Curatorem: A User-Friendly Movie Recommender Web-Application

Saleh Lootah

College of Engineering, Mathematics
and Physical Sciences, University of Exeter

2021

Abstract

The issue of content oversaturation is a well-studied area of data science and recommender systems are a widely implemented solution to this problem. This report details the research and development behind my attempt at developing a production movie recommender system called Curatorem. This problem was chosen to allow me to learn about the entire software development lifecycle as well as play as an introduction to the field of data-science. The main aim of the project was to create a platform where a user may learn of new movies to watch that they have not seen before through a fine-tuned machine learning algorithm. Curatorem can be divided into three main components: the database, the recommendation algorithm, and the front-end user interface. By examining the advantages and disadvantages of the many technologies that could be used to implement these components the finalized application consisted of a combination of the following technologies: Flask, PostgreSQL, and Apache Spark (PySpark). Microsoft Azure Cloud Services and Heroku were utilized for the deployment and hosting of the finalized application and GitHub was used as a version control platform.

I certify that all material in this dissertation which is not my own work has been identified.

Saleh Lootah
660071567

TABLE OF CONTENTS

1 INTRODUCTION

1.1 BACKGROUND INFORMATION & INTRODUCTION

A recommender system is a tool or technique used to provide suggestions of interest to a user [1]. With the rapid development of computing and storage technology, recommendation systems have become more and more relevant to the point of revolutionizing the way products are marketed. In the past, a shop would be constrained by its limited physical storage capability and logistics. In-turn forcing the shop to prioritize the most popular/mainstream items in order to gain the most profit [2]. This issue was largely mediated with the rise of online shopping, where a small storefront on a popular shopping street would not limit the storage capability of the business, as it was no longer a necessity due to the ability to market products on the internet [3]. The business would simply use a larger scale storage solution (warehouse instead of storefront on the high-street) and market the much larger selection of products on their website. However, this had an unexpectedly adverse effect on sales, where a customer would be overwhelmed by the seemingly unlimited number of choices and in-turn not make any purchases [4]. This un-foreseen side-effect called for some sort of marketing strategy reform prompting the use and development of recommendation systems. Recommendation systems allow online businesses to cater the products to each individual users' interests, just like a salesman who understands the customer and tries to sell them things they might be interested in. While recommendation systems have their place in the world of e-commerce, they are widely used in a plethora of different industries and are one of the most important tools used in any online business, from social media platforms to streaming services [5].

1.2 THE PROBLEM

The entertainment industry consists of a combination of 'sub-industries' that aim to produce some form of amusement for their consumers. As of 2019 the global theatrical and home/mobile entertainment market is valued at USD 101 billion [6]. In recent years many on-demand movie streaming services have seen a surge in their user-base and with the success of these services an increasing number of movies is regularly added into the pool of movies available to the user [7], [8].

With the ever-growing selection of movies to watch, users may feel overwhelmed and unable to decide what to watch [4]. For this reason, many of the on-demand movie streaming services implement recommender systems in their applications that suggest movies that may be interesting to the user. For my project I wanted to learn the inner workings of these systems, and therefore build one of my own.

1.3 THE PROPOSED SOLUTION: CURATOREM

The aim of Curatorem is to address the above stated issues of movie over-saturation by allowing users to rate movies and providing suggestions based on these ratings. The main focuses of the application are to generate viable movie suggestions, provide descriptions for any selected movie and provide the user with an intuitive user experience through a well-designed and simple user interface.

Curatorem will establish a platform that enables people to discover new movies to watch based on their personal taste. To use the application a user must initially create a profile by supplying their email, name and a password. This information will allow the system to differentiate between users. The user will then be able to access the list of movies that they may rate as well as read more about. Once a user has rated a sufficient number of movies the system is able to start recommending movies to watch.

