School of Computing

FACULTY OF ENGINEERING AND
PHYSICAL SCIENCES

UNIVERSITY OF LEEDS

Final Report

Using TrueSkill™ to Compare Relative and Absolute Methods of Ranking Movies

Louis Bennett Boswell

Submitted in accordance with the requirements for the degree of
BSc Computer Science (Digital & Technology Solutions)

2023/24

COMP3932 Synoptic Project

The candidate confirms that the following have been submitted*:*

Items	**Format**	**Recipient(s) and Date**
Final Report	*PDF file*	*Uploaded to Minerva (09/05/24)*
Scanned participant consent forms	*PDF file*	*Uploaded to Minerva (09/05/24)*
Link to online code repository	*URL - GitHub*	*Sent to supervisor and assessor (09/05/24)*

The candidate confirms that the work submitted is their own and the appropriate credit has been given where reference has been made to the work of others.

I understand that failure to attribute material which is obtained from another source may be considered as plagiarism.

Louis Bennett Boswell

Summary

Reviews and ratings are an integral piece of how the movie industry operates. Good reviews may encourage prospective movie goers to see a movie, with bad reviews dissuading them. This feedback mechanism is vital to the health of the industry and must therefore be built upon a rating system that is fit for purpose.

The common approach for displaying user sentiment and satisfaction towards a movie are absolute scales – where users give a movie a score out of a predefined scale (1-5 or 1-10 usually). The approach leads to many issues including inconsistency, difficult translating ratings between different users and lack of a mechanism to account for bias.

This project attempts to investigate a novel system for ranking movies – using relative comparisons to calculate a score for each movie using TrueSkill, a proprietary matchmaking algorithm created by Microsoft. The data collection platform will select five random movies from the users' viewed movies and ask the user to order them best to worst. Once submitted, TrueSkill updates the rating of these movies both for the user and globally across the system – creating a new novel relative ordering for this users' movies. This new order will then be compared to users' previous absolute reviews to help understand if a new relative ranking system can provide more effective, accurate and reproducible results.

Acknowledgements

I would like to my supervisor Dr Hui Lau for his assistance with this project.

I would like to thank all 12 users who participated in the user testing and data collection phases of this project for their endurance and patience, completing over 2500 movies comparisons.

Table of Figures

Table of Contents

Chapter 1
Introduction and Background Research

1.1 Introduction

User reviews are a vital part of the movie, television, and music industries – providing a rough outline on what to expect from a piece of entertainment media. The most common forms of ratings are star ratings out of 5 or numerical ratings out of 10, with some providers opting to show these as a score out of 100.

This common practice works for formulating a group consensus on a piece of media; however, it does not lead to a normative distribution of reviews. Users tend to gravitate towards 7 being average and 5 being bad, neglecting the entire 1-10 (or 1-5) scale, evidenced in **Figure 1** below.

This investigative project aims to propose an alternative method for formulating and understanding public opinion by comparing pieces of media relatively. All comparisons submitted by all users will also be logged in the database, creating a large dataset of film comparisons, to be investigated later in the report.

1.2 Background Research

1.2.1 Impact of Reviews on Movie Success

Movie critic reviews and the success of a movie are not independent, with critic reviews predicting the success of mainstream movies and directly influencing the success of art house movies (Gemser et al, 2007).

Eliashberg and Shugan (1997) find that critical reviews are statistically insignificant at the beginning of a movies life but do significantly correlate with its late box office revenue and cumulative receipts. They argue this suggests that critics are less prominent than other factors in motivating people to attend the cinema.

In a more recent study looking at online consumer reviews, as well as critics, Jacobs et al. (2015) find that negative reviews can lower enjoyment of movies, regardless of the credibility of the reviewer. They found that consumer reviews had a more significant effect on the viewers' enjoyment of a movie when compared to critic reviews.

It is worth noting however, most research on the subject of reviews' effect on media success are arguably outdated in the everchanging landscape of media consumption.

When considering Letterboxd, a platform self-described as 'a global social network for grass-roots film discussion and discovery' (Letterboxd, 2024), users will usually discover movies through friend / critic reviews. Letterboxd's top 250 movies by rating lists, ratings on friend's logged films and rating distributions visible on both user profile and movie pages promote the rating of movies as an intrinsic feature of the platform – especially when it comes to movie discovery. One could argue that in the new age of entertainment media, the predictive factor outlined above may be elevated due to the perceived importance of movie ratings and interconnectivity on Letterboxd.

The accessibility and popularity of media reviewing sites (Letterboxd, IMDb, Rotten Tomatoes etc.), may have shifted the way reviews are perceived from prospective movie viewers following a single critic, to now searching for an aggregated user score. The rise of streaming has also had significant impacts on how we consume media – especially cinema.

1.2.2 Flaws of Absolute Rating Systems

As investigated in the previous section, media reviews play a vital role in predicting the success of a movie or a viewer's enjoyment of said movie. Given the importance of reviews and user ratings, any rating system in place must be ***consistent***, ***translatable*** and ***accurate***. ***Consistency*** in a rating system refers to its ability to provide clear and understandable distinction between items and their minimum and maximum rating. A rating system can be considered ***translatable*** if media ratings from different users occupy the same scale and distribution, and therefore can be compared accurately. ***Accuracy*** refers to the ability for a system to allow for high granularity and distinction between movies of similar user preference.

The most common systems for rating movies are absolute systems, where movies are rated on an arguably arbitrary scale (1-5 stars, 0-100% etc.).

Rotten Tomatoes is a popular movie review site, that aggregates critic reviews from various sources and user reviews, to give two different scores for a movie – Tomatometer and audience score. The aggregated scores are then displayed as either a fresh tomato (over 60% positive reviews) or a green splat (less than 60% positive reviews) (Rotten Tomatoes, 2024).

The first issue with this type of aggregation is the binary positive or negative classification of a critic review. Critics reviews are provided in all sorts of different formats 1-4, 1-5, 1-10 or even A-E, further abstracting and complicating this metric. Using this method, both a review of *masterpiece* and *worth a watch* are equally weighted, neglecting all the nuance in the

aggregated reviews. It is theoretically possible on Rotten Tomatoes a movie could have 59% 5/5 critic reviews but would be considered rotten if the other 41% were 2/5 – the same movie would have an expected score of 7.54 on IMDb or 3.8 stars on Letterboxd.

Audience score on Rotten Tomatoes is calculated using a similar metric, a movie will be considered fresh if 60% of the reviews are 3.5 or above. The personal differences in review scales are important to mention here, whilst a 1–5 star rating scale will always be consistent across users, their distribution of ratings within that scale may not be. Some users may reject to the idea of giving a movie 5 stars as their review, due to the perceived impossibility of perfection, or below 2 stars, as they would not consider watching movies that bad.

The average annual Rotten Tomatoes score has annually been increasing, according to data collected by Gross (2024). The average review score has increased from 44.7% in 2004 to 66.9% in 2022, representing an increase of 66.8%. Due to Rotten Tomatoes dividing line between fresh and rotten being at 60%, this leads to 60.4% of movies released in 2021 being considered fresh – compared to 27.6% in 2004. According to Gross (2024), Rotten Tomatoes say that the increase comes from diversifying critics – but this still doesn't explain why one set of critics would be more positive than another.

Recency bias is described as a cognitive bias where a greater importance is place on more recent events – causing user's to favourably rate movies they have viewed recently. It's unlikely Rotten Tomatoes increase in rating is due solely to recency bias as recency bias would affect all reviews at all years.

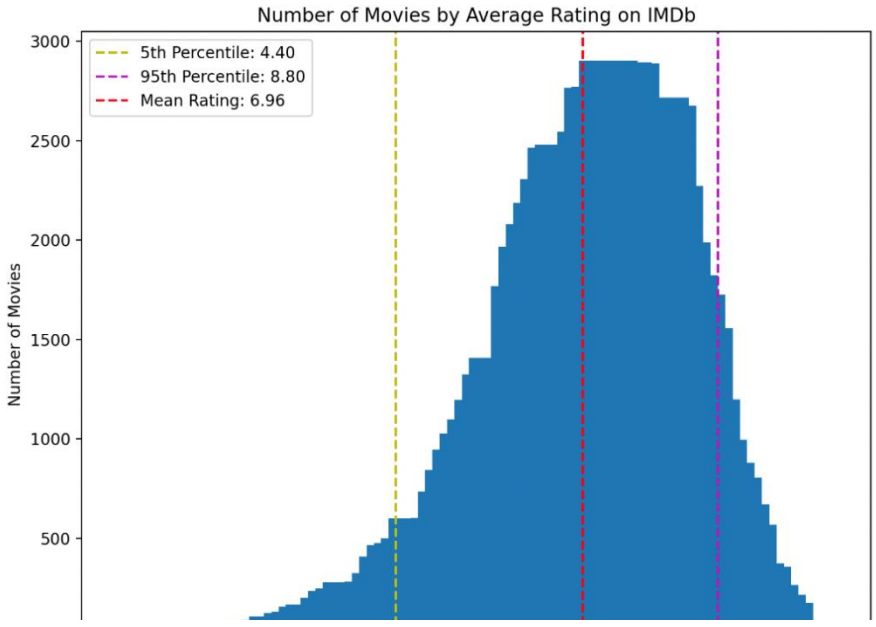

Figure 1 – Distribution of Movies by Average Rating on IMDb

These issues are not unique to Rotten Tomatoes, IMDb has a similar issue with users overrating movies. **Figure 1** is a visualisation created by the student of IMDb's publicly available dataset of movie ratings, filtered by movies with more than 1000 reviews (IMDb, 2024). From this visualisation, we can see that whilst the distribution of average ratings roughly follows a normal distribution – the ratings are heavily skewed to the top end of the scale.

Whilst 6.96 being an average rating for a movie may be intuitive to a user, it reflects a greater problem of granularity. If 90% of movies occupy only half the scale, either that 10% must be different enough to warrant the rest of the scale – or there are issues with the utilization of said scale.

The key purpose of the rating scale is to differentiate movies based on quality or preference. When 90% of movies occupy only 44% of the scale, it becomes difficult to differentiate between movies that are average, slightly above average or *perfect* – leading to inconsistent granularity across the entire scale. Furthermore, if ratings cluster around the 7 rating, it suggests that the entire scale is not being fully utilized. This method of aggregating user reviews is common across movie review websites, with IMDb and Letterboxd both using private algorithms to aggregate and average user reviews – further abstracting user sentiment.

A similar example of this phenomena can be found on Airbnb (an online marketplace for short / long-term homestays), where almost 95% of properties have an average rating of 4.5 or 5 stars (Zervas et al, 2015). This is a more extreme example, showing lack of utilization of the entire scale by users – and therefore it's difficulty in differentiating between reviews if the vast majority occupy a small portion of said scale. It is also important to note, both movies and holidays are similarly seen as positive experiences – where something overwhelmingly negative would have to happen to shift most people's perceptive to negative overall. This could help to describe why most reviews on both systems occupy the high end of the scale, as negative reviews would be more impactful to the reader's perception.

In an ideal scale, the highest rated movie and lowest rated movie would be 10 and 0 respectively – with all other movies normally distributed between. This would not only better utilize the entire scale (allowing for more precise distinction between similar movies) but also make comparing ratings between different users with previously different distributions easier.

When considering the issues with absolute ranking systems, it is difficult to consider changes that could be applied to mitigate the accuracy and translatability issues. For example, a simple binary *good* / *bad* review system would be translatable between users but would not allow for any distinction between user preferences within those categories. An ordered set of movies

would objectively be the most accurate display of user preference, as each movie would occupy its own rating, but would not be possible using the current absolute systems. Relative ranking algorithms such as Elo and TrueSkill, can be used to convert many unique comparisons between movies to create an ordered set of movies and provide novel, normally distributed ratings between them.

1.2.3 Elo Rating System

The Elo rating system is a method for calculating the relative skill of different competitors in chess and other zero-sum games (Elo, 1967).

The Elo system uses the difference in ratings between two players as a prediction for the outcome of a match between them. Each player has a rating which is used to calculate an expected score of a match between the two – an expected score of 0.60 represents 60% chance of that player winning, 40% chance of drawing and 0% change of drawing.

If two player have ratings R_A and R_B, then the formula for the expected scores (E_A, E_B) for each are as follows.

$$E_A = \frac{1}{1 + 10^{(R_B - R_A)/400}}$$

$$E_B = \frac{1}{1 + 10^{(R_A - R_B)/400}}$$

Once a match is completed, the result is used to update both players ratings, with the winner's score increasing and the loser's decreasing. The K factor, denoted K is a figure used in the Elo algorithm to set the maximum possible skill rating gain / loss per game.

Following completion of a match players ratings are updated as follows.

$$R'_A = R_A + K \cdot (S_A - E_A)$$

Where R'_A is the players updated rating, R_A is the players previous rating, K is the K factor, E_A is the expected score defined previously and S_A is the points scored by the player (Elo, 1967).

The Elo system is incredibly powerful for judging the relative ability of players in two player zero sum games, due to its dynamic nature and ability to straightforwardly predict the outcome of a match based on difference of player ratings.

1.2.4 Applications of Elo Rating System Outside Traditional Usage

Despite its majority application being in chess, the Elo rating system has successfully been applied in many other scenarios.

