Plymouth University

School of Computing, Electronics and Mathematics

PRCO304

Final Stage Computing Project

2017/2018

BSc (Hons) Computer Science

Jack Tantram

10487343

Methods and Tools for Building Recommender Systems

Acknowledgements

First, I would like to express my sincere gratitude to my project supervisor, Dr Marco Palomino, for his overwhelming support and guidance throughout this project. His encouragement and enthusiasm was invaluable.

I would also like to thank my girlfriend, friends, and family for their continuous support during my time at University.

Abstract

Recommender systems have gained a lot of interest over the past decade. Many companies are looking to adopt these systems to gain a competitive advantage by offering personalisation. This report describes a software development project that investigates how to do this. It tries to understand the processes and procedures for how to implement such a system, the aim was to create an open source web application that demonstrates a range of algorithms.

The report commences with an investigation into existing products and systems. This analysis is used to align the project objectives and deliverables. Further analysis is then made towards technologies, architecture, and design.

The development approach was incremental, which meant that requirements were added gradually so that everything was met and tested. The main development technologies were Python (Flask), React.js, Redis, and Neo4j. Other supporting technologies such as Docker, Webpack were also used. A description of this with a critical evaluation are presented.

The final sections of the report provide a critical evaluation of the final outcomes. As well as comparing the results from user testing two algorithms; Matrix Factorisation, and TF-IDF. Included, are the challenges it faced and suggestions for how the project could be improved in the future.

Details of interim project deliverables and other documents are presented in various appendices.

Table of Contents

Word count: 10634
Code Submission URL: `https://liveplymouthac-my.sharepoint.com/:f:/g/personal/jack_tantram_students_plymouth_ac_uk/EvxV6c_YDBNBoI9YHzHXahwBqn_Fzhfij-bJmCbGSOxeTA?e=bCwfBg`

1 Introduction

For over a decade, E-commerce has made a major impact to businesses around the world. In the UK, 82% of the population in 2017 had made an online purchase, compared to 57% in 2008 [1]. This has encouraged new businesses to sell their services online.

Visibility is problematic for many businesses, they can offer a wide range of products but customers do not see them. A study found that 91% of users do not go past the first page of results [2], which suggests if they're not presented with something relevant at the beginning the consumers will not purchase anything.

A way to challenge this is by applying personalised recommendations. This means that websites are more tailored towards consumer needs, which results in finding content much faster. It was estimated by McKinsey & Company [3] that 35% of Amazon's sales, and 75% of Netflix are driven by their product recommendation algorithm. Both of these companies are extremely successful and many businesses want to adopt similar strategies. To do this, extensive research has been made towards recommendation algorithms.

This project will evaluate these algorithms and understand the methods and tools to create recommender systems. To be able to achieve this, a recommendation website will be produced that demonstrates different algorithms. This will be used to evaluate overall performance and aim to understand how to improve accuracy. The project will then critically assess choices that were made and analyse methods to resolve them in the future.

There are many solutions online that describe recommendation algorithms, but there are little that explain how to create and end-to-end solution. The project contributes guidance on how to implement an architecture that can deal with these algorithms.

1.1 Background

1.1.1 Existing Products

The project began by providing an evaluation towards existing recommendation websites; YouTube, Netflix, Amazon, and Pandora. They all offer a range of services and calculate their recommendations in different ways; Appendix C.1 provides a full analysis.

1.1.2 Recommendation Algorithms

A recommender/recommendation system is a tool that aims to provide predictions towards user ratings or preferences [4]. There are a variety of recommendation algorithms, but they stem from 3 main approaches; Collaborative, Content-Based, and Hybrid.

Collaborative Filtering (CF) analyses large amounts of user behaviour, such as rating patterns [5]. It attempts to predict what users like based on similarity metrics, the type of data can vary per system. It can be implicit, such as page clicks, as well as time spent on a particular page. Or explicit, which could be ranking an item on a scale, in addition to if they like or

dislike an item.

CF is very versatile and is easier to apply to a cross domain, although it suffers from an issue called the cold start problem. This refers to the occurrence of a sparse user-item rating matrix and results in the algorithm having a limited knowledge base.

In contrast, the Content-Based (CB) algorithm involves comparing the characteristics of an item. Such as keyword attributes, or particular properties i.e. song pitch, or bass at certain intervals [6]. This is beneficial because the algorithm has no dependence towards users. In addition, it doesn't suffer from the cold start problem because new items do not need to be rated to be recommended. However, the performance of the algorithm is closely linked to the amount of information it can retrieve. If there are limited characteristics then the recommendations can suffer because there isn't enough to extrapolate from.

Furthermore, retrieving information for a CB system can be incredibly time-consuming. Music streaming provider Pandora ran the music genome project, which was used to identify the genetic mark-up of a song [7]. The study highlighted that there are approximately 450 individual genes that make up a song. The characteristics themselves are extremely useful towards recommending songs with similar genetic markup, however it takes an Pandora employee 20-30 minutes to identify a 4 minute song, which poses threats to scalability.

Finally, a hybrid system is a combination of CF and CB methods. The model itself can be generated in a range of ways, such as combining both predictions or having the prediction separate. They are beneficial because they have higher accuracy, although they can be difficult to implement and take a lot of time to perfect.

1.2 Aims, Objectives & Deliverables

The main aim of the project was to create a system that provides recommendations based off user interest. The goal was to create a tool that was well documented and extensible, so that it could be used for future work.

1.2.1 Project Objectives

The following objectives were created at the start of the project:

1. Analyse different recommendation algorithms
2. Evaluate methods and tools for building recommender systems
3. To provide recommendations for users
4. Create a robust and extensible system architecture
5. Create a responsive web application
6. Follow best practices and produce well documented code

7. Implement a testing framework, to ensure stability
8. Create a system that can be deployed easily

1.2.2 Deliverables

There are a number of deliverables for the project:

- User Guide
- Project Initiation Document
- Web application
- Source code available on Bitbucket and OneDrive link attached
- Project poster
- Fully documented API with unit tests

2 Method of Approach

2.1 Project Management

This section will examine the management approach that was followed. To decide on the appropriate methodology an evaluation was made (Appendix B.4). Highlight reports were an important part in keeping the project in track. They summarised what work had been completed and allowed reflection for what had to be done. It was beneficial because it helped maintain communication between the project supervisor and notified them how the project was progressing.

2.1.1 Agile

The agile methodology [8] was adopted because it provides much more flexibility than traditional methods. The projects requirements were expected to change frequently, which is something agile is effective with.

The project employed sprints, which refer to a period where work has to be completed [9]. In order to estimate how long a task will take, story points were used [10]. This assigns a value to estimate how difficult a particular task will be to complete; the higher the points, the longer it should take to finish. This is advantageous as it enables better time estimations during sprints.

2.1.2 Kanban

The agile methodology can be extended into a particular framework such as kanban and scrum [11]. The kanban approach describes that there are no prescribed roles and focuses on continuous development, whilst scrum delegates roles to team members and the product owner provides the goals and objectives. Any changes during the sprint are strongly discouraged.

On the basis that the project was being run individually and that requirements were changing frequently, the kanban framework was chosen. This was important because there was an expectation that adjustment during sprints could happen. For instance, if using scrum and a bug was found it would have to wait until the next sprint. Kanban would mean that there is no penalisation for change.

2.2 Prototyping

Prototyping is a commonly adopted approach in the Software Development Life Cycle (SDLC). It is extremely important because it enables the developer to understand how to approach a problem; this provides guidance when assessing feasibility. Jupyter notebooks (JN) [12] were used extensively in the project to create prototypes. JN are documents that contain both code and rich text and assist in visualising code. Fig 1 demonstrates the JN's file structure, with all the notebooks that were created in the project.

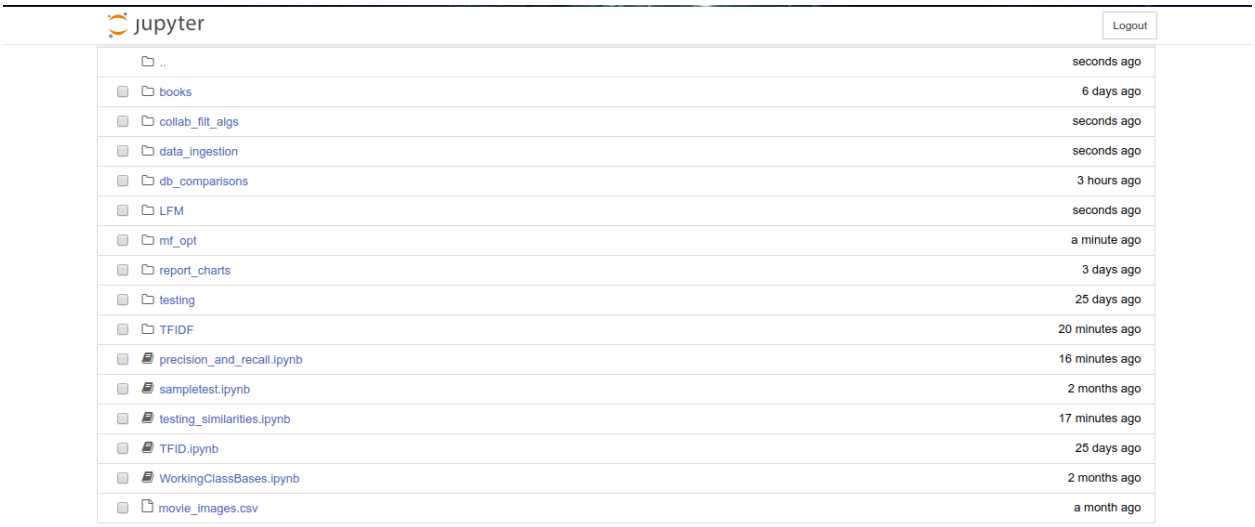


Figure 1: Jupyter Example

2.3 Version Control

To maintain the projects source code Bitbucket [13] was chosen. It enables code changes to be tracked at reverted easily.

2.4 Technologies

The project consisted of a variety of technologies, a full evaluation was made in appendix E.

Python [14] was chosen as the primary back-end language incorporating Flask [15] as the web framework. Flask is very lightweight in comparison to another popular Python framework, Django [16], this allows a lot more freedom in development; this is the reason it was chosen. The rationale for choosing Python was that it comprises many data science libraries, that could aid in the development of the recommendation algorithm. The chosen front-end framework was React.js [17], because it benefits by providing reusable components and improves code quality and speeds up development. In order for React.js to work it requires a bundler or task runner; Webpack [18] was chosen as it provides of benefits such as transpilation, uglification, and optimisation.

Finally, the databases that were chosen were Neo4j [19] and Redis [20] because it is advantageous for caching data. Neo4j was adopted for a variety of reasons, a full evaluation is made in Section 4.2.

3 Requirements

3.1 Functional Requirements

The method of approach used to create requirements was the MoSCoW method [21]. This refers to labelling requirements as 'Must Have, Should Have, Could Have, and Won't Have'. It is a useful way of labelling requirements and provides prioritisation.

The user requirements are:

Must have:

- Be able to create an account
- Password resets
- Be able to view recommendations
- Be able to rate movies
- Accessible on either tablet or computer

Should have:

- A user should be able to search by movie title
- A user should be able to view more in depth information about a movie
- A user should be able to search by genre

- A user should be able to view more in depth information about a movie

Would have:

- A user won't be able to integrate their social media interests

3.2 Non-Functional Requirements

Indifferent from functional requirements, non-functional aim to evaluate the operations of a system, rather than what it will try to implement. MoSCoW can be used to explain non-functional requirements, however for the project FURPS was followed [22]. FURPS was chosen because it was felt that it could define the non-functional requirements in more detail and provide greater structure. The approach consists of 4 main categories (Functionality, Usability, Reliablity, Performance, and Supportability) and uses them to classify attributes for software, and system quality [23]. The following table illustrates the non-functional requirements for the project.

Requirement Type	**Explanation**
Availability	The system should be available 24 hours, everyday
Compatibility	Will be compatible on modern browsers such as Chrome, Firefox, and available on tablets, and desktop
Performance	The website must react quickly to queries and return content in a reasonable amount of time (max 5 seconds).
Reliability	The system should function under all stated requirements
Maintainability	The system should be fully documented and be easy to make new changes.
Scalability	The system should be distributed so that the website has the ability to expand in the future
Usability	Most importantly the interface must be easy to understand and make sense to the user
Security	In the case of handling user accounts, the system should encrypt information and not store content like passwords as plaintext
Testability	There must be testing in place, such as unit testing to ensure modules work.

4 Design

4.1 Architecture Design

During the design phase a lot of emphasis was put in place to create a robust and scalable architecture. A typical web applications design is the monolithic approach. This describes an application in which everything is combined into a singular program or platform [24].

Initially, this is advantageous as it allows rapid development due to everything being in one

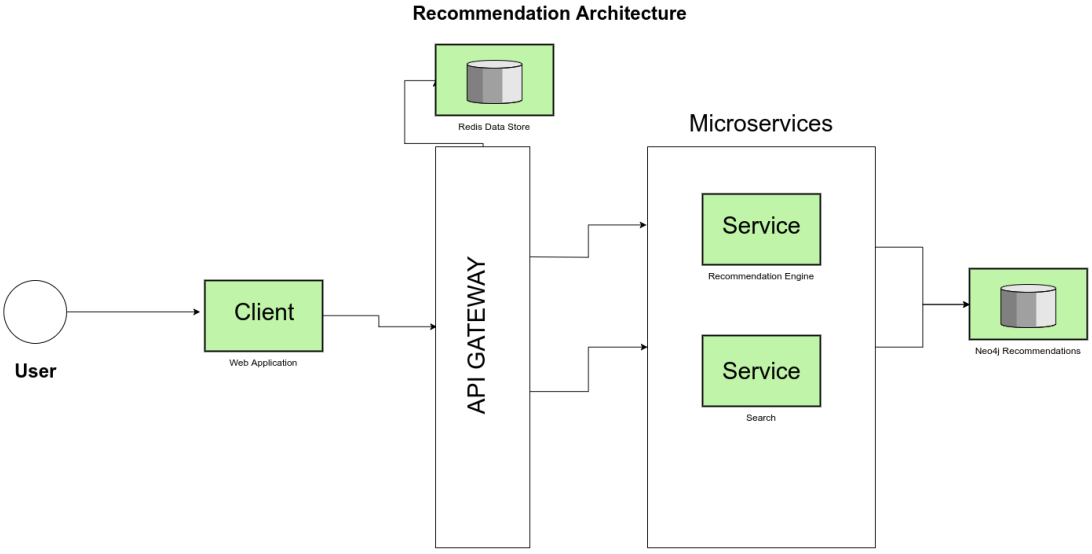
code-base. However, many companies have suffered due to this design. Deliveroo in 2017 [25] went through a massive redesign. This was because they were experiencing increasing performance issues and build times. Consequently, due to having a monolithic design.

The solution that was adopted like many others [26] [27] was the microservice architecture. This involves decomposing an application into a set of smaller services, which results in everything being completely distributed [28]. This is favourable because the application is decoupled and provides higher reliability as faults are isolated. Moreover, it provides freedom in the technology stack because each service is independent with the only requirement of communication through a HTTP channel.

The projects high-level architecture has been divided up as 3 core elements; the client, API gateway, and services. The client communicates through the API gateway, which acts as a middleware between services and clients. This is beneficial as it allows new clients to be created easily.

The projects architecture changed a couple times during the project, however there was still a focus on a microservice architecture. Originally (Appendix G), it consisted of the client, API gateway and its services; there were 3 services in total. However, due to using Auth0 [29] there wasn't the need to create a user service.

Figure 2: Final Architecture



4.2 Database Design

There are a range of databases to choose from and considerations were made to decide what was the most suitable for the project. There are two major types, relational and non-relational. Relational store data in tables and rows and use the query language SQL. Meanwhile, non-relational can represent data in a range of different formats and employs NoSQL [30].

A NoSQL Db can be subdivided into further categories. One in particular is a graph database (GDB). This consists of a structure that contains nodes, edges and properties to store data. An interesting concept is that it uses relationships to connect nodes. For the nature of the project this seemed beneficial because of wanting to understand the relationship between items to make recommendations.

To understand whether a NoSQL database was better suited, research was conducted to compare whether this would be the correct choice. To achieve this, two types of databases were tested with a set of queries. The relational database was PostgreSQL (Postgres) [31], and Neo4j for NoSQL.

Both databases were populated using same dataset so that queries could be evaluated fairly. To be able to calculate the overall query time a profiler was used. For Postgres this used the '*Explain, Analyze*' statement which returns the overall execution time [32]. The '*Profile*' command was used for Neo4j [33]. The profiler was executed multiple times for each query and the average was taken to ensure that there was consistency in results (Appendix E.3.1).

Overall, it suggested that Neo4j was faster than Postgres. The most highlighting factor was the queries themselves. The same query in Neo4j was a lot smaller than Postgres, and was found easier to write.

Medhi and Baruah [34] also investigated the differences and found that increasing the amount of objects in their query showed Neo4j outperforming. In addition, it was noted that Neo4j was much easier to design in comparison to SQL; it would expect a lot of joins. Therefore, this could potentially increase development time for the project.

Overall, Neo4j was considered to be the best option due to time and performance factors. Below illustrates the final property graph data model for the project. The data model comprises 5 core nodes, Users, Movies, Genres, Links and feedback. They are linked using relationships that describe their attributes between each other.

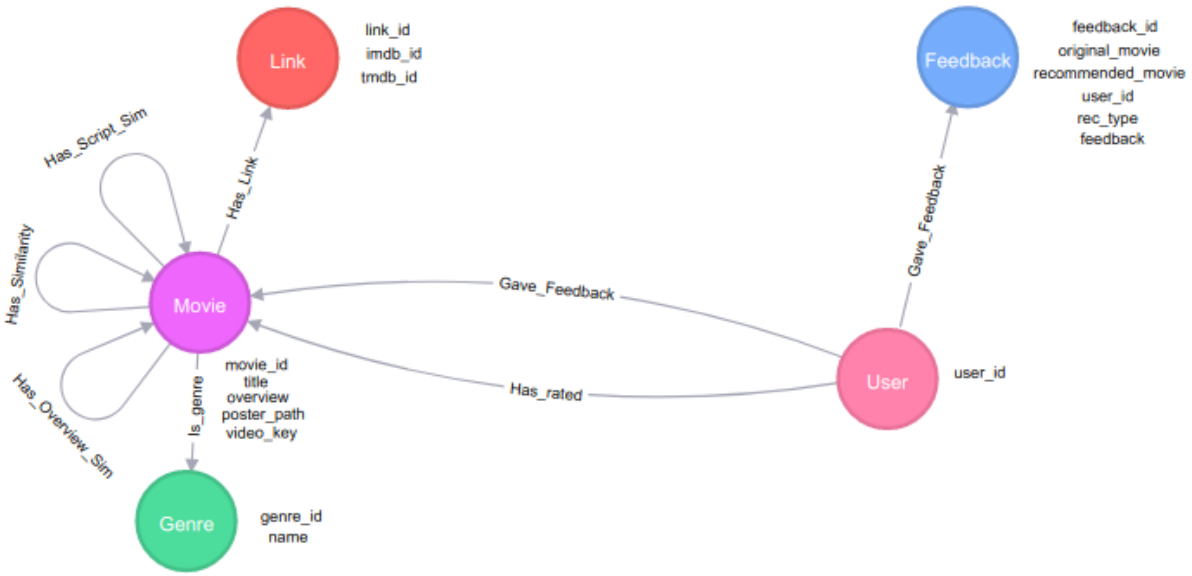


Figure 3: Graph Database model

4.3 User Interface Design

Human-Computer Interaction (HCI) is an important part of the design process. It observes how computers are utilised and tries to understand how to create a system that is best suited for users [35].

There are a number of reasons why HCI should be investigated. The average tolerable wait time for information retrieval was found to be 2 seconds [36]. If the users are not notified they will leave, which is something that should be avoided.

The project aimed to follow good design principles. A well-known method is Schneiderman's golden rules of Interface Design [37]. It outlines the core elements for designing a system. An example of these principles would be striving for consistency, offering informative feedback and simple error handling. Therefore, when designing the user interface these factors were taken into account (Appendix F.2).

5 Recommendation Algorithms

5.1 Overview

This section will discuss the research that was conducted to identify a suitable recommendation algorithm. As discussed in section 1.1.2, there are 3 major types of recommendation algorithm; CF, CB, and Hybrid. This section will aim to understand them.

5.2 Dataset evaluations

Analysis was conducted deciding a potential dataset for the project. This investigation compared different CF and CB datasets.

CB systems evaluate the characteristics of items and users to discover similarities between them. In order to achieve high quality recommendations a CB algorithm is reliant on using a quality rich dataset in order to create beneficial recommendations.

There are available CB datasets online, such as [38] that contains one million with audio features from 1922-2011, but others are hard to find. They are considerably harder to produce because it involves a strong understanding of the subject domain.

In contrast, there are a wide range of user item rating datasets online that would be suitable for a CF approach [39] [40] [41]. Appendix C.2 illustrates the benefits and limitations of each dataset.

Overall, comparisons showed that either MovieLens (ML) or book crossing was suitable for a CF approach as they had a lot of user-reviews. The Amazon dataset was disregarded due to the issues with using the ASIN to retrieve more information about an item.

The ML dataset was promising because it provides links which can be used to retrieve more content, such as overviews or cast members. Due to this it was chosen as it is the type of data a CF algorithm expects and enables the ability of accessing more in depth data which could be used for CB approach; enabling a hybrid system.

5.3 Collaborative Filtering Approach

A CF algorithm is typically split up into two categories model-based and memory based.

A memory-based (MB) algorithm approaches CF by using the entire database [42].It attempts to find users that are similar by using a similarity metric, such as cosine similarity, or Pearson's Correlation Coefficient (PCC). Generally, MB algorithms are simpler and provide higher accuracy. However, due to needing to use the whole database they struggle with scalability [43].

In comparison, a model-based approach does not store everything in memory. It extracts information from the dataset and creates a model that it can then use to produce recommendations. It is much faster than MB because it does not need to use the entire database and makes it more scalable. However, it can be inflexible as a model can take a while to generate. It also means that in the scenario of a new item it relies on the model to be regenerated to perform accurate predictions.

5.3.1 Latent Factor Model

Commonly, CF systems are user and item based approaches. However, these systems can be inflexible due to having high time complexity and scale poorly [44]. Alternative approaches using factorisation have been sought after because they are more scalable and efficient.

A Latent Factor Model (LFM) [45] is an unsupervised learning technique which aims to explain ratings by trying to characterise both users and items. It's based on the assumption that there are hidden latent factors that identify an item. For example, in films there could be dimensions for the type of genre, or the person who directs the film. In a sense, it is similar to a CB algorithm, but in LFM it does not have to know all the characteristics.

The LFM algorithm aims to predict known user ratings so that it can accurately predict unknown ratings. It is a form of Matrix Factorisation (MF), where dataset R, is decomposed of a mxn matrix of user, item ratings. The matrix is separated into two matrices P and Q. Their size is determined by a factor K which identifies the number of latent factors the algorithm is trying to find. Therefore, P is derived as $P^{(num\_users*k)}$ and $Q^{(k*num\_items)}$.

A rating is predicted by the dot product of both matrices. This is given by the equation:

$$r_{ui} = q_i^T p_u \tag{1}$$

In order to calculate the complete predicted rating matrix, both matrices are multiplied together; fig 4 illustrates this.

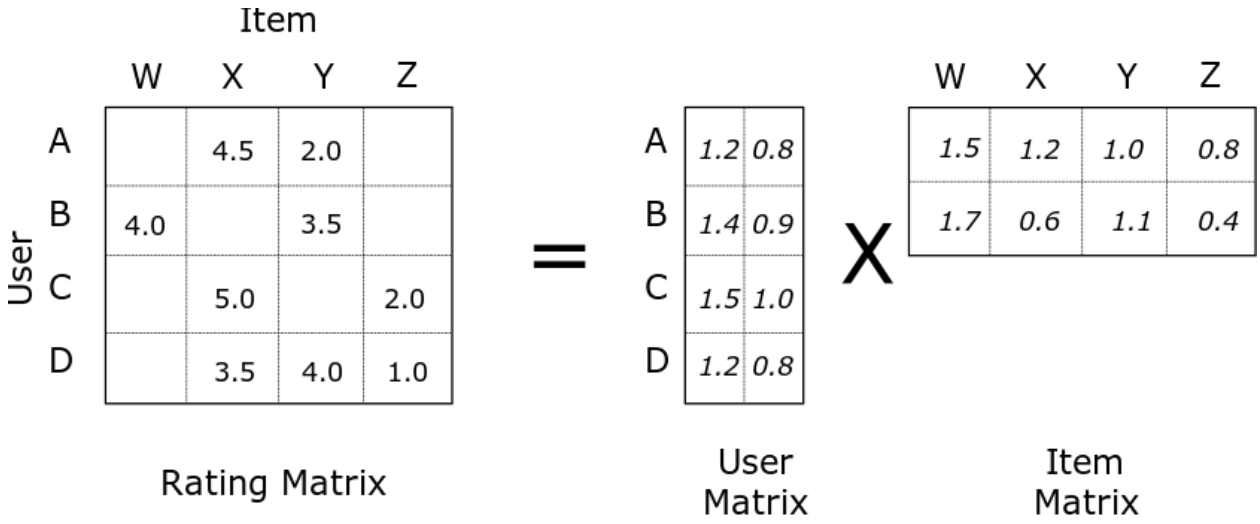


Figure 4: LFM Matrix [46]

To accurately learn the feature vectors the algorithm wants to be able to generalise previous ratings to predict the future. In order to do this it must minimise the error between the

prediction and known rating. This is given by:

$$e_{ui} = r_{ui} - q_i^T p_u \tag{2}$$

To approach the minimisation problem, an optimisation algorithm must be used. A common method is stochastic gradient descent (SGD). This uses a regularisation parameter γ during optimisation which helps deal with overfitting. It gives the overall formula:

$$\min q^*, p^* \sum_{(u,i)\epsilon K} (r_{ui} - q_i^T p_u)^2 + \gamma(||q_i||^2 + ||p_u||^2) \tag{3}$$

SGD is an iterative approach which aims to minimise the objective function. For LFM it uses two hyper-parameters, α which controls the learning rate and γ which is the regularisation property. Algorithm 1 demonstrates the pseudocode. To evaluate performance a common approach is calculating the root mean squared error (RMSE) which provides the error between prediction and actual value. It is only calculated for known ratings because RMSE is predominately looking at how close the predicted value is to the actual value; if a user doesn't give a rating for an item then it cannot be predicted.

An issue with the basic LFM approach is that it does not take into account bias. In a rating matrix certain users rate harshly than others which can influence the results. Koren [47] highlighted the importance of this and found a method of dealing with bias. This involves including the global average rating and two vectors to keep control of item and user bias.

Algorithm 1: SGD Matrix Factorisation

```
1  P = rand(num_users, K);
2  Q = rand(K, num_items);
3  R = rating_matrix;
4  function train (steps);
5  foreach row u ∈ R.row do
6      foreach col i ∈ R.cols do
7          r_ui = R[u, i];
8          if r_ui != 0 then
9              e_ui = r_ui − (Q[:, i] · P[u, :]);
10             Q[:, i] = Q(:, i) + α(e_ui ∗ P[u, :] + γ · +Q[:, i];
11             P[u, :] = P(u, :) + α(e_ui ∗ Q[:, i] + γ · +P[u, :];
12         end
13     end
14 end
```

A common problem that can be faced using MF is data sparsity, and there are many research papers that have experimented with ways of dealing with this. Kim et al [48] evaluated including tag information to improve recommendation quality. It was demonstrated that it can strengthen results and that the solution performed significantly better than general

MF. Yu et. al [49] also discovered including an attributes coupling with information such as director,actor, and genre details it can vastly better accuracy.

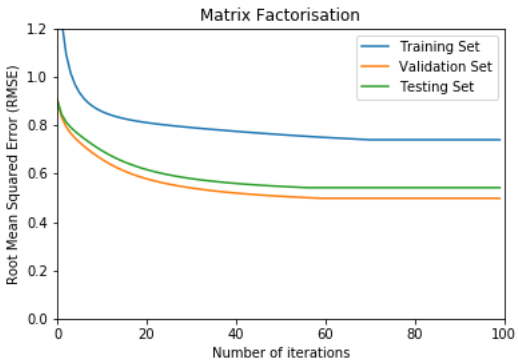
5.3.2 Experiments

To understand the effectiveness of the LFM algorithm an investigation was made to see if was able to produce accurate predictions. The dataset was split using cross validation (CV). This allows performance to be evaluated across multiple samples, so that it could be representative when new data enters the system. The dataset was split into training, validation, and test samples. The data were split using the Pareto principle, which is a 80/20% divide between training and testing. It is used commonly in machine learning and states that roughly in events 80% of effects come from 20% causes [50].

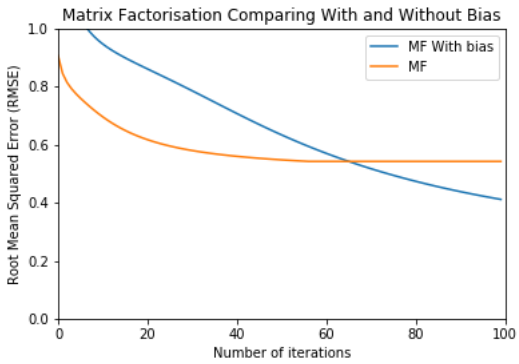
The training set was used to evaluate the performance of the algorithm without any prior optimisation, whilst the validation set would optimise the hyper-parameters α, γ.