2 LITERATURE REVIEW SUMMARY

2.1 RECOMMENDER SYSTEM BASICS

In general, a recommender system is a pipeline whereby a large amount data from a database is inputted and fewer items are then filtered and outputted depending on the filtering technique and the goals for the system. A typical recommender system consists of: A database, a filtering taxonomy, an item scoring technique, prediction ranking method and an evaluation metric. The end-goal of a recommender system is to make a prediction of what a user's interest in an item is, for that a user's interest must be quantified. This can be achieved in many different approaches; one such example may be rating items from 1-5. In doing so, the recommender system's aim becomes predicting what the user is going to rate other un-rated items based on previous ratings, the higher the predicted rating the more likely it is to be recommended.

2.2 COLLABORATIVE FILTERING

Collaborative filtering is essentially a generalization of the classification and regression problems, where the recommendations are based on similar users' content interaction/interest [9]. This approach excels in generating novel recommendations allowing a user to discover new interests because other similar users may be interested in an item [10]. However this approach suffers from what is known as the cold-start problem, where if a new item is introduced into the system, it cannot be recommended as no users have interacted with it, the same also applies to a new user [11]. The core of the collaborative filtering model is the user-item interaction information, which is expressed as a user-item matrix with the ratings given by the users as the matrix entries.

	Item 1	**Item 2**	**Item 3**
User 1	NaN	4	NaN
User 2	3	NaN	NaN
User 3	4	NaN	2

Item ratings

Figure 1 User-Item rating matrix example

Many collaborative filtering implementation instances have been developed in the past three decades which lead to the categorization of collaborative filtering into two types: model-based and memory-based collaborative filtering. However, the latest implementations of these systems mainly utilize model-based stand-alone latent factor methods based on matrix factorization[9], [12], [13]. To achieve matrix factorization an algorithm is used to decompose matrices. Its benefits

in the field of recommender systems were unveiled by Simon Funk in his blog post [14] explaining his entry to the famed Netflix prize challenge; where Netflix pledged 1 million dollars to the person who is able to develop a system that would beat the accuracy of their Cinematch recommender system at the time [15].

3 PROJECT SPECIFICATION

This section is divided into two segments: Functional requirements and Non-functional requirements. The first section defines the feature set that must be present in the final application to fulfill the primary goals of the project. The second section establishes a set of non-functional requirements that may be used to assess the operation of a system.

3.1 FUNCTIONAL REQUIREMENTS

The functional requirements define what the application should do and how this will be achieved. As briefly explained above, Curatorem aims to provide four main services to the user: Register to the application, browse movies, rate movies and suggest new movies to watch.

3.1.1 REGISTERING AND LOGGING INTO THE SYSTEM – A login system must be implemented to allow the system to distinguish between different users. The system will query the user for their name, email and password and then store them into the database.

3.1.2 BROWSING THE MOVIE DATABASE – Once a user has registered and logged in, they will be able to access a list of all of the movies present on the database. Once a user has located the movie they want, they will be able to rate it and learn more about it.

3.1.3 SUGGEST NEW MOVIES – A recommender algorithm must be trained to provide suggestions to users and run on a regular basis providing adjusting the recommendations with increasing user ratings.

3.2 NON-FUNCTIONAL REQUIREMENTS

There are 3 non-functional requirements. Non-functional requirements specify how the system must operate.

3.2.1 USABILITY – It is of utmost importance that the application be simple and easy to use. To achieve this a non-cluttered user interface must be designed and implemented. Another essential part of the usability is the reliability of the interface, where in the user must be able to register and start using the system without any obstacles hindering them. Curatorem must be viewable and functional on varying screen sizes/types (responsive design).

3.2.2 PERFORMANCE –Speed is essential when generally interacting with the system as a whole.

4 EVALUATION CRITERIA

This section specifies the evaluation criteria for each of the fundamental services that make up Curatorem. This section will be split by the requirements stated in the previous section, where each requirement will be mapped to its respective evaluation criteria.

4.1 LOGIN & REGISTRATION SYSTEM

For the login system to be considered properly functional it must:

- Securely and correctly capture and append new user information into the database.
- Securely allow users to log in.
- Not allow one user to access another's data.