Pelánek (2016, pg. 1) states that the 'Elo rating system is a simple, robust and effective tool for adaptive educational systems'. Adaptive educational systems are described as learning environments where data is tracked, and used, to provide personalized learning experiences tailored specifically for each student (Cavanagh et al., 2020). To cater to these environments a modified version of the Elo system is used, where instead of two unique players – both the students and questions have ratings on the same scale. Once the student attempts the question, the skill rating of the student and difficulty of the question are both updated according to how correct the answer is (instead of the points scored by the player in previous example). Pelánek notes the strengths of the Elo system, including the ease in adding a new item to the system – where the system will automatically learn and calibrate ratings of said items (Pelánek, 2016).

This is a particularly attractive property when considering an Elo systems potential application to movie ratings – where new movies are released regularly. When considering the annual increase in Rotten Tomatoes ratings explored in the previous section, new movies added would be calibrated against other movies present in the system. This could have one of two effects: calibrating new releases against older movies may cause a dampening effect on this inflation; or alternately further increase this inflation as reviewers may continue to favour newer releases – and thus in comparisons, reduce older ratings even further.

There are still problems with applying the Elo system, or any other relative rating algorithm to movie ratings. Many would argue that comparing movies of different genres, eras etc. is a trivial task – as all movies have different goals, all critics have different metrics and placing them all on the same scale is impossible. On the contrary, this is something that already happens. Whilst users on Letterboxd may have multiple movies reviewed 5 stars across different genres and eras, Letterboxd will still maintain a global ordering of these movies (Vis, 2024).

1.2.5 TrueSkill™

TrueSkill is a proprietary skill-based matchmaking algorithm developed by Microsoft research for usage in Xbox live competitive games. Unlike the Elo system, TrueSkill is natively designed for comparisons with more than two players.

In TrueSkill, a player's skill is composed of two variables, a mean value of μ and a variance of σ. μ refers to the players perceived skill and σ refers to how uncertain the system is in the aforementioned μ. TrueSkill represents these variables a normal distribution N where $N(x)$ is interpreted as the probability the players actual skill is x (Herbich and Graepel, 2006).

When a comparison between players is completed, all players' μ value is increased or decreased depending on the result, with σ updating based on the disparity of the players. If a player with a lower perceived skill wins against a player with a higher perceived skill, σ (the uncertainty in both players ratings) will increase as the result is contrary to what the TrueSkill system predicted (Moser, 2010).

TrueSkill has also been developed as a Python library under the BSD license, allowing for easy integration into Python development. Using the TrueSkill module in Python, new Rating objects can be easily created with default μ and σ values (of 25 and 8/3 respectively) or custom variables. Using these rating values, complex comparison / game formats can be constructed easily – with TrueSkill supporting many forms of comparison (N vs N team match, N player free-for-all etc.) (Lee, 2016).

Like the Elo system, TrueSkill requires a minimum number of comparisons to accurately estimate a player's skill. Game modes where the individual has minimal impact in a team (such as two teams with eight players per team) take considerably more matches to calibrate than individual free-for-all formats (Minka et al., 2005). When considering the Elo system and TrueSkill for the task of normalizing movie reviews, this is a considerable benefit.

1.2.6 Coupon Collector Problem

The coupon collector problem describes a hypothetical scenario that states: when selecting one item randomly from a set of n distinct items, how many selections m are required to have selected each item at least once (Motwani and Raghavan, 1995). This can be applied to our theoretical relative ranking system, where movies would need to be calibrated against each other until all movies are ordered.

Using the following estimation, we can calculate the number of comparisons required to calibrate all movies $E(T)$ for each system:

$$E(T) = n \cdot H_n$$

Where H_n is the n-th harmonic number as below (Sondow and Weisstein, 2024):

$$H_n = 1 + \frac{1}{2} + \frac{1}{3} + \cdots + \frac{1}{n} = \sum_{k=1}^{n} \frac{1}{k}$$

Assuming a user with 300 movies, we can approximate $E(T) \approx 1886$ trials to select each movie at least once. This approximation assumes we are only selecting one movie at a time, and that we do not need multiple comparisons to calibrate a movies ranking – however is useful for visualising how quickly number of trials will scale with more movies. Increasing the number of movies compared in a comparison, as well as ensuring movies that are considered calibrated cannot be selected until all are calibrated will decrease the number of comparisons required. TrueSkill's ability to compare more than two movies at once, combined with the increased rate at which it can calibrate new movies, make it objectively faster than Elo for our intended purpose.

1.2.7 Applications of TrueSkill™ Outside Video Games

Like the Elo system, TrueSkill has also been used outside of the competitive games environment. Sakaguchi et al. (2014) used a modified version of TrueSkill to order the human evaluation of machine translations, stating TrueSkill 'outperforms other recently proposed models on accuracy' and 'significantly reduces the number of pair wise annotations required'. TrueSkill was also tested with 5-way free-for-all matches for comparing the machine translation, finding that 5-way comparisons were advantageous for lowering the number of comparisons required – whilst keeping the same accuracy. This shows the utility of TrueSkill for creating accurate absolute rankings from relative comparisons and helps to identify and understand TrueSkill's strengths when applied to normalizing movies reviews.

When considering the minimum 5 comparisons required to calibrate an item in TrueSkill, this has two important outcomes. If a user has already calibrated all their movies and is simply adding a new movie to calibrate, only 5 comparisons will be required to calibrate said movie. However, to calibrate all their imported movies they will need to complete as many calibrations as movies imported (5 movies in a comparison, 5 comparisons per movie). For most users this would amount to over 300 comparisons and identifies an issue with the comparisons required to get the system accurately updating ratings. One potential solution to this would be to lower the number of comparisons required, instead focusing on faster comparisons rather than more accurate comparisons.

Another issue with using TrueSkill to aggregate relative rankings, comes from the competitive design of TrueSkill. In a competitive video game environment if a player wins 10 games in a row against the same 4 opponents, their skill rating will increase after every one of these games. This is not ideal for a relative ranking system for movies, as movies with more

comparisons may be skewed to be positive due to the number of comparisons. If a user imports only 5 movies – all comparisons will be the same and each subsequent comparison will increase or decrease the movies ratings despite this exact comparison already being appropriately factored in the first comparison.

1.3 Project Objectives

Objective	Description
Efficient Web Scraping	To provide movies for the user to compare, we will extract the data using web scraping. This web scraping module must be efficient, ethical and have minimized usage.
Comparison Web Application	To allow users to complete comparisons of their movies, we will provide a web-based application. This application must be responsive, secure, and scalable with more users.
Accurate Chain of Comparisons	To conduct complex investigative analysis, it is not enough to purely have the database of ratings after the user testing. Storing the data as a chain of comparisons allows us to not only recreate a final rating for all movies with custom parameters, but also test different relative ranking algorithms on this chain of comparisons.
Data Analysis of Results	To further understand the utility of TrueSkill as a relative ranking system for movies, we must investigate it's results and compare to the current absolute methods.

Chapter 2
Methods

This section discusses the design of a novel system to investigate the utility of using relative ranking algorithms for creating movie rankings.

2.1 Solution Definition

The data collection platform aims to collect user comparisons to investigate the viability of relative ranking systems for ordering media. Users can create an account, import their logged movies from their Letterboxd username and compare them relatively. These relative comparisons update a TrueSkill rating for the user's profile, and the global rating of the movie. Users can view all their movie ratings, other users' ratings and the global rating. Further pages will include information on movies and options to add / remove movies for comparison. All comparisons from all users will be collected and further analysed in this report.

The platform aims to be highly intuitive, responsive and modern. Primary user testing will be used to ensure the data collection platform is intuitive and responsive. Industry standard practises and frameworks will be utilised to ensure the platform obeys modern web development standards.

2.1.1 Application Design

The data collection platform will be developed as a modern web application comprised of three layers: presentation, application and data. Tiered architecture, most commonly three-tier on N-tier, is an industry standard way of designing and implementing a web application. Separation of data, logic and presentation components allow for greater scalability and modularisation in modern applications.

The data layer, a SQL Database, is responsible for handling all persistent data accessed or stored in the system. Further information on the models and schema of this database can be found at **2.1.2 Application Models**.

All complex logic and communication with the data layer will be handled by a Flask API backend. This application layer acts both as a connecting layer between the presentation and data layers, and as a handler for all complex logic present in the system.

The data collection platform is accessible through a React web application that functions as our presentation layer. Users interact directly with this layer, with React calling the necessary API routes of the backend.

Figure 2 describes the tiered approach used to develop the modern data collection application.

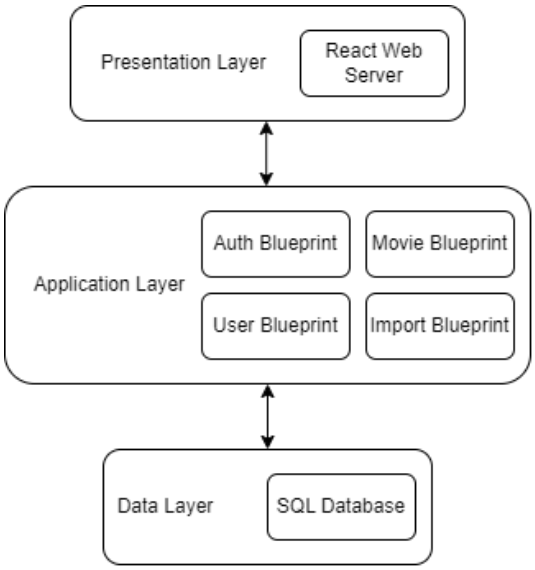

Figure 2 – Architecture Tier Design

2.1.2 Application Models

As data is transferred between multiple components in this project, we must define models to ensure consistency when communicating between different components.

These models will be defined both in the SQL database and Flask SQL Alchemy ORM, with any POST requests from React requiring custom serialization.

2.1.2.1 User Model

The User Model **(Appendix C - Figure 8)** stores all information relating to users created on the data collection platform. User ID and Username are both required to be unique, with User ID autoincrementing and serving as the primary key for the model. All passwords are stored hashed using Bcrypt.

2.1.2.2 Movie Model

The Movie Model **(Appendix C - Figure 9)** stores all information relating to movies imported from Letterboxd on the data collection platform. Movie ID and Title are unique, with Movie ID autoincrementing and acting as primary key for the model. Film URL Pattern is the string used for the sub-route of the movie on Letterboxd (for example, *The Godfather* is the-godfather), and is used for both the movie's individual page and querying TMDB (The Movie Database). It's worth noting for movies that share names Letterboxd will append the year to this string –

making it incredibly useful for referring to movies where titles are identical. Image Link refers to the movie poster link hosted on TMDB. All other figures are either imported from the Letterboxd movie page or used during comparisons.

2.1.2.3 User Movie Model

The User Movie Model **(Appendix C - Figure 10)** is used to store information on the movies a user has imported from Letterboxd. Once a movie is imported, a User Movie is created between the Movie object and the User object – and then used to store all information on comparisons that user completes including that movie.

2.1.2.4 Comparison Model

The Comparison Model **(Appendix C - Figure 11)** stores information on a single comparison completed by a user. It contains foreign keys to the user and all 5 movies in order (Movie 1 is rated first / best). The timestamp ensures that the comparison table can be used to reproduce an exact chain of comparisons, as they took place, for data analysis and dataset creation / replication. Other information such as the number of movies a user has imported and the number of movies they have chose to compare to aid analysis later.

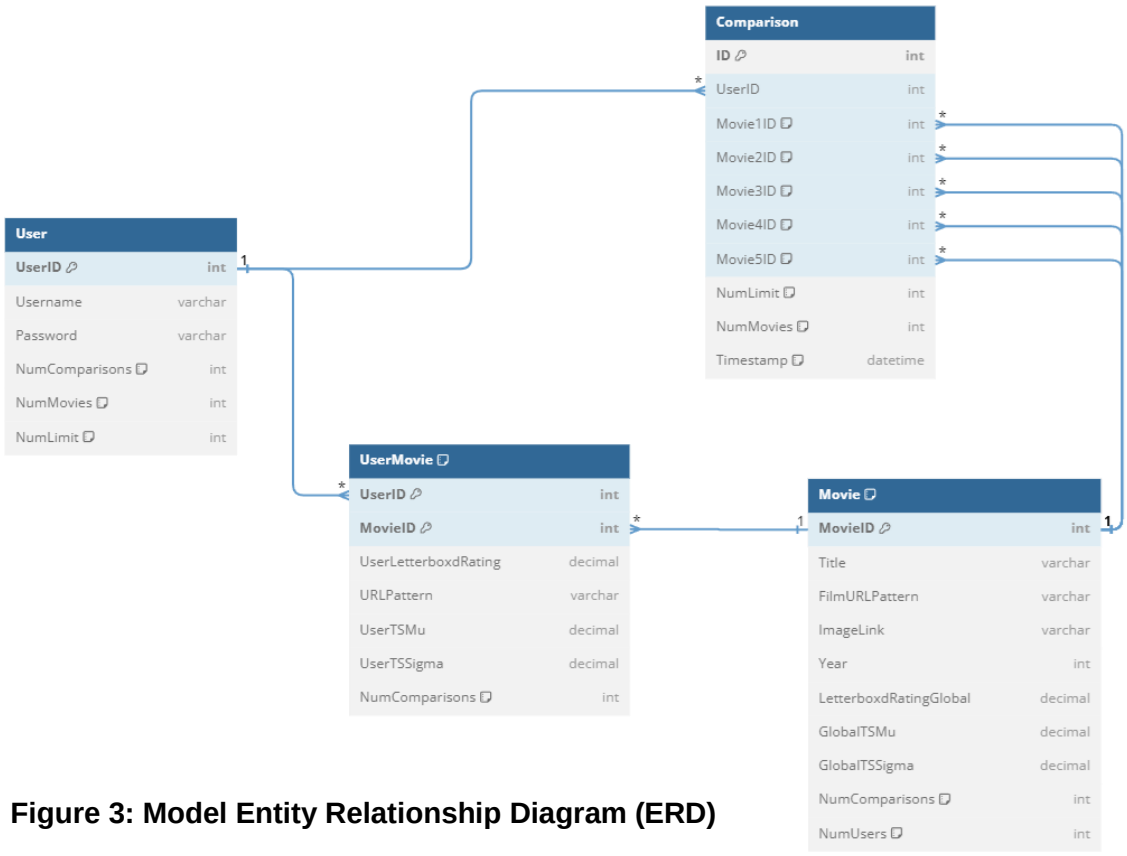

Figure 3: Model Entity Relationship Diagram (ERD)

2.1.2.5 Entity Relationship Diagram

The relationships between these models can be seen in **Figure 3**. Both the Azure SQL Server database and Flask SQL Alchemy ORM require this schema to be always followed.