To optimise the hyperparameters a grid search was used. This is an exhaustive search that goes through a set of parameters to find the optimum result. Fig 5a highlights the final results with optimisations for the training, testing, and validation sets. The graph suggested a drop in RMSE from the original training set value and shows that the optimisation also improves the testing set.

An additional experiment was run to compare the RMSE with and without bias. Overall it highlighted that the inclusion of bias reduced the RMSE value. After 100 steps it had still not converged which suggests that the inclusion of bias will take longer but will produce better results. Fig 5b shows the results for normal MF and MF with bias.



(a) Matrix Factorisation RMSE



(b) MF Comparing With and without bias. Final 0.411 RMSE

Figure 5: MF Cross Validation with comparison against MF with bias

After comparing the difference between MF with and without bias, a grid search was executed to optimise K and view the error rate with and without bias. Overall Fig 6 demonstrates that by increasing the value of K the error rate decreases to the point that it is almost 0. MF

bias suggests that it converges a lot quicker and $K = 100$ seems to be the best performing in 100 steps.

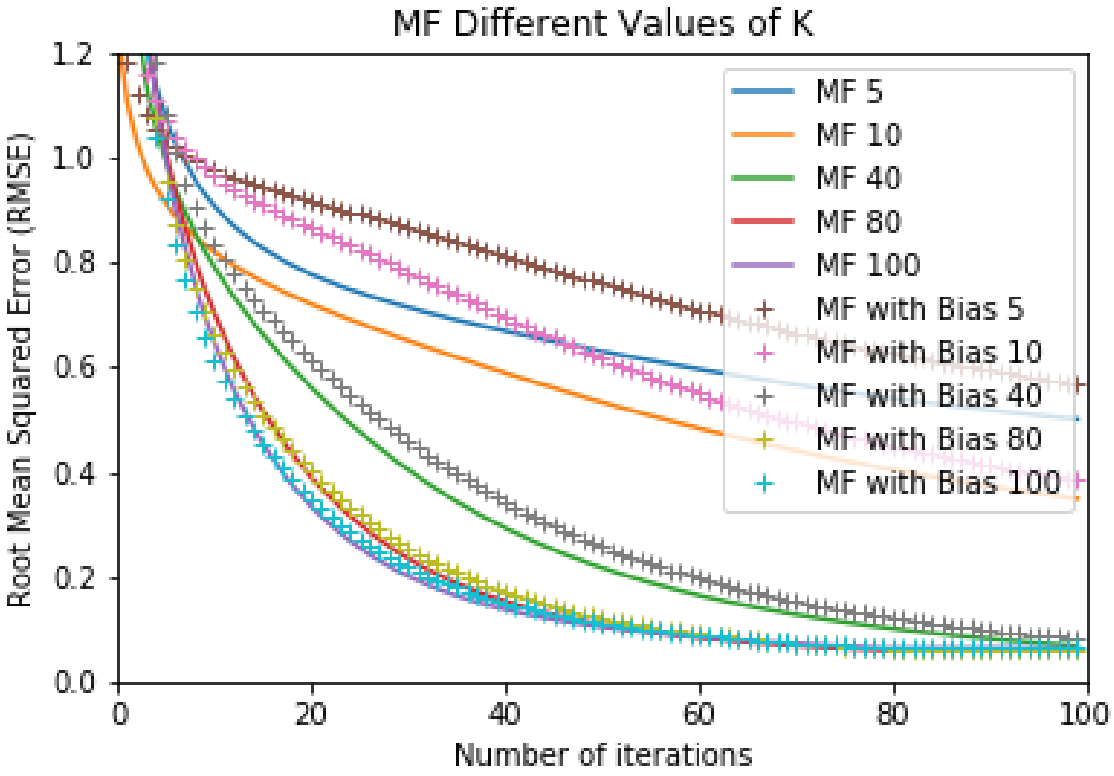


Figure 6: Comparison of K values with and without bias

Fig 7 highlights a list of recommendations using the optimised bias parameters.

```
Movie recommendations for Star Wars: Episode IV - A New Hope (1977)
Star Wars: Episode VI - Return of the Jedi (1983): Sim: 0.781841
Star Wars: Episode V - The Empire Strikes Back (1980): Sim: 0.690428
Shakes the Clown (1992): Sim: 0.635915
Spy Game (2001): Sim: 0.629563
Grand Day Out with Wallace and Gromit, A (1989): Sim: 0.628004
Jimmy Neutron: Boy Genius (2001): Sim: 0.615096
Sisters (1973): Sim: 0.612495
Raiders of the Lost Ark (Indiana Jones and the Raiders of the Lost Ark) (1981): Sim: 0.610067
Ever After: A Cinderella Story (1998): Sim: 0.609056
Phantom Tollbooth, The (1970): Sim: 0.602413
```

Figure 7: Optimised Bias Results

After K was determined to be 100, a series of tests were run to view the output of recommendations for a sequence of films. To calculate the recommendations the similarity metrics PCC and cosine similarity were used on the predicted ratings matrix (Appendix H.1).

Overall the differences between both of the metrics were not significantly different. Cosine similarity showed to be slightly better in all the trial recommendation outputs from the

researchers opinion, which meant that it was chosen as the metric for the project. To fully evaluate the algorithm's performance, user testing was conducted and the results can be found in the results section. Interestingly when looking deeper in the recommendations, movies were being grouped by year. Additionally, with *Jumanji* the recommendation for *Richie Rich* may not sound obvious at first, but delving deeper shows they have similar actors (Jonathan Nash).

5.4 Content-Based Approach

The CB approach recommends items based on similarities between characteristics. It does this by categorising items and understanding their properties.

5.4.1 TF-IDF

One type of CB algorithm is term frequency–inverse document frequency (TF-IDF) [51]. It is a statistical method for reflecting how important a word is in a corpus. The algorithm is widely used by search engines so that it can find websites with similar content. It uses a weighting factor so that it can work out what is important. The TF-IDF value increases proportionally to how many times it appears in the corpus, which means that the most important words are weighted the highest.

For the aspect of the project the idea was to see if comparing similarities between movie scripts and overviews was feasible. The original dataset did not contain any information about movie scripts or overviews, this meant that two scripts had to be written to retrieve these details. To obtain overviews, The Movie Database (TMDb) [52] was used and the Internet Movie Script database (IMSDb) [53] for screenplays.

After obtaining the relevant information a comparison was made to understand whether either algorithm could produce relevant recommendations. The implementation of the TF-IDF algorithm used the TF-IDF Vectoriser module from scikit-learn [54]. Each data sample were vectorised and a similarity matrix using cosine similarity was produced.

The measure of performance and accuracy is different from a MF approach due to not creating predictions, which means RMSE cannot be calculated.

5.4.2 Experiments

The experiment began by creating a cosine similarity matrix for movie scripts and overviews. Each matrix then produced a set of 10 recommendations, which were then evaluated to see if they were relevant (Appendix H.2).

The first observation was that similarity values coming from the movie scripts were significantly higher than overviews. For example, the top score in overviews for *Star Wars: A New Hope* was 41% similarity, in contrast movie scripts showed 81%. Further analysis showed that this was common across all tests. Additionally, for recommendations for *Alien* the top suggested

was *Talented Mr Ripley*, which is plausible due to the main character being named Ripley in both films. However, this is not a relevant suggestion and is actually ranked higher than *Alien Resurrection* which would be seemed more suitable.

It was distinguishable that comparing overviews showed more films that related to the topic of the original movie. *Alien* using overviews also showed recommendations for films which are related to space, a theme that is common with the franchise.

Overall, it is difficult to say which performs better, this is why the project went through user testing to evaluate opinions at a greater scale. It was noticed that the feature names generated using TF-IDF for scripts contained random numbers and letters. Incorporating a method for sanitising the text might improve the overall performance as it would disregard certain elements from the script.

6 Development

6.1 Quality Control

In order to maintain high quality a number of software practices were observed. By following best practices it can provide a range of benefits. Firstly, the code that is produced is easier to read and maintain. This is also not only beneficial for the developer writing the software but for anyone in the future that could pick it up.

There are a range of software best practices and more specialised standards for a particular language; for example PEP8 in Python. To ensure that the code for each language conformed to their standards, PyCharm was used. The application provides built in linters which aid towards writing code to their recommended guidelines. In addition, PyCharm's built-in formatter ensures that code is indented correctly.

Furthermore, the Don't Repeat Yourself (DRY) principle [55] was followed as much as possible throughout the project so that code could be easily reused and reduced the amount of duplication.

In web development there are a range of standards for the front-end and back-end. The project tried to follow these as closely as possible.

6.1.1 Documentation

Everything in the project has been documented, where every class and function contains relevant information about its usage. It has been written to conform to the language style guide so that developers can easily understand the project.

6.1.2 API Documentation

The API gateway acts as a middleware between clients and services. In order to make it reusable it has been documented using Swagger [56] which is a tool that helps developers document RESTful web services. This allows developers to easily understand how the API is broken up and what each parameter does. Effectively, it is split up into multiple endpoints which then correlate to services.

By using swagger it is a really effective tool for keeping everything separate, it also has functionality to test the endpoints from the tool which means developers can see the expected responses with ease.

The documentation is accessible via the API gateways url. Fig 8 highlights the different endpoints that are available in the API gateway. Each endpoint contains a description about its purpose and its request type; GET, POST, etc.

Swagger provides the ability to test endpoints and view the list of possible parameters for an endpoint. For security purposes certain endpoints require a valid JWT authorisation token and require a valid token in order to get the request to work **fig show evidence**.

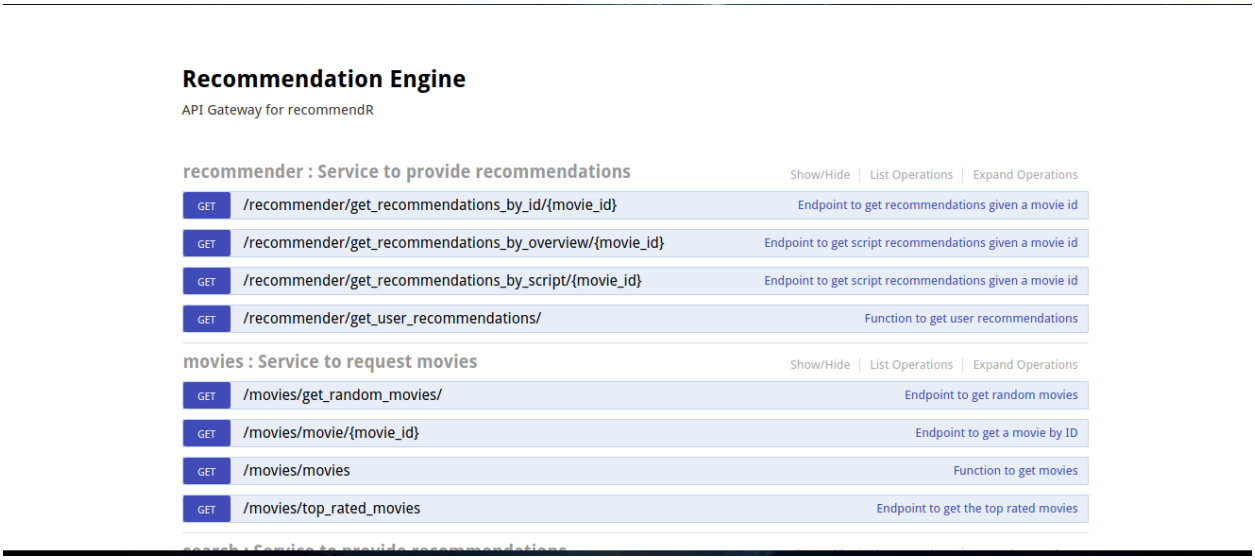


Figure 8: API Gateway

6.2 Security

To ensure the system was secure certain approaches were followed. For account creation Auth0 was used. They offer a range of services such as encryption, password hashing, and a variety of tested and verified identity standards.

The accounts stored in Auth0 are encrypted and means that no one is able to view information such as passwords. The API gateway uses Auth0 and for certain endpoints requires a valid

json web token (JWT) to allow access to the endpoint. This could be extended further by allowing only certain endpoints based on role, or user type. However, due to time constraints this was not added.

In order to ensure that external API's such as client ID's and secrets are not exposed in the source code, environment variables have been used so that the credentials are not accessible. This ensures that the likelihood of breach for these services are unlikely as nothing has been committed to source code, so they will not be able to see them.

Due to using Heroku free tier, additional security steps to the infrastructure have not been made as they are only available to enterprise members. If there was a enterprise membership or was deployed to something like AWS, IP whitelisting would be used so that only the API gateway would be able to access the services it communicates with. In conclusion, the chance of a breach in one of these services would be unlikely.

7 Deployment

Due to a microservice design the system requires some extra steps in order to get it to work. The live production of the website has been deployed to Heroku [57] which is allows free web hosting. However, Heroku has some limitations such as availability and performance.

All the microservices have been dockerised in the eventuality of wanting to deploy in a different environment. Each service makes use of environment variables so that it is able to connect to the correct service.

To keep the MF algorithm up-to-date *run_matrix_factorisation.py* in the recommendation engine app has to be executed once a day, so that the model can improve.

7.1 Local Deployment

For local deployment a Docker compose script was created which enables all the microservices to be instantiated by a single command. This is extremely useful as it means that the entire applications dependencies can be installed in a particular container and the only requirement is that the host machine can support Docker [58].

An example command for creating an environment would be:

```
docker-compose run up -d
```

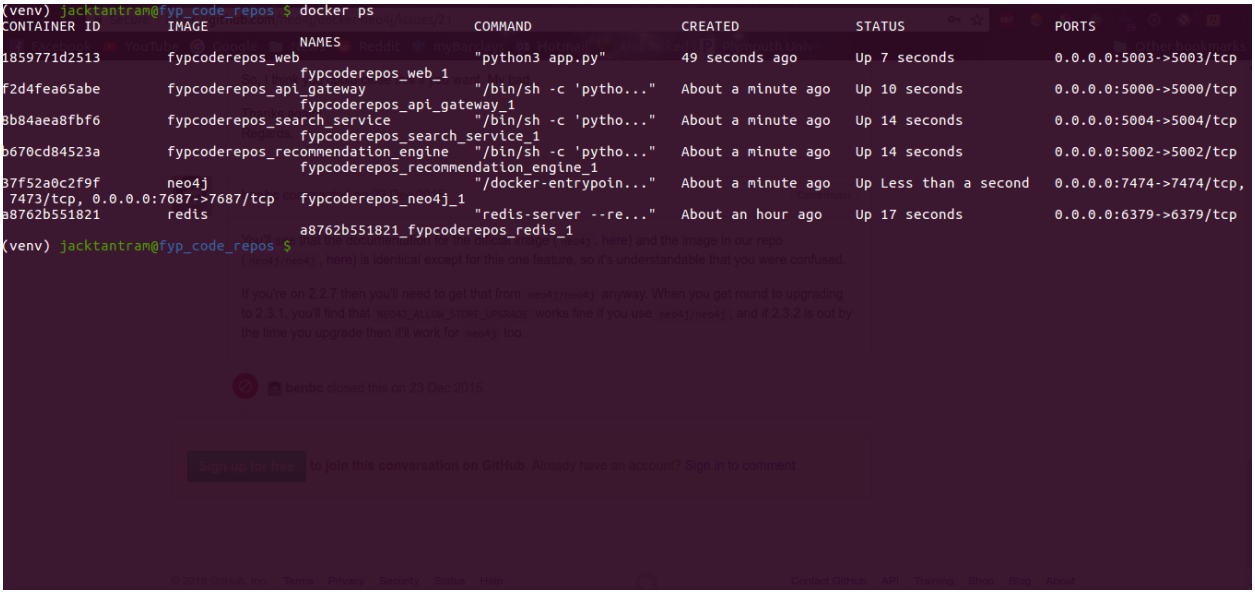


Figure 9: Demonstration of all running services

7.2 Production Deployment

For production deployment a docker compose script has been created, however due to using Heroku for the project it is not being used for the deployed website. Although this would be useful if wanting to migrate to AWS.

8 Sprint Iterations

8.1 Sprint 1

Aim:

The first sprint focused on gaining an understanding of existing recommendation algorithms. This would provide guidance towards the most important requirements. In addition to this, the sprint also wanted to get an understanding for the potential datasets that were available to see what was feasible.

If the initial research went well and the datasets had been evaluated thoroughly, the sprint also aimed to create requirements so that critical tasks were prioritised.

Outcome:

Overall, the sprint went well everything was completed on time. A document was created outlining different types of recommendation algorithms, which included their benefits and disadvantages C.3. Fundamentally, the research highlighted the three main types of recommendation algorithm and put emphasis on finding a suitable dataset. The research for these datasets

is outlined in section C.2.

Due to the finishing tasks faster than expected, there was enough time to create requirements. The functional requirements were created using MoSCoW because it wanted to be used as a prioritisation technique and the non-functional followed FURPS because it provides greater detail.

Deliverables:

- Recommendation algorithm analysis document
- Dataset Evaluation
- Functional and non-functional requirements

8.2 Sprint 2

Aim:

The second sprint aimed to create the systems architecture, and create an initial scaffold for its implementation. Additionally, the sprint aimed to conduct research towards the algorithms that could be used for the project.

Outcome:

The architectural design took longer than expected because the requirements kept changing. Once formalised it was much easier to produce the architecture design. The services were setup using Flask and had basic HTTP communication between apps; this was deployed on Heroku.

It was clear that the production database could be problematic, due to limitations with Heroku's free tier. There were a range of Neo4j hosting services, GrapheneDB [59] was the best option. Due to using free tier it presented several memory limitations that were made aware of.

Finally, research was conducted towards potential algorithms. Basic prototypes were implemented in Jupyter to understand how the algorithms worked. This compared item-item, user-user relationships with different similarity metrics. Also, further research had been conducted towards matrix factorisation and a basic notebook was created.

Deliverables:

- System Architecture (Appendix G)
- Flask microservices, with basic HTTP communication.
- Deployed apps to Heroku

- Jupyter notebooks comparing user-user similarities and item-item
- Initial notebook of Matrix factorisation in NB.

8.3 Sprint 3

Aim:

The main aim for the third sprint was to improve the matrix factorisation algorithm and integrate it with Neo4j and the recommendation engine service. If there was enough time then it would also be integrated with the API gateway, so that it could be easily tested.

Outcome

There were a couple difficulties combining the algorithm with the back-end. The initial idea was to execute the LFM algorithm using the ML dataset to gain predicted rating values and insert them into the database; then Neo4j would calculate the similarity metric at run-time.

However, due to memory limitations as aforementioned this was not feasible because creating a new ratings relationship between users would require (movies) 9066 * 671 (users). After some considerations, it was decided that the similarity matrix would be precomputed and the top 20 recommendations for a movie would be inserted as a relationship (Has_Similarity). Amazon have implemented something similar [60], where they pre-process their recommendation information offline, so that it isn't being completely stored in the database.

Furthermore, it aided in scalability because the system does not have to store the complete relationship matrix. After successfully integrating the MF script with the database, work began to provide communication between the API gateway and recommendation service. The queries themselves to retrieve the data were not difficult, but it was found that the overall query times were slow. Therefore, Redis was added to cache queries so that the gateway would not have to repeatedly call the recommendation service.

The next sprint was aiming towards creating the website, which meant that UI designs were produced also in sprint 3.

Deliverables:

- MF integration with database
- Integration with API Gateway - beneficial for testing
- Redis implementation for caching queries
- Redis query improvement
- UI Designs (Appendix F)

8.4 Sprint 4

Aim:

Sprint 4's goal was to implement the UI designs and connect the website to the API gateway. This would require a method of retrieving relevant information for movies as the ML dataset does not contain information such as images, or video links.

Outcome:

To be able to obtain images, overviews and additional fields, the TMDb API was queried. The script produced a CSV which was then ingested to update the relevant fields in the database. However, there were cases where certain fields were unavailable. This has been noted as a limitation in the system because there isn't a feasible alternative to obtain for all the films in the system.

The website integration started off quite slow due to not having any experience with React.js. In order to use React, there was a requirement for using Webpack to transpile the JavaScript. This caused a bit of delay but once implemented it was beneficial because Webpack provides a range of functionality, such as minifying CSS, and obfuscating JS which is desirable in a production environment.

Due to the requiring personalised recommendations, the first task for the website was to implement user account creation. Initially, Flask Security [61] was chosen as a method to create accounts. However, it became apparent that this would require a lot of extra work as Login and Signup pages would need to be created. Also, there would need to be password resetting and regular expressions to validate credentials. Therefore, the service provider Auth0 was used because the functionality is included, and they provide industry standard authentication. The unique identifier that Auth0 supplies would then be stored as a reference in the database.

Once login was implemented, the initial ratings and home page were constructed so that the users could view recommendations.

It became a lot easier to see recommendations using MF were not perfect. This provided the acknowledgement that the script needed to be improved. Alternatively, there was an idea to use TF-IDF based on movie overviews to see if it could produce relevant recommendations. A notebook was created to test this; it suggested that TF-IDF was feasible due to producing relevant recommendations.

Deliverables:

- Extra movie content integration (image url, video url, overviews)
- User account creation
- Initial ratings page, main page for recommendations

- Webpack integration
- TF-IDF implementation for movie overviews

8.5 Sprint 5

Aim:

The goal was to continue with the website integration to incorporate search and provide another page to show a movie individually with its recommendations. Additionally, to provide further research into how to improve the matrix factorisation algorithm.

Outcome:

After conducting research, there were a range of methods shown to improve MF accuracy, such as including additional input sources. This led to investigating whether incorporating genres and additional feedback such as 'thumbs up' or 'thumbs down' could be beneficial. The MF algorithm is quite slow and there was evaluation to see if this could be improved. Fangling Li et al [62] described a method to distribute SGD which looked promising, but it was evaluated that there wasn't enough time to implement.

Alternatively, TF-IDF provided useful recommendations, however they weren't ideal. This led to the implementation of using TF-IDF on movie scripts to compare whether this would produce better recommendations. Predominately due to scripts containing a larger corpus than overviews.

Overall, the initial comparison showed that neither TF-IDF implementation was proportionally better. This highlighted that user testing would be crucial to evaluating the performance of the algorithm.

Deliverables:

- MF research towards improving performance
- TF-IDF implementation using movie scripts
- Further website integration. (Individual movie page, search)

8.6 Sprint 6

Aim:

The final sprint aimed to tidy up all the bugs in the system and validate the unit, and user tests. The sprint also aimed to add feedback buttons so that a user could signify a relevant recommendation. This would then be used to evaluate performance.

Outcome:

The final sprint completed everything it aimed for. All the noticeable bugs were eliminated and user feedback buttons were added. This would signify if a user liked a recommendation. If the recommendation received positive feedbacks it would increase the similarity score, alternatively if there was 50 negative responses then the relationship would be deleted. Due to the website design being ready, user testing was conducted to receive feedback and evaluate the performance of the algorithms.

Deliverables:

- Unit test validation
- User feedback buttons
- User testing

9 Testing

The application was tested using a range of methods. It used exploratory, unit, and user testing. To ensure a code quality elements were tested before they were integrated back into the master branch. Due to the system being split up into a set of microservices it meant that tests had to be implemented in all systems.

9.1 Exploratory Testing

Throughout the application there was a lot of manual testing. This involved making changes to code and testing its expected. The debugger of choice was PyCharm as it provides code breakpointing which means it is easier to inspect code. For the website to work it is heavily reliant on the API gateway communicating with its services. This led to a lot of testing through the swagger tool so that all endpoints that the website could use worked as expected.

9.2 Unit Testing

Unit testing is a principle in software engineering which refers to code being split up into units, where they are individually scrutinised to see if they fit their purpose. For this project, each microservice had a set of unit tests written to ensure that all of its endpoints were working correctly. The type of testing was split up into acceptance and failure.

To test for acceptance it aimed for something positive to happen. For instance, each unit that expected a HTTP response should return status code of 200; this stands for OK. If the request returned this then the test would pass.

On the other hand, testing for failure would expect a status code of 400; bad request and should return an appropriate error message. This was to ensure the application did not crash

if sending incorrect HTTP parameters for instance.

By implementing unit tests it is beneficial, especially with microservices as it provides visibility into what endpoints are working.

9.3 User Testing

Once the website interface had been completed, user tests were conducted with 5 University students on campus. Originally, it was planned for earlier on in the project but due to delays it was carried out much later. The study complied with all the conditions stated in the Approved Ethics form.

The participants had two tasks to complete. For the first task they had to interact with the website and supply feedback via TypeForm [63] (Appendix J.1) The users were asked 10 short questions, such as, the user experience, things they liked and areas that they would change.

Firstly, they were asked what they though the website was about. All users answered they thought it was about recommendations, although one user stated they weren't sure what the recommendations were for. The landing page only contained the website's logo so alterations were made to provide more information.

Secondly, what they liked about the website. They were positive comments about the carousels which displayed the film posters and noted that they were intuitive. Although, the initial ratings page was found to be counter-intuitive as certain users struggled to understand what they had to do.

Overall, the reviews were positive with a couple notes on appearance such as the ratings, and fonts. The layout is important because users will be more likely to return if they can easily find the information they're looking for [64]. Therefore, feedback was taken into account and alterations were made.

The second part of the test consisted of users being presented with 3 different carousels that showed recommendations. The users had to provide an evaluation (positive, negative) for a particular recommendation. Each carousel depicts recommendations that were retrieved using a different algorithm; MF, and TF-IDF. TF-IDF similarity was calculated using movie scripts and overviews. The overall efficiency of the algorithm was calculated by comparing how many positive recommendations versus negative.

The task involved users giving feedback for 5 films; *Toy Story*; *Star Wars: The Empire Strikes Back*; *Aladdin*; *Pulp Fiction*; *Austin Powers (1997)*. Each algorithm produced 10 recommendations and the users had to press the feedback button to indicate what they thought of the recommendation. Typically, the user interface would show what algorithm the recommendation came from, but this was redacted when shown to the users so that it could somehow influence their opinion. During the task a couple users commented on the

phrasing of how the recommendations were presented, 'Films similar to'. They user's were not sure whether it should be the same, or something related, this is most likely didn't have an effect on the results.

Fig 10 shows an example of what the user saw when participating in the test.

Figure 10: Example feedback test for *Star Wars: The Empire Strikes Back*. MF (A), TF-IDF Scripts (B), TF-IDF Overviews (C)

10 Results

To evaluate each algorithm's efficiency, users were shown a series of films and asked to rate whether they thought it was a relevant recommendation or not. It was noticed afterwards that some feedback buttons were not pressed. This meant that instead of receiving a total

of 750 (5 users * 5 films * 30 recommendations), there was 741. This is not a significantly large number, but could have influenced the overall results slightly.

Fig 1 illustrates the user testing results. It was calculated by taking the number of positively identified items in a list i.e. if a user states 7 items out of 10 were correct. The average was calculated across all users for each film.

Overall, the results showed to be relatively low for all algorithms, with MF ranking highest. Therefore, results were calculated for the first 5 recommendations that were ranked the most similar. The results suggested that all algorithms improved. Overall MF was ranked the highest with an average value 0.55, $N = 5$.

Table 1: User Testing Results

	MF N=10	MF N=5	TF-IDF Movie Scripts N=10	TF-IDF Movie Scripts N=5	TF-IDF Movie Overview N=10	TF-IDF Movie Overview N=5
Toy Story	0.68	0.72	0.04	0.08	0.24	0.48
Star Wars Episode V	0.3	0.6	0.46	0.4	0.50	0.8
Aladdin	0.38	0.56	0.02	0.04	0.48	0.72
Pulp Fiction	0.36	0.52	0.14	0.17	0.06	0.04
Austin Powers (1997)	0.20	0.36	0.20	0.38	0.20	0.42
Average	**0.38**	**0.55**	**0.17**	**0.21**	**0.30**	**0.49**

11 Discussion

The accuracy between all algorithms was quite low, there are a couple of speculations to why this may be. After being notified at the end of the experiment what each carousel represented, the users stated that the carousels using TF-IDF for the first couple items did highlight similar films. Also, the recommendations were stated to be too obvious because they were typically a film in the same series, such as another *Star Wars* film. It meant that there is no sense of novelty and the recommendations become useless [65]. During the experiment a couple users got confused with the wording 'Similar to' which could have influenced their results slightly, however it shouldn't have been affected by much.

Furthermore, for certain films they do not contain an image due to the limitations of using an external API. This meant that the only descriptor for the film was the title in the carousel. Sunderesan et, al. [66] discovered that images serve as an important factor for potential buyers as it helps increase attention . Therefore, this could also translate to recommendations and would be interesting to discover how much images and their quality matter when making providing suggestions to users.

The MF algorithm showed to provide relevant recommendations and the types of films shown were noted as less obvious. However, after the first 5 items performance started to degrade. A reason for this could be that the algorithm does not use any additional input sources. Many of the suggestions after 5 items seemed random. They were from different genres or targeted towards a unrelated age category. There has been research in incorporating tags during MF, which has proven to be beneficial and should improve the overall accuracy. Creating a vector representation that took into account information such as director, age range, and common actors could have produced better results.