4.2 RECOMMENDATION SYSTEM

While the proper evaluation of any type of recommender system is user-based, as it is a subjective matter in the end and only the user can properly judge whether a system is effective or not, there are various widely used metrics that determine the accuracy of predictions. For my project I would plan of evaluating the quantified accuracy of my predictions by root mean squared error (RMSE).

For the recommender system to be considered properly functional it must:

- Provide relevant and sufficiently accurate suggestions (Low RMSE).
- Generate different suggestions for each user.

4.3 USABILITY

The usability of the application will be marked against the following criteria:

- The graphical user interface of the application must prove to be easy to use.
- The interaction flow must be smooth and intuitive.
- The application should be free of major bugs.
- The application must be responsive and functional on varying screen sizes.

4.4 PERFORMANCE

The application performance will be evaluated following the below criteria:

- Users can navigate the application in a timely manner.
- The recommender system generates updated recommendations on a regular basis.

5 PRODUCT DESIGN

This section is divided into 6 sub-sections describing the design choices made for each component of the application. The overarching paradigm for all the design choices was to try to keep the application's user interface as simple and intuitive as possible while being viewable at varying screen sizes. This is achieved by following a user-centered design approach. A user-centered design approach is an iterative process where each design decision is made with the user's needs in mind [16]. An important concept of this approach is that each object the user may interact with provides a clear and unique function, free of any unnecessary redundancies and complexities. This plays into the aesthetic-usability effect, which concludes that users often perceive aesthetically pleasing, less cluttered design as a design that is more usable [17]. Another important rule that I applied through-out the design process was to abide by Hick's Law, which states that the more choices a person is presented with, the longer the person will take to reach a decision [18]. In minimizing the page count, usability is increased allowing for quicker decision making when navigating the application.

5.1 AN OVERVIEW OF THE APPLICATION STRUCTURE

With a clear design paradigm to follow established, the next step is to plan out the user interaction flow process. The front-end user interface of Curatorem consists of 6* pages; A welcome screen, log-in page, sign-up page, personalized user profile, a move list page (every movie has a designated 'more details/rate' page) and a recommendations page. When a user first enters the application, they are presented with a welcome screen that concisely explains what the application does and provides helpful instructions on how to proceed. The user must then either sign-up or log-in. Once the user is logged in, they are redirected to a personalized profile page, of which allows them to review all of the movies they have already rated. From here they are able to view their recommendations, rate movies by browsing the list of all movies in the database.

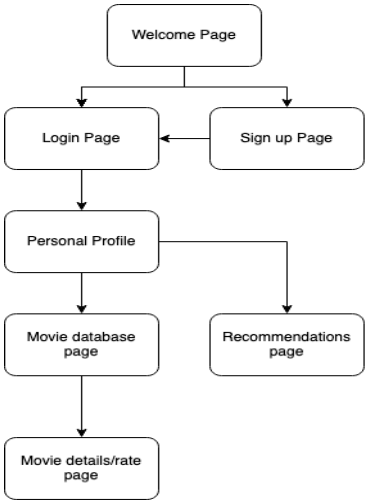

Figure 2 User interaction flow of the pages in Curatorem and the interaction between them

5.2 WELCOME, LOGIN AND SIGN-UP PAGES

The welcome page is the first page new users view upon visiting Curatorem. Having a welcome page plays an important role in making a good impression on a new user, as it allows the user to gain an understanding of the purpose of the application. Returning users on the other hand go directly to their personal profiles cutting any time wasted logging in or viewing welcome pages. This decision follows the UCD approach as it keeps takes new users into consideration, whilst increasing efficiency for users who are familiar with the application. A sign-up button is also present on the welcome and login pages. The sign-up page consists of four input fields asking for the users email, name and password.

5.3 USER PROFILE

Once a user has successfully logged into the application, or alternatively returns to the application prior to logging in, they are greeted with a welcoming personalized profile page as seem in Figure 4. On this page a user may: view their rated movies, navigate to the movie database page, or view the recommendations that are generated by the recommender system. Earlier iterations of the page implemented a recommender directly into each user's profile page, however upon acquiring user feedback it was decided to separate the pages as it allowed for improved distinction between the rated movies and the suggested movies.