2.2 Flask API Application Layer

Flask is a synchronous WSGI web framework, primarily used for creating APIs.

In this project, a Flask API server is used as the application layer – housing all application logic and computation in the system. All authentication, communication with the database and logic happen in this Flask server. Flask's small and lightweight design is perfect for quickly developing simple applications, whilst also housing the tools required for larger, complex applications.

All Flask applications function as a set of routes, accessed through HTTP that execute different code. Specification of routes required in the Flask API can be found in **Appendix D – D1 Flask API Specification**.

2.2.1 Flask Blueprints

Flask Blueprints are a tool used to compartmentalize these routes. Each blueprint has a specific sub route, for example *movie* or *auth*, with each route in that blueprint appended afterwards. This project will be comprised of 5 blueprints, split up for simplification of the API.

2.2.2 JSON Web Tokens

JWTs (or JSON Web Token's) are a common way of authenticating users and sessions. JWT's are Base 64 strings created by the backend upon authentication that are returned to the frontend, either to be stored as session cookies or as a header in any subsequent requests to the backend.

For this project, we opt to use HTTP-Only Secure JWT tokens. This means the tokens are not accessible by JavaScript running on the presentation layer and are only sent in HTTPS requests. Further information on the authentication protocol used in the data collection platform can be found at **3.1.1.4 Authentication**.

2.3 React Presentation Layer

React is an open-source JavaScript library used for building web applications out of components. This project uses React as the frontend webserver hosting the presentation layer, and therefore all user interaction. Vite was used for the development engine due to its instant server starting and quick reloads during development.

2.3.1 React Router

React Router will be used to navigate between different pages on the web application. All accessible information will be split into separate pages, with elements that appear frequently, or across multiple pages, developed separately as components – to best utilize React's modular strengths.

2.3.2 Contexts

Contexts are used to store information that remains persistent outside of the router. For example, when switching pages the user's choice of dark or light mode must remain consistent between pages. A context exists outside of the routers and call be get or set from these pages inside the router, allowing for persistence of information.

2.3.3 Cookies

Cookies will be used minimally in this project. To ensure security of authentication, cookies are stored in the browser as access tokens. The users chosen theme are also stored in the local browser to ensure consistency between refreshes, or different sessions.

2.4 Web Scraping

2.4.1 Beautiful Soup

Beautiful Soup (or BS4) is a Python library used for extracting information out of HTML files. This can be used in conjunction with the Requests library to query a web page and parse its results – extracting certain information.

2.4.2 Selenium

Selenium is a Python library used for automating actions and retrieving information on web pages using a web driver. Unlike BS4, Selenium requires a web driver and opens a web page when executing. As Selenium opens the web page using a web driver, all JavaScript is loaded on the site – whereas BS4 returns all HTML prior to JavaScript being loaded.

2.4.3 Web Scraping Implementation

Both Selenium and BS4 can be used to web scrape effectively. Selenium is more powerful, allowing for automation of actions. Selenium accesses the site identically to a user accessing through a web browsers development tool, making it simple to find elements to scrape.

When considering web scraping for this project, two vital considerations were made. Web scraping must be ethical, only accessing publicly available and exportable information, and efficient, minimizing requests as much as possible to not overload servers.

Selenium was used to populate the original movie table in the database. This process is only executed once, locally on the development machine. The benefit to Selenium over BS4 here is that all fields required for a movie can be scraped from the list of movies page. Some fields such as movie rating on this page are loaded after JavaScript, meaning BS4 cannot access them and would require a different request for each page. In this case Selenium requires 71 less requests than BS4 per page scrape.

BS4 is used for scraping a user's logged movies from Letterboxd, this requires one initial scrape of the user's profile to find how many movies the user has logged. As each list of movies contains 72 movies per page, the number of scrapes per import is 1 + (number of movies / 72) rounded up.

2.4.4 Python Global Interpreter Lock

Python uses a Global Interpreter Lock (GIL) to lock only one thread to executing the Python interpreter at any one time. This is used to ensure multiple threads cannot read, write, or modify data at the same time to avoid data corruption. As the web scraping of user information accesses a different page each requests, we can safely optimise this by multithreading the requests – requesting multiple pages at once instead of all pages sequentially on one thread. A Thread Pool Executor is used to launch this scraping parallelized, dividing the pages to request into multiple queues to be executed by different threads simultaneously. After implementation this took the time to import an account with 1,700 movies from 8 minutes to under 8 seconds.

2.5 Data Collection

All comparisons performed on the data collection platform are stored in the database. The Comparison object, shown in **Appendix C - Figure 11**, stores the users, all 5 movies in order, information on the user and the timestamp of the comparison. These timestamps can be used to recreate an accurate chain of all comparisons completed on the data collection application, allowing for more complex analysis over just analysing the results of the final database.

The Number Limit of movies was implemented as an attempt to mitigate the issues discussed in **1.2.6 Coupon Collector Problem,** allowing users with many movies to focus on calibrating a subset instead of partially calibrating all movies. This will limit the number of movies calibrated in the final dataset but will ideally mean the movies included are more accurate.

Number Limit does however have a downside of allowing users to artificially inflate a movies rating by limiting to a small subset of movies and repeating multiple comparisons between these movies. Any comparisons after the initial calibration comparisons will still update the movies compared, meaning repeat comparisons of the same 5 movies will increase the rating

of the best and worst movies with all repeat comparisons. All analysis must only use the first 5 comparisons of each movie for each user to negate this.

The data collection application uses the global Letterboxd rating to show as information to the user. This should not be used during any data analysis as it is not representative of our user sample – instead all comparative analysis must be using aggregated Letterboxd ratings from our users.

2.6 Project Methodology

2.6.1 Version Control

The software version control system used for management of the data collection platforms development is Git, hosted on GitHub. Version control was used to allow the student to develop on multiple devices, track and modify changes and work on multiple sections at once. GitHub was specifically chosen due to its ease and ability to develop CI / CD (Continuous Integration / Continuous Deployment) integrations with Azure services.

GitHub actions is a tool used for automating software workflows, primarily CI / CD. After pushing to the *master* branch, both the React and Flask applications are automatically deployed on Azure by GitHub Actions. Further information can be found on the CI / CD platform used in deployment at **3.1.2.4 CI / CD**.

2.6.2 Scrum Project Management

The development of the data collection application was carried out in accordance with industry standard Scrum methodology where all development is carried out in distinct sprints, with all features and tasks split up into tickets to aid development (Srinivasan and Maloney, 2024).

Each ticket was assigned a sprint, with tickets not completed in their assigned sprint moved forward to the next sprint. This allows for manageable goals to be set, spreading work evenly across the entire development windows – and accounting for any issues or setbacks that may occur during development.

All development was planned out and organised in a Gantt chart prior to beginning development and can be found in Appendix C - Figure 6. This Gantt chart does not exactly match the development timeline for the project as significant time was taken to develop a more complex version of the movie importing function present in the system – this is discussed further at **3.1.1.2 User Movie Import Function**. A retrospective Gantt chart has been included, showing the actual time spent on each section and feature, in Appendix C - Figure 7.

2.6.2.1 Kanban Board

A Kanban board was used to assist development of the data collection application. All features and development items were split up into smaller *tickets*, with a manageable number of tickets assigned to each sprint, allowing for flexible and pragmatic development.

The Kanban board was hosted on Trello, a free web-based Kanban board application, with 6 distinct categories for tickets. The 6 categories present on the Kanban are as follows: backlog, sprint planning, current sprint, in development, development complete and testing complete. Items are not required to move linearly through the Kanban board and can be moved backwards (an example would be a development complete item failing during the testing stage).

2.6.3 Testing

Due to the complexity of multiple software layers interacting with each other, testing was required at all stages of development. An automated testing suite was developed in Postman. Postman is an application used for testing web APIs, assisting users to create requests and repeat them. This testing suite is a list of all possible routes that can be executed on the backend. Once the testing suite is run, all routes are called sequentially – testing all possible interactions with the backend and asserting certain values (response code, movie id etc.).

Appendix C - Figure 12 is an example of one of these automated Postman test routes, developed in JavaScript. In this example, a movie is queried using the */movie/<URLPattern>* GET route with a correct URL Pattern. The tests are written in JavaScript and ensure that the response is a 200-success code and contains 8807 – the Movie ID of Barbie. These tests cover all possible outcomes of all routes including 500 execution errors, incorrect query parameters and unauthorised routes. This testing suite is run prior to any backend changes being merge into the remote master repository, to ensure all new code functions as intended.

After the application was deployed, full user testing was completed by a set of user testers. These testers were taken through a walkthrough of the application, noting any bugs or confusions they found on the website. Further information on the user testing stage can be found in **3.2 Primary User Testing**.

Chapter 3
Results

3.1 Project Implementation

All features described in **Chapter 2 Methods** have been implemented. A detailed specification of the final developed Flask API can be found at **Appendix D – D1.**

3.1.1 Implementation Decisions

3.1.1.1 TrueSkill™ over Elo

When originally developing the data collection platform, Elo was implemented as the proof-of-concept rating algorithm. This meant all movies comparisons were comprised of two movies. This worked for initial testing but was found to require an unfeasible number of comparisons to calibrate all movies imported. An ordered Elo set was created using pairwise comparisons generated from the submitted TrueSkill comparisons and is discussed at **3.3.4 Data Analysis**.

This was updated to implement 4 movies in a comparison, using TrueSkill to update both user and global ratings for movies in the comparison. This was modelled as a 4-player free-for-all with the movie in the first ranking being considered 1st and the last movie considered last.

Prior to data gathering, comparisons were updated to contain 5 movies. This increased the speed at which users can calibrate their movies, whilst also providing a useful *middle / average* movie in the 3rd place spot. 5 movies per comparison also simplifies the calculation required to find the number of comparisons required to calibrate a set number of movies. As users compare 5 movies per comparisons and must compare each movie 5 times to calibrate it, the number of comparisons required is equal to the number of movies on an account (given movies greater than or equal to 5 comparisons cannot be selected till all are calibrated). This simple calculation is used on the user's profile page to show the user's progress towards calibrating all their movies.

3.1.1.2 User Movie Import Function

The user Letterboxd username import function was heavily changed during development, as seen in **Appendix C - Figure 13**. Originally, when a user enters their username, all movies were scraped from their profile – and any not found present in the system will be scraped and added to the system. This was designed to ensure users could compare all movies they have logged on Letterboxd, providing greater coverage of users with relatively unpopular movies.

The student was aware that this would be considerably more computationally intensive, given the 4 additional web requests required per movies not found in the database, however initial

testing found it unfeasibly slow. It was assumed as more users import – and thus generate – movies that most popular movies would already be present in the system, and each import following would require less imports. As this function only returns a response once all movies were either added or skipped, users with >1000 movies would have to wait multiple minutes to import. Given that this original design was not only efficient, but also unnecessarily complex, development shifted to a providing a predefined set of movies.

Appendix C - Figure 14 describes the current function for initially populating the database with movie objects. It scrapes 250 pages of Letterboxd's most popular movies, populating the database with 17083 movies. This reduces the time required to import a user's logged movies, as no movies are created; and concentrates the user comparisons across a smaller subset of movies, potentially aiding future analysis.

3.1.1.3 CORS

Cross-Origin Resource Sharing (or CORS) is a mechanism that allows web applications to make requests to domains other than the one originally requested by the user. CORS enables web applications to specify which domains are allowed to access their resources, and thus prevent unwanted cross-origin interaction.

In the data collection application, the Flask backend API is configured to only receive cross-origin requests from the React frontend server. This is accessed through the frontend's deployed URL, with all responses from the backend containing the required headers. These URLs are updated depending on whether the project is run locally (development environment) or on deployment. More information on environments can be found at **3.1.2.3 Environments**.

3.1.1.4 Authentication

In the data collection application, user and session authentication is handled through the Auth blueprint. All user accounts information is stored in the database, with passwords hashed using Bcrypt. When a login in request is made, if both the username and password are correct, a JWT HTTP-Only Secure access token is returned in the response.

HTTP-Only means this cookie cannot be accessed by any JavaScript running in the browser, with Secure meaning cookies are only sent over HTTPS (where all HTTP requests are encrypted). After this cookie is returned, all requests from the frontend will contain the header *withCredentials* set to *True.* This means all requests will contain the user's authentication credentials, including the access token mentioned before.

These JWT tokens have an automatic time out period, set at 1 hour, where any requests after the time out period will require reauthentication by the user. The backend callback function is executed after every request, checking for a JWT expiring in the next 30 minutes, and creating

a new one if the supplied one is close to expiring. This is designed so users can extend their session authentication and do not have to log in every hour.