Finally, testing was conducted using a very small sample. This questions the validity of the experiment because most of the participants hadn't seen all the films, and meant that they were making guesses based off of the poster images. This could have definitely impacted the results and would require further testing with a larger sample size.

12 End Project Report

12.1 Project Summary

The project aimed to understand the methods and tools for creating a recommendation system. Its main purpose was to evaluate commonly used approaches and analyse how to implement them in a production environment. During initial research it highlighted that there are 3 main approaches for recommendation algorithms. They are defined as CF, CB, and hybrid, which is a combination of the first two. Each approach outlines distinct advantages and limitations and the project's goal was to understand them.

In order to achieve this, the project experimented using a range of recommendation algorithms. A single-paged web application was produced that suggests movie recommendations. It allows users to rate and give feedback, which aids in the overall performance.

Recommendation systems operate at a large scale and incorporate a considerable amount of data, this means that there architecture has to be scalable. Therefore, the project evaluated methods to do this and adopted a microservices architecture which enables the system to be distributed and extensible.

12.2 Evaluation

The following highlights the initial project objectives and indicates if they have been achieved:

- ☑ Analyse different recommendation algorithms
- ☑ Evaluate methods and tools for building recommender systems
- ☑ To provide recommendations for users
- ☑ Create a robust and extensible system architecture
- ☑ Create a responsive web application
- ☑ Follow best practices and produce well documented code
- ☑ Implement a testing framework, to ensure stability
- ☑ Create a system that can be deployed easily

Objective 1:

This objective was met by conducting extensive research into different types of recommendation algorithms. It was carried throughout all stages of the project, making comparisons with existing websites and implementations. Assessments were conducted to understand the advantages and limitations of various approaches, whilst using experiments to evaluate performance. Unfortunately, the project didn't manage to incorporate any extra methods to improve MF, but the report highlights methods to do so.

Objective 2

This objective was method by evaluating different algorithms and formalising an architecture for a recommendation system. The report, source code, and guide will help readers create their own systems.

Objective 3:

The website allows users to rate content and receive personalised recommendations based on what they have rated. It is presented to the user in an easy to understand format and has aimed to follow HCI principles. The objective itself is quite broad but the algorithms implemented mean that they can be applied to different domains, and not just films.

Objective 4:

A lot of research went into creating a robust, and extensible architecture. It involved comparing different architectures to understand what be the best for the project. A microservice approach was chosen, which meant that everything was distributed and makes it a lot easier to expand in the future. The choice of architecture was difficult to implement and resulted in tasks taking longer to complete. An understanding of the complexity earlier on would've been better to make more accurate time estimations.

Objective 5:

A critical factor when developing a web application is responsiveness. If it doesn't scale properly on devices, users will not use the site. The application uses Bootstrap to aid with this and scales correctly on laptop displays, tablets, and monitors. It is accessible on mobile, but the user experience is poor as this would add more complexity to the project.

Objective 6:

This project aimed to follow best practices by documenting code, conforming to language specific standards, such as PEP8, and focused making code as reusable as possible.

Objective 7:

This objective was met by implementing unit tests and ensured that services were communicating correctly. Unfortunately, the project did not include any supporting front-end testing frameworks due to time constraints, but would in the future. An initial goal was to use continuous integration to ensure functionality in deployment but this didn't happen due to difficulties setting up a testing database.

Objective 8:

This was adhered to by using Docker. This allows the whole environment to be instantiated with a single command, it also means that it is machine agnostic. It conforms to documentation as it is completely described in the project user guide.

Must have:

- ☑ Be able to create an account
- ☑ Password resets
- ☑ Be able to view recommendations
- ☑ Be able to rate movies
- ☑ Accessible on either tablet or computer

The project also managed to adhere to all the must have user requirements. It almost met all the 'should haves' apart from being able to search by genre. The website can filter by genre, but you cannot include it as an extra filter parameter in the search page.

13 Project Post-Mortem

The project was successful, however there were a couple areas that could have been improved on. For instance, understanding the content in scientific papers towards recommendation algorithms proved to be confusing. A lot of them were complicated and took a while to understand due to a variation in notation.

It was clear that requirement's analysis was essential for the success of the project. By highlighting the functional and non-functional requirements at the beginning it provided a good indication for what needed to be done. The MoSCoW method helped provide guidance and ensured that everything that was essential should be completed. Due to not having a client, it meant that a lot of research had to be made to assign and prioritise the requirements effectively. Incorporating user feedback earlier on would have been beneficial, as it would've helped when creating requirements.

The adoption of the agile methodology proved to be effective. Especially as requirements

changed quite a lot during the project and didn't cause too much impact on what was being delivered. Trello felt quite cluttered at times. It became difficult to understand what needed to be done and wasn't to view how the project was progressing. The free version of Trello lacks in a lot of functionality and is quite simplistic, whilst other tools such as Jira would've been more effective when managing the project because they allow more detail.

Overall, it appeared that the technology choices for the project were suitable. Python was a good choice as the back-end language, due to it having a lot of built in libraries for the data science domain which helped reduce development time. It meant that code could be prototyped in Jupyter notebooks which proved to be extremely beneficial when creating prototypes. However, as dataset sizes increased it was apparent that the algorithms could take a considerable amount of time to run, 2 hours in Python. Writing code in parallel would definitely improve the time it would take to complete the SGD algorithm. Although, this would require a different approach and would need some redesign. For the use of the 100k dataset the current implementation works well.

React.js was quite difficult to understand due to not having any prior experience, however it demonstrated to be extremely useful as the project progressed with its component based system. This meant that code reuse was strongly encouraged and kept the codebase a lot cleaner.

In addition, most of the user interface was consistent with the original designs (Appendix F). The login and registration page were not created because Auth0 provides this. The major difference was the initial ratings page and search, because the design was too complex to create within the time constraints.

The choice of database seemed to the best option after experimenting between a GDb and relational database. It was a lot easier to write Cypher than SQL, which reduced complexity when designing queries. The only downside for the project was that all the similarity metrics were calculated offline due to memory limitations. It would've been interesting if it was possible to experiment it alternatively to see the results.

The architectural design changed a couple times during implementation but the aim for a microservice approach was still followed. Overall, it did impede development and brought in extra costs. This was primarily due to everything having to be designed separately and then integrated. By using Docker it helped a lot because it meant that all the logs could be accessed in a centralised place. It also meant that there was reassurance when deploying because it wasn't machine specific as all the dependencies would be containerised in Docker.

The choice of algorithms seemed to be relevant. MF proved to produce accurate recommendations, but there were difficulties understanding the methods in which to improve it. There was research that evaluated different methods for improvement but unfortunately wasn't able to implement them. The ML dataset includes a separate file that contains movie tags and would've been interesting to integrate this but due to time this didn't happen. However, the website did implement feedback buttons that improve recommendation accuracy using user

feedback, so this means that the website performance could improve over time.

There were a range of lessons learned, one would be to put a greater emphasis on testing. Particularly, recommender systems require a lot of user feedback in order to understand whether there are good recommendations. Testing was left late and the project would've benefited from it at an earlier stage. Secondly, it was found when testing in the production system a lot of errors would prop up that wouldn't in the local system. This was primarily due to a difference in datasets as the Heroku limited the neo4j instance to 1000 nodes. A way to approach the problem would be using a testing database and continuous integration, this would ensure that everything is working correctly before it is deployed.

14 Conclusion

The project set out to understand the methods and tools to create a recommender system. Research suggested that there are a variety of recommendation algorithms and their interest is rising. To be able to create such a system, evaluations were made to assess different architectures and their implementations.

This highlighted the 3 core types of recommendation algorithm and the importance of architecture design. Therefore, to demonstrate findings, the project created a web application that provides recommendations using a combination of algorithms. The project was successful, the main deliverable satisfies all core and desirable requirements. Furthermore, the system has a robust, extensible architecture that can be developed for future use, as well as guide for other systems.

References

[1] Eurostat. Eurostat - data explorer. `https://goo.gl/ZSnXJe`. (Accessed on 05/06/2018).

[2] Alexander JAM Van Deursen and Jan AGM Van Dijk. Using the internet: Skill related problems in users' online behavior. *Interacting with computers*, 21(5-6):393–402, 2009.

[3] Mckinsey & Company. How retailers can keep up with consumers | mckinsey & company. `https://www.mckinsey.com/industries/retail/our-insights/how-retailers-can-keep-up-with-consumers`. (Accessed on 05/06/2018).

[4] Francesco Ricci, Lior Rokach, and Bracha Shapira. Introduction to recommender systems handbook. In *Recommender systems handbook*, pages 1–35. Springer, 2011.

[5] Loren Terveen and Will Hill. Beyond recommender systems: Helping people help each other. *HCI in the New Millennium*, 1(2001):487–509, 2001.

[6] Michael J Pazzani and Daniel Billsus. Content-based recommendation systems. In *The adaptive web*, pages 325–341. Springer, 2007.

[7] Robert Prey. Nothing personal: algorithmic individuation on music streaming platforms. *Media, Culture & Society*, page 0163443717745147, 2017.

[8] Microsoft. What is agile? - azure devops | microsoft docs. `https://docs.microsoft.com/en-us/azure/devops/agile/what-is-agile`. (Accessed on 05/17/2018).

[9] What is sprint (software development)? - definition from whatis.com. `https://searchsoftwarequality.techtarget.com/definition/Scrum-sprint`. (Accessed on 05/17/2018).

[10] MountGoatSoftware. Scrum methodology and project management. `https://www.mountaingoatsoftware.com/agile/scrum`, no date. (Accessed on 17/04/2018).

[11] Kanban vs. scrum: What are the differences? | leankit. `https://leankit.com/learn/kanban/kanban-vs-scrum/`. (Accessed on 05/17/2018).

[12] Jupyter. Project jupyter | documentation. `http://jupyter.org/documentation`. (Accessed on 19/05/2018).

[13] Bitbucket. Bitbucket | the git solution for professional teams. `https://bitbucket.org/product`. (Accessed on 20/05/2018).

[14] Python. Our documentation | python.org. `https://www.python.org/doc/`. (Accessed on 19/05/2018).

[15] Flask. Welcome to flask — flask 0.12.4 documentation. `http://flask.pocoo.org/docs/0.12/`. (Accessed on 19/05/2018).

[16] Django. Django documentation | django documentation | django. `https://docs.djangoproject.com/en/2.0/`. (Accessed on 19/05/2018).

[17] React.js. React - a javascript library for building user interfaces. `https://reactjs.org/`. (Accessed on 19/05/2018).

[18] webpack. `https://webpack.js.org/`. (Accessed on 20/05/2018).

[19] Neo4j. The neo4j graph platform – the #1 platform for connected data. `https://neo4j.com/`. (Accessed on 19/05/2018).

[20] Redis. Redis. `https://redis.io/`. (Accessed on 19/05/2018).

[21] Sarah Hatton. Choosing the right prioritisation method. In *Software Engineering, 2008. ASWEC 2008. 19th Australian Conference on*, pages 517–526. IEEE, 2008.

[22] Peter Eeles. Capturing architectural requirements. *IBM Rational developer works*, 2005.

[23] Takeshi Nakajo, Katsuhiko Sasabuchi, and Tadashi Akiyama. A structured approach to software defect analysis. *Hewlett-Packard Journal*, 40(2):50–56, 1989.

[24] Rod Stephens. *Beginning software engineering*. John Wiley & Sons, 2015.

[25] InfoQ. Moving deliveroo from a monolith to a distributed system. https://www.infoq.com/news/2017/03/deliveroo-monolith-distributed, 03 2017. (Accessed on 05/02/2018).

[26] Twitter. The infrastructure behind twitter: Scale. https://blog.twitter.com/engineering/en_us/topics/infrastructure/2017/the-infrastructure-behind-twitter-scale.html. (Accessed on 19/05/2018).

[27] Netflix Technology Blog. System architectures for personalization and recommendation. https://medium.com/netflix-techblog/system-architectures-for-personalization-and-recommendation-e081aa94b5d8. (Accessed on 19/05/2018).

[28] Sam Newman. *Building microservices: designing fine-grained systems.* " O'Reilly Media, Inc.", 2015.

[29] Single sign on & token based authentication - auth0. https://auth0.com/. (Accessed on 20/05/2018).

[30] Pluralsight. Relational vs. non-relational databases: Which one is right for you? | pluralsight. https://www.pluralsight.com/blog/software-development/relational-non-relational-databases. (Accessed on 05/17/2018).

[31] PostgreSQL. Postgresql: Documentation. https://www.postgresql.org/docs/. (Accessed on 05/17/2018).

[32] PostgreSQL. Postgresql: Documentation: 9.5: Explain. https://www.postgresql.org/docs/9.5/static/sql-explain.html. (Accessed on 05/17/2018).

[33] Neo4j. 3.6.2. profiling a query - 3.6. query tuning. https://neo4j.com/docs/developer-manual/current/cypher/query-tuning/how-do-i-profile-a-query/. (Accessed on 05/17/2018).

[34] Surajit Medhi and Hemanta K Baruah. Relational database and graph database: A comparative analysis. *Journal of Process Management. New Technologies*, 5(2):1–9, 2017.

[35] Alan Dix. *Human-Computer Interaction*, pages 1327–1331. Springer US, Boston, MA, 2009.

[36] Fiona Fui-Hoon Nah. A study on tolerable waiting time: how long are web users willing to wait? *Behaviour & Information Technology*, 23(3):153–163, 2004.

[37] Ben Shneiderman, Catherine Plaisant, Maxine Cohen, Steven Jacobs, Niklas Elmqvist, and Nicholas Diakopoulos. *Designing the user interface: strategies for effective human-computer interaction*, pages 88–89. Pearson, 2009.

[38] Thierry Bertin-Mahieux, Daniel PW Ellis, Brian Whitman, and Paul Lamere. The million song dataset. In *Ismir*, volume 2, page 10, 2011.

[39] Julian McAuley. Amazon review data. http://jmcauley.ucsd.edu/data/amazon/. (Accessed on 01/05/2018).

[40] Grouplens. Book-crossing | grouplens. https://grouplens.org/datasets/book-crossing/. (Accessed on 01/05/2018).

[41] Grouplens. Movielens | grouplens. https://grouplens.org/datasets/movielens/. (Accessed on 01/05/2018).

[42] Badrul Sarwar, George Karypis, Joseph Konstan, and John Riedl. Item-based collaborative filtering recommendation algorithms. In *Proceedings of the 10th international conference on World Wide Web*, pages 285–295. ACM, 2001.

[43] Fidel Cacheda, Victor Carneiro, Diego Fernández, and Vreixo Formoso. Comparison of collaborative filtering algorithms: Limitations of current techniques and proposals for scalable, high-performance recommender systems. 5:2, 01 2011.

[44] Dheeraj kumar Bokde, Sheetal Girase, and Debajyoti Mukhopadhyay. Role of matrix factorization model in collaborative filtering algorithm: A survey. *CoRR, abs/1503.07475*, 2015.

[45] Yehuda Koren, Robert Bell, and Chris Volinsky. Matrix factorization techniques for recommender systems. *Computer*, 42(8), 2009.

[46] Computing recommendations at extreme scale with apache flink™ - data artisans. https://data-artisans.com/blog/computing-recommendations-at-extreme-scale-with-apache-flink. (Accessed on 20/05/2018).

[47] Yehuda Koren. Collaborative filtering with temporal dynamics. *Communications of the ACM*, 53(4):89–97, 2010.

[48] Bu Sung Kim, Heera Kim, Jaedong Lee, and Jee-Hyong Lee. Improving a recommender system by collective matrix factorization with tag information. In *Soft Computing and Intelligent Systems (SCIS), 2014 Joint 7th International Conference on and Advanced Intelligent Systems (ISIS), 15th International Symposium on*, pages 980–984. IEEE, 2014.

[49] Yonghong Yu, Can Wang, and Yang Gao. Attributes coupling based item enhanced matrix factorization technique for recommender systems. *arXiv preprint arXiv:1405.0770*, 2014.

[50] Jürgen Backhaus. The pareto principle. *Analyse & Kritik*, 2(2):146–171, 1980.

[51] Jure Leskovec, Anand Rajaraman, and Jeffrey David Ullman. *Mining of massive datasets*. Cambridge university press, 2014.

[52] TMDb. The movie database (tmdb). https://www.themoviedb.org/?language=en. (Accessed on 19/05/2018).

[53] IMSDb. The internet movie script database (imsdb). http://www.imsdb.com/. (Accessed on 19/05/2018).

[54] Scikit-learn. sklearn.feature_extraction.text.tfidfvectorizer — scikit-learn 0.19.1 documentation. http://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.TfidfVectorizer.html. (Accessed on 19/05/2018).

[55] Andrew Hunt and David Thomas. *The pragmatic programmer: from journeyman to master*. Addison-Wesley Professional, 2000.

[56] Swagger. Documentation | swagger. https://swagger.io/docs/. (Accessed on 19/05/2018).

[57] Heroku. Cloud application platform | heroku. https://www.heroku.com/. (Accessed on 20/05/2018).

[58] Docker - build, ship, and run any app, anywhere. https://www.docker.com/. (Accessed on 21/05/2018).

[59] GrapheneDB. Neo4j cloud hosting, neo4j hosting | graphenedb. https://www.graphenedb.com/. (Accessed on 20/05/2018).

[60] Brent Smith and Greg Linden. Two decades of recommender systems at amazon. com. *IEEE Internet Computing*, 21(3):12–18, 2017.

[61] Flask Security. Flask-security — flask-security 3.0.0 documentation. https://pythonhosted.org/Flask-Security/. (Accessed on 20/05/2018).

[62] Fanglin Li, Bin Wu, Liutong Xu, Chuan Shi, and Jing Shi. A fast distributed stochastic gradient descent algorithm for matrix factorization. In *Proceedings of the 3rd International Conference on Big Data, Streams and Heterogeneous Source Mining: Algorithms, Systems, Programming Models and Applications-Volume 36*, pages 77–87. JMLR. org, 2014.

[63] Turn data collection into an experience | typeform. https://www.typeform.com/. (Accessed on 21/05/2018).

[64] Maristella Matera, Francesca Rizzo, and Giovanni Toffetti Carughi. Web usability: Principles and evaluation methods. In *Web engineering*, pages 143–180. Springer, 2006.

[65] Liang Zhang. The definition of novelty in recommendation system. *Journal of Engineering Science & Technology Review*, 6(3), 2013.

[66] Wei Di, Neel Sundaresan, Robinson Piramuthu, and Anurag Bhardwaj. Is a picture really worth a thousand words?:-on the role of images in e-commerce. In *Proceedings of the 7th ACM international conference on Web search and data mining*, pages 633–642. ACM, 2014.

A User Guide

recommendR User Guide

Jack Tantram

User guide for replicating recommendR environment, testing, and website interactions.

Introduction

This project assumes that you are using a linux based system with Docker and Docker Compose installed.

There is at least 5gb of available space available on the machine and 4gb RAM.

This is also under the assumption that the code repos have been installed and the docker-compose script has been set at the base of the project.

Installation

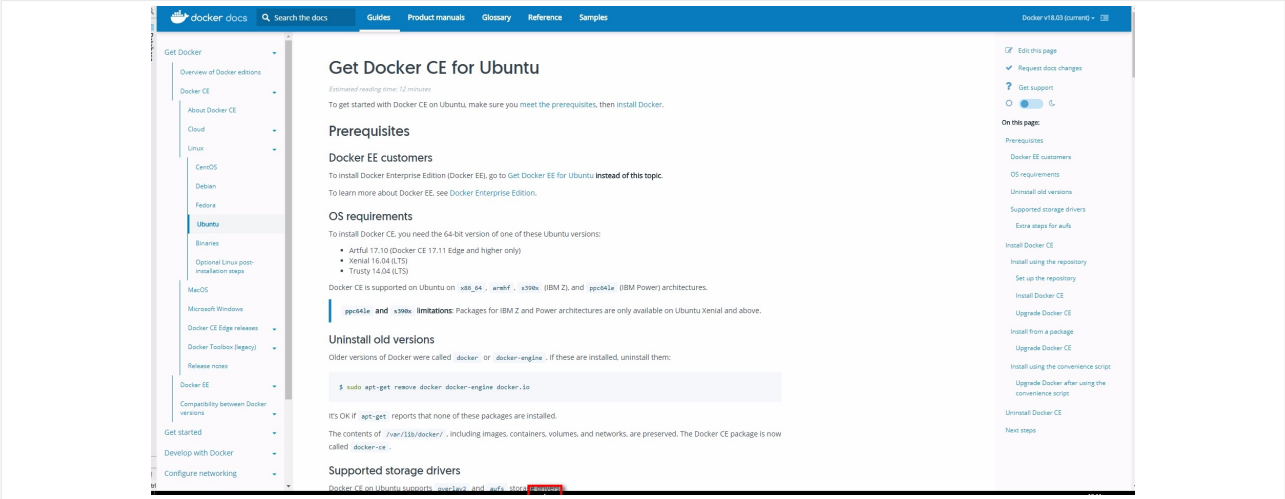
Go to URL for code repo

Go to for project source code:

https://liveplymouthac-my.sharepoint.com/:f:/g/personal/jack_tantram_students_plymouth_ac_uk/EvxV6c_YDBNBoI9YHzHXahwBqn_Fzhfij-bJmCbGSOxeTA?e=bCwfBg

Once the project has been downloaded go into the source_code folder for installation. Jupyter notebooks for the project are available in the jupyter_notebooks folder.

Install Docker for Linux

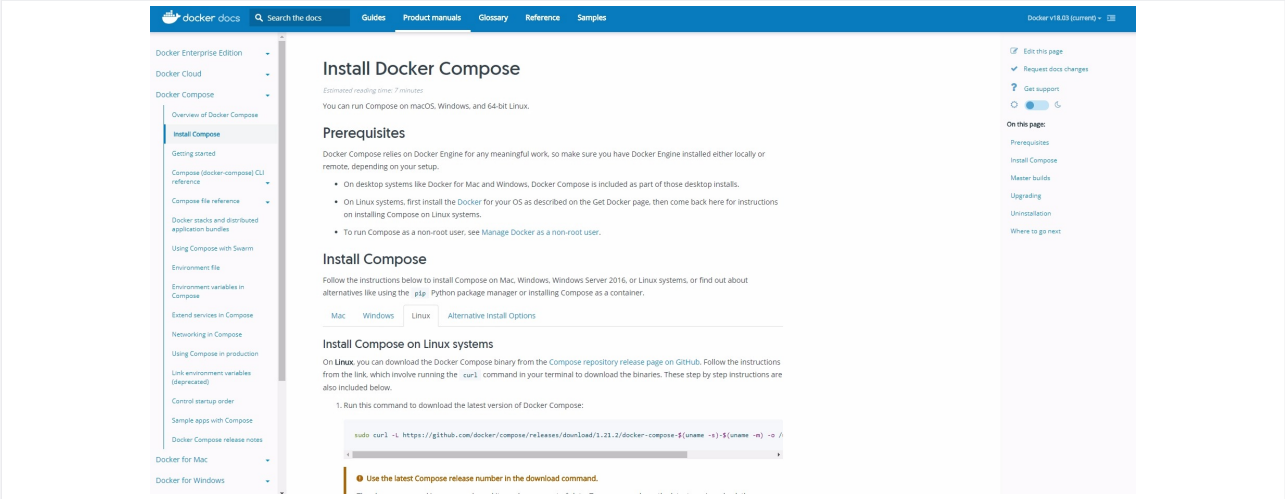


Docker can be installed at for Ubuntu:

https://docs.docker.com/install/linux/docker-ce/ubuntu/

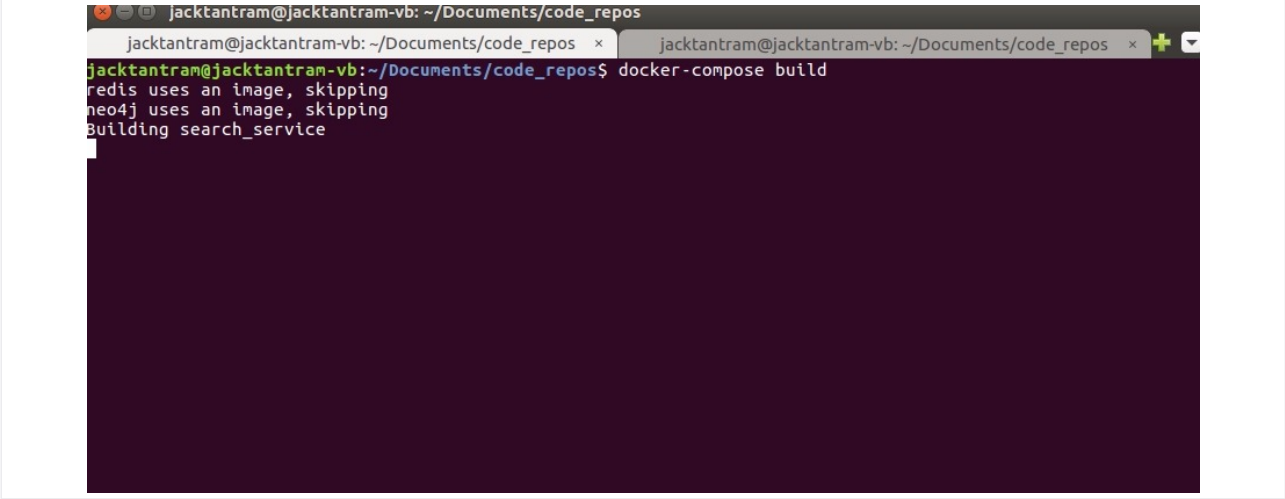
This project assumes you are using a linux based system. It is possible for a windows installation however.

Install Docker Compose



Install docker-compose for appropriate OS

Run docker-compose build

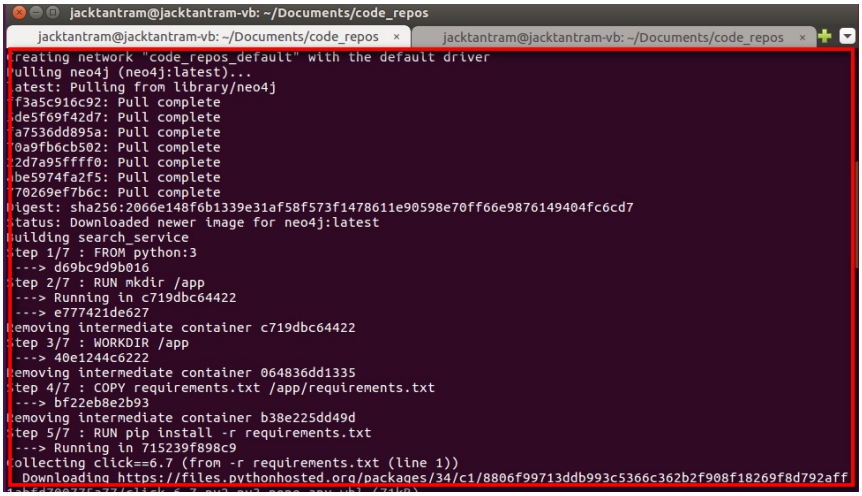


At the root folder of the project run the following commands:

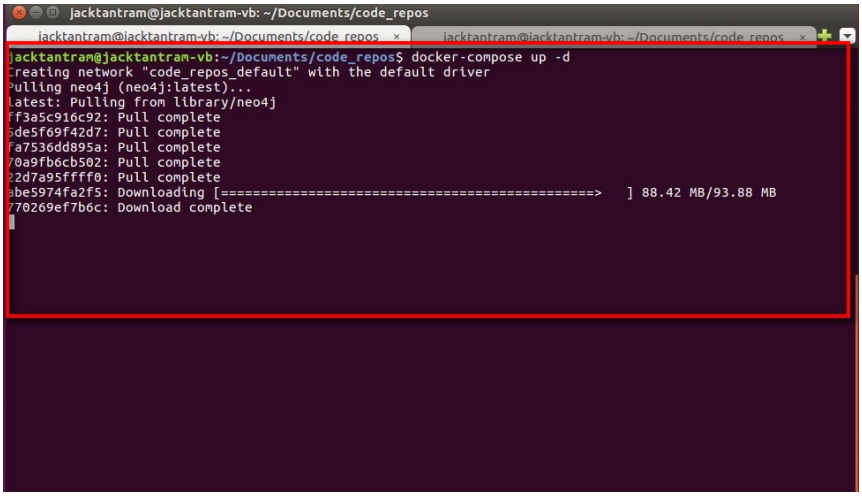
Run docker compose up -d to build and run docker-compose script

-d - signifies to run in the background, or daemon

Wait for dependencies to install



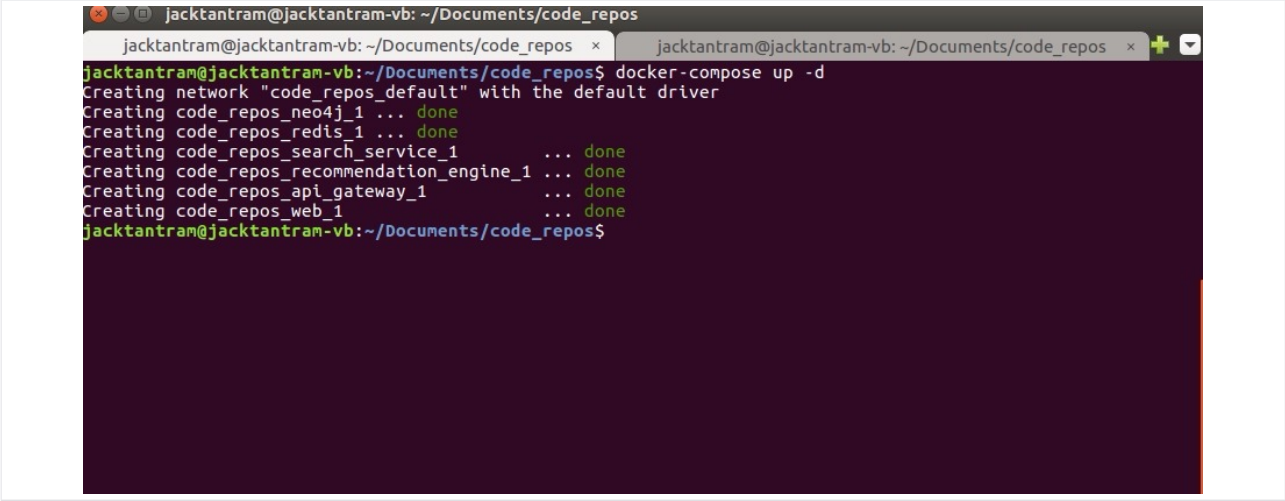
Run all services



Run docker-compose up -d to start all services and run them.