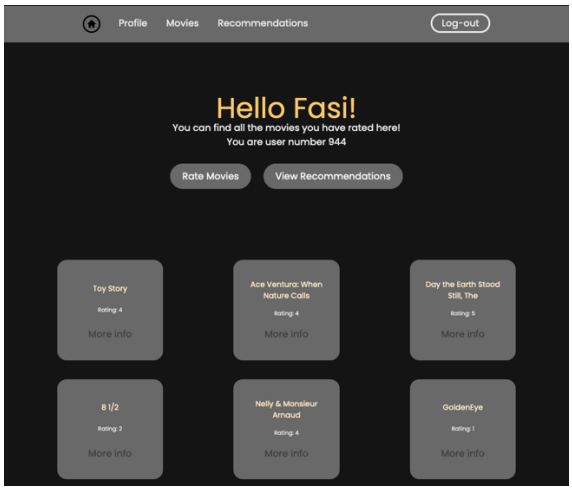

Figure 3 Curatorem personalized profile page

5.4 MOVIE DATABASE & RECOMMENDATION PAGES

The movie database and recommendation pages are the core of Curatorem and play similar roles. Each page must present movies in a clear and easy to comprehend manner as to allow the user to quickly understand what they are looking at. These pages allow users to interact with the presented movies and navigate to their respective details/rating pages. Each movie is clearly presented and listed in a manner that allows the user to access its details page quickly.

In designing these pages, Jakob's law of internet user experience was taken into consideration as well as the earlier stated paradigms. Jakob's law was coined by Jakob Nielsen which explains that users spend most of their time on other sites, meaning that users prefer your site to work the same way as all the others they already know [19]. The movie database page was designed following a simple grid layout, that is inherited from many other extremely popular websites such as YouTube.

5.5 MOVIE DETAILS AND RATING PAGE

The key premise of Curatorem is that it is a user-based application; it relies on users' movie ratings in order to provide recommendations, for this reason when designing the movie details and rating pages the top priority was to establish a clear and bold design language that is instantly intuitive to the user. The page simply consists of extra information about each movie and a big rating drop down bar that allows the user to quickly rate a movie and go back to the previous page exactly

where they left off to continue rating more movies or view their recommended movies. The simplicity of the design allows for a very intuitive user interaction process that is also efficient.

5.6 RECOMMENDATION SYSTEM DESIGN

The biggest factor in the design of a recommender system lies within the source of data used to train the algorithm that powers it. For my application I decided to utilize the MovieLens 100k movie ratings dataset [20]. The MovieLens 100k public dataset describes 5-star rating and free-text tagging activity from the MovieLens website. It contains 100836 ratings and 3683 tag applications across 9742 movies created by 610 users between March 29, 1996 and September 24, 2018.

With large number of users, a collaborative filtering based approach proves to be the best fit according to my literature review. Collaborative filtering systems are focused on detecting patterns in user behavior; for example, if two users have similar rating histories, it is presumed that they will act similarly in the future [9]. This allows for novelty in the recommendations as it garners new suggestions based on movies different people have watched, which in most cases will include movies that a user has never seen. This is in opposition to a content-based technique which is limited to generating recommendations solely based on the content each user consumes in turn making it difficult to discover new movies that may not be related to the content the user consumes directly.

Collaborative filtering can be accomplished in a variety of ways, but one popular approach is to use dimensionality reduction and matrix factorization techniques. Algorithms like Alternating Least Squares (ALS), Singular Value Decomposition (SVD), and Stochastic Gradient Decent (SGD) are good working solutions for building production-ready applications with large volumes of data, and these algorithms, in combination with distributive computing and parallel processing, are good working solutions. For my engine, I decided to use the ALS algorithm as it scaled well with increasing amounts of data and for its computational efficiency.