The Authentication Context in the frontend contains a method for requesting */auth/validate,* setting the logged in user if one is found. This context is requested from the custom Protected Route component used for certain routes in the frontend. Once a protected route is accessed, the auth context ensures that the current user is authenticated and allows them to access – also refreshing the expiry time on the user's JWT if it is >30 minutes time to expiry. If auth context finds a user is not authenticated, the protected route will instead redirect to the login page.

3.1.2 Deployment Approach

As the data application platform requires users to complete comparisons, an always online hosted deployment approach was favoured. This allows users to complete their comparisons in their own time, on their own machine and anywhere in the world.

3.1.2.1 Azure Static Web Apps

Azure Static Web Apps is a service that builds and hosts web applications on Azure. This was chosen as the hosting platform for the frontend React presentation layer due to its simplicity to deploy, and the students experience working with Azure previously.

A custom domain, Cinerank.xyz, was acquired from a registration provider and set as an alias for the data collection platform. Users can access the data collection platform's presentation layer through this URL, with all communication encrypted using a provided SSL Certificate. This domain was selected to provide a simple, short referable name for users to remember and access the website through.

3.1.2.2 Azure Serverless Functions

Functions as a Service (or FaaS) is a modern cloud computing service that allows functions to be hosted on the cloud and called in response to events, bypassing the management of infrastructure required for cloud applications (Roberts, 2018). Serverless functions on Azure automatically scale based on demand and are only charged per usage (unlike monthly costs for virtual machines / hosting), making it scalable and affordable for the data collection platform.

This Flask server is wrapped in an Azure Serverless Function, allowing for effective scaling during peak times. Originally, the import function seen in **Appendix C - Figure 13** was planned to be a separate micro service – allowing users to compare a subset of their movies whilst the

rest are asynchronously imported. This idea was reverted when the adding of movies to the database was made manual, as importing no longer took infeasible amounts of time.

Azure App Service is Azure's cloud-based platform for hosting custom and complex web applications on the cloud. This service is designed to work simply with Flask, but after consideration was found to be more expensive and less scalable than using Serverless Functions for the backend (Azure offers 1 million function free requests for new accounts).

3.1.2.3 Environments

When running both the Flask backend and React frontend on a local machine, CORS parameters and request URLs are must be updated to communicate with each other across the local machine host. However, when both are deployed on the cloud the CORS parameters and URLs are different – requiring manual updates or environments.

Two different environments, development and production, were used to aid development. All requests from the React presentation layer are called with a base URL accessed from the relevant *.env* (*local* or *production*) file, selected dynamically when running the application in deployment or debug mode using npm. All instances of the URL or CORS parameters in the Flask application layer are stored and accessed as environment variables, set in the local console or in application settings on Azure. Confidential parameters and settings such as database connection URL and application key are also set as environment variables to obscure them from potential attackers.

3.1.2.4 CI / CD

Continuous integration and continuous development / deployment (or CI / CD) describes the action of automated integration and simultaneous deployment of code changes.

GitHub Actions is a CI / CD platform that connects automatically both Azure Serverless Functions and Azure Static Web Apps to a GitHub repository. When the student pushes code changes to the remote master branch, these changes are automatically integrated and deployed to both the React frontend and Flask backend. These integration and deployment procedures and defined by *yml* files found in the GitHub workflows section of the repository.

3.2 Primary User Testing

3.2.1 Methodology

Once all work items were development complete and passed integration testing, a primary round of user testing was completed. Users were given a path to follow through the application, rating each page on usability, making comments of confusing aspects and noting any bugs or

unexpected behaviour. The exact structure of the user testing questionnaire can be found in **Appendix D – D3 Cinerank Ease of Use Survey.**

Most feedback from the user testing was positive, with some pages requiring minimal or no change. Following this, all user feedback was added to the backlog, developed and rigorously tested before the same users were able to begin submitting their final comparisons.

3.2.2 Outcomes

As users will always first interact with the log in or register pages, an appropriate error notification was added to these pages if an operation failed due to the database being asleep – prompting the user to try again in a few minutes once the database has been reactivated. User's reported confusion with the email field being present on registration but not usable to log in. The email field was an artifact from a planned reset password feature, before the scope of the application was reduced. This field was removed to streamline the registration process and minimize user data kept on the system. A toggle button was added to allow users to view their password as they enter it, a requested addition from the survey.

When importing a Letterboxd username, users found the feedback confusing in some cases. Three special responses were added to rectify this: an appropriate error message if the username is incorrect; a special response if the logged in user has already imported all movies on the associated username (no new movies to import); and a special case for importing one movie where previously the plural term 'movies' was used.

Many users reported confusion with the movie comparison page. Movie ranking direction being left to right (best on the left, descending) was not intuitive and the ability to drag movies was not made apparent to new users. This was rectified with clear text explaining the ordering, and how to move the movies into order. Some users misunderstood the application, hoping it would have more information on each movie – acting more as a movie discovery site rather than a data collection platform. This was originally not intended for the site, however a link to the Letterboxd page of a movie was added to each movie page.

A numerical input field was added to the limit slider in settings, to allow for more accurate input. Previously developed settings options were reworded to be clearer, and a new option to reset imported movies was added, following user request. Other fixes such as spelling errors, unclear error messages and inconsistent use of movie and film were also rectified following this testing.

3.3 Data Analysis

3.3.1 Defining Objectives and Questions

As discussed in **1.2.1 Flaws of Absolute Rating Systems**, the current absolute systems employed for ranking and rating media are not perfect. The following analysis aims to investigate whether a novel TrueSkill relative ranking system can accurately categorise user preference around a movie.

Our system must be ***consistent***, providing clear and under stable distinction between items and their minimum and maximum rating.

Our system must be ***translatable*** between users. In a traditional absolute rating system, it is infeasible to compare media ratings between different users as their distribution and definition of the scale may be different. In a rating system where all user's ratings occupy a normal distribution with a mean at the midpoint, a movie 2.5 stars occupies the same percentile across all users – indicating it is the *average* movie seen by said user.

Our system must be ***accurate***, allowing for high granularity and distinction between movies of similar user preference. When considering Letterboxd's scale, there is no differentiation between a users' 5-star movies. TrueSkill's larger range and accurate calculation of these ratings help ensure all movies are not only ordered but also assigned an accurate skill rating.

3.3.2 Data Collection

Over the two-week data gathering period, 12 users performed 2600 comparisons across 1743 distinct movies. Of these 1743 movies, 1061 received more than 5 comparisons and are therefore considered calibrated by TrueSkill. Only the 784 movies that are calibrated using comparisons from more than two users are used for further data analysis, negating the issue of some movies being rated inconsistently due to one user's personal preference

Number of users	**12**
Movies compared at least once	**1743**
Movies globally calibrated	**1061**
Movies globally calibrated across multiple users	**784**

Using the timestamped chain of comparisons, a final database state was constructed – with global ratings and rankings for all movies.

Comparisons submitted	**2600**
Comparisons containing movies that are globally calibrated	**2202**

Comparisons containing movies that are globally calibrated across multiple users	**1208**

In an ideal data set, all movies would be compared multiple times by multiple users to ensure the TrueSkill rating reflects the entire userbase.

3.3.3 Data Cleaning and Preparation

All movie comparisons of form ($A > B > C > D > E$) are split into 10 distinct pairwise comparisons, where A > all other movies, B > all movies except A etc.

When comparing the users' pairwise comparisons in for movies A and B, where $A > B$, there are a few assumptions we must make:

1. No two movies can be rated the exact same. In a pairwise comparison, there is a distinct positive preference for A over B by this user.
2. Ideally, all comparisons form a larger transitive relationship. A group of movie comparisons are transitive if $A > B$, $B > C$ and therefore $C > A$ – thus ensuring consistency between all pairwise comparisons and ordering.
3. A pairwise movie comparison $A > B$, describes the related user's preference - regardless of outside context, timescale or otherwise indicated absolute rating.

All analysis was developed and executed using a Jupyter notebook, allowing for sequential development and dynamic execution of analyses functions.

These 1208 5-movie comparisons are then split into 12080 pairwise comparisons to find and categorize the number of intransitive cycles and direct contradictions submitted by users.

Pairwise comparisons	**12080**
Pairwise contradictions	**32**
Pairwise intransitive cycles	**40**

Pairwise contradictions are defined as unique instances where in one comparison a user rates a movie A higher than B, and then rates B higher than A in another comparison – thus *contradicting* their initial comparison. For analysis of these pairwise comparisons going forwards, all contradictions will not be removed from the dataset. As both contradictory comparisons (A > B, B > A) are submitted by the user it would be impossible, without further discussion with said users, to determine their actual preference between the related movies. Considerations were made to check for which of the comparisons is more common, however almost all contradictions are made equal times in either direction – indicating user error / small differences in preference between both movies.

An intransitive pairwise cycle refers to a relationship between movies A, B and C where in any number of comparisons containing these movies a circular relationship forms – such as A > B > C > A. This is contradictory as we would assume A > C. Due to the low number of intransitive cycles present across the pairwise comparisons, we would assume this is user error rather than a disproof of the expected transitive property of user preference. The comparisons that contain cycles are not removed from the dataset as they are uncommon and difficult to remove without removing multiple 5 movies comparisons from the dataset. The most common movie *Batman Begins* appears in 4 distinct cycles, with most movies in the 40 cycles only appearing in one each. Due to the limited amount of data collected, priority was placed on maintaining as much usable data as possible – if more data was collected for comparison, it could be worth analysing the set without these cycles or performing analysis on these cycles themselves.

Global Letterboxd ratings, where the score is aggregated from all users on the platform, are used for quick feedback to the user on the data collection platform – but are not accurate or indicative of the absolute sentiment of the test group. For each of these 784 movies, a new absolute Letterboxd rating is created by averaging the user ratings of all users who were involved in calibrating that movie.

3.3.4 Data Analysis

Using the aggregated user Letterboxd ratings and TrueSkill ratings calculated using the pairwise comparisons for each movie, we can create two ordered sets – both acting as our final states for absolute and relative user preference.

A third ordered set of ratings was created using the Elo system using the pairwise comparisons. This set's distribution is visible in **Figure 5**, however due to time constraints TrueSkill was preferred over Elo for comparison with Letterboxd. On average, each movie in the Elo ordered set was 135 places away from the Letterboxd average ratings set – with a lower correlation coefficient than TrueSkill of 0.72. This does not disprove that an Elo rating system constructed on the pairwise comparisons could be viable, but instead that effort was placed on analysing TrueSkill due to its initial strengths and time constraints.

The average absolute position difference between movies in both sets is 101.25 places. This means that on average a movie in position X in our results, will be *on average* $\pm$ 101 positions away in the Letterboxd absolute ratings set. With a sample of 784 movies, 101 positions may seem like an extreme figure but one must consider the flaws of the absolute set we are referring to. 77 movies occupy the Letterboxd 3.5 – 3.99-star range, meaning any movie rated 0.5 points higher compared to the absolute set will be at least 77 positions away. As the absolute ratings are calculated and aggregated from such a small set of users, many movies

occupy very similar categories – meaning any meaningful distinction TrueSkill provides between these movies will cause a larger position difference.

Spearman's rank correlation coefficient is a non-parametric measure of rank correlation, used in this example to understand and assess the relationship between our TrueSkill ordered set and Letterboxd aggregated rating set. Spearman's returns two important values: the correlation coefficient that indicates the strength of a relationship between the two sets (+1 being strong positive, -1 being strong negative, with 0 suggesting no linear correlation); and a P-value to determine the statistical significance of this correlation coefficient (P > 0.05 indicates the correlation is statistically significant) (Spearman, 1904). The correlation coefficient between our both of our sets is 0.81, indicating a strong positive correlation, with a P-value of $1.37 \cdot 10^{-186}$ – an incredibly small number that indicates our correlation coefficient is statistically significant.

On the data collection platform, all TrueSkill ratings are shown as a 0.5 – 5 rating, calculated by mapping all ratings between the highest and lowest ratings. This was designed to make comparing between TrueSkill mu ratings and aggregated Letterboxd ratings simpler for users and is used in the charts and tables below.

The aggregated Letterboxd ratings provided by the test user base demonstrate the mean and median rating around 7 phenomena discussed in **1.2.2 Flaws of Absolute Rating Systems**. This underutilizes the 0.5 – 5-star scale and is visible in that no movie in the aggregated Letterboxd set is rating 0.5 stars. TrueSkill however occupies the entire scale, with movies at 0.5 and 5, with a mean and median at 2.73 – meaning the entire scale is intuitive and utilized.

	Letterboxd Ordered Set	TrueSkill Ordered Set	TrueSkill Mapped Ordered Set	Elo Ordered Set	Elo Mapped Ordered Set
Mean	**3.46**	**24.7**	**2.73**	**1000**	**2.58**
Median	**3.50**	**7.29**	**2.73**	**999**	**2.58**
Standard Deviation	**0.71**	**7.29**	**0.76**	**63.4**	**0.63**
5th Percentile	**2.25**	**12.5**	**1.45**	**906**	**1.67**
95th Percentile	**4.50**	**36.3**	**3.94**	**1120**	**3.74**

3.3.5 Data Visualization

The below scatter plot **Figure 4** shows all 784 calibrated movies in the database, plotted according to their Letterboxd aggregated rating and TrueSkill relative rating. The red line is an Ordinary Least Squares (OLS) regression trendline of form $L = 0.77 * T + 1.37$, where L is the aggregated Letterboxd rating and T is the relative TrueSkill rating. This line predicts that a movie's Letterboxd rating will always be greater than the TrueSkill rating, again visible in the mean and median values for both distributions. The black line below intersects the graph along $y = x$, meaning any plot below is higher rated on TrueSkill than Letterboxd, and the reverse for above. This majority of movies occupy the space above this line, meaning that most movies are overrated on Letterboxd when compared to the relative TrueSkill ratings. This is expected behaviour given the disparity in mean and median values for both sets.

