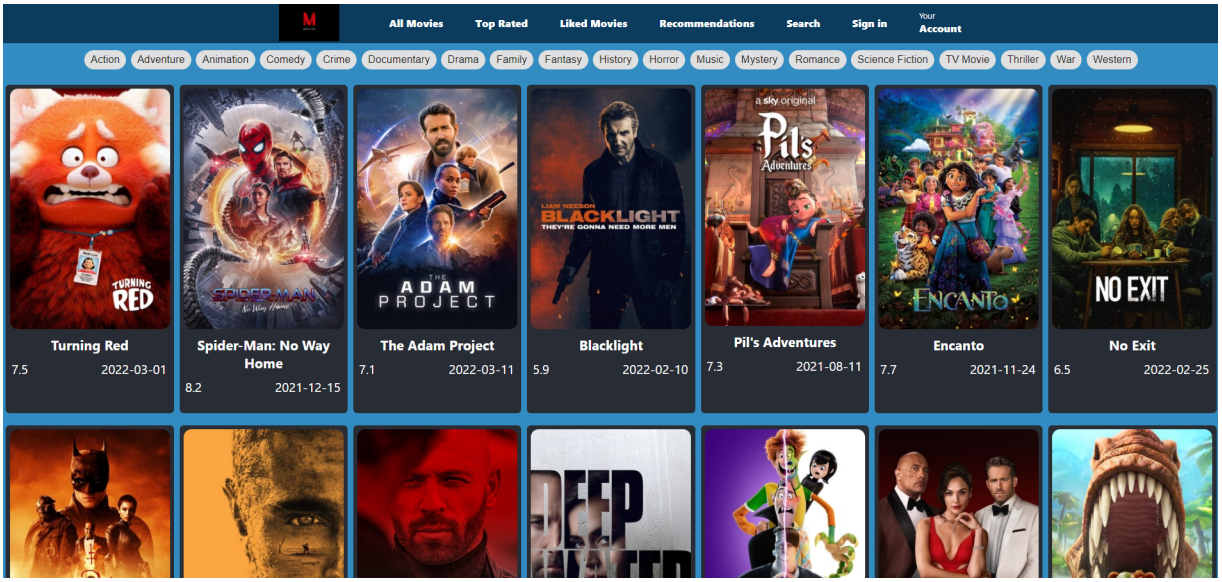

Figure 4.3: Front Page

4.2.1 Home Page

4.2.2 SingleMovie and MovieModal

This is the front page of my application. The front end consumes TMDB's API to get movie information. When the user first loads this page, the 'Discover' component sends a fetch request to TMDB's /discover/ endpoint which returns a list of 20 movies. It then saves them to a state variable. The data in the variable is then looped over and its details are decomposed and sent to a component called SingleMovie as props. After this is done, the SingleMovie component uses containment [35] to pass its values into the MovieModal component. This in turn creates a clickable SingleMovie that when clicked, opens the MovieModal component. SingleMovie shows the movie's title, poster, rating and release date. The result of this is shown above in figure 4.3.

When the user clicks on the SingleMovie, the MovieModal is opened. This modal appears on top of the home page. It is created using the Modal component from Material UI[36]. The Movie Modal shows the title along with the year of release, a movie tagline and its description. Also it provides a button that leads to a trailer for that movie on Youtube. This is how all movies in the web application are displayed, no matter what tab it is.

When designing this page, I wanted to let the user scroll infinitely through a list of movies. However, as I mentioned above, TMDB's API only returns

Figure 4.4: Movie Modal

20 movies at a time and as such I needed a workaround. I implemented a function that increments a 'page' state variable whenever the user scrolled to the bottom of a page. It fires when the user's position on the page is more in pixels than the height of the visible document in pixels. This function is inside a useEffect Hook, which rerenders the page and calls the 'updateMovies' function whenever the 'page' variable changes. The 'updateMovies' function uses the page state variable in the API call to retrieve the next page of results from the API.

Figure 4.5 in the next page shows a sequence diagram of operations that I have just described.