6 DEVELOPMENT

6.1 DEVELOPMENT METHODOLOGY

Prior to development, the most important decision to make was which software development lifecycle to use, and to do so, I looked at the requirements for the end solution and the various components needed to build the application. As Curatorem is a full-stack web application it required distinct separate technologies that would be integrated together to form the final application, for this reason I chose to apply the Waterfall model. The Waterfall model divides project activities into linear sequential phases, each of which is dependent on the previous phase's deliverables and corresponds to a task specialization [21], this allows for the development of each component in a step-by-step basis. Seeing as my requirements were set early on in the project lifecycle, all that was needed to do was to set out a timeline to accomplish each phase. This allowed for a streamlined development process that required minimal supplementary resources. Additionally, the separation of the phases allowed for clear goals to be set at each developmental stage.

The traditional waterfall model life cycle is divided into 6 phases: analysis, design, development, testing, implementation and maintenance [21]. Initially, I set out to clearly specify the aim of my project and the requirements needed in-order to achieve that aim. From there I designed mock-ups of the user interface and in-following the user centered design approach I accumulated user feedback with regards to various UI layouts and applied the appropriate changes iteratively until I reached a suitable design. For the development stage I chose to further divide it into 'sub-phases' with respect to the components that made up the application. Having each component represent a sub-phase allowed for further refinement of the goals which in-turn allowed for the testing phase to be interlinked with the development phase by splitting the testing phase into component level sub-phases as well. In doing so each of the components is developed and tested thoroughly allowing for a smooth implementation phase where all the components are integrated together. The maintenance phase would simply consist of further minor adjustments to the completed product.

6.2 APPLICATION TECHNOLOGIES

As briefly stated earlier, Curatorem is a full-stack web application, this means that it consists of a front-end (client side) and a back-end (server-side). Several technologies were required to build Curatorem. The front-end was developed with Flask; a python based web framework [22], while the back-end consists of a combination of a PostgreSQL database [23] and Apache Spark for the recommender system (PySpark) [24]. These technologies were primarily chosen because they utilize the python programming language, this is with the exception of the PostgreSQL database, which is utilizes SQL syntax.

6.2.1 VERSION CONTROL

Choosing a version control system was of a crucial step to take prior to starting the development of the application code. For this project I chose to utilize GitHub's version control services, this is for two main reasons: it's quick and robust integration with the Heroku application deployment solution, and due to the fact that I have used it in the past with various projects throughout my time at university and thus I am familiar with it. By iteratively backing up the progress made on the code base on my GitHub repository I was aided with the advantage of easily identifying bugs that may have been introduced into the code base by reverting to previous versions of the file and reviewing what changes caused the error. Additionally, the presence of several backups of the entire code base proves to be of immense help in the event of file corruption or failure. Throughout the development phases I utilized a main branch that would hold the tried and tested code and used a development branch where I would edit the code with potentially unstable updates that I would later test and if the code proves stable, merge the branch into the main branch. Using the GitHub version control platform in this manner allowed for an organized development process.

6.2.2 CLIENT-SIDE

The client-side application consists of the front-end user interface and a method of communicating with the back-end. The user interface is served from the server when a client requests to access Curatorem on a browser. Web applications are simply interactive HTML and CSS websites. They are made interactive through the use of web frameworks. As briefly stated earlier, Curatorem's front-end was coded utilizing the Flask web framework. Flask is a code library that speeds up and simplifies web development by providing common patterns for creating dependable, scalable, and

maintainable web applications. By utilizing a web-framework one is able to access common functions without the need of coding them from scratch, this allows for increased efficiency and access to external libraries that allow for added functionality such as: URL routing, Input form handling and validation, access to templating engines, database connection configuration and data manipulation using an object-relational mapper (ORM), web security against common malicious attacks and session storage /retrieval [25].