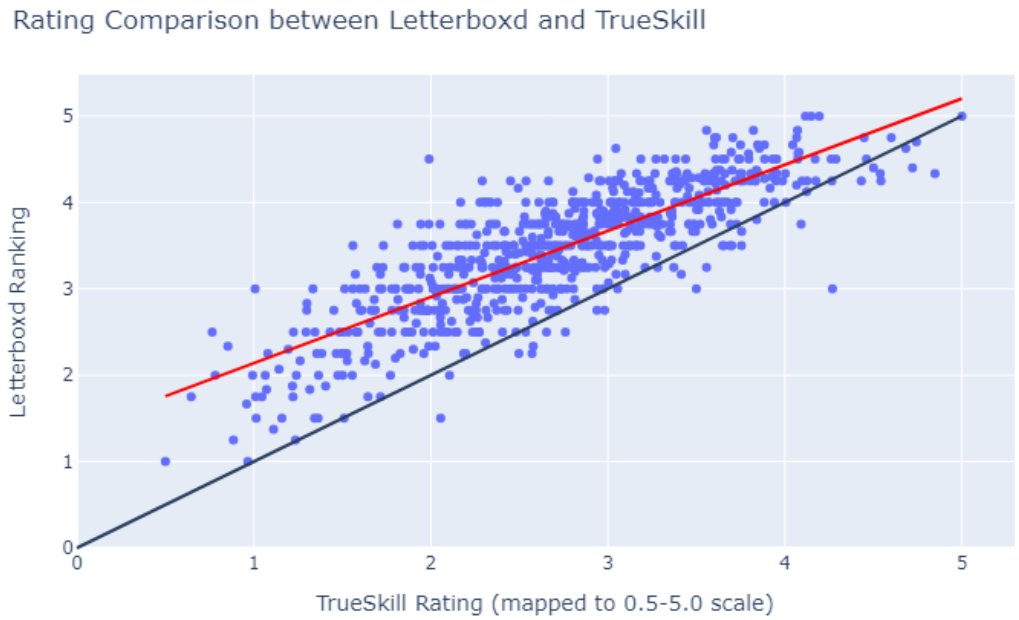

Figure 4 – Scatter Plot of Aggregated Letterboxd Ratings against TrueSkill Mapped Ratings

Figure 5 is a bar plot of the distributions of the 784 aggregated Letterboxd user ratings, TrueSkill relative ratings and Elo relative ratings. The large spikes in the Letterboxd distribution are mostly a symptom of the aggregation process, where most movies only have 1-3 absolute ratings to average. This chart helps to visualize the distribution differences between Letterboxd, TrueSkill and Elo. The aggregated Letterboxd ratings mirror the issues described and discussed in **Figure 1**, most notably the skewed mean at 3.40 and taller distribution – with only 5% of movies within the range 0-2.25.

Whilst having the closest mean and median to the midpoint, 90% of movies in the Elo ordered set occupy 41.4% of the scale (49.8%, 45% for TrueSkill and Letterboxd respectively). This is an undesirable trait as it makes it harder for users to distinguish between similarly rated movies.

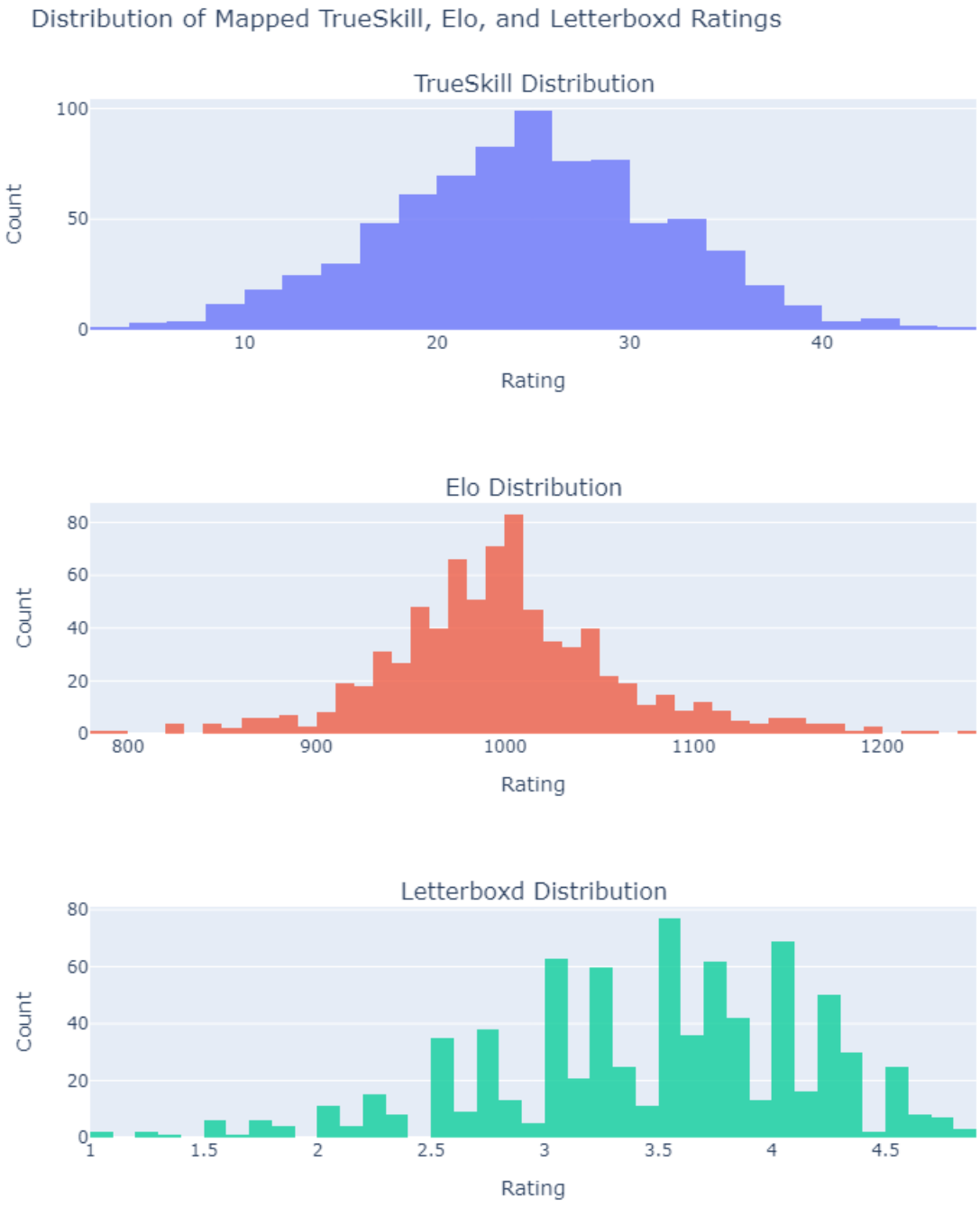

Figure 5 – Bar Plot of the Rating Distribution of all 784 Movies on Elo, TrueSkill and Aggregated Letterboxd Ratings

Chapter 4
Discussion

4.1 Conclusions

When originally discussing current absolute ranking systems and their issues, three main criteria were developed for a novel rating system. This new system must be ***accurate***, ***translatable*** and ***consistent***.

Accuracy refers to how well the rating system matches the users actual expressed preferences, especially when factoring in aggregated global user ratings. This experiment demonstrates successfully that when compared to the test base's aggregated absolute reviews, TrueSkill accurately calculates statistically significant correlated results (evidenced by the Spearman rank correlation). This proves the ideal and optimal behaviour of movies that would be highly ranked on an absolute scale are also highly ranked on our novel relative scale.

Translatability refers to how well a user's distribution of ratings can be compared against another's distribution to determine information about either's movie preferences. As our novel system tends towards a mean of 2.5, with a user's favourite and least favourite movie being 0.5 and 5 respectively, a movie's rating now correctly describes its preference relative to a user's favourite movie, least favourite movie and *average* movie. This however may become intuitive when comparing users with a large disparity in movies watched and their preference between those movies.

Consistent is used to describe systems where the tools or metrics used to rate a piece of media are always the same for all users. TrueSkill returns consistent ratings for all users – always occupying a normal distribution of ratings between 0 – 50 Mu rating (mapped to 0.5 – 5 stars) with a mean and median of around 25. However, one could argue that the way comparisons are created in TrueSkill is not consistent, as the order movies are compared may affect their final rating. Due to TrueSkill's original use in online video game matchmaking, if a movie receives low ratings in its first 5 (calibration) ratings, it may be placed lower than if those 5 ratings occurred at a later stage in that movie's list of comparisons.

It's also worth considering that calibrating a movie rating may take away from a user's freedom to voice their preference and further abstract the rating process, leaving user's unhappy or in disagreement with the calculated rating. When considering all major rating website's movies averaging around a 7 or a 3.5, it's worth noting that whilst this is a poor utilization of the scale and is unintuitive mathematically, it may be intuitive for a user. User's seeing their average movie as a 2.5 using TrueSkill instead of a 3.5 may be confusing, as it is not the rating they would expect or potentially desire for their *average* movie.

In conclusion, this investigative analysis shows the potential viability of TrueSkill, or relative ranking systems, as an alternate or supplementary system to traditional flawed absolute rating systems.

4.2 Ideas for Future Work

If one were to conduct further analysis into the viability of using relative ranking systems to rate media, the first step would be to conduct data gathering with a much larger user base. As this study only contained 12 users, very few of the 784 calibrated movies have over 2 users or more than 20 comparisons. More comparisons per movie decreases the uncertainty (σ) TrueSkill has in those movies' ratings, and the more users involved in these comparisons decreases the chance one user's perception over-impacts global rating.

As our original hypothesis states that absolute systems for ranking and rating media are not perfect, it is difficult to understand and compare other methods as there is no absolute preference to compare to. Ordered sets could be created using different relative ranking algorithms, or the same algorithms with different parameters, and then compared to the TrueSkill set to understand if relative ranking algorithms all tend towards a similar ordering, or if each algorithm creates a completely new unique ordering. This will help to understand if relative ranking systems determine more accurate and precise rankings based on the user preferences, or instead are as unreliable as absolute methods.

TrueSkill, being a proprietary algorithm owned by Microsoft, would have significant costs to implementing into a movie review website. Further research into alternate relative ranking algorithms such as Openskill would be required to understand if these results are repeatable using a different relative ranking algorithm. Some analysis was performed on an Elo ordered set, constructed using the pairwise comparisons extracted from the larger user submitted comparisons. However, it would also be possible to calculate these ratings based on other algorithms designed for two player zero-sum games such Glicko or even TrueSkill set up for 1v1 competitions. Further analysis would be required to understand the strengths and weaknesses of these systems, especially when compared to relative ranking algorithms that factor in multiple items at once.

List of References

Cavanagh, T., Chen, B., Lahcen, R.A.M. and Paradiso, J. 2020. Constructing a design framework and pedagogical approach for adaptive learning in higher education: A practitioner's perspective. *The International Review of Research in Open and Distributed Learning*. **21**(1), pp.172–196.

Eliashberg, J. and Shugan, S.M. 1997. Film Critics: Influencers or Predictors? *Journal of marketing*. **61**(2), p.68.

Elo, A.E. 1967. The Proposed USCF Rating System, Its Development, Theory, and Applications, pp.26–31.

Gemser, G., Van Oostrum, M. and Leenders, M.A.A.M. 2007. The impact of film reviews on the box office performance of art house versus mainstream motion pictures. *Journal of cultural economics*. **31**(1), pp.43–63.

Gross, D.A. 2024. *Current Movie Industry Charts.* [Online]. [Accessed 5 September 2024]. Available from: https://www.franchisere.biz/movie-industry-charts-and-trends/.

Herbrich, R., Minka, T. and Graepel, T. 2007. TrueSkill: A Bayesian Skill Rating System *In*: B. Schölkopf, J. Platt and T. Hofmann, eds. *Advances in Neural Information Processing Systems 19*. The MIT Press, pp.569–576.

IMDb 2024. IMDb Non-Commercial Datasets. [Accessed 5 September 2024]. Available from: https://datasets.imdbws.com/.

Jacobs, R.S., Heuvelman, A., Ben Allouch, S. and Peters, O. 2015. Everyone's a critic: The power of expert and consumer reviews to shape readers' post-viewing motion picture evaluations. *Poetics (Hague, Netherlands)*. **52**, pp.91–103.

Lee, H. 2016. *TrueSkill — trueskill 0.4.5 documentation* [Online]. [Accessed 9 May 2024]. Available from: https://trueskill.org/.

Letterboxd. 2024. *Frequent questions.* [Online]. [Accessed 9 May 2024]. Available from: https://letterboxd.com/about/faq/.

Minka, T. and Cleven, R. 2018. *TrueSkill 2: An improved Bayesian skill rating system.* [Online]. [Accessed 9 May 2024]. Available from: https://www.microsoft.com/en-us/research/uploads/prod/2018/03/trueskill2.pdf.