4.2.3 Genres

One of my identified objectives was the ability of the user to filter movies by genre. I achieved this by making two arrays for genres. One array called genres displays all genres that are available. These are retrieved from TMDB via a fetch request to the /genres/ endpoint. These genres are then displayed on the home page using a Chip component from Material UI. When a user selects a genre, it is removed from the genres array and inserted into the

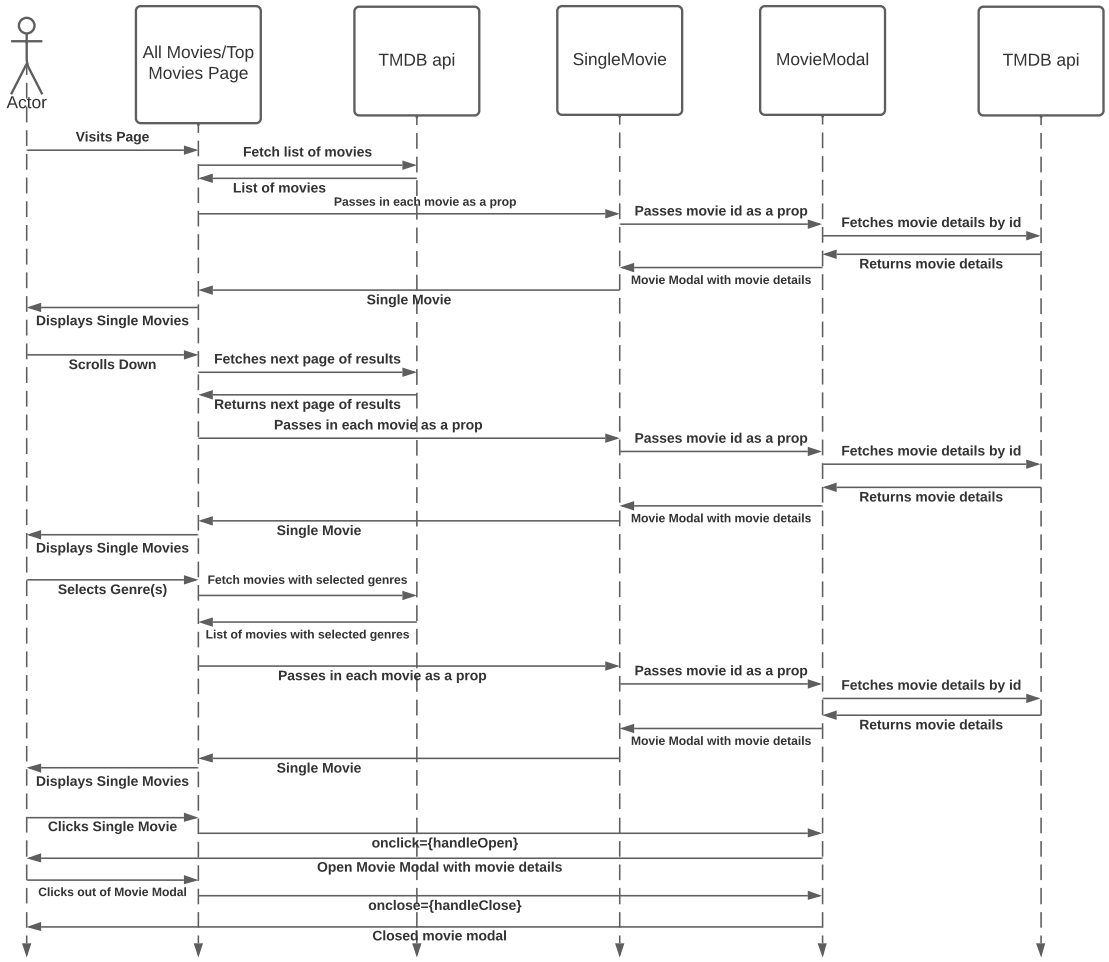

Figure 4.5: Sequence Diagram

selected genres array. This array's "Chips" are displayed in a different color to the genres array, signifying that they are selected. When the user unselects a genre.

UseGenres

I created a custom hook called useGenres to take the selectedGenres array as an input and format it into the format required by the API, which is the genres id followed by a comma. The result of this is stored in a state variable that when changed causes a useEffect hook to call the updateMovies function, which queries TMDB API with the selected genres. The custom

```
const useGenres=(selectedGenres)=>{
    if(selectedGenres.length<1){
        return "";
    }

    //extract genre id from selected genre array
    const GenreIds= selectedGenres.map((genre)=> genre.id)

    //Add all the genre ids into a string seperated by commas.
    //accumulator is the value being returned.
    return GenreIds.reduce((accumulator,current)=>accumulator+","+current);
}

export default useGenres;
```

Figure 4.6: UseGenres custom hook

Figure 4.7: Redux Values

hook can be seen in figure 4.6 above.