6.2.3 SERVER-SIDE

The server-side technologies are the components of the application that reside on the server permanently. Curatorem's backend consists of two components: a database and a recommender engine. For the database PostgreSQL was the obvious choice as it provided an open-source and robust solution that has been proven to reliably integrate with the Python programming language. Additionally, the PostgreSQL documentation is extensive and thorough. The construction of the database relied heavily on the structure of the data from the MovieLens dataset, which consisted of four tables: Movie table, User table, Rating table and Recommendation table. The Movie table holds information about the movies and consists of a unique movie ID, movie title, the movie release date and a IMDB URL linking to the IMDB page of the respective movie. The User table consists of a user ID, user email, username, password, age of the user and their zip-code (the age and zip code are inherited from the MovieLens dataset and are not required for registration). The Rating table represents the relationship between a user and a movie. It is a One-to-Many table where one user may rate many movies, it consists of a rating ID, user ID, movie ID and movie score. Similarly, the Recommendation table follows the same table structure with slight modifications in the form of a recommendation ID instead of a rating ID and a predicted score instead of a score.

For the development of the recommender engine, I opted to utilize Apache Spark, specifically the PySpark interface for Apache Spark. Apache Spark is a dynamic piece of software that has powerful ways of processing large volumes of data and separating the processing from the disk. The benefit of Spark is that it allows for large computations to be run and processed in a distributed manner (distributed computing/processing), such that it may take a 1 million row dataset slice it up into smaller pieces and run the algorithm simultaneously on these smaller pieces of data before compiling them back together. While the MovieLens dataset that I chose to use for this project is not the largest, the efficiency provided by Spark's ability to parallelize the workload proves beneficial to the overall performance of the recommender engine, providing headroom for the possible increase the dataset size in the future.

6.2.4 DEPLOYMENT

Deployment involves packaging the web application and hosting it production environment capable of running it [26]. As Curatorem consists of three components, each needs to be deployed separately. When deploying an application, one is faced with 4 options, the deployment on: "Bare metal" servers, Virtualized servers, Infrastructure-as-a-service or Platform-as-a-service, as time and resources were scarce the best option for deployment was the Platform-as-a-service, whereby I utilized the Heroku, and Microsoft Azure cloud hosting services. Heroku was chosen as the hosting platform for the front-end Flask application as it presented a free of charge hosting option as well as robust and responsive integration with GitHub. Microsoft Azure was used to host the ALS recommendation algorithm as well as the PostgreSQL database.

6.3 DEVELOPMENT OF FUNCTIONALITY

This section explains the main functions of each component that makes up Curatorem. This consists of the login/registration system, movie rating system and the recommendation generation engine.