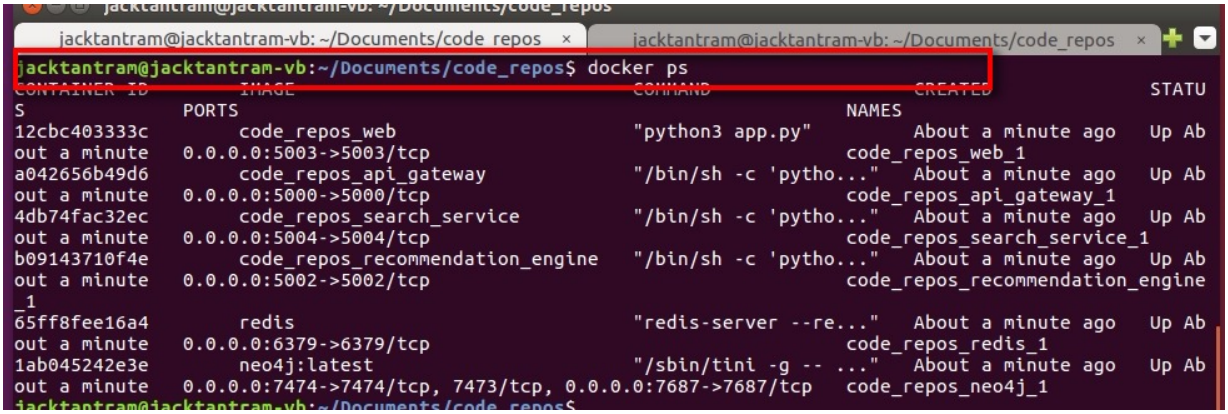
-d - signifies background and daemon mode

Indication of completion



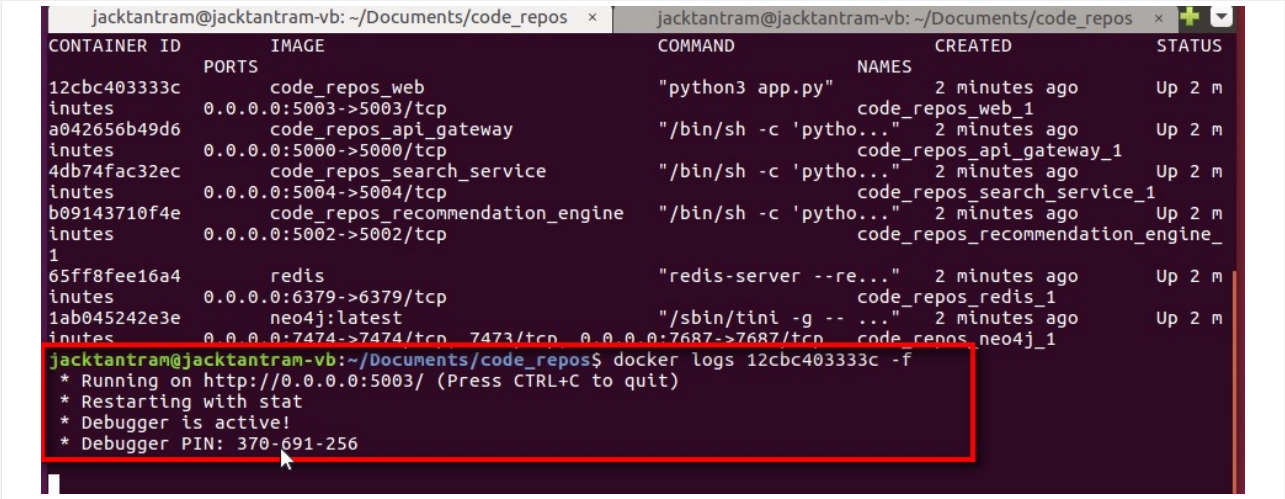
All will highlight done when complete and instantiated

View all running services



Run docker ps to view running services

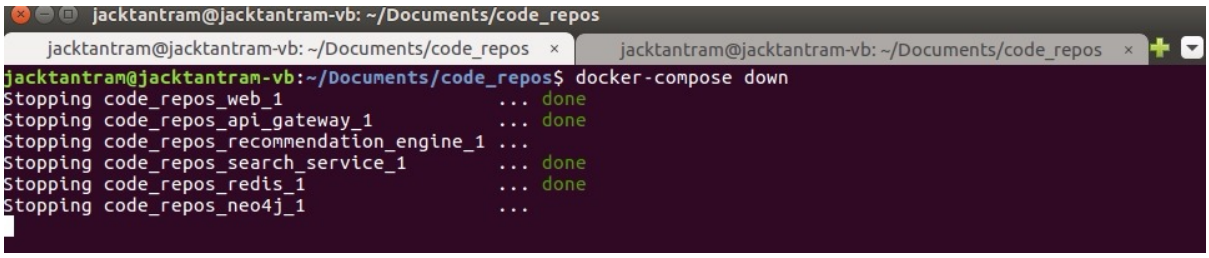
Viewing services logs



To view logs use docker logs <container_id> -f

-f - follow

Finally, to bring down the services



Run docker-compose down

Importing Data

To populate the Neo4j instance with movie data from the training set.

Use the command:

docker exec -it <code_repos_recommendation_engine> bash

Then execute:

python recommendation_engine/database_creation.py

Replicate production environment

Replicating Heroku environment

To replicate environment in Heroku, create a repository for each service.

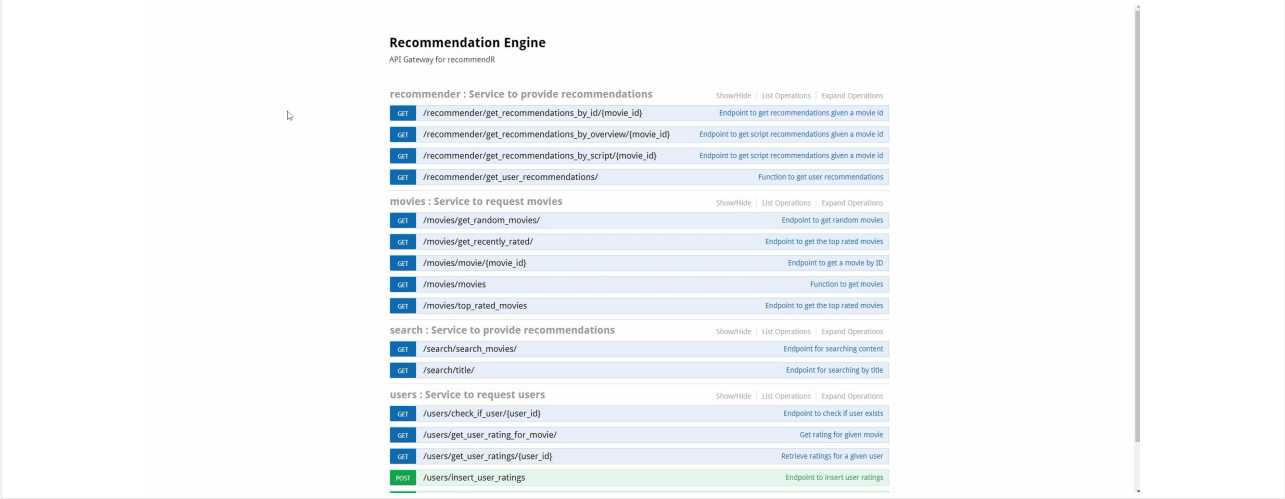
Web, API Gateway, Recommendations, and Search.

Then create an individual app for each service in Heroku and push code to remote.

Change api url in service_config file of each dependent service to be able to connect to the correct environment

Testing

View endpoints and api gateway

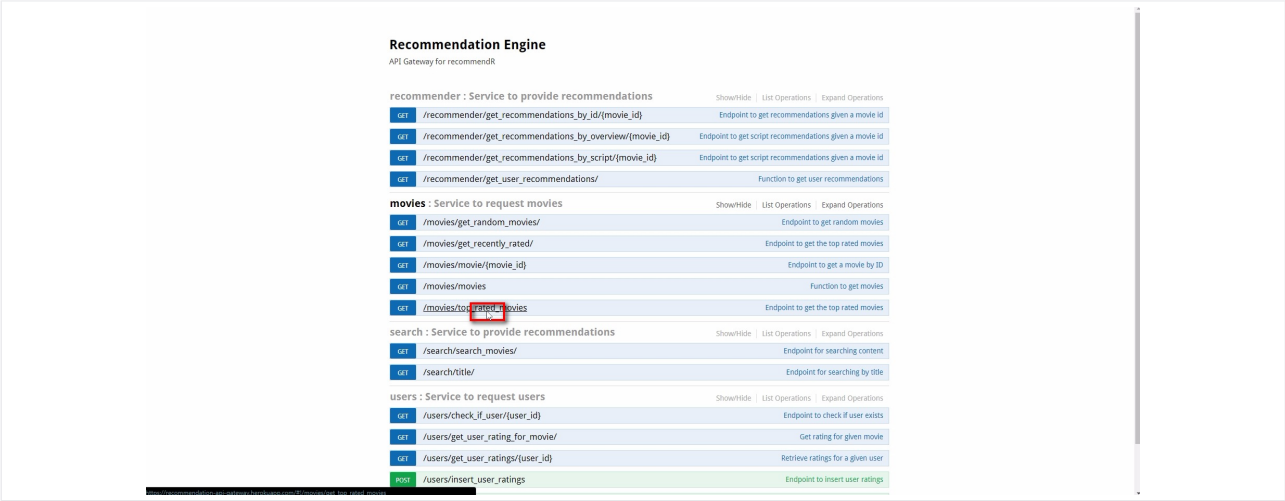


Go to either https://recommendation-api-gateway.herokuapp.com/ in production

or

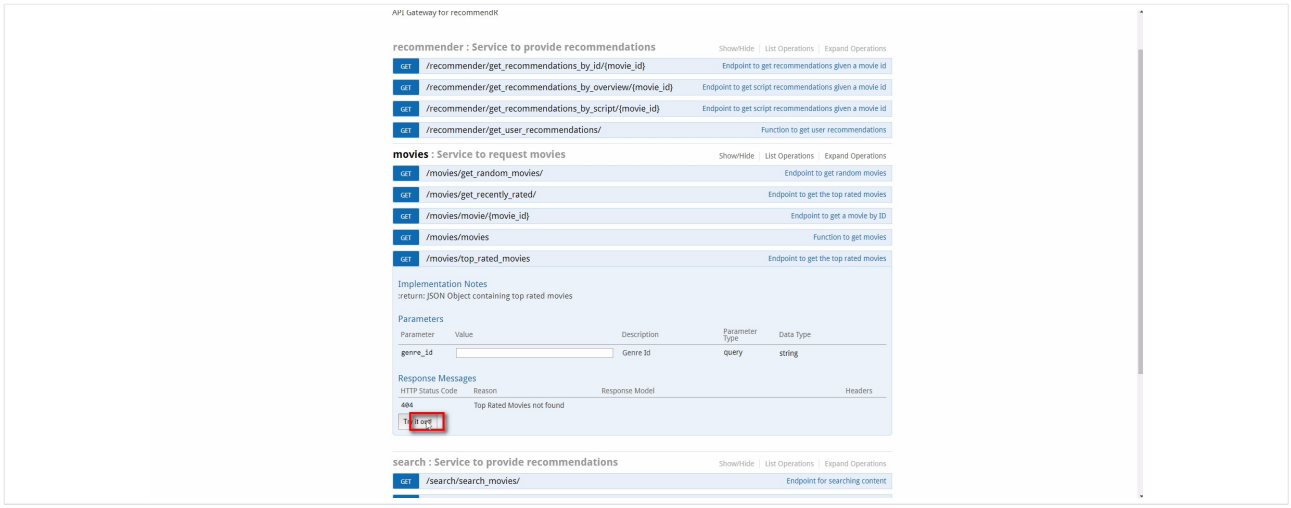
http://localhost:5000/ in development to view API gateway

Click on an endpoint to test



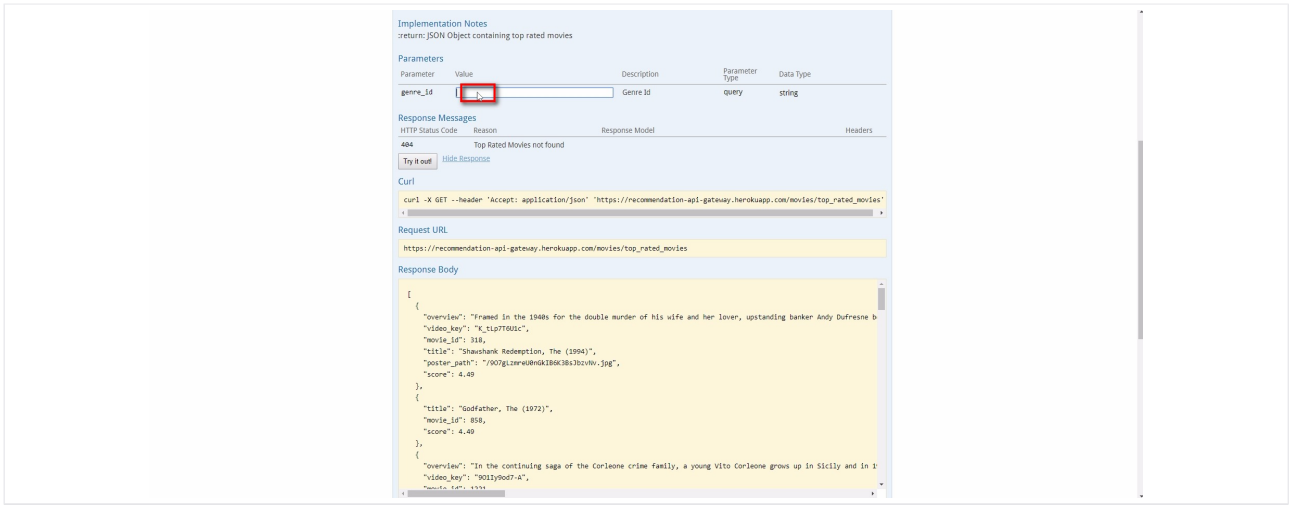
To view more information for an endpoint click on it to expand the list

Run query

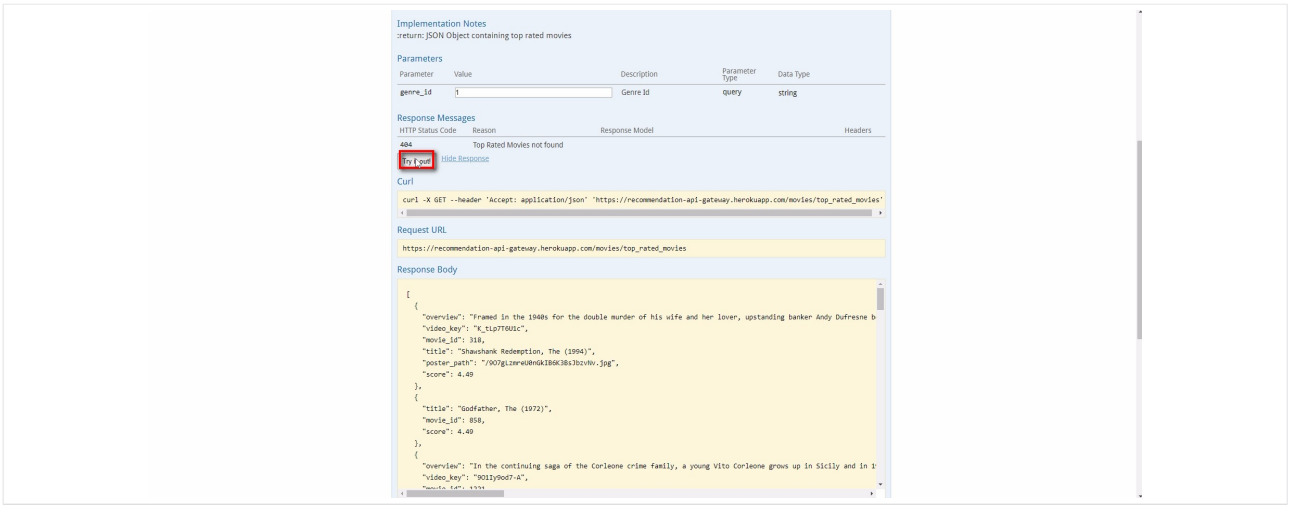


Press try it out to submit query

Assign optional param for results

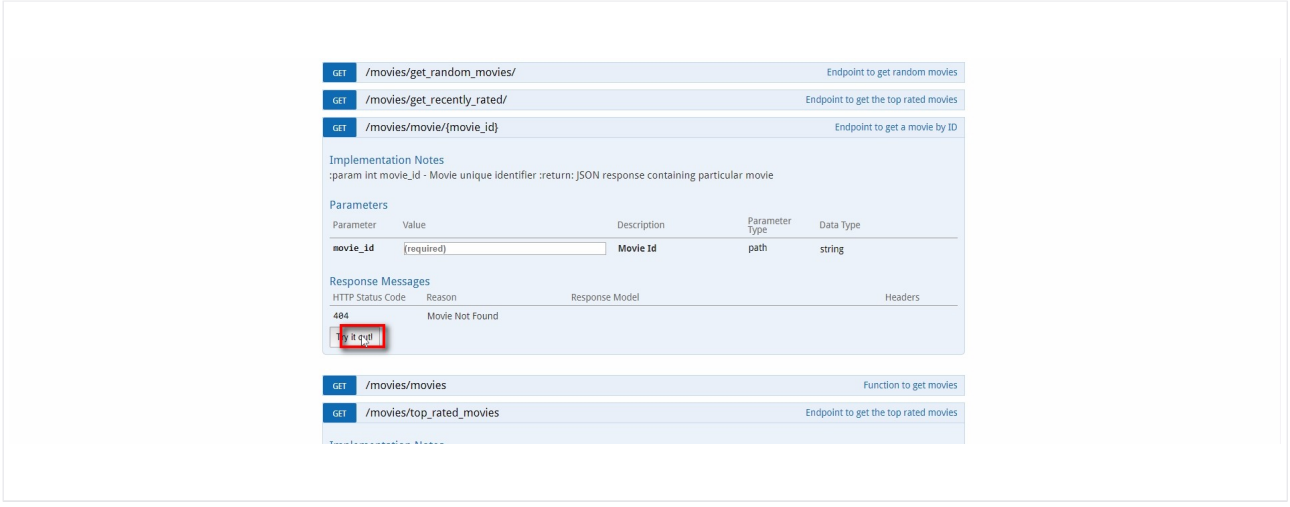


Submit to view response

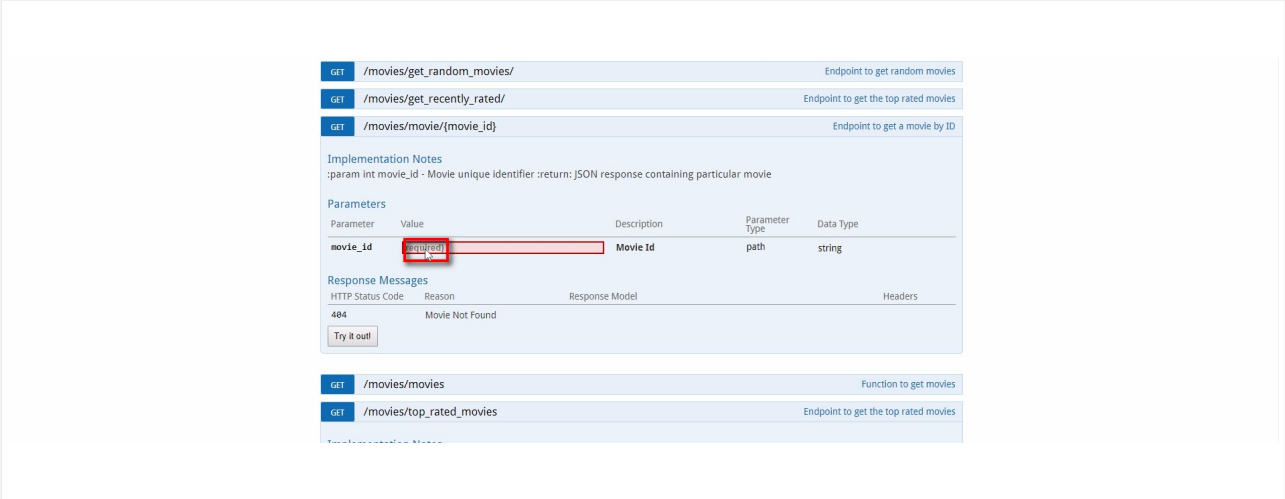


Click 'Try it out!'

Submitting without required parameter

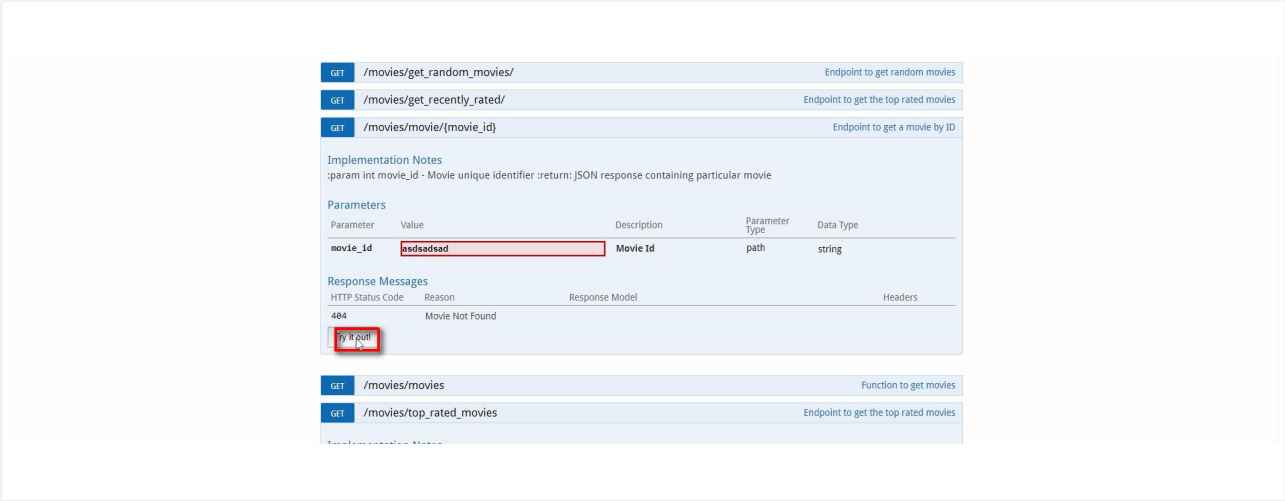


View error



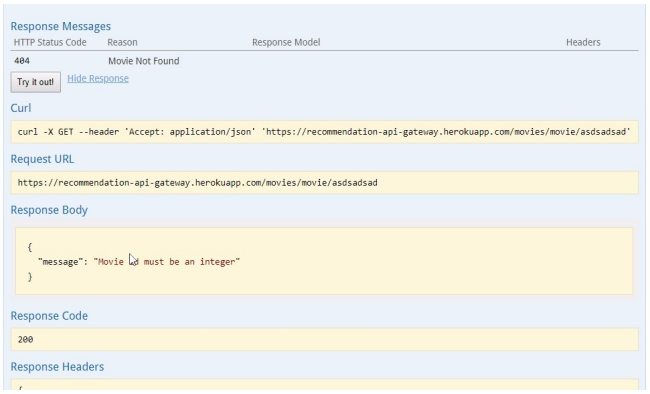
Box turns red to signify required

Try it out with incorrect values



Type in string for movie id

View appropriate error message



View HTTP error message

Website Interaction - Account Creation

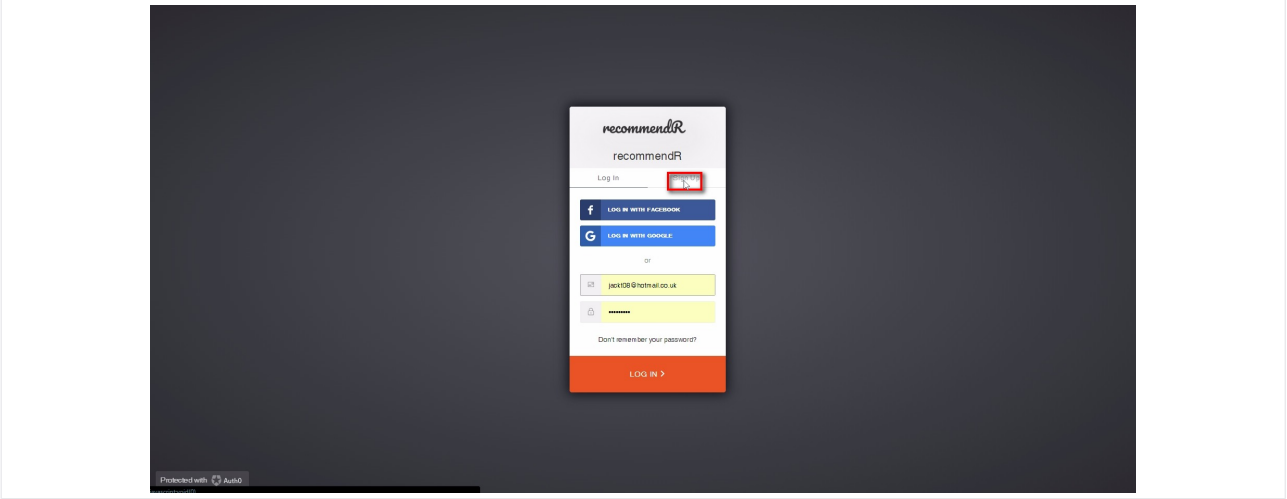
Accessing the website

The production environment is accessible at:

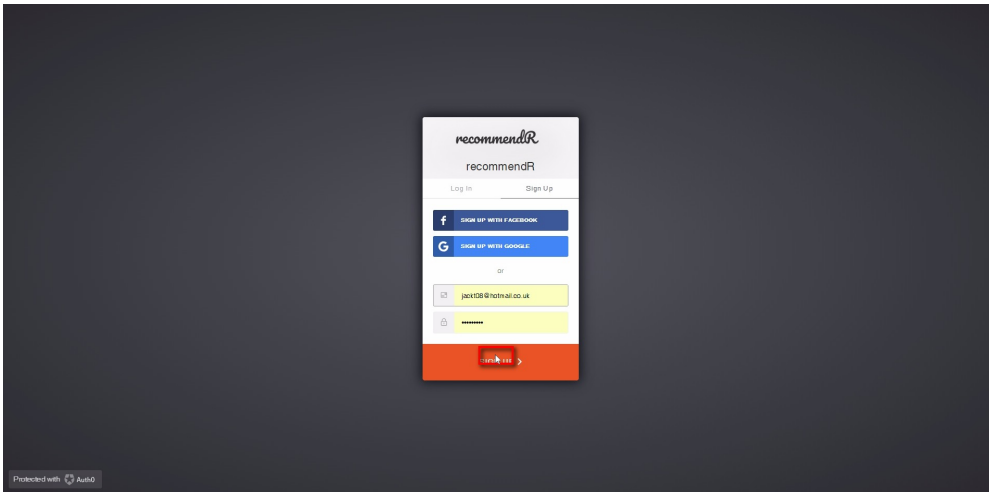
https://recommendation-web-app.herokuapp.com

Or locally at http:localhost:5000/

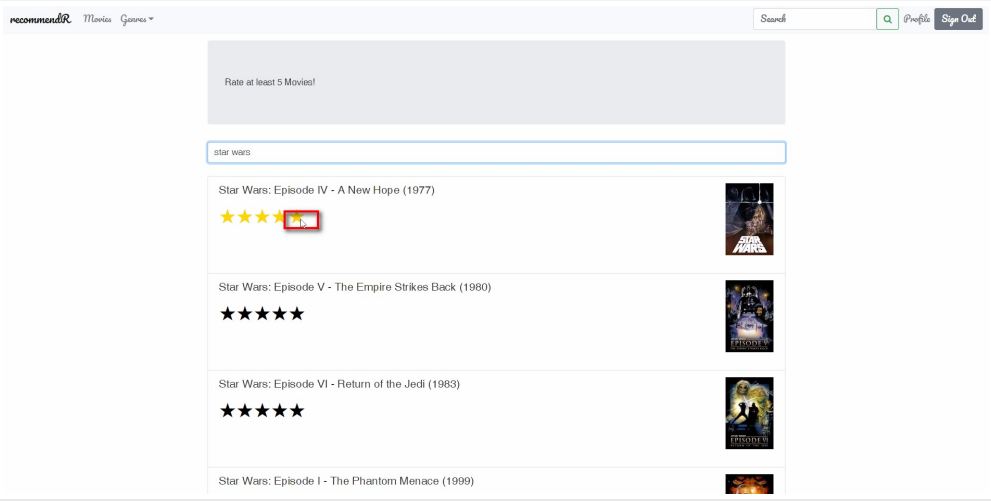
Signup and create an account



Click to create account



Providing initial ratings

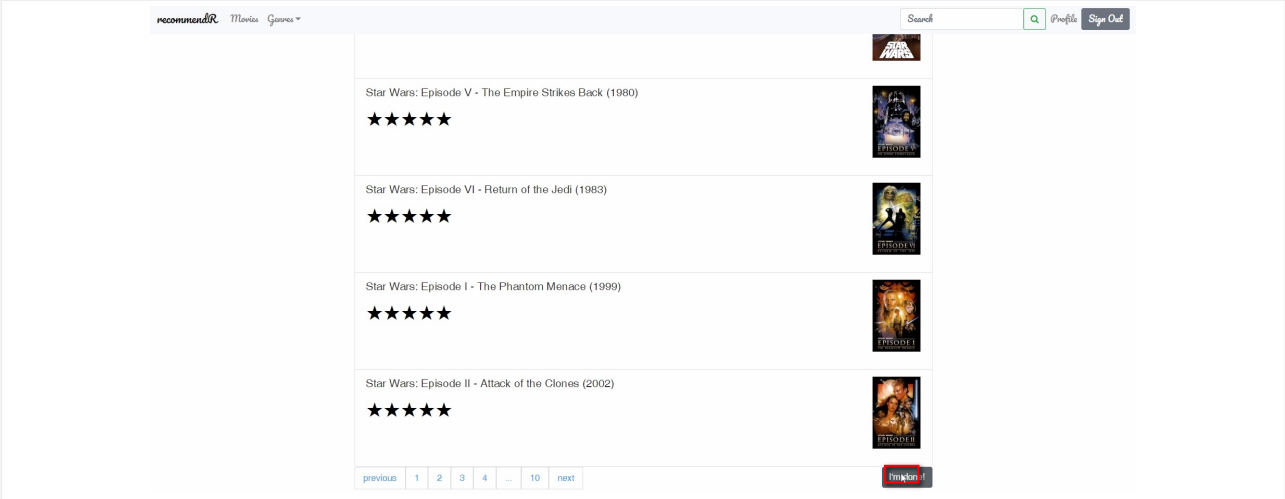


Users are expected to rate 5 movies to be able to view any other pages in the website.