Moser, J. 2011. *The Math Behind TrueSkill.* [Online]. [Accessed 9 May 2024]. Available from: https://www.moserware.com/assets/computing-your-skill/The%20Math%20Behind%20TrueSkill.pdf.

Motwani, R. and Raghavan, P. 1995. Coupon Collector Problem *In*: *Randomized Algorithms*. Cambridge: Cambridge University Press, pp.57–63.

Pelánek, R. 2016. Applications of the Elo rating system in adaptive educational systems. *Computers & education*. **98**, pp.169–179.

Roberts, M. 2018. *Serverless Architectures.* [Online]. [Accessed 9 May 2024]. Available from: https://martinfowler.com/articles/serverless.html.

Sakaguchi, K., Post, M. and Van Durme, B. 2014. Efficient elicitation of annotations for human evaluation of machine translation *In*: *Proceedings of the Ninth Workshop on Statistical Machine Translation*. Stroudsburg, PA, USA: Association for Computational Linguistics.

Spearman, C. 1904. The proof and measurement of association between two things. *The American journal of psychology*. **15**(1), pp.72–101.

Srinivasan, R. and Maloney, B. 2024. *What is scrum.* [Online]. [Accessed 9 May 2024]. Available from: https://www.scrumalliance.org/about-scrum.

Rotten Tomatoes. 2024. *About.* [Online]. [Accessed 9 May 2024]. Available from: https://www.rottentomatoes.com/about.

Vis, D. 2024. *Film list: Official Top 250 Narrative Feature Films.* [Online]. [Accessed 9 May 2024]. Available from: https://letterboxd.com/dave/list/official-top-250-narrative-feature-films/.

Weisstein, E.W. and Sondow, J. 2024. *Harmonic number.* [Online]. [Accessed 9 May 2024]. Available from: https://mathworld.wolfram.com/HarmonicNumber.html.

Zervas, G., Proserpio, D. and Byers, J.W. 2021. A first look at online reputation on Airbnb, where every stay is above average. *Marketing letters*. **32**(1), pp.1–16.

Appendix A
Self-appraisal

A.1 Critical self-evaluation

This project outline was specifically designed to allow me to grow my web development skills, designing and deploying a modern multi layered application. The web scraping and data analysis sections allowed me to broaden my technical skills, gaining new experience in tools such as Jupyter, Selenium and Beautiful Soup.

Originally, I had envisioned a platform where users complete their comparisons and receive tailored feedback on how similar (or dissimilar) their relative comparisons were to their Letterboxd logged movies. This idea was flawed in multiple ways: it was more passion project than research project – with any insights not really furthering the field; and offering relatively little answers to my original question. After discussion with my supervisor, the scope was changed to include analysis of the comparisons completed on the site.

When considering the data analysis to be completed later, much effort was put into ensuring the comparison objects would have more than enough information to extract insights. Once the decision was made to transition from an Elo system to TrueSkill, the Comparison schema was the first thing designed with data analysis in mind. Although fields such as Number Movies and Number Limit ended up not being used by any users, the fields were included in the Comparison object to ensure analysis could be conducted upon them if required.

Overcomplicating the movie import feature slowed down development a lot during the early stages – with the original intended workflow having little benefit other than seeming more *scalable*. In retrospect, the benefits of this feature versus its development time and cost should have been properly investigated before developing. This would have saved a lot of wasted development time and is something that I plan to do in future projects.

Despite the unexpected time taken to populate the movie database, my original Gantt chart and timescales were mostly accurate. Development was sporadic to fit around other coursework and data collection which wasn't too much of an issue as a sole developer but is something I will have to focus on and improve when developing in larger teams in future.

All features originally written in the specification were developed in both the Flask application layer and React presentation layer. Both the backend and frontend are hosted on a cloud provider with environments, CI / CD, custom domains and industry standard security.

The website was a success with my user base, with 12 users completing 2600 comparisons between them. A 2-week data gathering deadline was set, allowing me to catch up on report

writing and begin developing analysis functions. Switching from the original planned movie import function to a set number of movies also arguably was beneficial for the data collected (being more concentrated over a smaller number of movies).

The consideration put into the Comparison model prior to collection was invaluable when performing data analysis, allowing me to not only recreate a final state of the database but also perform further analysis on the chain of comparisons. Without this chain of comparisons, pairwise comparisons could not be created, and I would have been unable to analyse the Elo set.

A.2 Personal reflection and lessons learned

My project idea was born out of a curiosity and passion I had for the question of why movie ratings always average around a 7 – and how this could be fixed. I wanted to use this dissertation as an opportunity to broaden my knowledge of web development (developing and deploying a 3-tier application) and further experiment in other fields of computer science I had little experience in.

After moving towards a simpler application (the current data collection platform), I had assumed that this site alone would be enough to qualify for my dissertation work. Looking back on this now, this seems incredibly naïve. Upon direction from my supervisor and assessor I was directed to also conduct data analysis on the results of user comparisons. I am incredibly proud of the analysis and investigation performed in this report, especially considering the data was harvested through my own platform. The data analysis section of this project pushed me into an area of computer science I lacked experience in, learning and persevering at every opportunity and challenge.

I chose to use Python, and more specifically Flask, as my framework for the application layer due to my previous experience working with the framework. I had used Flask for Web Application Development module and continued learning and using it in other personal projects. At one point, at the beginning of the second semester and thus Web Services module, I considered developing the backend in Django – attempting to progress my dissertation while learning the content for Web Services. I abandoned Django quickly as the data collection platform was arguably one of the most complex applications I have ever built, and having the preexisting knowledge of Flask was invaluable. This taught me to remain confident and consistent in my original design and specification.

For my presentation layer, I chose to use React due to some previous experience working with it but more so due to a personal interesting in learning the framework. Due to its popularity, there are endless tutorials and documentation on React, making it a lot easier to learn than

other less popular frontend frameworks. This frontend was positively received by my userbase, and it was always exciting to see users sending me difficult comparisons they had been tasked with completed on the platform.

Originally, I had planned to release Cinerank to a wider audience, allowing for more users to submit comparisons and in turn more data for me to analyse. When discussing this with my tutor, I was met with the realisation that this would be a pretty difficult task when considering university ethics and guidelines. This was initially disappointing as it meant data collection would be limited, but meant I was able to ensure the ethics and consent requirements were met. This led to limited comparisons for analysis, meaning the set of movies would be significantly smaller than the 17000 present in the populated movie database.

This experience designing, building and deploying a complex web application I have designed is a testament to my ability to learn and adapt as a computer scientist.

A.3 Legal, social, ethical and professional issues

A.3.1 Legal issues

Despite being hosted and accessible globally, all users accessing the website were based in the UK. Therefore, the most relevant legal documentation is the UK Data Protection Act (DPA), based on the EU's General Data Protection Regulation (GDPR).

Minimal data is stored for each user – limited to username, password (hashed using industry standard algorithms) and any comparisons they submit. None of this information is considered sensitive information by the DPA. Users can delete their account at any time and all information stored on the site is openly visible to the user.

TrueSkill is a proprietary algorithm developed by Microsoft requiring a license for commercial applications and development. This project is not a commercial application, however if this system were to be adopted by a movie rating aggregator a license would be required to use TrueSkill.

All other software and services used in this project are open source to ensure proper and legal use in this project.

A.3.2 Social issues

When considering the social issues related to cloud computing and its unavoidable environmental impact, considerations were made to ensure any compute or resource usage was minimised.

During development the Azure SQL server was set to automatically sleep after 1 hour of inactivity, ensuring the database was not unnecessarily contributing to carbon emissions. An Azure Serverless function was preferred over an Azure App Service for the Flask application due to it's on-demand nature. These choices ensured that resources were not wasted when not in use.

If this application and it's userbase were to scale globally, consideration would be given to other languages. As all text and movie information is stored in English, localisation would be required to ensure the site is accessible and usable in regions where English is not the primary language.

The drag and drop functionality on the movie comparison page is not accessible for keyboard users as it requires precise mouse movement. An alternate arrow key ordering control scheme would be required for a more accessible application.

A.3.3 Ethical issues

As of completing this project, Letterboxd does not specify against the use of web scraping in their terms of use, and do not provide a robots.txt file. Many ethical considerations were made when considering the impact of the web scraping parts of this project.

Only publicly available, exportable information was web scraped from Letterboxd. Anyone with a Letterboxd account can export their activity and any lists (including the top 20000 movies). from Letterboxd. All scraping was minimised to ensure that Letterboxd servers are not overloaded or slowed by the traffic. The movie import function was specifically redesigned to ensure only essential web scrapes were used.

Alternate methods of attaining users' information were planned (including users uploading their exported information as a csv) in case the web scraping was unethical, or Letterboxd requested it to stop. In a larger application with more users, the number of web scrapes would grow exponentially and would require different methods of data collection, or access to Letterboxd private API.

All requests to The Movie Database (TMDB) for movie posters were completed in accordance with TMDB's API specifications.

A.3.4 Professional issues

This projects development followed the professional standards set out by the BCS Code of Conduct and the ACM Code of Ethics.

The BCS Code of Conduct specifically states to have due regard for the rights of third parties and only work within your professional competence should be undertaken. As outlined above, many considerations were made to reduce the impact of the application on Letterboxd's operations. All work completed in this project are within the student's professional competence, and no claims to a higher level of competency have been made in this report.

Appendix B
External Materials

This project is built upon many key technologies that are not the student's own work.

The database storage layer is deployed on Azure SQL Server. SQLAlchemy is used for all interaction between the Flask application layer and this SQL Server.

Git was used as the version control software for this project, with the remote master being hosted on GitHub. CI / CD workflows for both the application and presentation layer were both automated by GitHub Actions. Trello was used for the Kanban board.

The application layer is developed in Python, using Flask as a web framework. All authentication between the application layer and presentation uses Flask JWT Extended to create, authenticate and reset JWT tokens. Password hashing and comparing uses passlib, with Bcrypt being used as the hashing algorithm. Flask CORS is used for handling all Cross Origin Resource Sharing headers and interaction between the application layer and presentation layer. Python packages TrueSkill and Elo were used for any relative ranking calculations carried out on user comparisons.

Selenium, with Chrome WebDriver, was used for scraping the original movies for the database. Beautiful Soup was used for all web scraping of user information from Letterboxd. Tqdm was used for dynamic progress bars in some of the scraping functions.

A Jupyter notebook was used to encapsulate all data analysis. Pandas, numpy, scipy and statsmodels were all used for data analysis and investigation. Plotly and matplotlib were used for data visualization.

For the presentation layer, React was used as the JavaScript front end library and Vite was used as the local development server. React Router DOM was used to create the router, allowing multiple pages to be navigated between. Tailwind was used as the CSS framework for styling, with DaisyUI used for premade components and styling. Axios was used to simplify all requests made from the presentation layer. PasswordChecklist was used for the password requirements on the registration page. React DnD was used for the drag and drop functionality on the movie comparison page. Immutability helper was used to assist with updating of movie data on the comparison page.

All movie information was scraped from Letterboxd, with movie posters being acquired from The Movie Database's public API.

The report was written in Microsoft Word and converted to PDF.