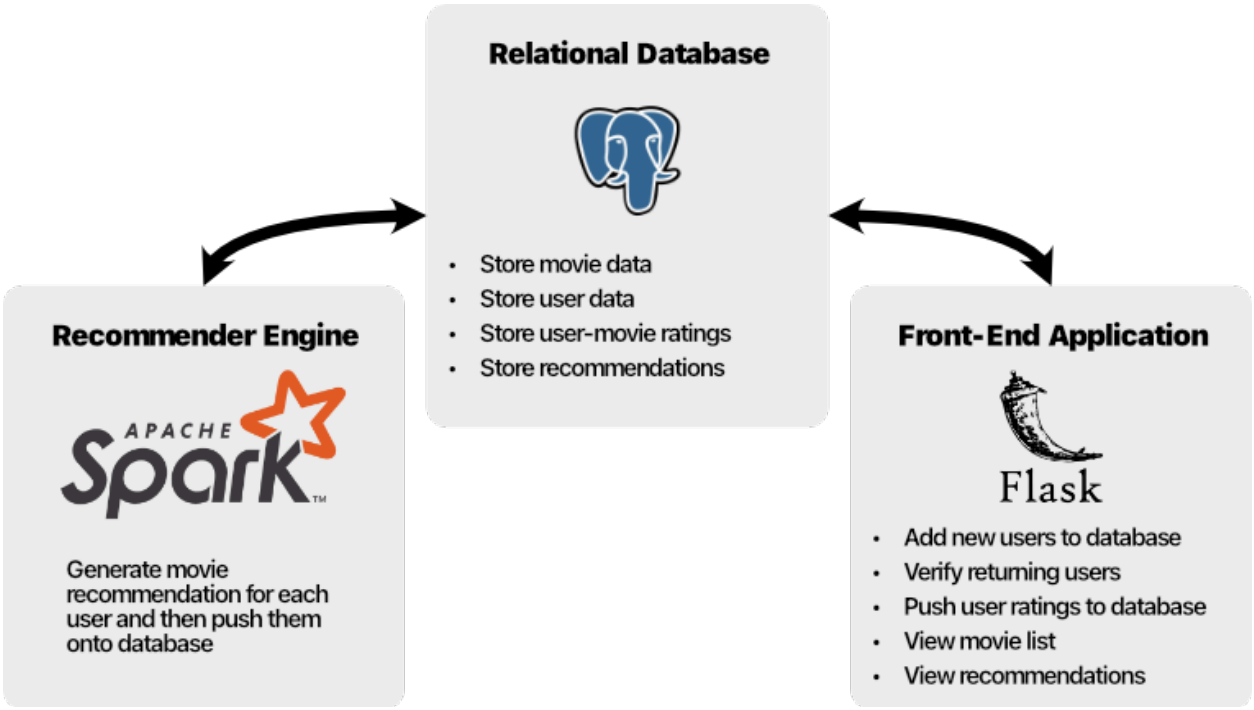

Figure 4 Overview of the relationship between each component of Curatorem where the arrows depict communication

6.3.1 LOGIN/REGISTER

When first opening the application, the user is required to either register their credentials or login with their registered credentials, allowing them to access Curatorem. When a user clicks the Sign-Up button, they are guided to the REST endpoint `/sign_up` which triggers a GET request to the application server which triggers the GET sign_up function which fetches the sign-up form. The sign-up form is where the user may input their email address, username and password (and confirmation password). Once a user finishes the form, they may click the submit button on the registration page, where a POST request is sent to the application server and the POST sign_up function is triggered checking the received request. The sign_up function houses a series of if statements that check whether the inputted information in the registration form abide by the correct standards and if all the information is correct, a User object is created with the information fetched from the form and is added onto the User table on the database. However, if some of the information fails to meet the set standards, an error is thrown in the form of an error banner that communicates to the user what the issue is.

The Flask framework has a login package that houses a set of functions and classes that streamline the configuration and set up process of the login system. When a user accesses the login page they navigate to the `/login` REST endpoint which implicitly sends a GET request to the application server, requesting an instance of the login page. When the user attempts to log in, a POST request

is sent to the application server and `login` function is triggered. The login function then queries the database for a user with the email address that was used to login, checking if that user exists in the database. If the user is found, the login function securely checks whether the inputted password hash matches the one on the database using the `check_password_hash` function provided by in the Werkzeug python library. If the password hashes match, the user may access Curatorem, otherwise an error prompt is raised communicating to the user what the error is.

6.3.2 MOVIE RATING SYSTEM

The other major functionality provided by the front-end Flask application is the movie rating system. The movie rating system allows each user to rate movies under their personal account. When a user selects a movie, they are transferred to the REST endpoint `"/moviedetails/<movie_id>"` where movie_id is the ID of the movie the user wishes to rate. Accessing this endpoint sends a GET request to the application server triggering the `movie_details()` function which simply queries the database for information about the selected movie and displays the information on the rendered page. This page also houses the rating selection box. When a user attempts to submit a rating, a POST request is sent to the application server triggering the `rating_handler()` function. The `rating_handler()` function checks whether the rated movie has been rated by the same user in the past, if so, it simply updates the previous entry in the database and prompts the user that the movie rating has been updated. If the user has not rated the given movie in the past, the `ratings_handler()` function creates a new entry in the ratings table.