In this example the user is searching for Star Wars.

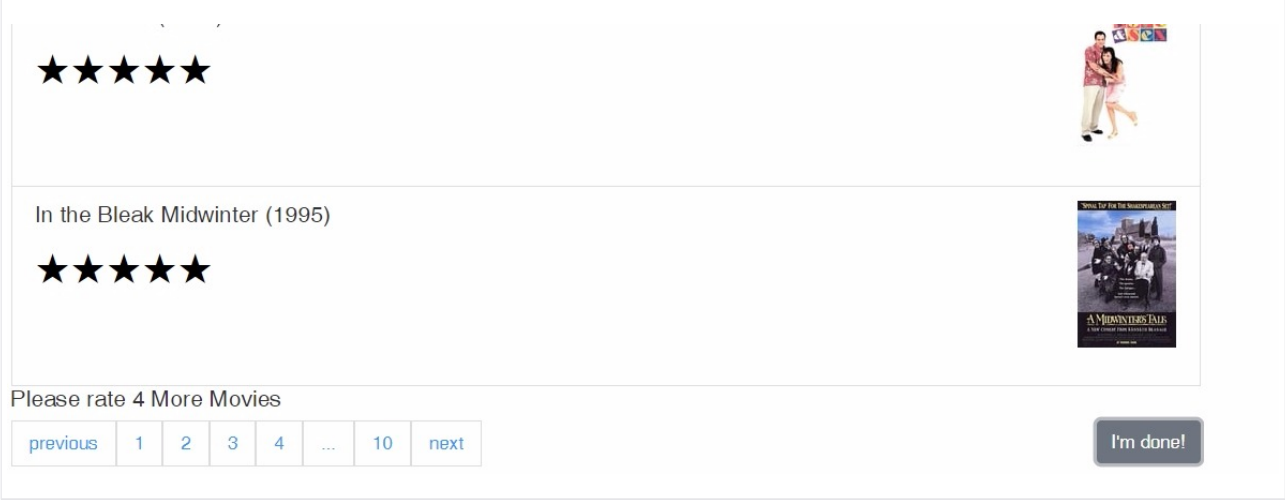
To provide a rating, click on the star rating component.

Submit rating



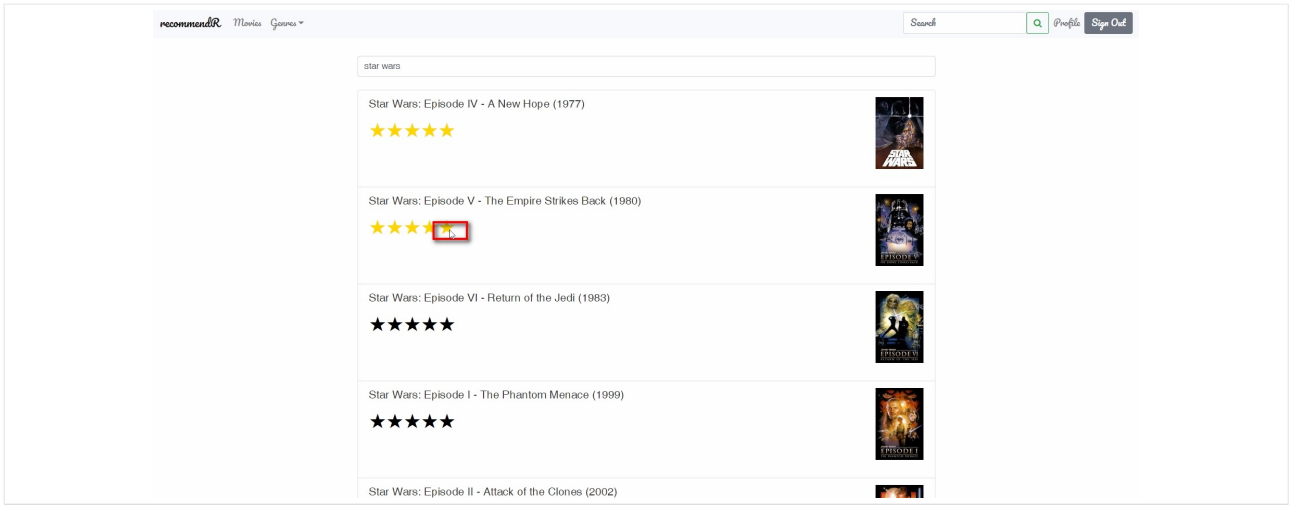
To submit a rating, press the 'I'm done' button

Inidication to rate more movies



If the user hasn't rated 5 movies then they will not continue to the next page

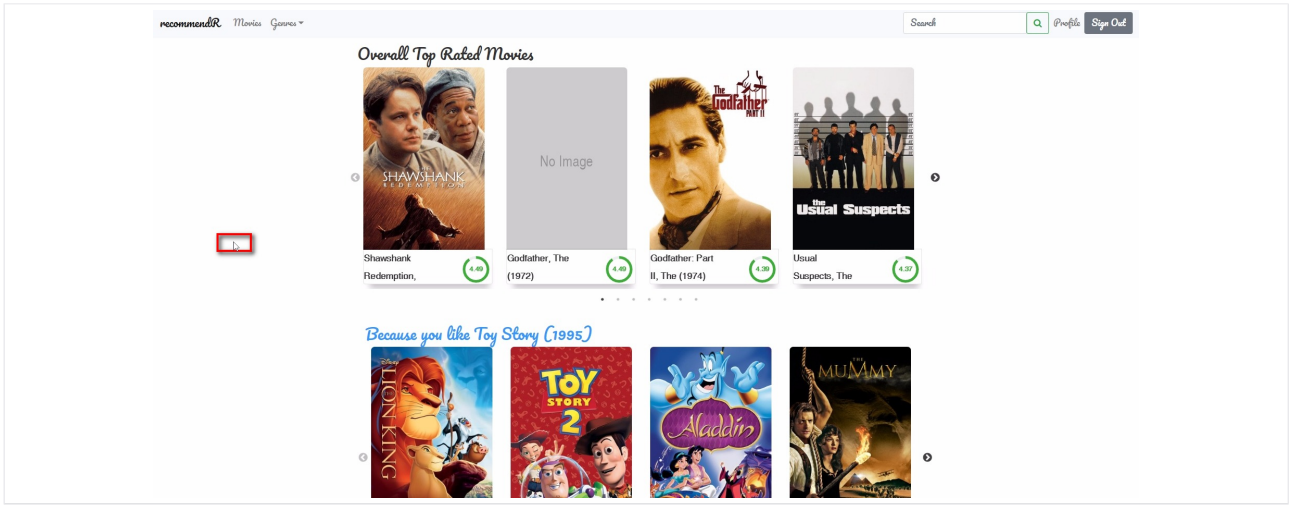
Rate more movies



User rating more movies

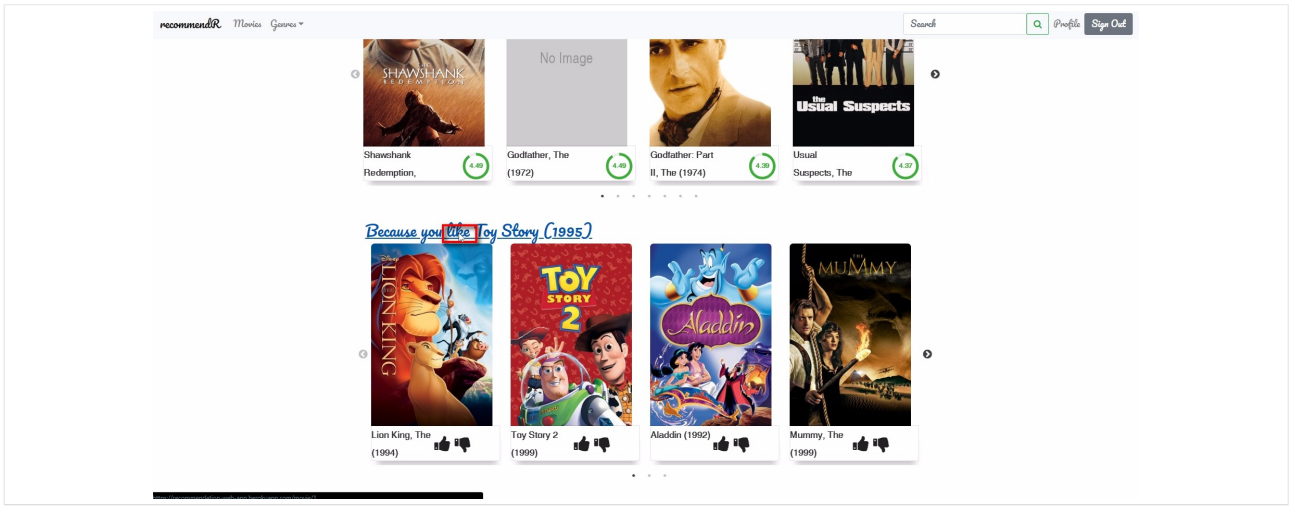
Website Interaction - Logging in

View top rated movies and recommendations

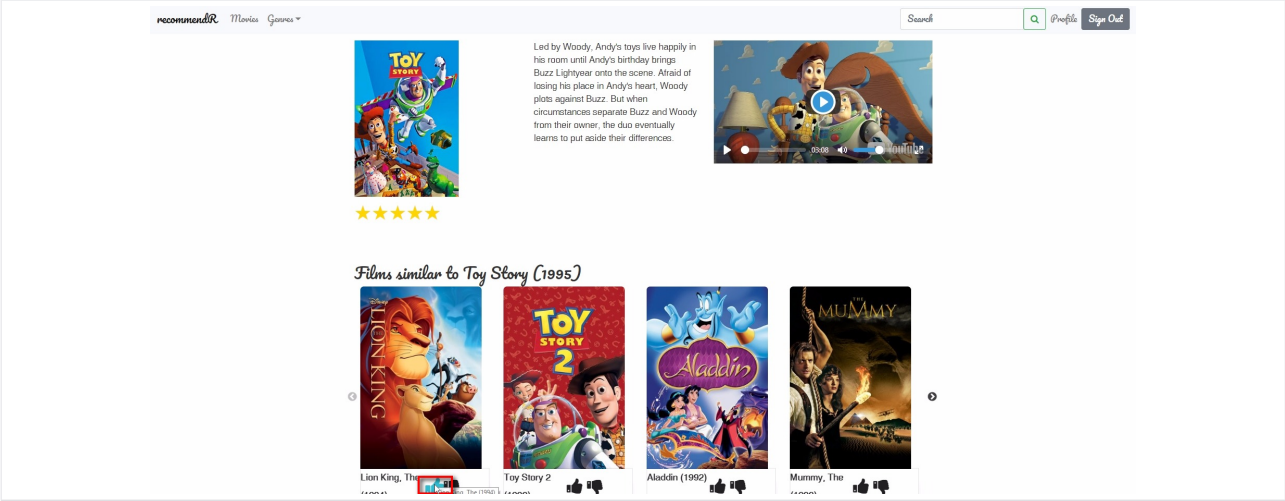


Once logged in the user can view top rated movies and their recommendations

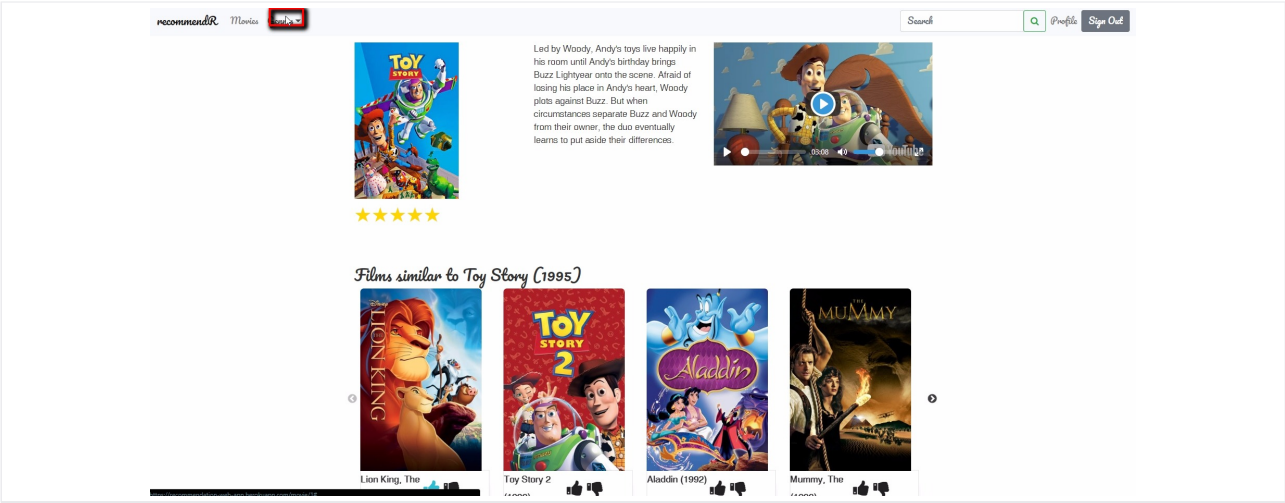
To view more details click link or carousel



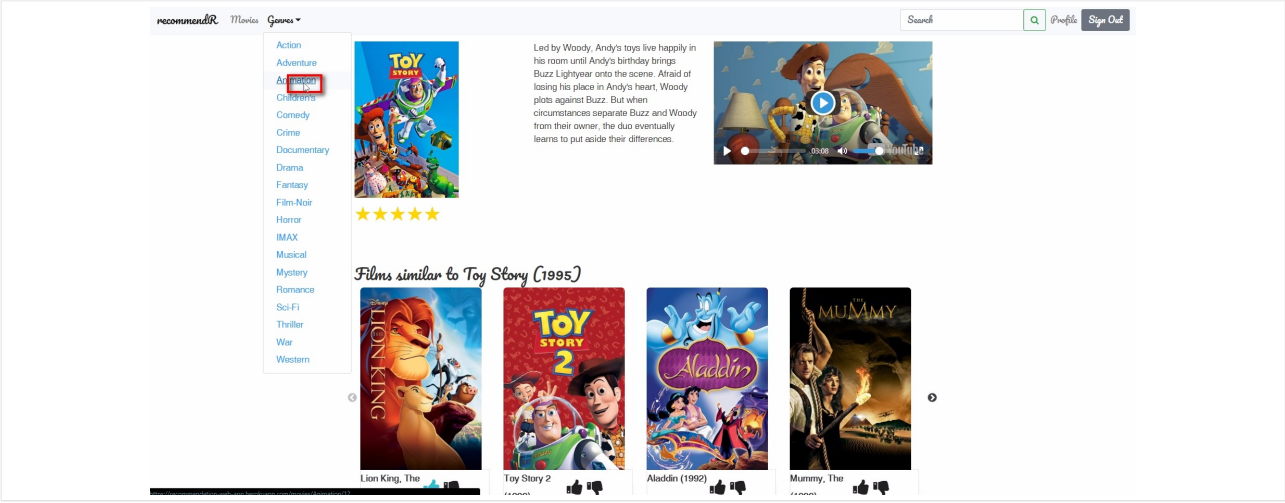
View recommendations for specific movie



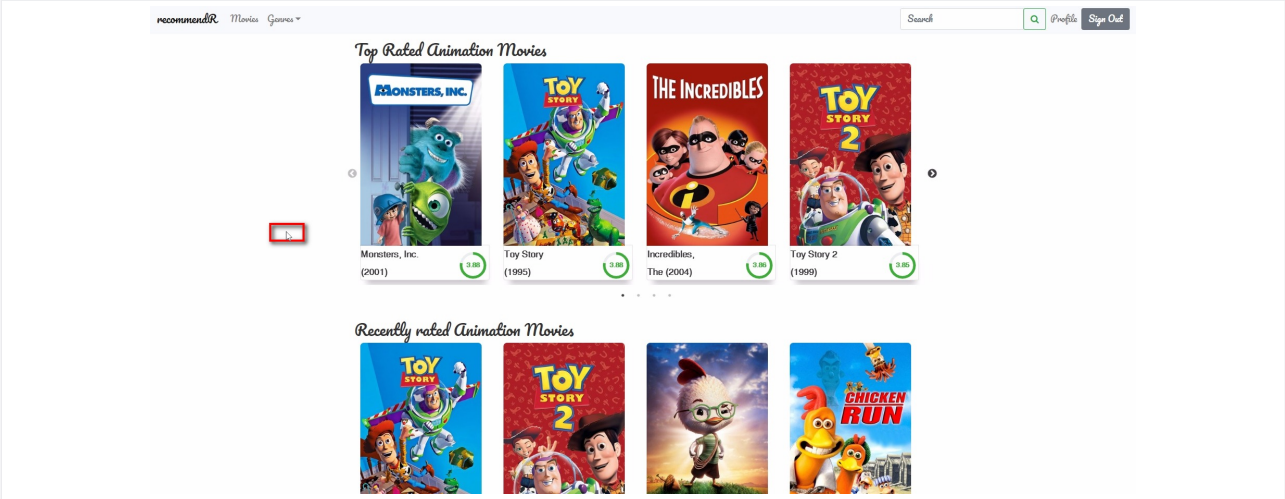
View by genre



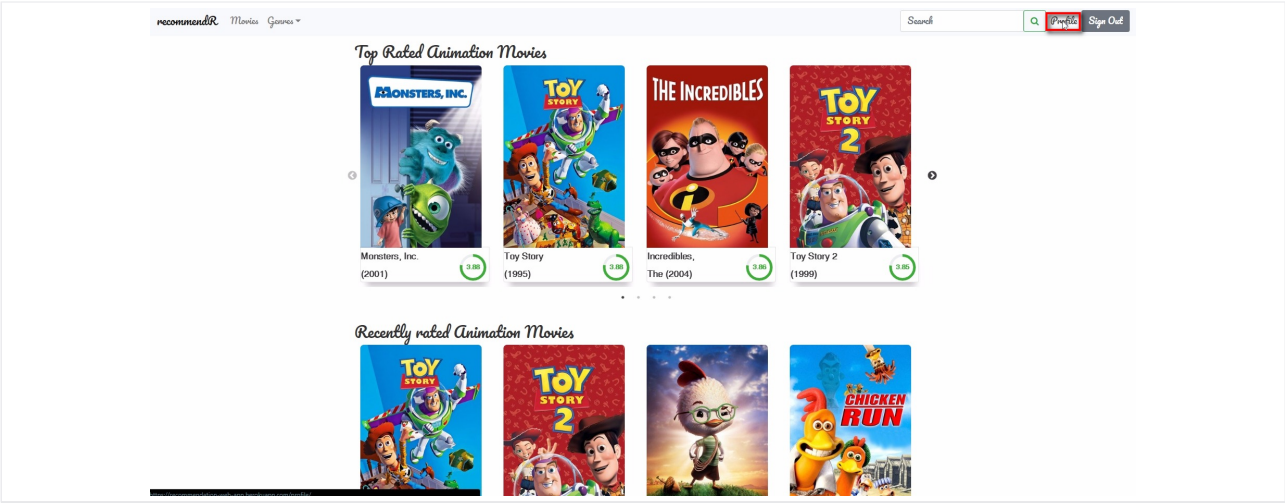
Select specific genre



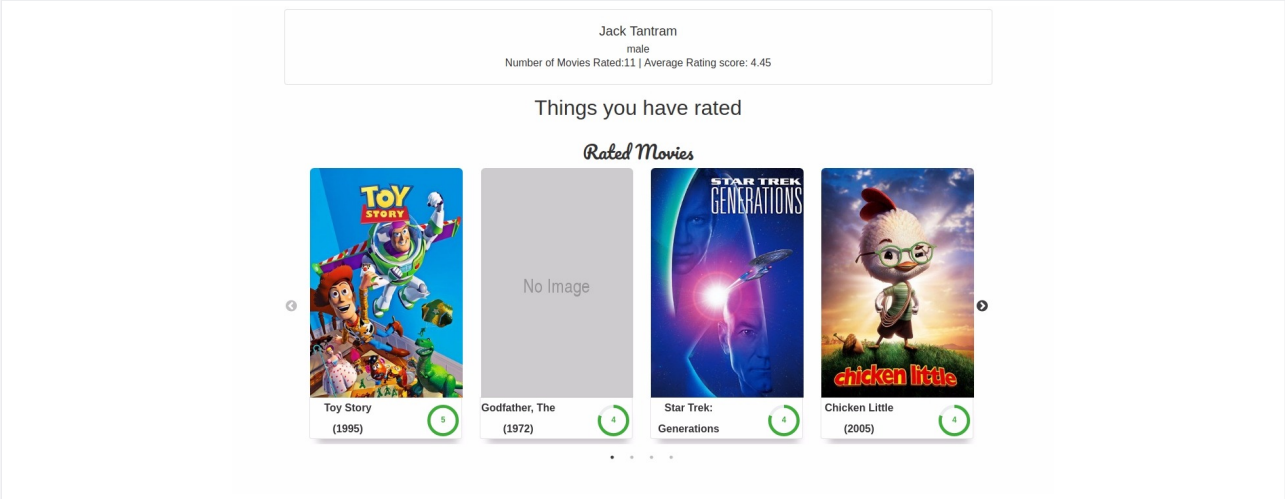
View results



View user profile

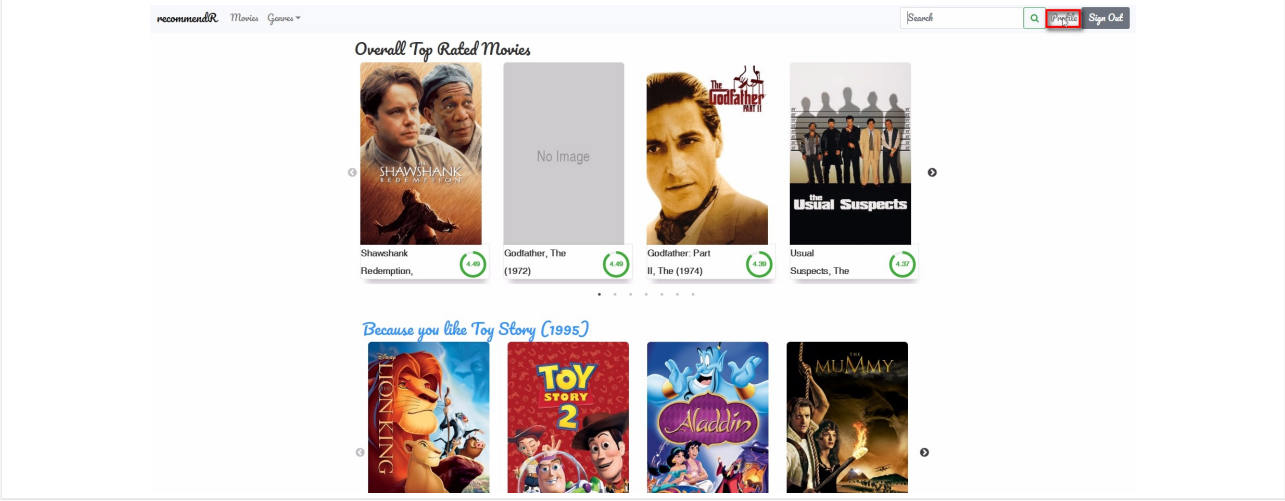


View user profile

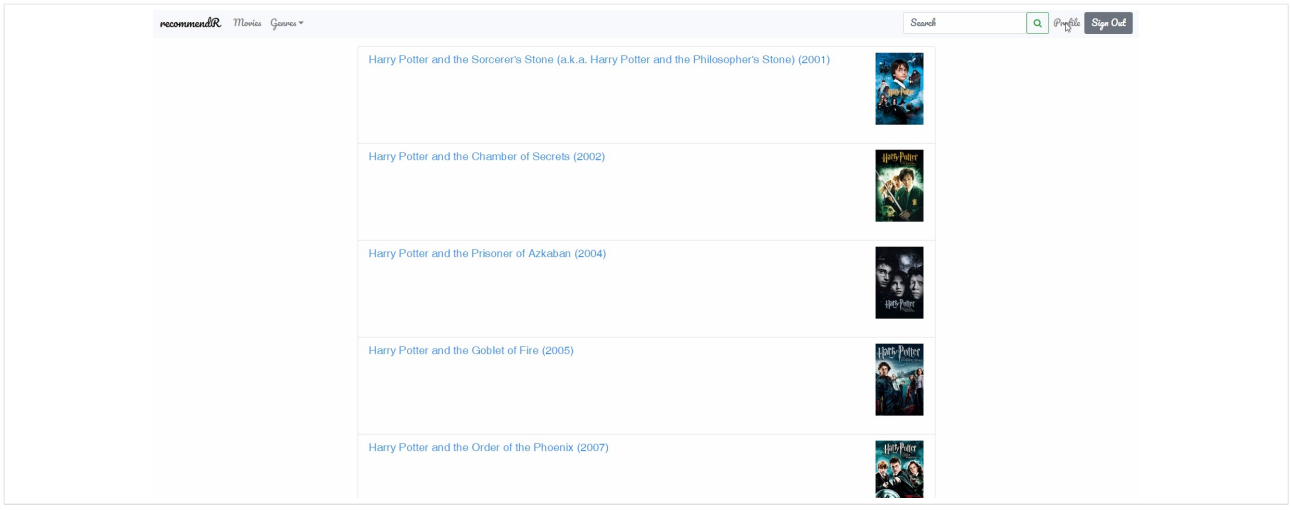


The user profile signifies movies in date order and shows the users average and number of movies rated.

Search for a movie



View results



B Project Management

B.1 Project Initiation Document

Project Initiation Document (PID)

Jack Tantram

April 24, 2018

Contents

1 Introduction

The main aim for the project is to provide methods and tools for building a recommender system. The final product will be a responsive web application that can propose suitable content to users based on their personal interests. The main goal is for the user to be able to create a profile that best fits their persona, and then offer recommendations for specific content.

The system should be flexible enough so that in the future additions to the architecture do not require a full redesign. For example, if the system wants to start making recommendations for food recipes there should be a well documented route in how to do so.

2 Background/Motivation

There are already a range of systems that can provide recommendations. These systems employ a range of models and tools, such as collaborative, or content-based filtering; at a high level, they provide recommendations by comparing items which are similar.

This project aims to create a system that provides recommendations for a variety of domains. It will evaluate the methods and tools for providing personalised recommendations, and take into account how to make the system ready in a production environment. Overall, it will begin by demonstrating insights on a small dataset, which will then follow by creating a system architecture, that is able to expand to other domains. This system should not be tightly coupled; i.e. alterations in the future, should not severely affect other parts of the system.