4.2.4 Redux

The user must register and log in to see recommendations and to like movies. Once the user completes a login/register form, the details are serialised into a JSON format and sent to the appropriate endpoint of the REST api in the Flask application. The back-end validates the details and, on login, returns the user's details in JSON format to the front-end. The front-end uses Redux to save the user's details. This includes the user's name, likes, dislikes and recommendations. This allows me to display each attribute of the user in the appropriate section of the application. The values that are stored in Redux can be seen in figure 4.7 above.

```
//when likes change then fetch movie details from TMDB.
useEffect(() => {
  if (likes != null) {
    const fetchMovies = async () => {
      //iterate through likes object to make concurrent requests
      axios
        .all(
          likes.map((u) =>
            axios.get(
              `https://api.themoviedb.org/3/movie/${u.id}?api_key=${process.env.REACT_APP_API}`
            )
          )
        )
        //take the result of all of those requests and set it to movies
        .then(
          axios.spread((...res) => {
            console.log(res);
            setMovies(res);
          })
        );
    };
    fetchMovies();
  }
}, [likes,user]);
```

Figure 4.8: Concurrent requests using axios.all

4.2.5 Liking

When a user presses the like icon on an open MovieModal, the user's like state array in Redux is updated with the movie's id using a reducer. When a user removes a movie from their likes, then that movie's id is removed from the user's like state array. When a movie is liked or a movie is unliked, its details are sent to the back-end where the back-end decides what to do with that information. When the user visits the likes tab, the likes array is retrieved from Redux and is then iterated through, and concurrent requests are made to TMDB to get information about each movie by its id. The result of all of those requests is then set to a state variable, which in turn is iterated over and displayed by passing its values as props to the SingleMovie component. This is shown in figure 4.8 above.

The same process of adding and removing using reducers is employed in the disliking functionality, the only difference being is that the only purpose the dislikes serve is that such movies are removed from the user's recommendations. They are not displayed anywhere in the application.

4.2.6 Recommendations

Recommendations work in a similar way. The Flask back-end returns recommendations in the form of a list of movie titles. The Recommendations component makes concurrent requests to get movie details based on the titles, and then displays loops over the resulting array and extracts its values, passing them as props to the SingleMovie component.

```python
class User(db.Model):
    __tablename__ = "users"
    user_id = db.Column(db.Integer, unique=True, primary_key=True)
    fname = db.Column(db.String, nullable=False)
    lname = db.Column(db.String, nullable=False)
    email = db.Column(db.String, unique=True, nullable=False)
    password = db.Column(db.LargeBinary, nullable=False)

    likes = db.relationship("Likes", backref=db.backref("users"))
    recommendation = db.relationship(
        "Recommendations", backref=db.backref("users"))
    dislikes = db.relationship("Dislikes", backref=db.backref("users"))

    def __init__(self, user_id, fname, lname, email, password):
        self.user_id = user_id
        self.fname = fname
        self.lname = lname
        self.email = email
        self.password = password

    @staticmethod
    def get_user(email):
        id = db.session.query(User.user_id).filter_by(email=email).first()
        user = db.session.query(User).get(id)
        return user

    @staticmethod
    def exists(email):
        exists = db.session.query(User.user_id).filter_by(
            email=email).first() is not None
        print(exists)
        return exists

    @staticmethod
    def get_password(email):
        return db.session.query(User.password).filter_by(email=email).one()
```

Figure 4.9: Database Models

4.3 Database

As mentioned in the Technology Review, the database used in this application is PostgreSQL. The database was created and maintained using Flask-SQLAlchemy. I created a models file in the Flask back-end that contained model classes which signify tables in PostgreSQL. As you can see in figure 4.9 above, a few relationships are defined in the Users class. The Users class is the foreign key in the likes, recommendations and dislikes table.

4.4 Back-end

I linked the front-end and back-end of this application by specifying a proxy in the package.json file of the React application. This allows the front-end to proxy its request to whatever address the flask application is on.

4.4.1 Structure

The Flask back-end is structured using Flask blueprints which I came across during research. Blueprints are Flask's way of creating reusable modular components. Blueprint functionality was separated by the database model that they perform actions on. For example the auth blueprint handles registering and logging in the user, the likes blueprint handles creating and deleting likes etc. In each blueprint there is a routes.py file that handles incoming requests from the React front-end. These blueprints are "imprinted" onto the main Flask application, giving it access to all the routes in the blueprints. This is how I created a REST API in the back-end of this project.

4.4.2 Registering

Once the user completes the register form in the front-end, the React application sends a POST request to the /register endpoint in the Flask back-end. The back-end validates the data received by querying the PostgreSQL database to check if a user with the email supplied by the front end exists, and checks whether the rest of the fields were valid. If the data passes the checks, the application then hashes the user's password using the Flask-Bcrypt's hashpw function and a randomly generated salt. Once this is done, the application creates a User object (defined in the models class) from the data and pushes it to the database.