Appendix C – Additional Figures

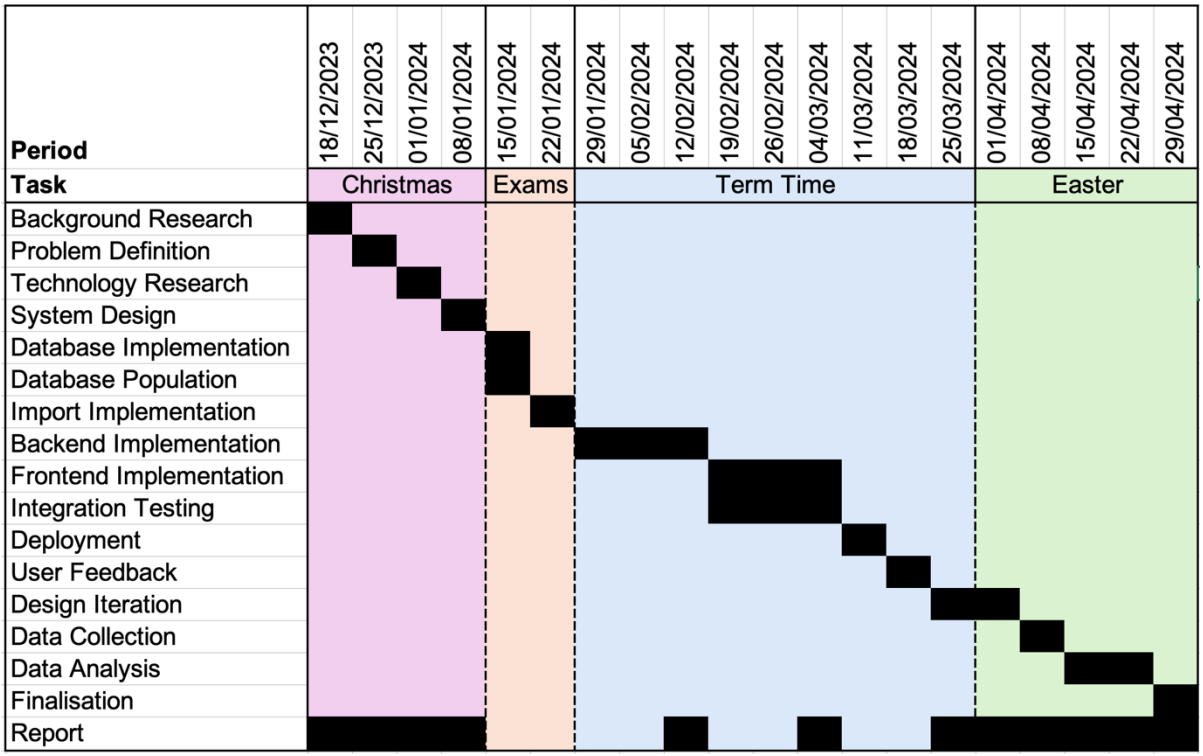

Figure 6 - Planned Development Timeline

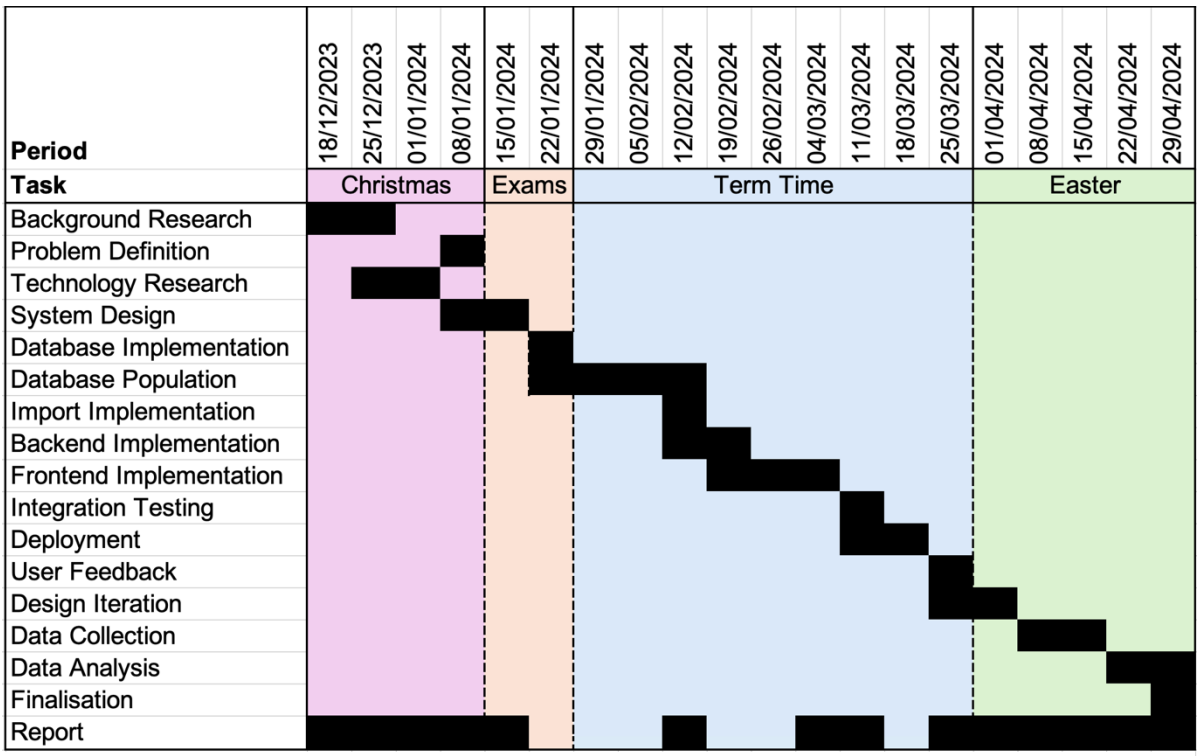

Figure 7 - Actual Development Timeline

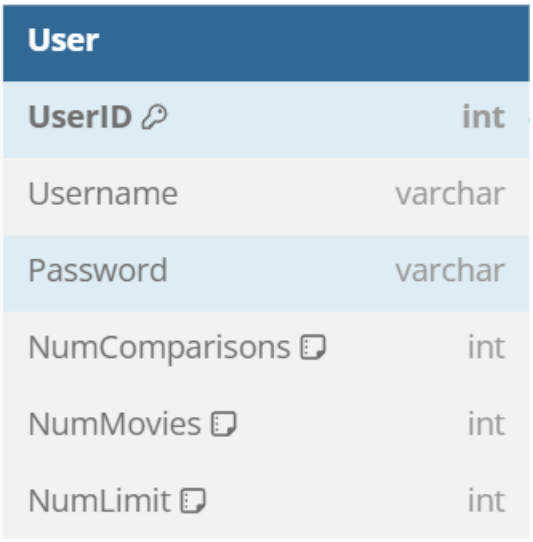

Figure 8 - User Schema

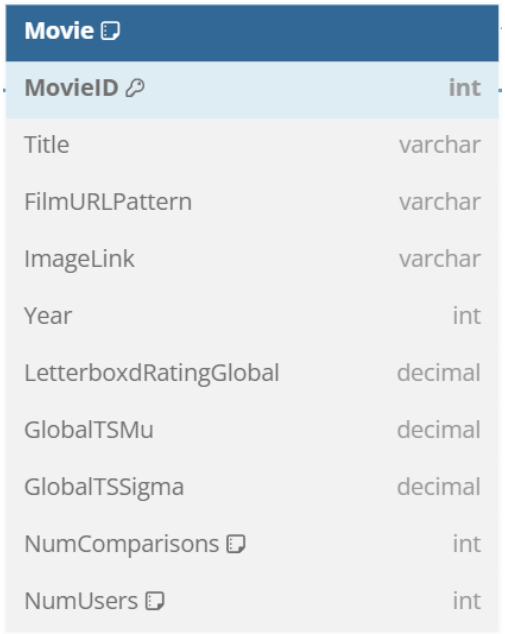

Figure 9 – Movie Schema

Figure 11 - Comparison Schema

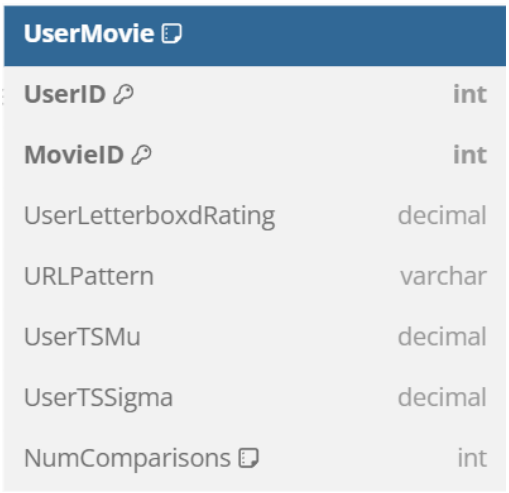

Figure 10 - UserMovie Schema

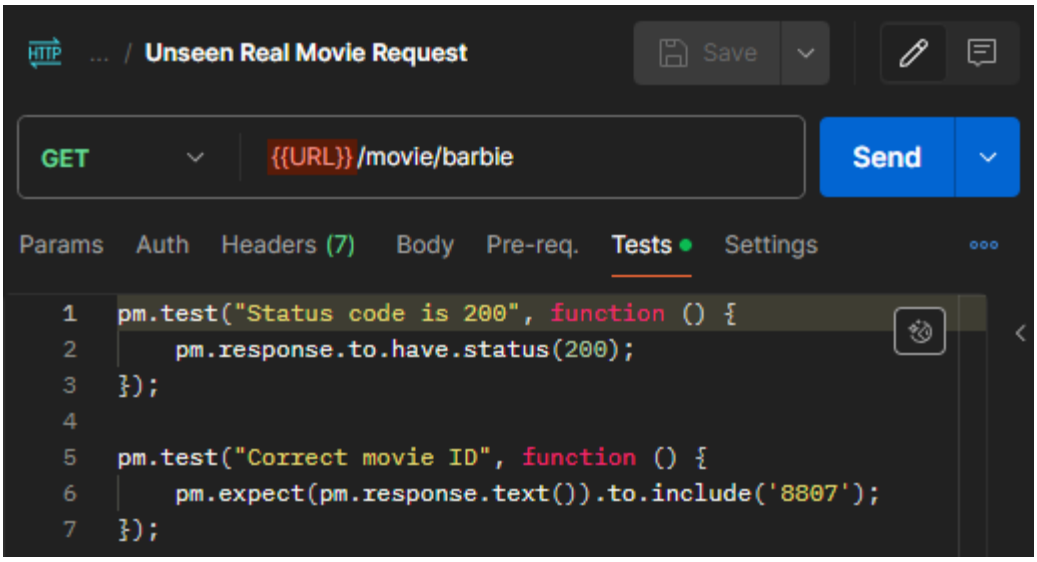

Figure 12 - Example Postman Test for Querying a Movie Successfully

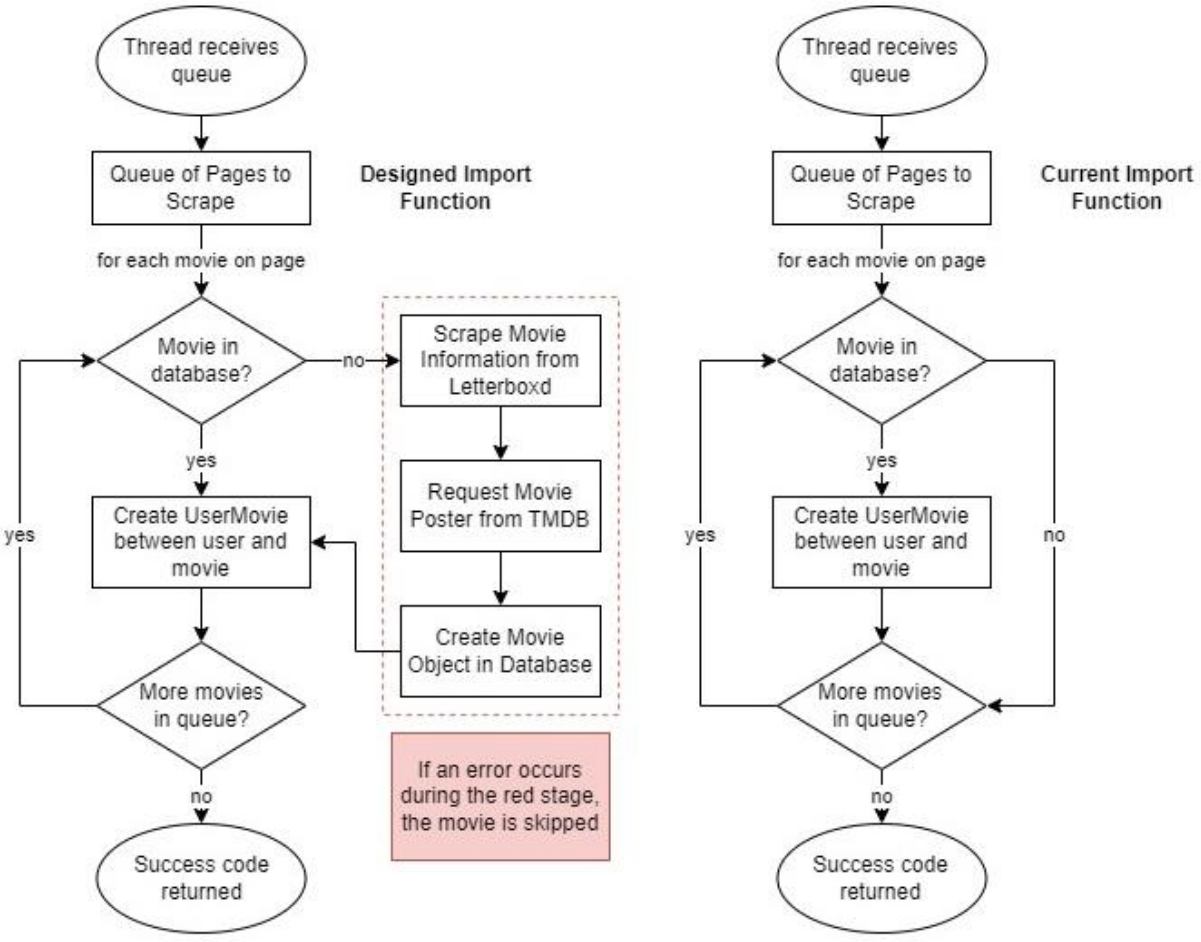

Figure 13 - Original Design of Import Function Compared to Developed Import Function