3 Project objectives

1. Provide methods and tools for building multi-level recommender systems
2. To provide recommendations for specific content towards a user
3. To create a system with an architecture that is expandable in the future; such as a microservice architecture
4. Create a responsive web application that users can interact with
5. To follow best practices and provide well documented code
6. Implement a testing framework to ensure stability
7. To create a system that can be deployed, and is well documented

4 Initial scope

This section will provide more details about the project objectives:

1. An initial training set will need to be found for system
2. Implementation of recommendation models will be identified through research; models could in theory be used in conjunction with each other (collaborative, content-based)
3. User requirements will be gathered using interview and development of user stories

4. Interface design will be determined by user testing
5. System architecture will be determined using UML activity diagrams
6. The proposed system will allow:
   (a) Users to be able to create an account that holds information about their interests
   (b) Users will be able to view content based on recommendations and be able to search for content
   (c) Functionality to expand the system content to offer recommendation in other domains
   (d) The ability to upload a new training set
7. The system should be fully deployable.

5 Resources and dependencies

The project is critically dependent on the following:

- A method of hosting the architecture.
- Finding a suitable initial training set for recommendations
- API's to keep information up to date, or method for adding content so that system is up to date

6 Method of approach

Software development will employ an iterative approach, where the application will be broken up into smaller chunks. With this approach, code is designed, developed, and tested until there is a fully functional application. Essentially, it will follow an agile approach and work will be completed in sprints; scope can change, but it must be agreed upon and overall time must be estimated.

Possible technologies are:

- Python - Recommendation Engine
- Flask (Python)/node.js - Web Server
- Angular.js/React.js - Client side web
- MongoDB/Neo4j - Database

However, a full evaluation will take place during the project to see which will be the most suitable technologies.

7 Initial project plan

Project Plan			
Stage	Expected Start Date	Expected Completion Date	Products/Deliverables/Outcomes
1. Initiation		1 Feb	PID
2. Investigation of initial requirements	30 Jan	6 Feb	Report of initial requirements; main requirements outlined; Evaluation of possible development technologies
3. Initial high level design	5 Feb	13 Feb	Design documents (Architecture; DB schema; UML diagrams)
4. Sprint 1	14 Feb	28 Mar	Develop backend functionality for recommendations; add database functionality
5. Sprint 2	1 Mar	15 Mar	Create a frontend to link to backend functionality;Begin investigation into hosting
6. System and user acceptance testing	12 Mar	16 Mar	User testing, and feedback. System tests creation
7. Sprint 3	16 Mar	30 Mar	Review user feedback and make changes;Further development and infrastructure testing
8. Sprint 4	30 Mar	13 April	Further system development
9. Final system and user acceptance testing	13 Apr	25 Apr	Test Results; final system; user training and review
10. Assemble and complete final report	26 Apr	12 May	PRC304 Report

7.1 Sprint Management

The details of the project plan should not be seen as set in stone. At the end of each sprint the intended objectives can be reviewed and a decision can be made accordingly on whether to change them.

Any requests will be documented accordingly, on a tool such as Trello. The time taken to implement the new request will be calculated accordingly; it will then be reviewed to see if there is enough scope to continue with the request.

This will then be added as an appendix section in the final report, and will contain each sprint objective and any changes that had to be made. This will then be discussed in the main body of the report to provide an overview of the overall process.

7.2 Control plan

The following control plan will be employed:

1. Weekly highlight reports as dictated by the PRCO304 module

2. End-stage reports
3. Weekly review meetings with project supervisor

7.3 Communication plan

Review meetings will be held with the supervisor in line with the Control plan. Further ad-hoc communications will take place as needed.

8 Initial risk list

Initial Risk List	
Risk	Management Strategy
Hosting issue	If there is an inability to host on an external server, action will need to be taken for local hosting and demonstrating that it would be able to deployable if a external server was found.
Technology failure	The system will be deployed using standard technologies and backups will be taken weekly to account for loss.
Scope creep	In the case for scope creep contingency will be provided in the project plan. Highlight reports and regular meetings will be made to address issues.
Gold plating	Additional features that are out of scope can be useful, however work should be throughout estimated before starting so that core functionalities are still implemented.
Corruption of work	Any substantial amount of work should be backed up every day using source control.
Planning and control	Everything will be planned accordingly to schedule and tasks will be broken up on a tool such as Trello.
Illness	If illness is to occur, then the task estimates will need to be adjusted accordingly.
Change of Scope	Scope will be agreed during project planning, any alterations will have to reviewed and time implications calculated.

9 Quality plan

Initial Quality plan	
Quality Check	Strategy
Requirements	Requirements will be checked to ensure that they are correct and relevant. Prototyping and user interviews will be employed throughout to ensure requirements are met.
Design validation	Designs will be drawn out and shown to potential users. They will then rate potential designs and give overall usability feedback.
Testing	Unit testing will be employed to ensure quality of software

10 Legal, social, ethical and/or professional issues

There should not be any present issues and conforms to the University's Ethics policy. For user testing, it will be conducted by University students in a group size smaller than 20, and on University grounds.

B.2 Highlight Reports

PRC304 Highlight Report

Name: Jack Tantram

Date: Feb 02, 2018

Review of work undertaken:

- Trello board created to document all tasks that need to be completed. These tasks are then broken down with subtasks and have a deadline
- Github repositories have been created
- An investigation in a database has been looked into. Neo4j seems promising
- System architecture has been drawn up on a whiteboard (picture on trello) and will be transferred to computer form
- Small prototype made in Jupyter notebook to ingest movie data and provide recommendations using neo4j
- Datasets have been found, such as Amazons dataset [39] or MovieLens [41]. or Book-Crossing Dataset [40]

A decision needs to be made to decide whether to purely use Amazon as the initial training set, or to use separate training sets for different use cases. The Jupyter implementation uses movielens.

Plan of work for the next week:

- Review comments made by supervisor on PID
- Create ERD.
- Complete formalisation of architecture and database
- Continue with requirement documents such as sequence, UML diagrams

If the UML diagram is completely formalised, begin backend implementation of recommender system

Date(s) of supervisory meeting(s) since last highlight:
This is the first highlight report. The first meeting was the 25th January

Brief notes from supervisory meeting(s) since last highlight:
This is the first highlight so this refers to previous communications.
Comments on the PID have been made and the project plan will need some adjustments.

Stage Review:
Everything is going to plan and the datasets look promising. Additionally, everything has been setup project management wise so should be organised

PRC304 Highlight Report

Name: Jack Tantram

Date: Feb 15, 2018

Review of work undertaken:

- Architecture formalised as microservices and has been uploaded to trello.
- A basic UML for the recommendation engine has been drawn and uploaded to trello
- Setup hosting for all services on heroku, with redis and neo4j working.
- Created a docker compose script for easy deployment on local machines.
- Basic implementation for api gateway, recommendation engine made with swagger documentation enabled on the api gateway.

The decision to use multiple training sets for different use cases has been made as the Amazon dataset uses ASINs; which are a unique identifier for Amazon. This would mean retrieving more information such as images would be difficult as the API to retrieve this information requires an Amazon associate account.
A script has been written for the movielens dataset that calls the OMDB API (unofficial) to retrieve the images from IMDB and store the urls into the database.

API Gateway URL: `https://recommendation-api-gateway.herokuapp.com`

Plan of work for the next week:

- Finalise requirements for website.
- Update PID
- Continue with backend functionality for services and API gateway
- Look into method for adding book dataset into system
- Review types of recommendations that could be made

Date(s) of supervisory meeting(s) since last highlight:
There hasn't been a meeting since the last highlight. However, there has been communication through email.

Brief notes from supervisory meeting(s) since last highlight:
Supervisor provided advice about hosting and mentioned Graphene to use for neo4j. Also communicated about potential hosting with head of year. There was also advice given over two potential architectures for the system.

Stage Review:
Been a bit of a learning curve as having to learn how to set up the microservices properly

PRC304 Highlight Report

Name: Jack Tantram

Date: Feb 22, 2018

Review of work undertaken:
I have begun reviewing recommendation algorithms. I have decided to focus on going down a collaborative filtering approach.

There are two kinds of collaborative filtering systems, model based, and memory based; both have their advantages and disadvantages which I have noted in detail in a document.

After some research I found a paper which discussed combining the two. I have decided that it would be interesting to see if I could try to implement this to improve the overall recommendations. I have managed to create a Jupyter notebook which implements Jaccard distance, and Pearson correlation coefficient to get a similarity metric between users (memory based); I am currently trying to figure out a way to test the performance of the algorithms.

Plan of work for the next week: I've been looking at splitting up my dataset to create a training set that the algorithm will learn from and then create a test set that learns from this model. Then see if I can apply an evaluation metric (such as RMSE) to see how close the predictions are.

Hopefully by next week I'll be able to get some of this implemented and display some graphs in Jupyter to analyse which is the better algorithm to use; such as Jaccard, or Pearson as a similarity metric. Once I've decided on what to use I will migrate the code that I have written in Python to Cypher(Neo4j) for better performance.

Date(s) of supervisory meeting(s) since last highlight:
16th February

Brief notes from supervisory meeting(s) since last highlight: Marco gave me advice that I should start focusing on the recommendation algorithm and focus on one domain, such as movies. If there is enough time then I can start looking into other domains. It was also advised to start adding content into the miscellaneous section of SPMS which I will start doing.

Stage Review:
N/A

PRC304 Highlight Report

Name: Jack Tantram

Date: March 1, 2018

Review of work undertaken:

After reviewing some recommendation algorithms I found that collaborative matrix factorisation seemed promising. They attempt to characterise items and users by a set of latent factors, which is inferred from known rating patterns. It is a unsupervised learning technique which is quite similar to content based filtering, whilst that has known labels. I have managed to implement a jupyter notebook which runs matrix factorisation with stochastic gradient descent for optimisation on the movielens dataset. I have also set-up a method for adding new users as typically matrix factorisation are static.

I have also implemented K-nearest neighbours which uses a similarity metric and returns K-nearest for the user to see. My main idea is to use this with matrix factorisation so that once the matrix is produced, neo4j runs the similarity metrics to show relevant data.

A further addition to the system could be implementing a content based approach which analyses things such as genres to improve efficiency, and to make it a hybrid system. However, this will be put on hold until everything else has been implemented.

I also have found a method that analyses the overall precision and recall, which I think could be useful at analysing the systems performance.

Plan of work for the next week:

- Begin integrating algorithm into main repository.
- Integrate this with neo4j.
- Draw up UI designs for website front-end
- Begin integration with web front-end
- Investigate precision and recall at K

Date(s) of supervisory meeting(s) since last highlight:

16th February

Brief notes from supervisory meeting(s) since last highlight: Following the advice Marco has given me on focusing on the algorithm. Main point of contact has been through email and will be discussing the algorithms that I have found.

Stage Review:

Everything seems to be going according to plan, it is clear that the website ui needs to be implemented as it is not functional yet.

PRC304 Highlight Report

Name: Jack Tantram

Date: March 8, 2018

Review of work undertaken:
Implemented matrix factorisation and split data set into training,testing and validation on the movielens dataset. I have optimised the hyper parameters and it seems to be making worthwhile results; for example recommendations for Star Wars is giving me other Star Wars films plus other related films. I have also attempted to add Boolean implicit feedback to help with the score, this hasn't been fully tested and optimised but it doesn't seem to be affecting the results badly. This could be useful in helping a cold start problem.

The RMSE is calculated by comparing the values that the user rated against predicted values. Fig 1 shows how the algorithm improves once optimised.

The current optimal values are: $\alpha = 0.01, \gamma = 0.0001, k = 40, Steps = 100$

I have created a graph property model and UI designs and will upload to miscellaneous once I show it to Marco in our supervisory meeting.
I am currently in the process of integrating my latent factorisation model of ratings into neo4j so that it can provide recommendations from there.
I have decided that initially for website I will create a main page which recommends films based on an item-item level. So that you don't necessarily need an account from the beginning, and once I get this to work to a certain degree start implementation with the creation of users so that it is more personalised.

Plan of work for the next week:

- Continue Integration with backend system
- Investigate additonal implicit feedback addition
- Integrate With Neo4j
- Begin writing endpoints that can be exposed by Api Gateway to get results
- Begin UI implementation if approved by supervisor

Date(s) of supervisory meeting(s) since last highlight:
2nd March

Brief notes from supervisory meeting(s) since last highlight: I spoke to Marco about my implementation of the algorithm and it was decided to focus on it further and try to optimise it. I was also told to create a graph property model which I have created and will be showing to Marco in the next meeting. There was emphasis placed on finishing the algorithm and then performing an evaluation later, where metrics for k were discussed and fallout ratios.

Stage Review:
Its been difficult to understand the algorithm through all the papers, struggling to see how to evaluate the metrics. However it is working at a basic level and integrating with the backend which is good.

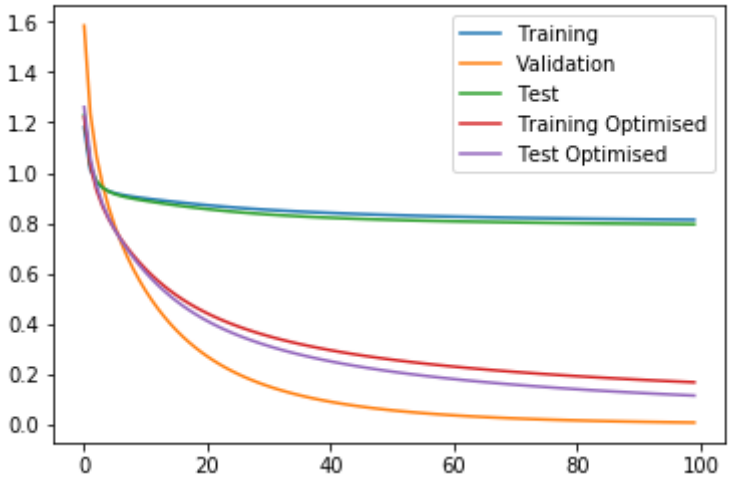


Figure 1: RMSE Over Time with Training,Testing,Validation Sets

PRC304 Highlight Report

Name: Jack Tantram

Date: March 15, 2018

Review of work undertaken:
Following Marcos comments I created user stories using MoSCoW prioritisation methods. I have created a new graph property model which I believe is almost there I just need to go over it to make sure.

- Migrated to later movielens 100k dataset for movies up to 2016, compared to only 1996 before.
- Created webpack configuration for dev and production which is deployed on heroku.
- Setup web frontend using react and bootstrap 4, with components and client side routing.
- Created basic frontend with carousels and movie detail page on click
- Integrated database with matrix factorisation
- Created script to get movie overview, video, and image links for movies from tmdb.

An important thing to note for the matrix factorisation integration is that I have decided to use item-item similarity metrics. I compute this from the array that is retrieved during matrix factorisation and then update the relationships according to this in the database.I am only storing the top 20 per movie because I want to reduce the size in the database. Otherwise, for example if I am computing at an item-item level similarities and there are 9000 movies I am then creating 9000*9000 relationships; my new way would be 9000*20 which is more efficient and reduces size.

Initially I wrote a script that would create a predicted rating relationship which was computed during matrix factorisation and was computing that for every user-item; however I decided that it would create too many relationships and the similarities would be the most important thing.

I have considered writing the matrix that is computed by the latent factor model into a csv and sending it to s3 so it can be kept for historical purposes, and so that you can analyse the performance over time. I am going to run the matrix factorisation script via herokus scheduler at 2am to update the similarities in the database; due to it being computationally expensive.

Website url: https://recommendation-web-app.herokuapp.com/

Plan of work for the next week:

- Continue with website integration
- Integrate database with frontend, this should be partially done by the 16th March
- Review possibilities of improving the algorithm
- Consider potential testing methods

Date(s) of supervisory meeting(s) since last highlight:
9th March

Brief notes from supervisory meeting(s) since last highlight: There was a discussion about implementing another algorithm, or comparing it with the current for a certain benchmark. It was recommended to create user stories using MoSCoW, and consider the Neo4j property graph model. It was discussed to start the website, and integrate the algorithm with the database.
Exploring the use of a SQL database and comparing it with the graph db was talked about too.

Stage Review:
React has quite a steep learning curve, and so has webpack which made development slow at first but the place started to pick up

PRC304 Highlight Report

Name: Jack Tantram

Date: 22 Mar, 2018

Review of work undertaken:

- Created integration with database and front-end showing top rated movies and what are similar
- Created basic search functionality, searching by movie title
- Implemented TFIDF using python library which compares similarity between movie overviews
- Implemented functionality for creating users, this allows password resets, constraints, signup and sign in functionality(auth0)

Following comments from last week I have managed to implement my website which shows recommendations at an item level; it shows a carousel for that movie when you go on the individual page.

I have managed to integrate auth0 for logins so that users can be created on my website. I now need to add the functionality for the users to be able to rate existing movies.

I have started to compare TFIDF and Matrix factorisation and am currently trying to see how I can evaluate both of them.

Plan of work for the next week:

- Continue with website integration for users
- Dedicate time for documenting code, and tidyup
- Integrate local to production(currently isn't working properly in prod)
- Evaluate TFIDF and MF

Date(s) of supervisory meeting(s) since last highlight:
16th March

Brief notes from supervisory meeting(s) since last highlight: We spoke about comparing another algorithm and I have managed to implement TFIDF with movie overviews and this produces recommendations. It was recommended to stay on the movie domain and continue with implementation and testing for that. There have been discussions on how could compare algorithms

Stage Review:
User integration is easier than expected due to Auth0 providing a lot of the functionality. Wasn't too bad linking database to show top rated movies, the microservice approach adds more complexity

B.3 Stage Objective Tasks

Table 2: Task list

Sprint No	Title	Deliverables	Story Points	Start Date	Estimated Completion	Completion Date
Sprint 1						
1	Setup project management tools	Created trello board	1	30-Jan	30-Jan	30-Jan
1	Create initial respositories	Repositories for code, and documents created	1	30-Jan	30-Jan	30-Jan
1	Investigate suitable database for the project	Database evaluations	5	01-Feb	03-Feb	03-Feb
1	Find appropriate training sets	Training set evaluations	5	01-Feb	09-Feb	09-Feb
1	Investigate basic collaborative filtering algorithms	Evaluation on CF algorithms	3	02-Feb	05-Feb	04-Feb
1	Investigate content-based algorithms	Evaluation on CB algorithms	3	02-Feb	05-Feb	04-Feb
1	Create prototypes of algorithms	Construct basic CF and CB notebooks	8	05-Feb	12-Feb	12-Feb
1	Create basic requirements	Initial requirements uploaded to trello	3	12-Feb	14-Feb	13-Feb
Sprint 2						
2	Draw initial architecture diagram	initial architecture diagram	3	14-Feb	17-Feb	17-Feb
2	Setup hosting services for Heroku and Neo4j	Initialised databases in Heroku	2	14-Feb	15-Feb	14-Feb
2	Create notebook that integrates with neo4j	Basic py2neo notebook	2	15-Feb	17-Feb	17-Feb
2	Create basic scaffold for microservices	Initial microservice setup	3	16-Feb	20-Feb	21-Feb
2	Review further recommendation algorithms	Investigate deeper into MF	5	16-Feb	22-Feb	21-Feb
2	Setup services to use environment variables	Code change	2	16-Feb	17-Feb	17-Feb
2	Create UI designs	Basic UI designs for website	2	22-Feb	23-Feb	23-Feb
2	Investigate applying basic algorithms to book dataset	Book notebook created	3	22-Feb	24-Feb	24-Feb
2	Setup apps with Docker	Docker script for each service	3	25-Feb	28-Feb	28-Feb
2	Follow basic react.js tutorials		2	26-Feb	28-Feb	28-Feb
Sprint 3						
3	Investigate methods for improving MF	MF evaluations	8	01-Mar	05-Mar	05-Mar
3	Implement redis for caching	Redis integration	2	05-Mar	07-Mar	08-Mar
3	Separate out API gateway into different namespaces	code changes	2	08-Mar	09-Mar	09-Mar
3	Add swagger docs to API gateway	Interactive API gateway with swagger docs	3	09-Mar	11-Mar	11-Mar
3	Setup docker compose	docker-compose.yml file	5	12-Mar	15-Mar	16-Mar
3	Webpack research	Investigate how to setup webpack	3	11-Mar	13-Mar	13-Mar
3	Create functional and non functional requirements	MoSCoW and FURPS requirements	3	12-Mar	15-Mar	15-Mar
3	Follow React.js tutorials on Pluralsight	Watch react.js videos	3	13-Mar	15-Mar	15-Mar
3	Run Cross validation and optimise MF parameters	Hyperparameter opt and CV graphs	5	13-Mar	15-Mar	17-Mar
3	Integrate MF with database	Code integration	5	14-Mar	17-Mar	16-Mar
3	Write script to retrieve additional content from ML	TMDB retrieval script	2	14-Mar	16-Mar	16-Mar
Sprint 4						
4	Webpack integration	Setup webpack	3	16-Mar	18-Mar	17-Mar
4	Basic UI implementation	Create basic UI pages	8	18-Mar	25-Mar	25-Mar
4	User account creation	Integrate user accounts	8	23-Mar	28-Mar	26-Mar
4	Create carousels	Create carousel component	5	23-Mar	25-Mar	26-Mar
4	Be able to insert ratings	Users can rate movies	3	24-Mar	27-Mar	27-Mar
4	Show recommendations for users	basic implementation to view data		26-Mar	29-Mar	29-Mar
4	Investigate TF-IDF for movie overviews	TF-IDF Notebook	3	26-Mar	29-Mar	28-Mar
4	Consider testing methods	Look into testing libraries (pytest, unittest, selenium)	2	28-Mar	29-Mar	29-Mar
4	Fix issues with production and local	Bug fixes	2	29-Mar	30-Mar	30-Mar
4	Document code and tidyup	documentation	2	29-Mar	30-Mar	30-Mar
Sprint 5						
5	Create script to get movie scripts	Use IMSDb website to retrieve data	2	30-Mar	01-Apr	31-Mar
5	Create indivudal movie page to show recommendations and info	Individual movie page	3	31-Mar	03-Apr	03-Apr
5	Compare TF-IDF on scripts and overviews	NB comparing performance of TF-IDF and overviews	8	03-Apr	06-Apr	05-Apr
5	MF research creating vector of additional input sources	Further MF research	5	05-Apr	10-Apr	11-Apr
5	Create search element	Search microservice created	3	05-Apr	08-Apr	09-Apr
5	Create profile page	Profile page	3	07-Apr	11-Apr	11-Apr
5	Integrate TF-IDF with DB	TF-IDF with similarity relationships created	3	09-Apr	11-Apr	11-Apr
5	Add TF-IDF and overviews to website	Integration	3	11-Apr	13-Apr	13-Apr
Sprint 6						
6	Create genre page	Create individual genre page	2	14-Apr	16-Apr	16-Apr
6	Create infinite scroll	Make home page infinite scroll	5	14-Apr	16-Apr	16-Apr
6	Add pagination to search	Add pagination to search	5	15-Apr	17-Apr	17-Apr
6	Add search to initial ratings page	Search element on initial ratings page	5	17-Apr	19-Apr	19-Apr
6	Create feedback buttons	Feedback buttons for carousel	5	20-Apr	24-Apr	26-Apr
6	Conduct user testing	User testing setting up typeform and getting participants	5	01-May	05-May	08-May
6	Evaluate user testing	Notebook written to compare results	5	08-May	10-May	10-May
6	Begin report	Final report		05-May	20-May	20-May
6	Create User guide	Guide created using stepshot	2	12-May	13-May	13-May
6	Bug fixes	Fix bug fixes and changes users requested	8	04-May	15-May	18-May
6	Create unit tests	Unit tests for all services	8	05-May	10-May	10-May

B.4 Methodology Evaluations

Table 3: Project management methodology evaluations

Type of methodology	Benefits	Disadvantages
Agile	- Flexible to requirements change - Encourages frequent communication with stakeholders, improves quality. - Reduced risks	- Flexibility can cause dysfunction with too many requirements changing - Not very predictable - Can be challenging at larger scale
Waterfall	- Enforces discipline, every stage has a designed start and end. - Timescales are kept - More control	- Not very flexible - Doesn't fit the customers needs - Longer delivery time - If assumptions are wrong then estimated time can exceed and fail to meet deadline
PRINCE2	- Predictable as it breaks down the project in stages - Standardises everything in the project so that everything is followed to a certain standard	- Presents a large overhead and is hard to apply in small projects - Not a lot of flexibility
Extreme Programming (XP)	- Highlights importance of communication - Reduces the amount of documentation and focuses on delivery - Constant feedback is encouraged - Can adapt to changing requirements	- Doesn't measure or plan Quality assurance - Generally practiced with pair programming which leads to duplication of code - Difficult practice to follow - Assumes constant involvement with customer - Too minimalist

C Initial Research

C.1 Existing Products

YouTube:

YouTubes recommendations are driven by Google Brain, which opensourced by TensorFlow. It uses deep neural network architectures and distributed training.

System:

Consists of two neural networks, one for candidate generation (CG) which it takes as input. CG uses implicit feedback of video watches to train the model. Explicit is the thumbs up or down button, they are rarer. Users watch history and using CF selects videos in the range of hundreds.

Google uses offline metrics for performance, final decision comes by using live A/B testing.

Second neural network is used for ranking the few hundred in order. Logistic regression to score each video and then use A/B testing to evaluate performance. Metric of performance is watch time

What does it recommend?

Videos that are similar to ones that the user have watched and people that the user is subscribed to.

Highlighting points

Their deep CF Model outperforms their previous MF approaches, with layers of depth. approaches used at YouTube

Testing

They use a range of offline methods, such as precision, recall, ranking, and loss. But A/B testing is the primary driver.

Netflix:

Netflix produces recommendations for movies and tv shows. Winner of the

Netflix Prize to produce recommendations SVD

System:

Singular Value Decomposition (SVD) is a form of matrix factorisation, used to predict ratings. Its similar to Principle Component Analysis (PCA). Have thumbs up and down system to help provide feedback. New image algorithm to project image it thinks you'll respond to. Incorporates additional information such as tags and things about a user to make recommendations.

What does it recommend?

Movies and tv shows that are similar taking into account user information and explicit, and implicit information.

Highlighting points

They use a lot of additional information to drive their algorithms, such as popularity as a baseline.

Testing

Uses RMSE to calculate overall error. A/B Testing, and other standard metrics

Amazon

Amazon provide a range of methods to recommend items.

System:

Amazon use a combination of user-user and item-item based CF methods. They originated using Pearson correlation coefficient, but have experiment with other metrics such as cosine similarity. By 2003 item-based was used in Amazon, mainly using featured recommendations based on past purchases. Search would recommend based on search, and also the shopping cart.

What does it recommend?

Items based on previous buys, page clicks, search recommendations and basket recommendations.

Highlighting Points

Theyve stated it to be very scalable, and simplistic which has helped them grow. By preprocessing offline it means that the recommendations are quick and more personalised to the user. Similar items is a sleek list, instead of using the whole dataset. Focuses on the users history instead of others so not sparsity.

Testing

A/B Testing and a variety of other methods

Pandora

Pandora's recommender system is quite different to others and is based on content-based algorithms

System

The system uses characteristics between songs and labelled 450 musical attributes altogether. They have a large music database with all this information labelled. Genes could be gender of vocalist, or the tempo of the chorus. Each genre has a different set of genes; e.g. Rock and Pop 150 genes.

Takes 20–30minutes per 4-minute song by a Pandora employee to construct a music genome. Their algorithm compares the genetic markup of every song in their database and then is placed and ranked in order of similarity using a distance function.

What does it recommend?

Recommends songs that are similar to each other and does it based on the genetic markup of a song.

Highlighting Points

Really in depth system with a ton of information about music. However, this is a very laborious task and requires experts to train it; 'Musicologists'.

Testing

Review process by music experts to assign appropriate characteristics. A/B Testing and other standard metrics

C.2 Dataset Evaluations

Dataset	Benefits	Limitations
Amazon Dataset	• Large variety of user item ratings • A lot of domains such as electronics,movies, and music	• Every item identified by unique ASIN(Amazon ID) • ASIN is difficult to lookup requires amazon associate account • Difficult to retrieve extra content such as images
Movielens	• Variety of different sized datasets • Large user item ratings • Provides extra information for details such as age,genres,tags • IMDB links, useful for retreiving images, and more content	
Book crossing	• Large user-item rating matrix for books • In depth characteristics such as publisher, or user demographics	

C.3 Algorithm comparisons

Type of algorithm	Advantages	Limitations
Content-based	- User independence - does not need to know anything about users - No cold start due to using the characteristics of an item - Can provide high accuracy - If model is constructed well	- Limited content issues - Struggles if limited knowledge - Over specialises , nothing is surprised - Difficult to obtain a relevant dataset - Expert knowledge required in subject domain - Not flexible towards other domains
Collaborative	- Flexibility across domains - Can uses human experiences for recommendations, such as rating biases - Easier to obtain/create dataset	- Suffers from the cold start problem - Can run into scalability issues when the amount of items and users increase - Data sparsity - If there are limited ratings, the system will suffer

D Requirements Analysis

D.1 Functional requirements

Must Have	Should Have	Could Have	Would Have
A user must be able to create an account	A user should be able to search by movie title	A user should be able to view more in depth information about a movie.	A user wont be able to have social integration
A user must be able to reset their password	A user should be able to search by genre	A user could be able to insert new content	
A user must be able to view recommendations	A user should be able to view more in depth information about a movie.		
A user must be able to rating movies	A user should be able to view more in depth information about a movie.		
A user must be able to access the website via a computer or a tablet.			