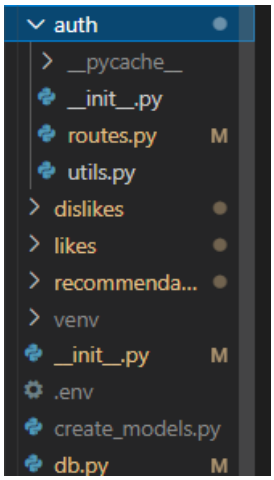

Figure 4.10: Structure of Flask application

4.4.3 Logging in

When the user submits a login form, the application queries the database to see if the user exists. If the user does not exist then Flask returns an error to the React app. If the user does exist then the application encodes the password provided by the front end into a utf-8 format, and compares it to the password stored in the database using Bcrypt's checkpw function. The encoding must be done because cryptographic functions only work on byte strings. If these passwords match then a JSON Web Token is created using Flask's JWT extension. The user's details and the JWT are then serialised and sent to the React front-end.

4.4.4 Liking

When a user likes or unlikes a movie on the front-end, the front-end sends the details of that movie to the back-end. If the like already exists, this means that the user has unliked that movie. The back-end removes that like from the Likes table in the database and removes any recommendations that originated from that movie.

If the like does not exist, the back-end creates a Likes object and pushes it to the Likes table in the database, with the user defined as a foreign key.

4.4.5 Disliking

Back-end dislike functionality works in a very similar way to the liking functionality. The only difference being that when a user dislikes a movie, the back-end checks if that movie is being recommended to the user and if it is, it removes the movie from the recommendations table. The difference is that when a like is removed, the back-end removes any recommendations that were made using that like. However, a dislike only removes recommendations with the same title.

4.4.6 Recommendations

The recommendation system that I implemented is a content-based recommendation system. It works by comparing the likes of the user to a dataset stored in the database of the application. The dataset I used can be found here: `https://www.kaggle.com/datasets/tmdb/tmdb-movie-metadata`. Since this dataset only contains information about movies and does not contain information about users and their likes, a content-based recommendation

system seemed to be the natural choice to build, rather than a collaborative system.

4.4.7 Pushing Dataset to PostgreSQL

This dataset contains two csv files. One contains details about the movie such as genres, overview and tags and the other contains details about the movie's cast and crew. In order to take the contents of the two CSV files and push them to my database, I had to write a script. In the scipt, I had to examine the data and extract the most relevant features out of it.

I merged the two files using the movie title through the use of pandas. Then, I deleted any rows that were empty. After this, I extracted the genres, keywords, overview, top three actors, and the director out of the dataset. I extracted these values specifically because in my opinion they are the most relevant details about a movie. A user may be a fan of a certain genre, or a certain actor/director etc. After this, I then merged each of these values into one column called 'tags' and deleted the rest of the irrelevant data.

Then, using the nltk package, I applied the Porter Stemming algorithm on the 'tags' column. Stemming is a way of normalising words. Words in English have many variations that mean the same thing. However, the program that I use to compare movies will not know that these different words actually mean the same thing. The use of the Porter Stemming algorithm returns just the root of a word. This in turn will make predictions more accurate because the program / algorithm will be able to better compare variations of words.[37]

Once this was done, I pushed the dataset into my database using pandas.

4.4.8 Cosine Similarity

The actual recommending was achieved through the use of cosine similarity. Cosine similarity is the measure of how similar two vectors are. The similarity is determined by the angle between the two vectors. If the angle is close to zero, this means that the two vectors are very similar. However, an angle is not a good measure of similarity. The similarity angle is transformed into a number between 1 and 0 using the cosine function, which gives a much better and readable indication of how similar two vectors are. After the angle is transformed, the closer the result is to one, then the more similar the two vectors are.[38]

In order for cosine similarity to work though, I had to convert the tags column from words into numbers that the computer could process. I achieved this by using Scikit-Learn's CountVectoriser. CountVectoriser converts a text

"to a matrix of token counts".[39] The result was a matrix containing each row of the tags column converted into numerical data.

When new recommendations are requested by the front-end, the back-end computes a similarity matrix using the cosine similarity function in Scikit-Learn.