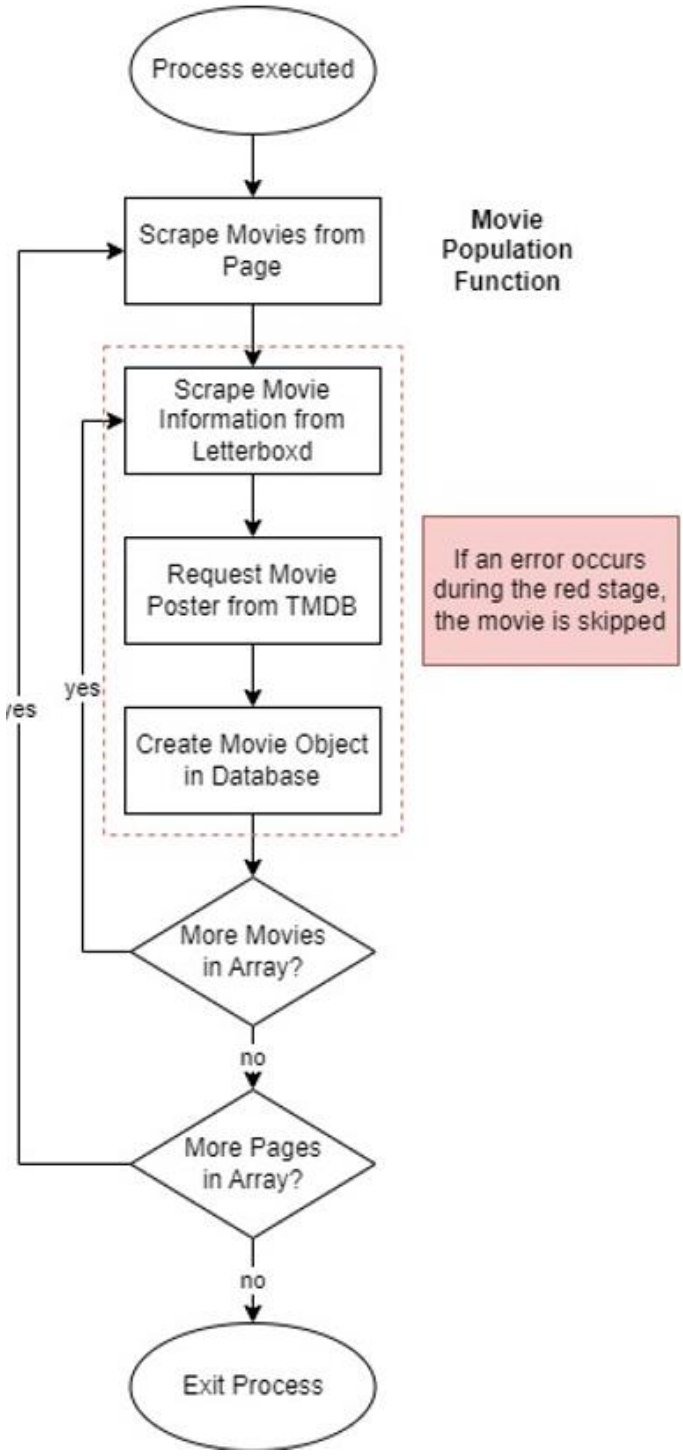

Figure 14 – Implemented Movie Database Population Function

Appendix D – Additional Resources

D1 - Flask API Specification

User Blueprint

The User blueprint uses the sub route *user* and houses all routes relating to getting information related to a user, updating user settings and further operations a user can make.

GET /user/leaderboard - Get Leaderboard

- *GET Method, JWT Authentication Required*
- Used for showing the leaderboard table on the Cinerank website.
- Returns the top 25 users (by number of comparisons completed) and their relevant information.
- Returns 200 code if successful.
- Returns a 500 code and appropriate error if route fails during execution.

GET /user/<username> - Get User Statistics from Username

- *GET Method, JWT Authentication Required*
- Used for showing a user's movies watched in order on their profile page on the Cinerank website.
- Takes a username string in the route and returns the user's top 100 movies and their relevant information.
- Converts TrueSkill μ ratings into star ratings, also returned in the above array.
- Returns 200 code if successful.
- If the logged in user is not found, a 404 invalid user error is returned.
- Returns a 500 code and appropriate error if route fails during execution.

GET /user/settings – Get User Settings

- *GET Method, JWT Authentication Required*
- Returns the number of movies, limit of movies and number of comparisons for the logged in user.
- Returns a 200 code if successful.
- If the logged in user is not found, a 404 invalid user error is returned.
- Returns a 500 code and appropriate error if route fails during execution.

POST /user/settings – Update User Settings

- *POST Method, JWT Authentication Required*

- Receives a new limit of movies number for the user and updates it in the database.
- Returns a 200 code if successful.
- If the logged in user is not found, a 404 invalid user error is returned.
- Returns a 500 code and appropriate error if route fails during execution.

POST /user/reset - Reset User Movies

- *POST Method, JWT Authentication Required*
- Resets all of users imported movies, setting rating values back to default and number of comparisons to 0.
- Returns a 200 code if successful.
- If the logged in user is not found, a 404 invalid user error is returned.
- Returns a 500 code and appropriate error if route fails during execution.

DELETE /user/delete – Delete All User Information

- *DELETE Method, JWT Authentication Required*
- Deletes all data connected to a user, including comparisons, user movies and account.
- Returns a 200 code if successful.
- If the logged in user is not found, a 404 invalid user error is returned.
- Returns a 500 code and appropriate error if route fails during execution.

DELETE /user/remove - Remove User Movies

- *DELETE Method, JWT Authentication Required*
- Deletes all user movies connected to a user – intended for users to reset their account if imported account was incorrectly typed.
- Returns a 200 code if successful.
- If the logged in user is not found, a 404 invalid user error is returned.
- Returns a 500 code and appropriate error if route fails during execution.

POST /watch/<URLPattern> - Add a Movie to User Watched

- *POST Method, JWT Authentication Required*
- Finds a movie from the supplied URL Pattern and creates a user movie between it and the logged in user.
- Returns a 200 code if successful.
- If no movie is found with the URL Pattern provided, a 404 invalid movie error is returned.

- If a user movie is found to already exist (the user has already watched the movie), a 400 movie already watched error is returned.
- If the logged in user is not found, a 404 invalid user error is returned.
- Returns a 500 code and appropriate error if route fails during execution.

DELETE /unwatch/<URLPattern> - Remove a Movie from User Watched

- *DELETE Method, JWT Authentication Required*
- Removes a movie of the supplied URL Pattern from the logged in user's user movies.
- Returns a 200 code if successful.
- If no movie is found with the URL Pattern provided, a 404 invalid movie error is returned.
- If no user movie is found (the user has not watched the movie), a 400 movie not watched error is returned.
- If the logged in user is not found, a 404 invalid user error is returned.
- Returns a 500 code and appropriate error if route fails during execution.

Movie Blueprint

GET /movie/<URLPattern> - Get a Movie by URL Pattern

- *GET Method, JWT Authentication Required*
- Returns all information on a movie from the supplied URL Pattern parameter.
- If the logged in user has seen the movie, information such as the user rating and number of comparisons are also returned from the connecting user movie.
- Returns a 200 code if successful.
- If the logged in user is not found, a 404 invalid user error is returned.
- If no movie is found with the URL Pattern provided, a 404 invalid movie error is returned.
- Returns a 500 code and appropriate error if route fails during execution.

GET /movie/<X> - Get X Random Movies

- *GET Method, JWT Authentication Required*
- Returns X random movies from the logged in user's user movies table.
- If the logged in user has no movies, a 404 no movies found error is returned.
- Returns a 200 code if successful.
- If the logged in user is not found, a 404 invalid user error is returned.
- Returns a 500 code and appropriate error if route fails during execution.

GET /movie/global – Get Global Movie Statistics

- *GET Method, JWT Authentication Required*
- Returns the top 100 movies with over 5 comparisons – ordered by their global TrueSkill μ rating.
- Returns a 200 code if successful.
- Returns a 500 code and appropriate error if route fails during execution.

POST /movie/user/comparison/ - Post a Movie Comparison

- *POST Method, JWT Authentication Required*
- Receives a formatted JSON array of 5 movies in order and completes a TrueSkill comparison both globally and locally for the logged in user.
- If the logged in user is not found, a 404 invalid user error is returned.
- If the comparison data JSON has less than 5 movies, a 400 invalid comparison data error is returned.
- Returns a 200 code if successful.
- Returns a 500 code and appropriate error if route fails during execution.

Authentication Blueprint

POST /auth/register/ - Register a New User

- *POST Method*
- Receives a formatted JSON of username and password, and then creates a user object in the database.
- If the username or password are not present in the JSON, a 400 missing arguments error is returned.
- If a user with the supplied username exists, a 400 username already exists error is returned.
- Returns a 200 code if successful.
- Returns a 500 code and appropriate error if route fails during execution.

POST /auth/login/ - Login as an Existing User

- *POST Method*
- Receives a formatted JSON of username and password.
- Queries the database for a user with a matching username, then compares the hash of the provided password with the password stored.
- If the user exists and the password hashes match, a JWT access token is created and returned to the user. This is then used by the frontend for authentication.
- If a user is not found with the provided username, or the password hashes do not match, a 401 invalid username or password error is returned.

- Returns a 500 code and appropriate error if route fails during execution.

GET /auth/logout/ - Logout as a Logged in User

- *GET Method*
- Used to logout an authenticated user, removing their JWT access token and effectively ending their session.
- Removes the JWT access token if it is found in the request.
- Returns a 200 code if successful.

GET /auth/validate/ - Test JWT Validation

- *GET Method, JWT Authentication Required*
- Used for testing JWT authentication during development.
- Returns the username of the current logged in user.

Import Blueprint

POST /import/<username> - Import User Movies from Letterboxd

- *POST Method, JWT Authentication Required*
- Scrapes all movies logged by Letterboxd user with username provided and add them to the logged in user's user movies.
- The number of logged movies is scraped for that user, then each 72-movie page is scraped by multiple different threads in parallel.
- If a movie scraped from Letterboxd is not found in the database or an error occurs during it's scraping, it is skipped, and no user movie is created.
- If the logged in user is not found, a 401 Invalid user error is returned.
- If an error occurs during the execution of the web scrape, comparing to movies present in the database or creating user movies a 500 Error message is returned with an appropriate error message.

D2 – User Consent Form

Using TrueSkill™ to Compare Relative and Absolute Methods of Ranking Media

Louis
Boswell

* Indicates required question

Consent Form (User Testing)

1. *Tick the box if you agree with the statement to the left* *

Check all that apply.

- [ ] I confirm that I have read and understand the information sheet dated 19/04/2024 explaining the above project and I have had the opportunity to ask questions about the project.
- [ ] I understand that my participation is voluntary and that I am free to withdraw at any time without giving any reason and without there being any negative consequences. In addition, should I not wish to answer any particular question or questions, I am free to decline. Insert contact details here of project student.
- [ ] I understand that my responses will be kept strictly confidential.
- [ ] I understand that my name will not be linked with the project materials, and I will not be identified or identifiable in the report or reports that result from the project.
- [ ] I agree for the data collected from me to be used in future research.
- [ ] I agree to take part in the above project and will inform the project student should my contact details change.

2. Name of Participant

3. Cinerank Username

4. Date

Example: January 7, 2019

D3 – Cinerank Ease of Use Survey

Cinerank Ease of Usage Survey

This survey will be used to assey and improve the user experience when comparing films on Cinerank.

Each page will have its own sections for:

- How usable and intuitive is it?
- Are there any confusing aspects (that may need clarifying)?
- Are there any noticable bugs?
- Extra fields for other feed back

Users without a Letterboxd account, will need to select this in the following section.

* Indicates required question

1. Did you use your own Letterboxd account, or the provided testuser account? *

   Mark only one oval.

   - ◯ Created Own Account
   - ◯ Test User

Log In and Register Pages

Pages to create a new account or log in as an existing user.

2. Are the password requirements clear?

   Mark only one oval.

   - ◯ Yes
   - ◯ No
   - ◯ Other: _____

3. How usable and intuitive are these pages?

Mark only one oval.

	1	2	3	4	5	
Unc	◯	◯	◯	◯	◯	Clear

4. Are there any confusing aspects (that may need clarifying)?

5. Are there any noticable bugs?

6. Extra fields for other feed back

Movie Ranking Page

Page where users films are ranked

7. Is the order you rank the films best to worst clear?

Check all that apply.

- [ ] Yes
- [ ] No

8. How usable and intuitive are these pages?

Mark only one oval.

	1	2	3	4	5	
Unc	◯	◯	◯	◯	◯	Clear

9. Are there any confusing aspects (that may need clarifying)?

10. Are there any noticable bugs?

11. Extra fields for other feed back

Movie information page

Page where information on a specific movie is stored, can be accessed double clicking a film in the ranking

12. How usable and intuitive are these pages?

Mark only one oval.

	1	2	3	4	5	
Unc	◯	◯	◯	◯	◯	Clear

13. Are there any confusing aspects (that may need clarifying)?

14. Are there any noticable bugs?

15. Extra fields for other feed back

Statistics page

Page where a users' movie preferences or global preferences can be found

16. How usable and intuitive are these pages?

Mark only one oval.

	1	2	3	4	5	
Unc	◯	◯	◯	◯	◯	Clear

17. Are there any confusing aspects (that may need clarifying)?

18. Are there any noticable bugs?

19. Extra fields for other feed back

User leaderboard page

Page where the most active users can be seen

20. How usable and intuitive are these pages?

Mark only one oval.

	1	2	3	4	5	
Unc	◯	◯	◯	◯	◯	Clear

21. Are there any confusing aspects (that may need clarifying)?

22. Are there any noticable bugs?

23. Extra fields for other feed back

Import movie page

Page where movies can be imported from a Letterboxd username

24. How usable and intuitive are these pages?

Mark only one oval.

	1	2	3	4	5	
Unc	◯	◯	◯	◯	◯	Clear

25. Are there any confusing aspects (that may need clarifying)?

26. Are there any noticable bugs?

27. Extra fields for other feed back

Setting page

Page where users can update their settings

28. How usable and intuitive are these pages?

Mark only one oval.

	1	2	3	4	5	
Unc	◯	◯	◯	◯	◯	Clear

29. How clear is it what each setting does?

Mark only one oval.

	1	2	3	4	5	
Unc	◯	◯	◯	◯	◯	Clear

30. Are there any confusing aspects (that may need clarifying)?

31. Are there any noticable bugs?

32. Extra fields for other feed back

Information page

Page where users can vew information on Cinerank

33. How usable and intuitive are these pages?

Mark only one oval.

	1	2	3	4	5	
Unc	◯	◯	◯	◯	◯	Clear

34. Are there any confusing aspects (that may need clarifying)?

35. Are there any noticable bugs?

36. Extra fields for other feed back