D.1.1 Non-functional Requirements

Requirement Type	**Explanation**
Availability	The system should be available 24 hours, everyday
Compatibility	Will be compatible on modern browsers such as Chrome, Firefox, and available on tablets, and desktop
Performance	The website must react quickly to queries and return content in a reasonable amount of time (max 5 seconds).
Reliability	The system should function under all stated requirements
Maintainability	The system should be fully documented and be easy to make new changes.
Scalability	The system should be distributed so that the website has the ability to expand in the future
Usability	Most importantly the interface must be easy to understand and make sense to the user
Security	In the case of handling user accounts, the system should encrypt information and not store content like passwords as plaintext
Testability	There must be testing in place, such as unit testing to ensure modules work.

E Technologies

Service	Technologies Used
Website	Back-end: Python 3 (Flask), Frontend: HTML5, React.js, CSS. Tools: Webpack
Api Gateway	Back-end: Python 3 (Flask), Database: Redis
Recommendations Engine	Back-end: Python3 (Flask), Neo4j

E.1 Front-end Evaluations

Technology	Benefits	Limitations
React.js	- Reusable components - Good developer tools - Great native framework for mobile - High readbility and maintainability - Flexibility - Continuous support - Great documentation	- Steep learning curve - Need to learn it - Requires a loader, such as webpack
Angular.js	- Strict - good for projects file structure and reuse - High emphasis on Object Orientated Programming - Reusable components - Modifies DOM directly - faster - Data binding functionality - no need for observable functions - Lot of support - Great documentation	- Only know up to angular 1.5 would require learning later version - Requires knowledge of Typescript also. - Steep learning curve - Later versions require loader, such as webpack - Strict - Have to do things the angular way
JQuery	- Easy to use - Large community - Great documentation	- Functionality can be quite limited - no reusable components - Not very clean - Large codebase - performance issues
JavaScript (core)	- Easy to write - Well documented	- Not very structured - Core JS libraries are limited - Code can be messy - lots of ways to do things

E.2 Back-end Evaluations

Technology	Benefits	Limitations
Python (Flask)	- Well Documented - Very lightweight - useful when having a lot of microservices - Simple, Flexible - A lot of developer support - Python itself has a ton of datascience libraries which would be useful for the algorithm - Good RESTful api features - great for making microservices	- No built-in ORM which could cause issues updating DB later - Requires a lot more work to get extra functionality. - Structure isn't fixed which could mean file structure is messy
Python (Django)	- Contains a lot out of the box - Is well documented - Provides admin functionality and everything built in - Useful ORM that means can change database quite easy - Rigid structure, which can be useful for control	- Typically designed for relational databases - RESTFul API isn't great could cause headache for multiple services
Node.js	- Fast - uses Chromes V8 Javascript engine - Emphasis on asynchronisity - Fullstack is JS - Consistency - Active community	- CPU Intensive computations can cause issues. - Data science libraries are not as good as Pythons - Quality- Lots of ways to do something in JS, can get messy - Lack of consistency, unstable API
Go	- Hardly any boilerplate code - Runs extremely fast optimised - Strong and static typed - Reliable	- No prior knowledge of Go, will take time to learn - Very strict syntax and rules - Small number of packages, would require a lot more work

E.3 Database Evaluations

E.3.1 Database Query Times

Query	Postgres	Neo4j
Similar Users	57ms	35ms
Top Rated Movies >10 number of ratings	261ms	250ms
Top Rated Movies >100 number of ratings	256ms	178ms
Top Rated Movies >500 number of ratings	263ms	212ms

E.3.2 Database Query Comparisons

```
1 select u1.user_id,u1.movie_id,u1.rating,u2.user_id, u2.rating
2 from
3 (select user_id,movie_id,rating
4 from ratings where user_id=1) u1
5 INNER join (
6 select user_id,movie_id,rating
7 from ratings) u2 on u1.movie_id=u2.movie_id WHERE u1.rating=u2.rating;
```

Figure 11: Postgres: Finding similar user

```
MATCH(u:User{user_id:1})-[r:Has_rated]->
(m:Movie)<-[r2:Has_rated]-(u2:User)
where r.rating = r2.rating
return m.movie_id, u.user_id,r.rating, r2.rating,u2.user_id
```

Figure 12: Neo4j: Finding similar user

```
select m1.movie_id,sum(m1.rating)::decimal/count(m2) as score
from (select movie_id,user_id, ratings.rating from ratings) m1
INNER JOIN(
 select user_id,movie_id,rating from ratings) m2
ON m1.movie_id=m2.movie_id
WHERE m1.user_id=m2.user_id
GROUP BY m1.movie_id
having count(m2)> 100 ORDER BY score DESC
```

Figure 13: Postgres: Top rated movies

```
MATCH()-[r:Has_rated]->(m:Movie)
WITH m, sum(r.rating) as sum_of_ratings, count(r) as num_ratings
WHERE num_ratings> 500
return m.movie_id, tofloat(sum_of_ratings)/num_ratings as rating
ORDER BY rating DESC
```

Figure 14: Neo4j: Top rated movies

```
select m1.movie_id,sum(m1.rating)::decimal/count(m2) as score
 from (select movie_id,user_id, ratings.rating from ratings)
 m1
 INNER JOIN(
select user_id,movie_id,rating from ratings  )
 m2 ON m1.movie_id=m2.movie_id
INNER JOIN movie_genres g2 ON g2.movie_id=m2.movie_id
INNER JOIN genres g3 ON g3.genre_id=g2.genre_id
WHERE m1.user_id=m2.user_id AND g3.name='Sci-Fi'
GROUP BY m1.movie_id  having count(m2)> 10 ORDER BY score DESC
```

Figure 15: Postgres: Top rated movies with 'Sci-Fi' genre

```
MATCH()-[r:Has_rated]->(m:Movie)-[:Is_genre]->(g:Genre)
WITH m, sum(r.rating) as sum_of_ratings, count(r) as num_ratings
WHERE num_ratings> 500 and g.name = 'Sci-Fi'
return m.movie_id, tofloat(sum_of_ratings)/num_ratings as rating
ORDER BY rating DESC
```

Figure 16: Neo4j: Top rated movies with 'Sci-Fi' genre

F User-Interface Design

F.1 Initial wireframes

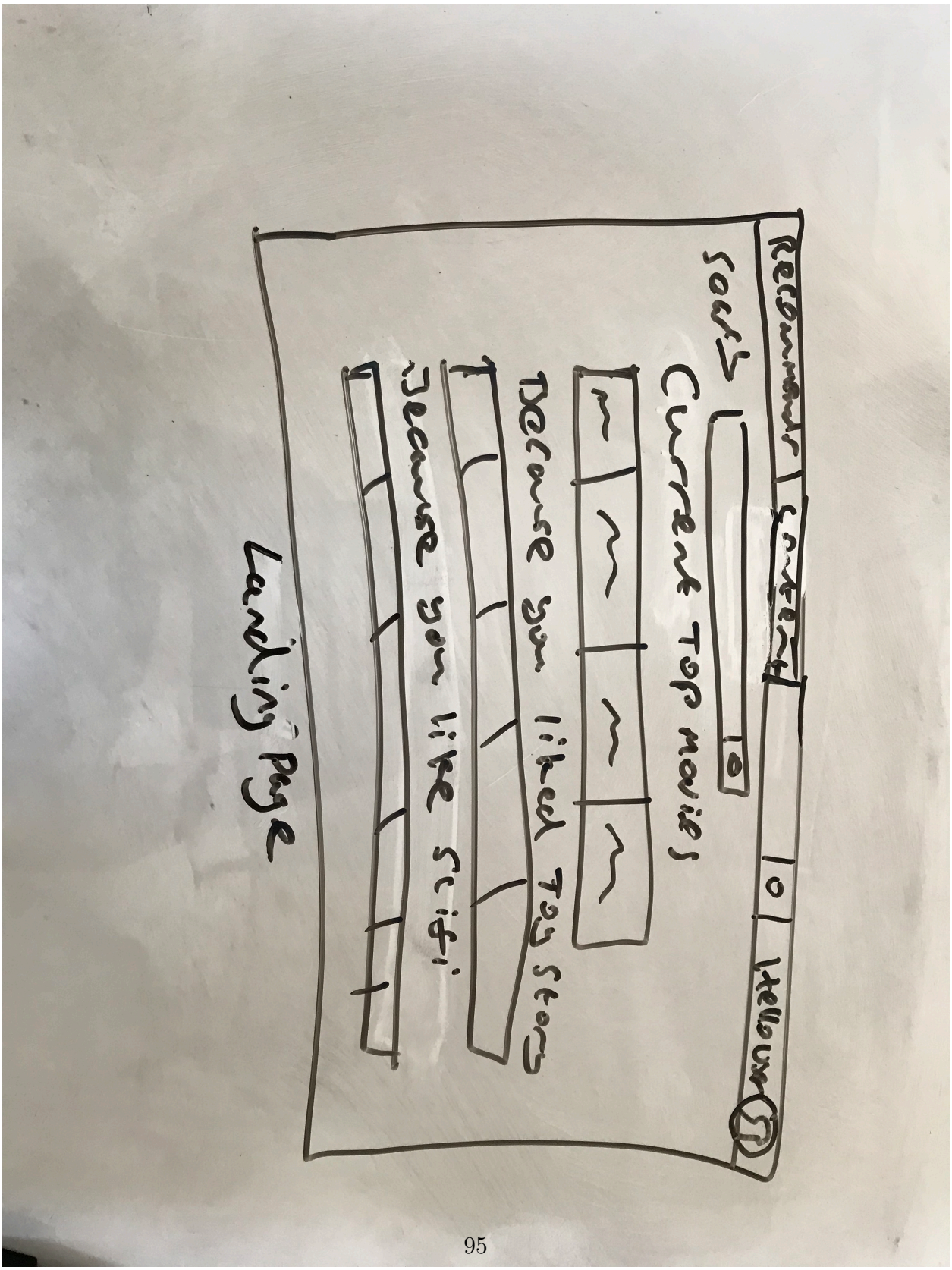


Figure 17: Home Page

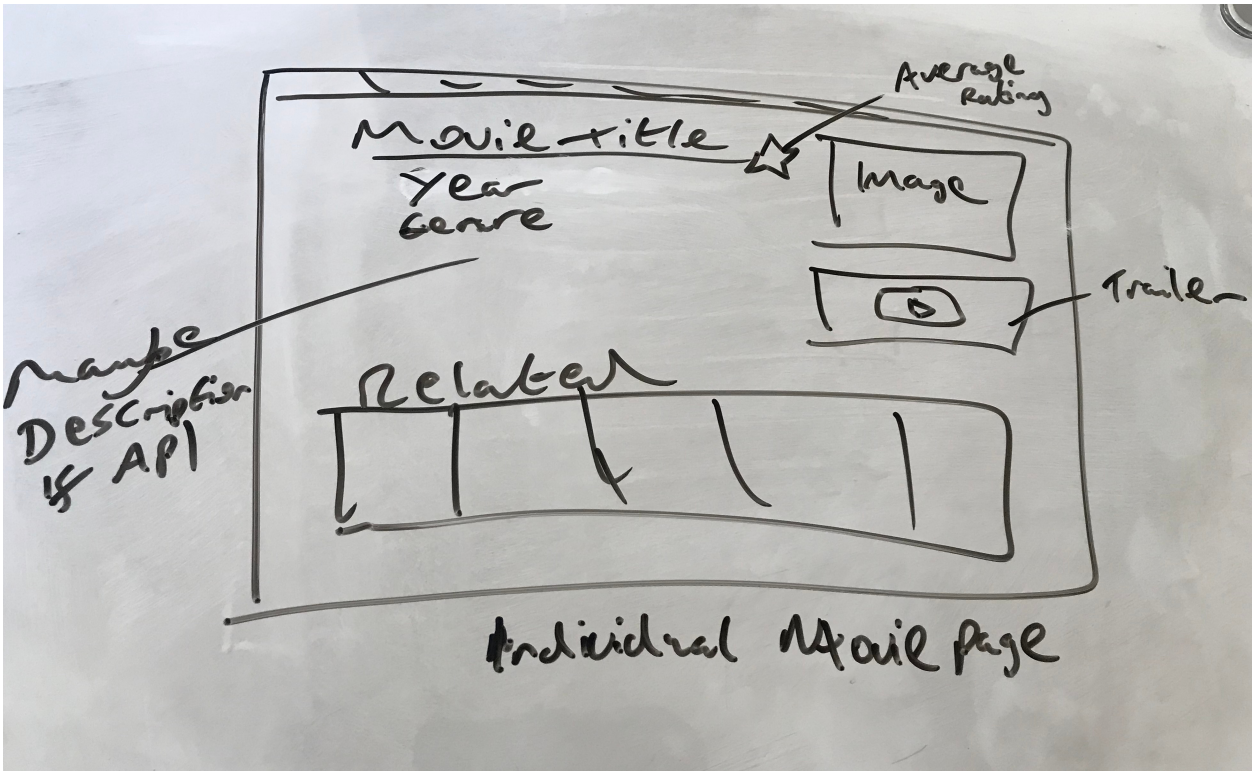


Figure 18: Individual Page

Figure 19: Initial Ratings Page

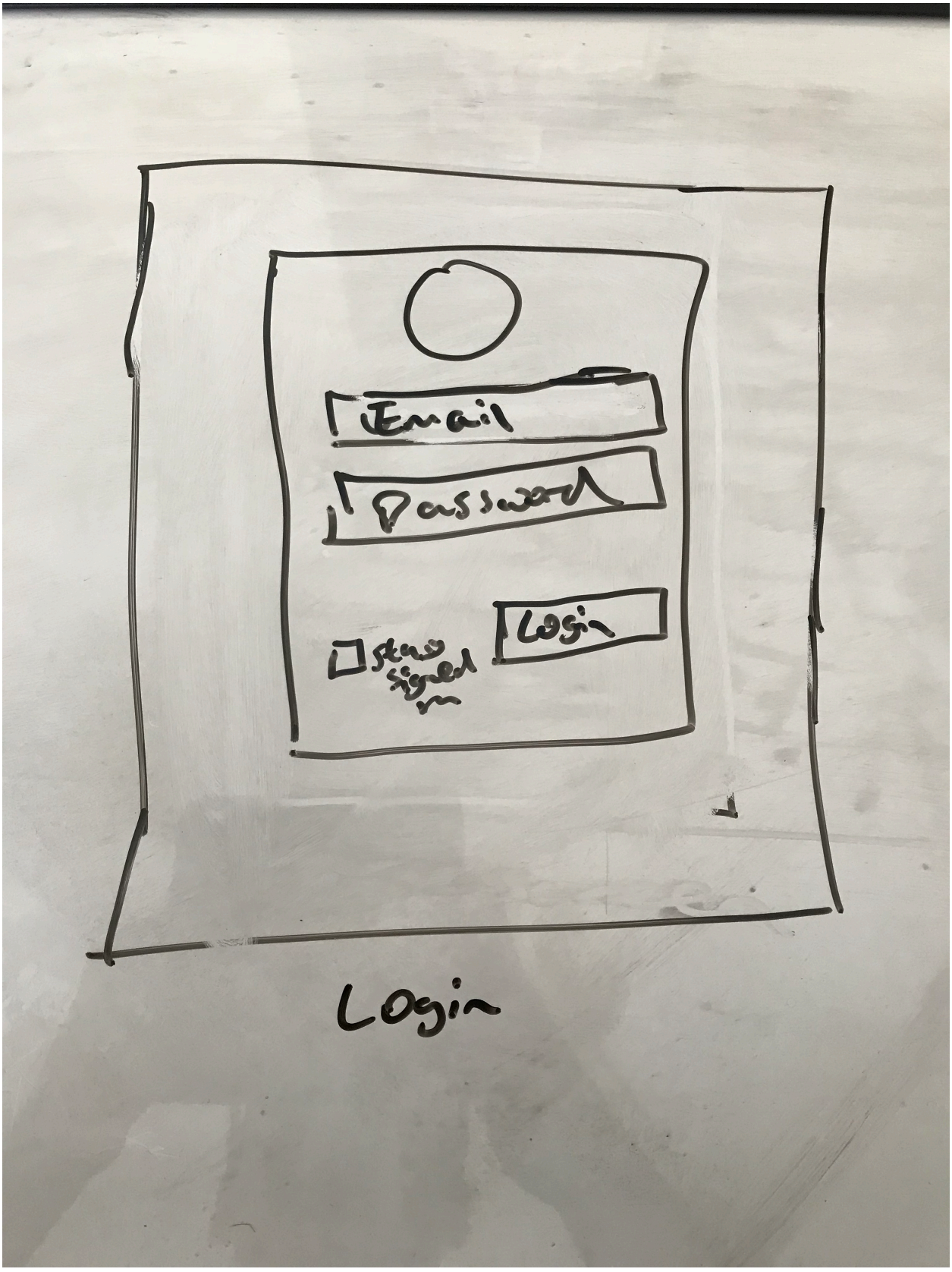


Figure 20: Login Page

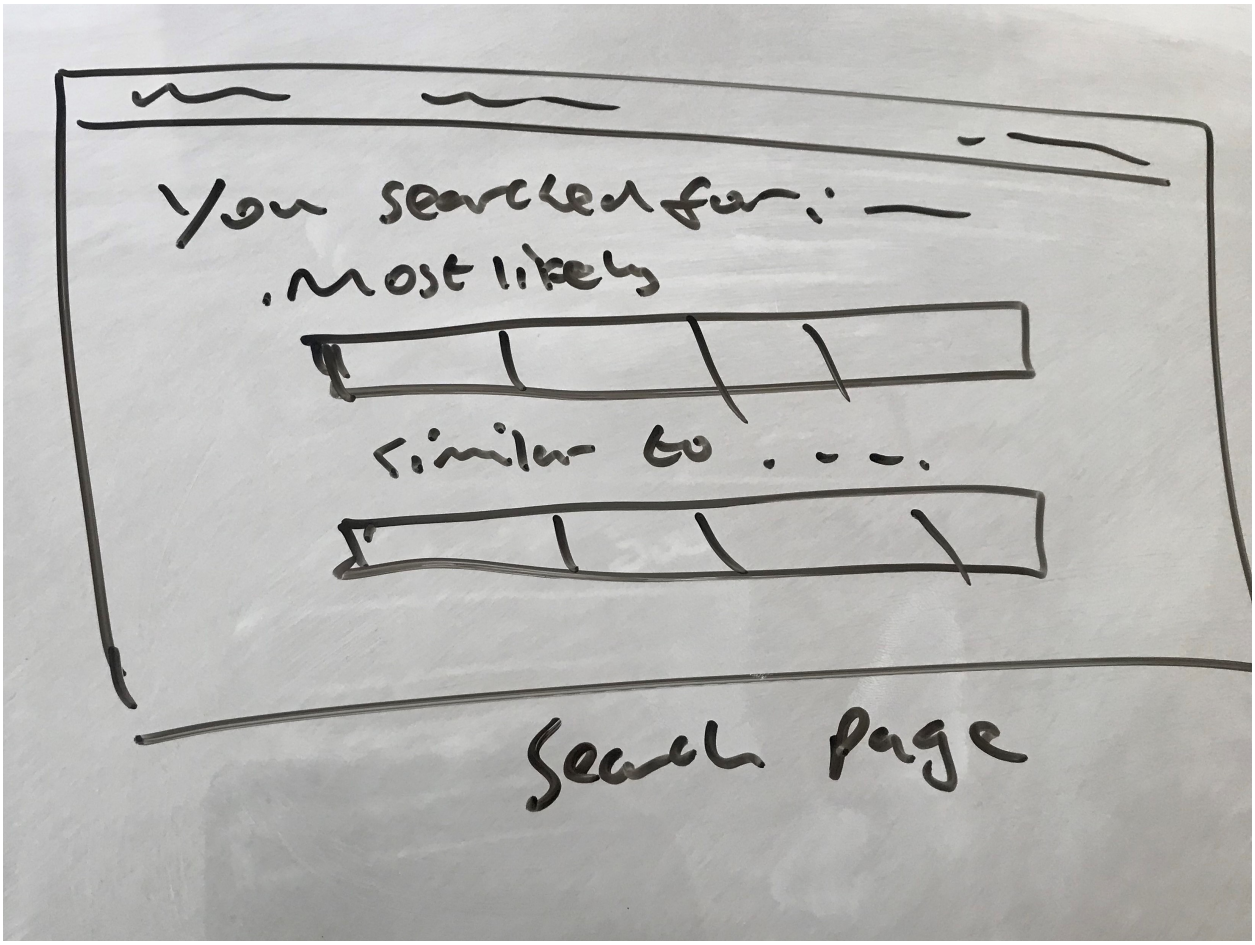


Figure 22: Search Page

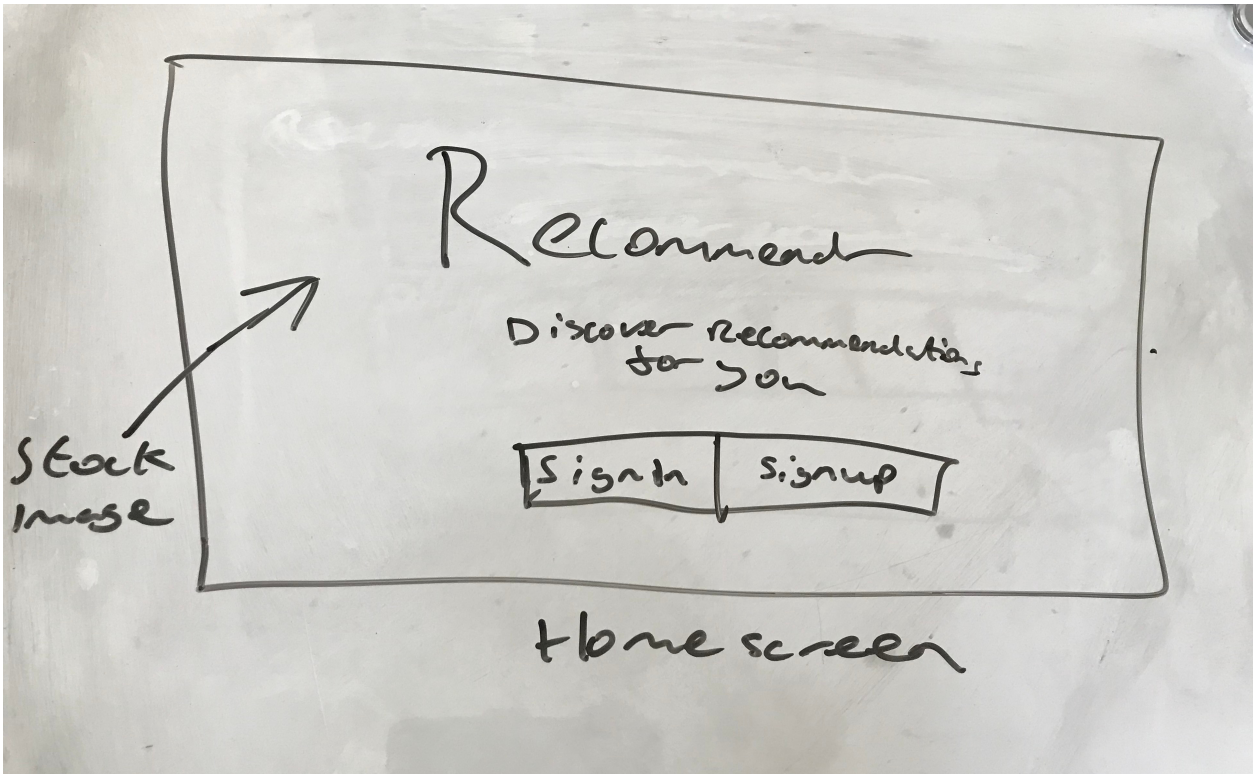


Figure 23: Landing Page

F.2 HCI Considerations

1. Offer Informative Feedback:

The website tries to offer a range of ways to provide informative feedback. For example, when users try and rate a movie the ratings bar will fill. In addition, when users are giving feedback towards recommendations, the 'thumbs' button will fill in which notifies the user that their option has been made.

Thirdly, on the main page it consists of an 'Infinite Scroll' element; this is used by many websites, such as Facebook. As a user scrolls, t loading text appears to notify them that more data is being retrieved. Once they reach the bottom (gone through all the users rated movies), they are signalled that they need to rate more films so that they are able to see more content.

Finally, when the item carousel reaches its end or beginning the directional buttons grey out, indicating that it cannot go any further.

2. Strive for Consistency

One of the main aims for the system was to strive for consistency, to illustrate recommendations carousels were used to show movie content. This would be used to notify the user that it was about item content and would either show a score relating to the movies top score, or the 'thumbs' to notify them that this is for giving feedback.

3. Design dialog to yield to closure

The most important element for designing a dialog to yield to closure, is that there should be informative feedback from start to finish. An example of this in the projects website is when a user enters the website for the first time.

The process starts by a user signing up. They are notified by their success by being taken to the initial ratings page; they also receive an email. Upon reaching the initial ratings page, the users are enforced to rate a subset of films. They are not able to visit any other page on the website until this is done, which enforces them to stay in the correct flow. Once they rate a subset of films they are notifed with success by being taken to the main page which presents them with a subset of carousels which are related to the films

that they rated.

4. Reduce short-term memory load.

Due to the website containing over 9000 movies when a user searches for an item it could return a significant amount of matches. To reduce the amount of information the user sees, the data is put into a list view and limited to 10 items. This has pagination which enables the user to view more content if they don't see what they were looking for on the first page. In addition, the 'Infinite Scroll' feature on the main page results in the user only needing to stay on that page to view overall recommendations which reduces the actions needed for the user to find recommended content.