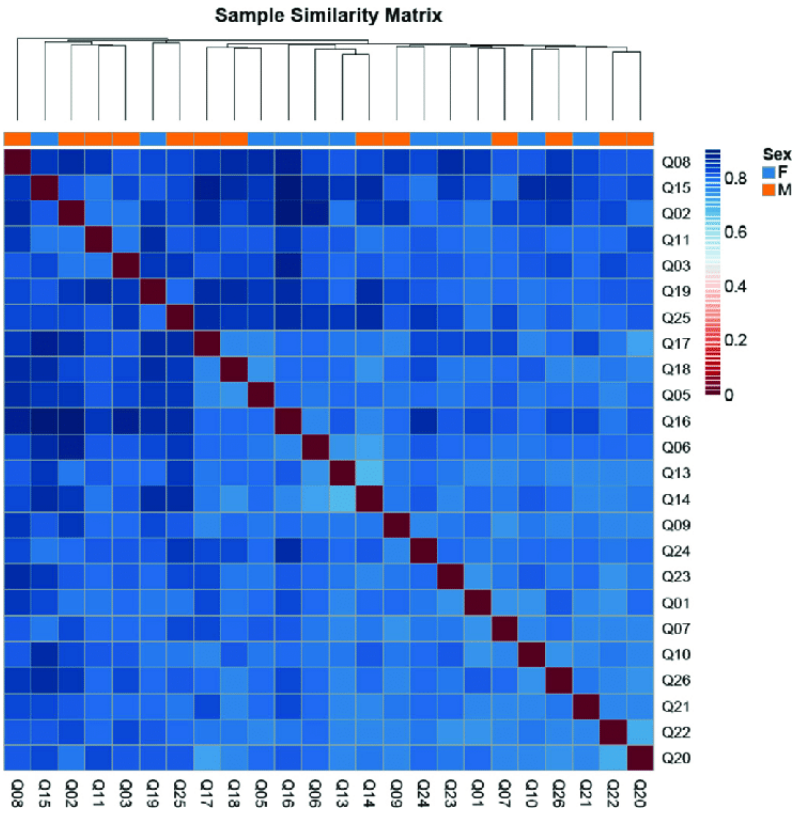

Figure 4.11: An Example of a Similarity Matrix

Once the user hits refresh on the recommendations page in the front-end, the Flask back-end fetches every like of that user and sends them to 'recommender.py' in the recommendations blueprint of the Flask application. This file then takes this list of likes, and individually figures out the index of that movie in the similarity matrix. Then, it gets the top 4 most similar movies from the similarity matrix and returns these in a list. Then, I make a dictionary with the original movie being the key and the array of recommendations for that movie being the value. After this, I push these values into the Recommendations table in my database.

What if a user likes a movie that is not in the similarity matrix?

When designing this recommender system, I had to consider what would happen if the user likes a movie on the front-end that isn't in the dataset in the back-end. This could happen because TMDB's API contains nearly 500,000 movies while the dataset I used only contains 5000. If this happened, no recommendations could be found since that movie wouldn't be in the similarity matrix. To prevent this from happening, whenever a user likes a movie, the back-end queries the 'data for model' table in the back-end and checks if it exists. If it doesn't exist, then that movie is processed in a similar way to which the CSV files were processed, and then it pushes that like to the 'data for model' table. This way, there will always be recommendations for movies and no errors will ever be thrown. I considered writing a script that would download the entire database from TMDB, but due to the API call limits on it, it would have taken over a week to download the entire thing. Due to time constraints, this was not possible.

4.5 Hosting on Heroku

The app was deployed to Heroku semi-successfully. Since there are technically two servers running; Node.js and the Flask server, I deployed two Heroku applications. One was for the React application, and one for the Flask application. Then I removed the proxy that was specified in the package.json file of the React application. Following this, wherever I made fetch requests to the Flask back-end in the front-end, I replaced the url with the url of the deployed Flask application on Heroku. Most features in the deployed version work, except for the recommendation engine. I realise that this is the main purpose of the application, and as such not being able to get this to work was disappointing. Whenever a user hits the refresh button in the recommendations tab on the React app, the Flask app on Heroku crashed due to it exceeding the total memory allowed by Heroku.

```
2022-03-31T18:48:00.460881+00:00 app[web.1]: in recommender
2022-03-31T18:48:00.460938+00:00 app[web.1]: likes [{'movie_id': 414906, 'title': 'The Batman'}, {'movie_id': 656663, 'title': 'Jackass Forever'}, {'movie_id': 760868, 'title':
'Black Crab'}, {'movie_id': 833425, 'title': 'No Exit'}, {'movie_id': 661791, 'title': 'The Grandmother'}, {'movie_id': 928381, 'title': 'Restless'}]
2022-03-31T18:48:09.509118+00:00 heroku[web.1]: source=web.1 dyno=heroku.255731339.714e9b11-2f93-41cb-b9e0-c0e80baab49c sample#load_avg_1m=0.00 sample#load_avg_5m=0.00
2022-03-31T18:48:09.661481+00:00 heroku[web.1]: source=web.1 dyno=heroku.255731339.714e9b11-2f93-41cb-b9e0-c0e80baab49c sample#memory_total=1145.59MB sample#memory_rss=511.98MB
sample#memory_cache=0.00MB sample#memory_swap=633.61MB sample#memory_pgpgin=361462pages sample#memory_pgpgout=234485pages sample#memory_quota=512.00MB
2022-03-31T18:48:09.701164+00:00 heroku[web.1]: Process running mem=1163M(227.3%)
2022-03-31T18:48:09.774801+00:00 heroku[web.1]: Error R15 (Memory quota vastly exceeded)
2022-03-31T18:48:09.805518+00:00 heroku[web.1]: Stopping process with SIGKILL
2022-03-31T18:48:09.975525+00:00 heroku[router]: at=error code=H13 desc="Connection closed without response" method=POST path="/recommend" host=cryptic-meadow-
```

Figure 4.12: Logs From Heroku on Application Crash

It seems to me that this is because anytime that the user hits refresh,

the Flask application has to pull in every row from the 'data for model' table into memory. This obviously takes up a vast amount of memory, which causes Heroku to end the process. It might also be due to a memory leak somewhere in my Flask application. Regrettably, due to time constraints and the multiple deadlines in final year, I could not remedy this issue. Fixing the issue would require me to completely redesign the recommendation system or to upgrade to a paid Heroku server which I am unable to do due to financial constraints.

Chapter 5

System Evaluation

In this chapter, I evaluate the project by comparing the implementation and the objective. I will also describe how it was tested and any flaws or limitations present in the system. Lastly I will discuss how I could improve and fix these in the future.

5.1 Objectives

In the introduction I outlined the main objectives of this project. The main objectives were to :

- Create a single page web application that recommends movies to a user.
- Display a wide selection of movies to the user.
- Implement a register/log in feature.
- Allow the user to like/dislike movies.
- Allow the user to filter movies by genre.
- Learn new skills/technologies that are widely used in industry.
- Deploy the project onto a cloud hosting service.

Upon reflecting on the objectives above and the implementation of the project, I feel that I have implemented and achieved each of these, with the exception of the last one. Users are able to log in and able to search a wide database of movies. In fact, if the movie exists, then they can find it. This is thanks to the use of TMDB's API. Users are able to like movies, and get recommendations based on them.

I have also gained knowledge in a vast amount of areas and technologies through the development of this application, which was a secondary goal of mine.

5.2 Testing

White Box Testing was conducted throughout the development of the project. It was conducted after any change had been made. This ensured that as I added code and components, the program would not break and would still function as expected. White Box Testing was conducted by using Postman to make requests to the API in the back-end. I then compared the output to the expected output. This made sense in the context of my chosen development methodology; feature driven development. In order for an incremental approach to work, components in the software must be tested rigorously to prevent unexpected errors. Although this method took a lot of time, it was necessary to ensure the success of the project and ensured its robustness. This project is robust due to the extensive testing performed, which eliminated any errors from the program.

If I were to undertake this project again, I would automate these tests with some kind of software such as the pyTest framework. I would also incorporate unit testing. Had I done this in the beginning, I would have saved a lot of time during development. My reasoning against automating the tests was that at the beginning of the development of this project, I was very inexperienced in using the Python language, the Flask framework and in functional React. I did not want to add an extra burden of writing unit tests in a framework which I am unfamiliar with, on top of learning everything else.

I tested the PostgreSQL database using pgAdmin4. This software allowed me to confirm that the expected data was indeed present in the database when rows were inserted by Flask.

5.3 Limitations

There are a couple of limitations present in this project, and they stem from the same issue. This issue is that whenever a user wants new recommendations from the back-end, the back-end must load in every single row from the 'data for model' table in PostgreSQL. This results in a semi-long wait time for the user on the front-end. Also, in the deployed version, this causes the application to run out of memory and to crash. This was an objective I

wanted to complete because it would allow me to show off my hard work to other people with ease and it was disappointing that I did not have enough time to remedy this issue.

In the future I plan to fix this issue. One approach I could take is to run a script on TMDB API to download all of its data. This would mean that the API on the front-end would not have movies that the dataset in the back-end does not have. This would allow me to create a similarity matrix out of this data, and then I could find a way to store this matrix in the database. This way I would not need to query the entire 'data for model' table, only the required row for the matching movie in that table.

One other thing this would allow me to do is to incorporate automatic recommendations. This would eliminate the need for the user to hit refresh on the recommendations tab, making the app more intuitive.

Those limitations aside I am happy with how the application works and functions. In my opinion, the UI has a sophisticated appearance and is styled well.