G Architecture

Figure 24: Originally proposed architecture

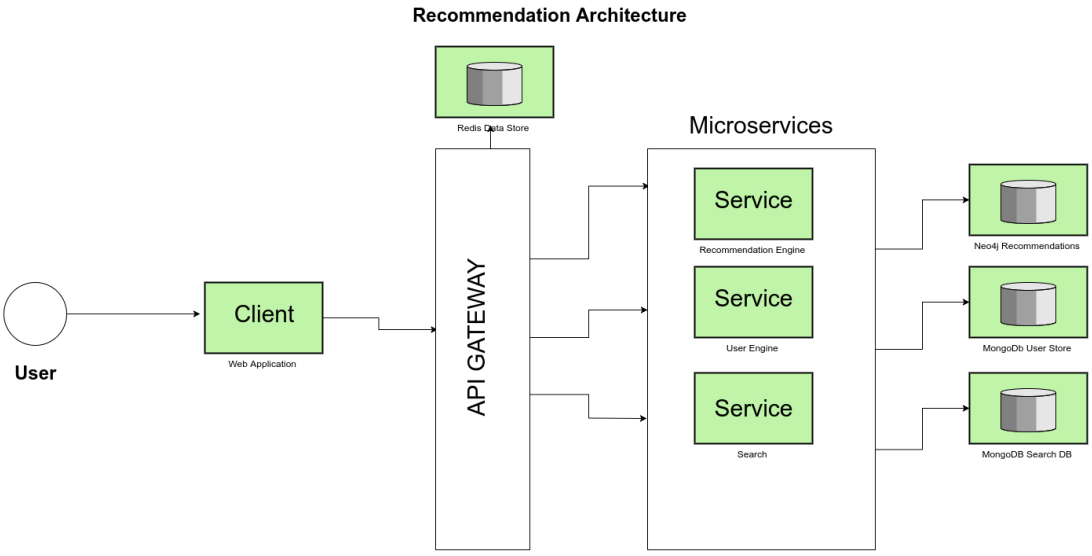
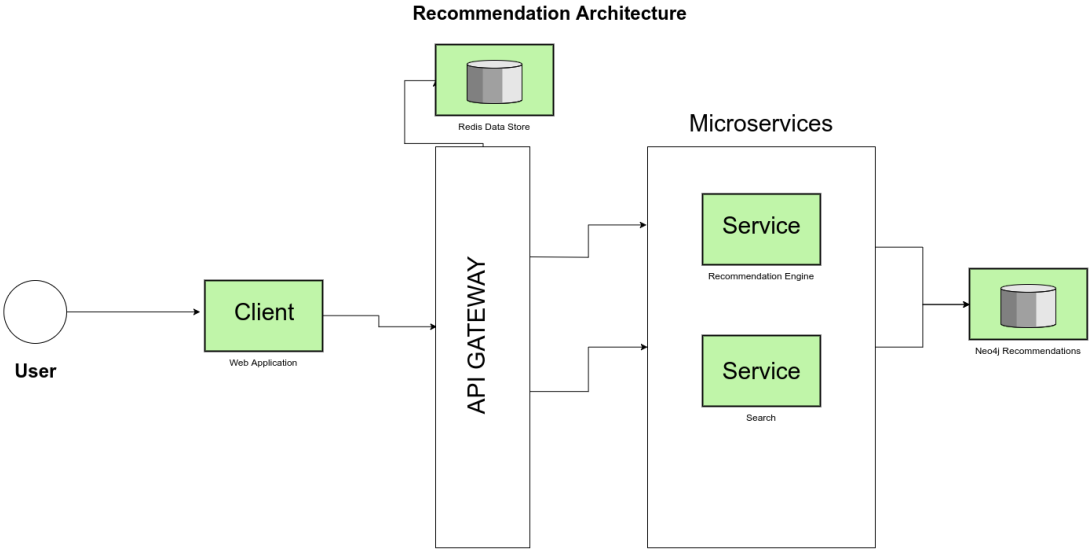


Figure 25: Actual Architecture



G.1 System Architecture Design

G.2 Database Design

The data structure was already normalised and format was taken from the movielens dataset. The final ERD:

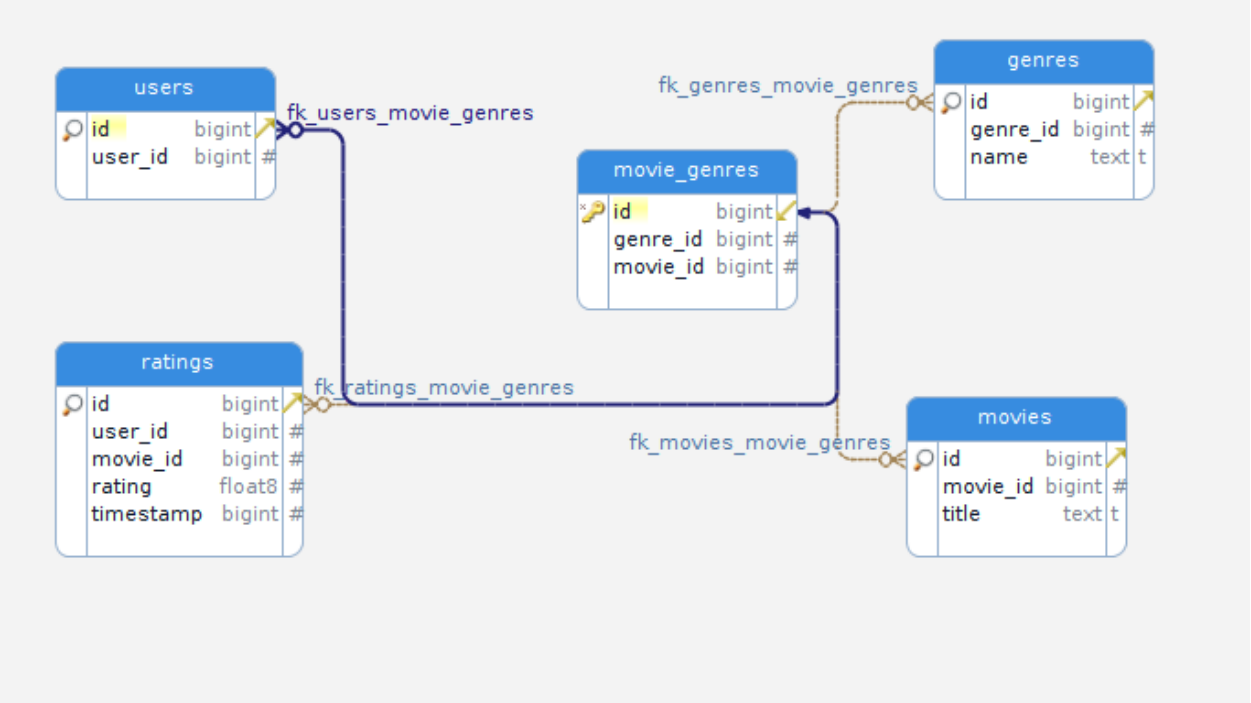


Figure 26: Postgres Relational Database ERD

The final design for the projects graph database:

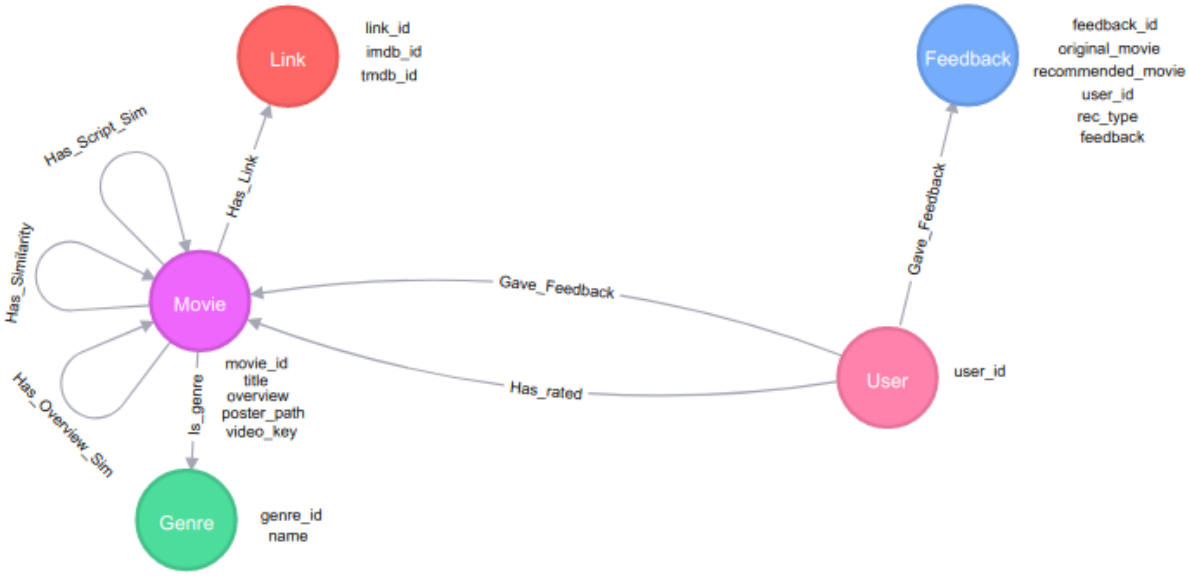


Figure 27: Graph Database model

H Experiments

H.1 Latent Matrix Factorisation

In order to understand the effectiveness of the LFM algorithm a series of experiments were run. sads

H.1.1 CV and Bias inclusion

H.1.2 Comparing Different values of K

H.1.3 Recommendation Outputs

```
Movie recommendations for Toy Story (1995)
Raiders of the Lost Ark (Indiana Jones and the Raiders of the L
Beyond, The (E tu vivrai nel terrore - L'aldilà) (1981): Sim: (
Toy Story 2 (1999): Sim: 0.974285
Shawshank Redemption, The (1994): Sim: 0.973758
Little Caesar (1931): Sim: 0.972558
Birds, The (1963): Sim: 0.971934
Big Daddy (1999): Sim: 0.971931
Blood Simple (1984): Sim: 0.971847
Usual Suspects, The (1995): Sim: 0.971753
Rain Man (1988): Sim: 0.971647
```

(a) Recommendations MF without bias

```
Movie recommendations for Toy Story (1995)
Lion King, The (1994): Sim: 0.972665
Apollo 13 (1995): Sim: 0.969511
Aladdin (1992): Sim: 0.969457
Crimson Tide (1995): Sim: 0.968869
Shrek (2001): Sim: 0.968588
Silence of the Lambs, The (1991): Sim: 0.968133
Batman (1989): Sim: 0.967977
Raiders of the Lost Ark (Indiana Jones and the Raiders of the Lost Ark) (1981)
Babe (1995): Sim: 0.967584
Schindler's List (1993): Sim: 0.967390
```

(b) Recomendations with bias

Figure 28: Recommendation for Toy Story, using cosine similarity (MF using bias $k = 100, \alpha = 0.001, \gamma = 0.005$)

```
Movie recommendations for Star Wars: Episode IV - A New Hope (
Star Wars: Episode V - The Empire Strikes Back (1980): Sim: 0.
Star Wars: Episode VI - Return of the Jedi (1983): Sim: 0.9856
Raiders of the Lost Ark (Indiana Jones and the Raiders of the
Usual Suspects, The (1995): Sim: 0.979144
Shawshank Redemption, The (1994): Sim: 0.979026
Pulp Fiction (1994): Sim: 0.978837
Definitely, Maybe (2008): Sim: 0.977772
Silence of the Lambs, The (1991): Sim: 0.976563
Seven (a.k.a. Se7en) (1995): Sim: 0.976409
Tropic Thunder (2008): Sim: 0.976190
```

(a) Recommendations MF without bias

```
Movie recommendations for Star Wars: Episode IV - A New Hope (1977)
Star Wars: Episode V - The Empire Strikes Back (1980): Sim: 0.988040
Star Wars: Episode VI - Return of the Jedi (1983): Sim: 0.985394
Raiders of the Lost Ark (Indiana Jones and the Raiders of the Lost Ark) (198
Godfather, The (1972): Sim: 0.974778
Fight Club (1999): Sim: 0.973837
Back to the Future (1985): Sim: 0.973651
Indiana Jones and the Last Crusade (1989): Sim: 0.972925
Matrix, The (1999): Sim: 0.972092
Jimmy Neutron: Boy Genius (2001): Sim: 0.971664
Godfather: Part II, The (1974): Sim: 0.971637
```

(b) Recommendations with bias

Figure 29: Recommendation for Star Wars: New Hope using cosine similarity (MF using bias $k = 100, \alpha = 0.001, \gamma = 0.005$)

```
Movie recommendations for Toy Story (1995)
Toy Story 2 (1999): Sim: 0.471547
Shrek (2001): Sim: 0.459054
Definitely, Maybe (2008): Sim: 0.444193
Lion King, The (1994): Sim: 0.434884
Indiana Jones and the Last Crusade (1989): Sim: 0.422768
Decline of the American Empire, The (Déclin de l'empire amér
Man Who Wasn't There, The (2001): Sim: 0.416141
Gladiator (2000): Sim: 0.415396
Men in Black (a.k.a. MIB) (1997): Sim: 0.398124
Babe (1995): Sim: 0.391966
```

(a) Recommendations using PCC

```
Movie recommendations for Toy Story (1995)
Lion King, The (1994): Sim: 0.972665
Apollo 13 (1995): Sim: 0.969511
Aladdin (1992): Sim: 0.969457
Crimson Tide (1995): Sim: 0.968869
Shrek (2001): Sim: 0.968588
Silence of the Lambs, The (1991): Sim: 0.968133
Batman (1989): Sim: 0.967977
Raiders of the Lost Ark (Indiana Jones and the Raiders of the Lost Ark) (1981)
Babe (1995): Sim: 0.967584
Schindler's List (1993): Sim: 0.967390
```

(b) Recommendations using Cosine similarity

Figure 30: Recommendation for Toy Story, using cosine and PCC matrix. (MF using bias $k = 100, \alpha = 0.001, \gamma = 0.005$)

PCC recommendation	**Relevant recommendation**	**Cosine recommendation**	**Relevant recommendation**
Toy Story 2	1	Lion King	1
Shrek	1	Apollo 13	0
Definitely, Maybe	0	Aladdin	1
Lion King	1	Crimson Tide	0
Indiana Jones	0	Shrek	1
Decline of the American Empire	0	Silence of the Lambs	0
Man Who Wasn't There	0	Batman	0
Gladiator	0	Raiders of the Lost Ark	0
Men In Black	0	Babe	1
Babe	1	Schindler's List	0
Total	**4**		**4**

Table 4: Toy Story recommendation evaluation for PCC and Cosine similarity

```
Movie recommendations for Star Wars: Episode IV - A New Hope (1
Star Wars: Episode V - The Empire Strikes Back (1980): Sim: 0.7
Star Wars: Episode VI - Return of the Jedi (1983): Sim: 0.71612
Colonel Chabert, Le (1994): Sim: 0.593292
To Catch a Thief (1955): Sim: 0.586694
Bajirao Mastani (2015): Sim: 0.582136
Camelot (1967): Sim: 0.581448
Sunset Blvd. (a.k.a. Sunset Boulevard) (1950): Sim: 0.581334
National Lampoon's Lady Killers (National Lampoon's Gold Digger
Raiders of the Lost Ark (Indiana Jones and the Raiders of the L
Butch Cassidy and the Sundance Kid (1969): Sim: 0.580284
```

(a) Recommendations using PCC

```
Movie recommendations for Star Wars: Episode IV - A New Hope (1977)
Star Wars: Episode V - The Empire Strikes Back (1980): Sim: 0.988040
Star Wars: Episode VI - Return of the Jedi (1983): Sim: 0.985394
Raiders of the Lost Ark (Indiana Jones and the Raiders of the Lost Ark) (198
Godfather, The (1972): Sim: 0.974778
Fight Club (1999): Sim: 0.973837
Back to the Future (1985): Sim: 0.973651
Indiana Jones and the Last Crusade (1989): Sim: 0.972925
Matrix, The (1999): Sim: 0.972092
Jimmy Neutron: Boy Genius (2001): Sim: 0.971664
Godfather: Part II, The (1974): Sim: 0.971637
```

(b) Recommendations using Cosine similarity

Figure 31: (Recommendation for Star Wars: New Hope, using cosine and PCC matrix. (MF using bias $k = 100, \alpha = 0.001, \gamma = 0.005$)

PCC recommendation	Relevant recommendation	Cosine recommendation	Relevant recommendation
Star Wars Episode V	1	Star Wars Episode V	1
Star Wars Episode VI	1	Star Wars Episode VI	1
Colonel Charbert	0	Raiders of the Lost Ark	1
To Catch a Thief	0	Godfather	0
Bajirao Mastani	0	Fight Club	0
Camelot	0	Back to the future	1
Sunset Blvd	0	Indiana Jones	1
National Lampoon's	0	Matrix	
Raiders of the Lost Ark	1	Jimmy Neutron	0
Butch Cassidy	0	Godfather Part II	0
Total	**3**		**5**

Table 5: Star Wars: New Hope recommendation evaluation for PCC and Cosine similarity

```
Movie recommendations for Jumanji (1995)
Richie Rich (1994): Sim: 0.730082
Bobby (2006): Sim: 0.714721
Little Colonel, The (1935): Sim: 0.712709
Blown Away (1994): Sim: 0.711150
Breaker Morant (1980): Sim: 0.710309
Man's Favorite Sport? (1964): Sim: 0.708590
Flight of the Intruder (1991): Sim: 0.707815
52 Pick-Up (1986): Sim: 0.706198
Country (1984): Sim: 0.705844
Friday the 13th Part VIII: Jason Takes Manhattan (1989): Sim:
```

(a) Recommendations using PCC

```
Movie recommendations for Jumanji (1995)
Men in Black (a.k.a. MIB) (1997): Sim: 0.969721
Dave (1993): Sim: 0.966884
Men at Work (1990): Sim: 0.966521
Rock, The (1996): Sim: 0.965965
Indiana Jones and the Last Crusade (1989): Sim: 0.965668
Temp, The (1993): Sim: 0.965367
Monty Python and the Holy Grail (1975): Sim: 0.965133
Breakfast Club, The (1985): Sim: 0.964632
Mrs. Doubtfire (1993): Sim: 0.964498
Under the Same Moon (Misma luna, La) (2007): Sim: 0.964153
```

(b) Recommendations using Cosine similarity

Figure 32: (Recommendation for Jumanji, using cosine and PCC matrix. (MF using bias $k = 100, \alpha = 0.001, \gamma = 0.005$)

PCC Recommendation	Relevant recommendation	Cosine Recommendation	Relevant recommendation
Richie Rich	1	Men In Black	1
Bobby	0	Dave	0
Little Colonel	0	Men at Work	0
Blown Away	0	Rock, The	1
Breaker Morant	0	Indiana Jones	1
Man's Favourite Sport	0	Temp the	0
Flight of the intruder	0	Monty Python	0
52 Pick-Up	0	Breakfast Club	0
Country	1	Mirs Doubtfire	1
Friday the 13th	0	Under the Same Moon	0
Total	**3**		**4**

Table 6: Jumanji evaluation for PCC and Cosine similarity

H.2 TF-IDF

The TF-IDF algorithm went through testing to make a comparison against the recommendations using movie overviews as an input parameter, and screenplay text.

```
Recommendations for Star Wars: Episode IV - A New Hope
Star Wars: Episode V - The Empire Strikes Back: Sim: 0.41
Star Wars: Episode VII - The Force Awakens: Sim: 0.25
Star Wars: Episode VI - Return of the Jedi: Sim: 0.19
Swan Princess, The: Sim: 0.13
Sleeping Beauty: Sim: 0.11
Princess Bride, The: Sim: 0.09
Mirror Mirror: Sim: 0.09
Topsy-Turvy: Sim: 0.09
Star Wars: Episode III - Revenge of the Sith: Sim: 0.08
Mummy, The: Sim: 0.08
```

(a) Recommendations using Overviews

```
Recommendations for Star Wars: Episode IV - A New Hope
Star Wars: Episode VI - Return of the Jedi: Sim: 0.80
Star Wars: Episode V - The Empire Strikes Back: Sim: 0.78
Metropolitan: Sim: 0.36
Gerontophilia: Sim: 0.36
Zerophilia: Sim: 0.36
Mission to Mars: Sim: 0.23
Star Wars: Episode VII - The Force Awakens: Sim: 0.20
Next Three Days, The: Sim: 0.18
Mirror Mirror: Sim: 0.12
MirrorMask: Sim: 0.12
```

(b) Recommendations using movie scripts

Figure 33: (Star Wars: New Hope: Recommendation results using TF-IDF for movie overviews and scripts)

```
Recommendations for Aliens
Alien: Resurrection: Sim: 0.26
Event Horizon: Sim: 0.14
Solaris: Sim: 0.14
Talented Mr. Ripley, The: Sim: 0.14
Mission to Mars: Sim: 0.12
Alien: Sim: 0.12
Alien Nation: Sim: 0.10
Day the Earth Stood Still, The: Sim: 0.09
Hitchhiker's Guide to the Galaxy, The: Sim: 0.09
Starship Troopers 3: Marauder: Sim: 0.09
```

(a) Recommendations using Overviews

```
Recommendations for Aliens
Talented Mr. Ripley, The: Sim: 0.65
Afro Samurai: Resurrection: Sim: 0.58
Alien: Resurrection: Sim: 0.58
Out of Sight: Sim: 0.14
I'll Do Anything: Sim: 0.09
Mexican, The: Sim: 0.08
Merchant of Venice, The: Sim: 0.08
Mechanic, The: Sim: 0.08
Giant Mechanical Man, The: Sim: 0.08
Gilda: Sim: 0.08
```

(b) Recommendations using movie scripts

Figure 34: (Alien: Recommendation results using TF-IDF for movie overviews and scripts)

```
Recommendations for Dawn of the Planet of the Apes
Battle for the Planet of the Apes: Sim: 0.26
Godzilla: Sim: 0.16
Phantoms: Sim: 0.13
Alive: Sim: 0.11
Cleopatra: Sim: 0.11
Conquest of the Planet of the Apes: Sim: 0.11
Inside Out: Sim: 0.11
Rise of the Planet of the Apes: Sim: 0.10
Omega Man, The: Sim: 0.09
World Is Not Enough, The: Sim: 0.09
```

(a) Recommendations using Overviews

```
Recommendations for Dawn of the Planet of the Apes
Backdraft: Sim: 0.00
Two Jakes, The: Sim: 0.00
Bonfire of the Vanities: Sim: 0.00
Map of the World, A: Sim: 0.00
Distinguished Gentleman, The: Sim: 0.00
Flawless: Sim: 0.00
Toy Story: Sim: 0.00
Fisher King, The: Sim: 0.00
Awakenings: Sim: 0.00
Cider House Rules, The: Sim: 0.00
```

(b) Recommendations using movie scripts

Figure 35: (Planet of the apes: Recommendation results using TF-IDF for movie overviews and scripts)

```
Recommendations for Star Trek: Generations
Star Trek: Nemesis: Sim: 0.26
Star Trek: First Contact: Sim: 0.17
Lady Eve, The: Sim: 0.09
Star Trek: Sim: 0.08
True Colors: Sim: 0.08
Cecil B. DeMented: Sim: 0.07
Destination Tokyo: Sim: 0.07
Star Trek: The Motion Picture: Sim: 0.07
Human Nature: Sim: 0.07
House of Frankenstein: Sim: 0.07
```

(a) Recommendations using Overviews

```
Recommendations for Star Trek: Generations
Star Trek: First Contact: Sim: 0.68
Star Trek: Nemesis: Sim: 0.66
Star Trek: The Motion Picture: Sim: 0.23
Starry Eyes: Sim: 0.23
Starred Up: Sim: 0.23
Star Trek: Sim: 0.23
Star Trek: Sim: 0.23
She's Out of My League: Sim: 0.22
Sandlot, The: Sim: 0.07
Landlord, The: Sim: 0.07
```

(b) Recommendations using movie scripts

Figure 36: (Star Trek: Recommendation results using TF-IDF for movie overviews and scripts)

I Testing

Below contains the output from the unit tests written for each app. The tests have been written to ensure that the services are communicating correctly. To run the tests go into the desired app folder and run the **pytest** command. Each tests can contain a set of further tests, to ensure that it reaches the acceptance criteria, such as returning a specific response, status code 200, or 400 with the appropriate error message.

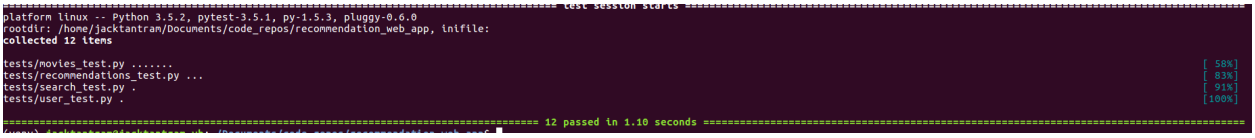


Figure 37: Website Unit tests

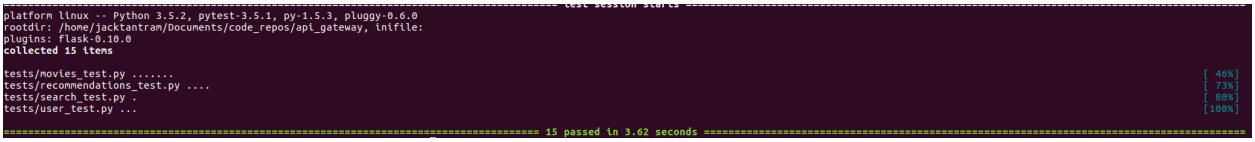


Figure 38: API Gateway unit Tests

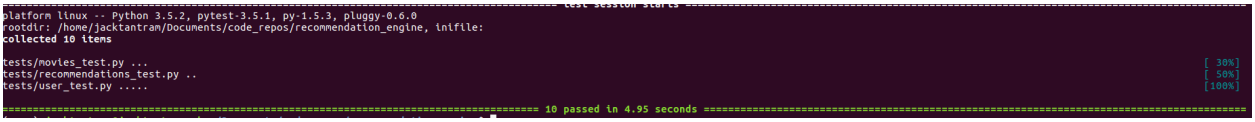


Figure 39: Recommendation engine unit tests

J User Testing

J.1 Feedback

Below are the questions asked and their responses:

1 **What did you think the website was about when you first entered?**

5 out of 5 people answered this question

Movie site

21 hours ago

Wasn't sure, not enough information on initial landing page

a day ago

I thought it was about movie recommendations

12 days ago

A film recommendation site

12 days ago

a movie suggestion site

13 days ago

2 **Is there anything you like about the user interface?**

5 out of 5 people answered this question

Looks professional

a day ago

It reminds me of Netflix

a day ago

I liked the images on the covers and nice layout

12 days ago

clearly laid out and easy to navigate

12 days ago

simple and easy to use, i liked the images of the films covers

13 days ago

3 **Were you able to create an account?**

5 out of 5 people answered this question

100%	Yes	**5** Responses
0%	No	**0** Response

4 **Did the process of creating an account make sense?**

5 out of 5 people answered this question

yes

a day ago

Yes

a day ago

Yes, although rating films was a bit fiddly with submit button at bottom

12 days ago

yes

12 days ago

yep

13 days ago

5 **What were you expecting the home page to look like?**

5 out of 5 people answered this question

Film recomendations

a day ago

Netflix

a day ago

Show movies i like

12 days ago

clearly laid out, with lots of films

12 days ago

pretty much how it looked, looked like netflixs etc

6 **Was anything too obtrusive?**

5 out of 5 people answered this question

Could not figure out how many ratings i made on the initial ratings page

a day ago

The curly text was a bit off-putting

a day ago

No

12 days ago

no

12 days ago

nope

7 **How did you find the layout of the site?**

5 out of 5 people answered this question

Very professional

a day ago

Good

a day ago

I liked the layout but the font for titles was a bit strange

12 days ago

clear and easy to use

12 days ago

easy to use, if the thumbs changed colour when you rated then it would be better. but all round pretty simple

13 days ago

8 **What would encourage you to visit the site again?**

5 out of 5 people answered this question

Up to date catalogue

a day ago

Email notification on new movies

a day ago

Not sure

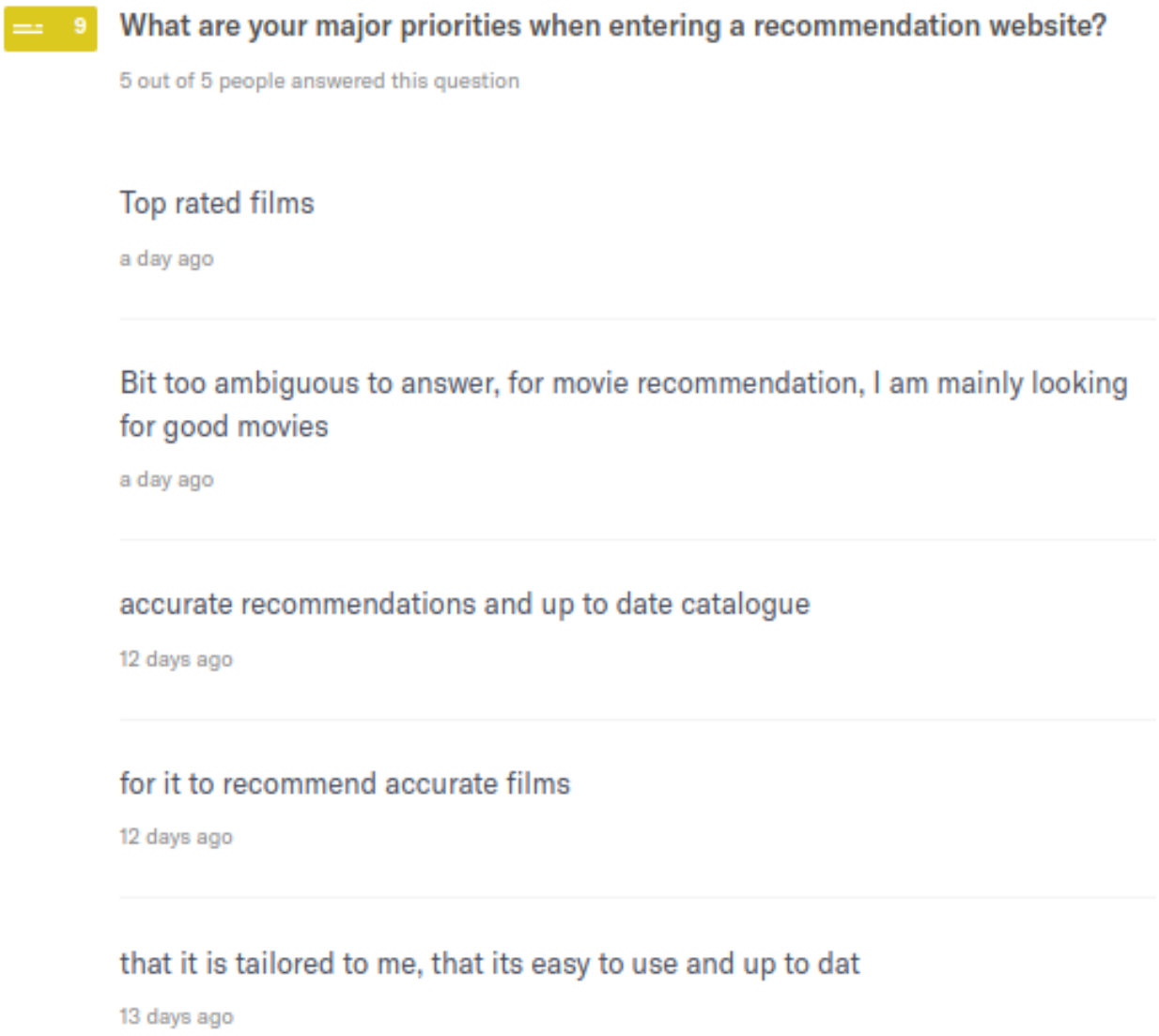
12 days ago

If it gave me recommendations of films I liked

12 days ago

i dont know

13 days ago



J.2 Results

Table 7: Results N=10

	Matrix Factorisation	TF-IDF Movie Scripts	TF-IDF Overviews
Toy Story	0.6	0.03	0.25
Star Wars: Episode V	0.3	0.46	0.5
Aladdin	0.36	0.02	0.48
Pulp Fiction	0.34	0.14	0.06
Austin Powers (1997)	0.18	0.20	0.20

Table 8: Results N=5

	Matrix Factorisation	TF-IDF Movie Scripts	TF-IDF Overviews
Toy Story	0.67	0.07	0.5
Star Wars: Episode V	0.6	0.4	0.8
Aladdin	0.52	0.04	0.72
Pulp Fiction	0.52	0.17	0.04
Austin Powers (1997)	0.32	0.38	0.42

Table 9: Global Average across trials

	Matrix Factorisation
Matrix Factorisation N=10	0.36
Matrix Factorisation N=5	0.53
Overview N=10	0.30
Overview N=5	0.49
Script N=10	0.17
Script N=5	0.21

K Jupyter Notebooks

All Jupyter Notebooks are available in the project source code folder