Chapter 6

Conclusion

This project was an extremely time consuming and challenging one. A lot time, frustration, effort and energy went into it. This is due to the fact that I was unfamiliar with most of the technologies incorporated in it, which meant a lot of time was spent reading documentation, trying to understand it and then applying what I learned into the project. However, in the end it was extremely satisfying applying new knowledge to my application. There is no feeling quite as good as finally figuring out why your code was not working after being stuck on an issue for a while. This project definitely solidified my love for programming. I developed a lot of invaluable skills that I will undoubtedly use later on in my career.

6.1 Skills Learned/Improved Upon

Python programming

Python is huge in industry and in machine learning nowadays.

REST APIs

I improved my skills in consuming and designing REST APIs. This is an essential skill because REST is the industry standard for APIs nowadays.

JSON data

I improved my skills in working with JSON data, meaning reading JSON data and serialising data into JSON format. JSON is extremely popular due to it being easily transferable between systems and due to it being extremely lightweight, which makes it perfect to send in HTTP requests.

JavaScript

JavaScript is such an important language to learn because there are so many job opportunities associated with it. JavaScript is used in every single website on the web. Every business, college, school, public service etc needs a website. I learned a lot about the fundamental concepts of JavaScript during the development of this application. Concepts such as the DOM and prototypal inheritance will no doubt come up in future interviews and will be useful to know about in the development of web applications.

React

React is another hugely important language due to its wide use in industry and in a large number of fortune 500 companies.

Writing a Dissertation

Prior to authoring this dissertation, I had not previously written an academic paper of this scale. The 'Research Methods' module in the first semester of final year, where I wrote a literature review, helped me tremendously with writing this paper especially with the technical review aspect of this dissertation.

6.2 Objectives

The main goal of this application was to create an application that recommended movies to users, allowing them to find movies similar to movies that they like. I am satisfied to say that I have achieved this goal.

The features and goals that I have implemented/achieved are:

- Creates a single page web application using React and Flask.
- Users are to register and log into the website.
- Users can log out of the application.
- Users can search for any movie they like.
- Users can see movie details.
- Logged in users are able to like movies.
- Logged in users are able to perform CRUD operations using the like feature.

- Logged in users are able to get recommendations for their liked movies.
- Learn new skills/technologies that are widely used in industry.
- I have learned a lot of new skills.

When comparing my achievements to the list of necessary goals, I am proud to say that I have achieved nearly all of them, with the exception being the cloud hosting which was mentioned in the previous chapter. Overall, I am very happy with how this application turned out.

6.3 Future Improvement

Firstly, I plan to fix the limitations that were outlined in the previous chapter of this paper. After I complete this I would like to use a different dataset that deals with user profiles and what movies those users like. This would allow me to implement a neural network as the engine of the recommendation system, which might increase the overall accuracy of the recommendations displayed to the users. That being said, I am still extremely happy with my the end result of this application. It was a challenging but very satisfying project to undertake, and I feel that I got a lot of value from the whole experience.

6.4 Thanks

I would like to thank Dr Dominic Carr who was my supervisor for this project. His advice was a significant help during the development of the project.

6.5 Credit to TMDB

I would also like to give credit to TMDB for their great API. While this application uses the TMDB API, it is not endorsed or certified by TMDB. All images in this application belong to The Movie DataBase(TMDB). All movie information is also provided by TMDB.

Chapter 7

Appendix

7.1 Link to Project Source Code

`https://github.com/antongolubev12/finalYearProject`

7.2 Installation

The steps you need to take to install and run this application are as follows:

1. Clone the repository; git clone https://github.com/antongolubev12/finalYearProject
2. Open the root of the repository in a command line environment and run yarn install to install all Node.js dependecies.
3. In the root of the repository, create a file called .env.
4. Declare the following variable in that file:
   REACT_APP_API=15a5b586e8467d37fb18ad12df7c3810
5. Navigate to the folder called project.
6. Create a virtual environment using the following command:

   ```
   python3 -m venv ./venv
   ```

7. Activate your virtual enviornment by using

   ```
   .\venv\Scripts\activate
   ```

8. Run the following command to install Flask dependencies:

```
python -m pip install -r requirements.txt
```

9. Create a file called .env in the root of the project folder with the following values:

   JWT_SECRET_KEY=21GF1T3Y84H39iujhbvft54edfvbghjui

   FLASK_APP=__init__.py

   FLASK_ENV=production

10. Install PostgreSQL on your machine.

11. Create a database on postgreSQL.

12. In db.py in the project folder, edit line 8 so that it matches this format

```
app.config['SQLALCHEMY_DATABASE_URI'] = 'postgresql://\
   YOUR_USER:YOUR_PASS@localhost/YOUR_DB'
```

    and also line 11:

```
engine = 'postgresql://YOUR_USER:YOUR_PASS@localhost/YOUR_DB'
```

    where YOUR_USER is the username you set up in postgreSQL, YOUR_PASS is the password and YOUR_DB is the name of the database you created.

13. Enter

```
python
```

    into your terminal.

14. Enter these two commands:

```
from models import db
```

15.
```
db.create_all()
```

    and then exit python

16. Import the csv into the dataframe by running

```
python prepare_csv.py
```

17. Run

    ```
    yarn start
    ```

    in the root of the movie folder.

18. Finally, run

    ```
    python __init__.py
    ```

    in a second console in the root of the project folder.

19. Open the localhost where node is running.

Bibliography

[1] S. Overflow, "Stack overflow devleoper survey," survey, Stack Overflow, 2020.

[2] HackerRank, "Hackerrank developer skills report," report, HackerRank, 2020.

[3] J. Hunt, "Feature-driven development," *Agile Software Construction*, pp. 161–182, 2006.

[4] J. Hughes, "Why functional programming matters," *The computer journal*, vol. 32, no. 2, pp. 98–107, 1989.

[5] M. D. Network, *MDN Web Docs Glossary: Definitions of Web-related terms : First-class Function*.

[6] T. Mikkonen and A. Taivalsaari, "Using javascript as a real programming language," 2007.

[7] L. Rei, S. Carvalho, M. Alves, and J. Brito, "A look at dynamic languages," *Porto, Portugal*, vol. 30, 2007.

[8] H. M. Gualandi, *Typing dynamic languages–a review*. PhD thesis, PUC-Rio, 2015.

[9] M. D. Network, *MDN Web Docs References: JavaScipt: Inheritance and the prototype chain*.

[10] M. D. Network, *MDN Web Docs References: JavaScipt: JavaScript lnaguage resources*.

[11] O. Foundation, *Node.js Documentation: The Node.js Event Loop. Timers, and process.nextTick()*.

[12] Oracle, *The Java Documentation: Deadlock*.

[13] O. Foundation, *Node.js Documentation: About Node.js.*

[14] META, *React Documentation: Introducing JSX.*

[15] jsx.github.io, *JSX: What is JSX?*

[16] META, *React Documentation: Introducing Hooks.*

[17] META, *React Documentation: ReactDOM.*

[18] M. D. Network, *MDN Web Docs References: Web APIs: Document Object Model: Introduction to the DOM.*

[19] META, *React Documentation: Virtual DOM and Internals.*

[20] META, *React Documentation: Reconciliation.*

[21] json.org, *Introducing JSOn.*

[22] M. D. Network, *MDN Web Docs: Guides: JavaScript — Dynamic client-side scripting: Introducing JavaScript objects: Working with JSON.*

[23] jwt.io/introduction, *JSON Web Token Introduction.*

[24] wiki.python.org, *Why is Python a dynamic language and also a strongly typed language.*

[25] T. E. Oliphant, "Python for scientific computing," *Computing in Science Engineering*, vol. 9, no. 3, pp. 10–20, 2007.

[26] J. W.-B. Lin, "Why python is the next wave in earth sciences computing," *Bulletin of the American Meteorological Society*, vol. 93, no. 12, pp. 1823–1824, 2012.

[27] A. Nagpal and G. Gabrani, "Python for data analytics, scientific and technical applications," in *2019 Amity International Conference on Artificial Intelligence (AICAI)*, pp. 140–145, 2019.

[28] flask.palletsprojects.com, *Flask Documentation: Modular Applications with Blueprints.*

[29] jd data.io, *JS Data Documentation: Data Mapper Pattern.*

[30] db Engines.com/en/ranking, *DB-Engines Ranking.*

[31] S. Overflow, "Stack overflow devleoper survey," survey, Stack Overflow, 2021.

[32] T. Register, *Postgres fills Ellisonian black hole.*

[33] PostgreSQL, *PostgreSQL Documentation: Chapter 9. Multi-Version COncurrency Control.*

[34] PostgreSQL, *PostgreSQL Documentation: CREATE INDEX.*

[35] META, *React Documentation: Composition.*

[36] MUI, *Material UI Documentation: Modal.*

[37] tartarus.org/martin/PorterStemmer, *Porter Stemming Algorithm.*

[38] A. Wooditch, N. J. Johnson, R. Solymosi, J. Medina Ariza, and S. Langton, "Getting to know your data," in *A Beginner's Guide to Statistics for Criminology and Criminal Justice Using R*, pp. 21–38, Springer, 2021.

[39] scikit learn.org, *Scikit Learn Documentation: CountVectorizer.*