6.3.3 RECOMMENDATION ENGINE

The recommendation engine is the core of the Curatorem project. With the given set up of the application, ensuring that new user recommendations were incorporated into my collaborative filtering recommendation engine was of utmost importance. The recommendation engine is powered by the Alternating Least Squares algorithm, this is due to 3 main reasons: generating recommendations using the data provided in the MovieLens dataset using the collaborative filtering approach is most optimally achieved using matrix factorization and dimensionality reduction. The ALS algorithm is computationally efficient allowing for a form of future proofing should the dataset increase in size. PySpark provides clear documentation surrounding it's ALS function, and allows for distributed computing, further increasing efficiency.

The recommendation process consists of 5 general steps: Importing the data from the PostgreSQL database, setting up the ALS model, fine-tuning the model parameters, fitting the model to the training dataset and finally generating movie recommendations for all users.

First, a JDBC connection must be established with the PostgreSQL database that houses the ratings table (the table that links the users to the movies with user-movie ratings). This allows for access to all of the data from the ratings table in the Databricks notebook. Once a connection is established, we must build out a PySpark dataframe schema for the ratings table contents assigning the appropriate datatypes to each column. This allows the data fetched from the database to be converted into a PySpark dataframe and read in the notebook. After the ratings are imported into as a PySpark dataframe we can start setting up the ALS algorithm which relies on the imported data to make movie rating predictions for each user registered in our database. Using the PySpark ALS function, this process becomes trivial. We must first divide the original dataframe into 3 'sub-

dataframes': 60% of the rating data is taken for training the model, 20% for validation and 20% for testing. The validation dataset is necessary to the fine-tuning process of the model parameters while the test dataset is only used to compute the root mean square error (RMSE) on our final, optimized model. Utilizing the PySpark ALS, evaluation and tuning functionality that is part of the `pyspark.ml` package we can initialize a base ALS model where we must assign our model components to their respective column names in the ratings table and set some base model parameters.

```
from pyspark.ml.recommendation import ALS
from pyspark.ml.evaluation import RegressionEvaluator
from pyspark.ml.tuning import TrainValidationSplit, ParamGridBuilder

als_dt = ALS(maxIter=5, regParam=0.25, userCol="user_id", itemCol="movie_id",
ratingCol="score", coldStartStrategy="drop", nonnegative = True, implicitPrefs =
False)
```

Figure 5 Code initializing the base ALS model with 5 maximum iterations, 0.25 initial regularization parameter, setting the cold start strategy to drop, establishing that the ratings must be non-negative and disabling implicit preference

With the base ALS model set, we can proceed to fine-tune our model in-order to obtain the best parameters and rank for our ALS computations. Hyper-parameter tuning is a highly recurring task in many machine learning projects, for this reason I utilized a tuning function written by GitHub user Kevin Liao [27]. This function performs a grid search to select the best model based on the RMSE value of hold-out data. The function takes arguments: training data, validation data, max number of iterations, float array of regularization parameters and integer array of ranks. For each rank, we iterate over all of the inputted regularization parameters training the model, generating predictions, comparing the predictions to the validation data and finally calculating the RMSE value for each parameter configuration. This allows for an automated tuning process that returns the parameters that generate the smallest RMSE value. We then use these parameters to build our final optimized model and fit our training dataset to it.

The final step is to generate recommendations for all of the users in our database. Conveniently, PySpark houses a `recommendForAllUsers()` function that given N integer value, generates N recommendations for each user as a dataframe. However, before pushing the output of the `recommendForAllUsers()` function onto the PostgreSQL database, we must first filter out movies that the user has already watched. This is done by checking whether the user has previously rated a recommended movie, if so, it is truncated from the final list which is then pushed onto the PostgreSQL database as the Recommendation table.

7 TESTING

This section is divided into four sections: UI testing, recommendation engine testing, integration testing and system testing.

7.1 UI TESTING

Following the set requirements, it was crucial to develop a responsive, intuitive and simple to use UI ensuring a good user experience. For this reason, I utilized Adobe XD, a vector-based user experience design application to plan out the application structure and flow prior to coding. Using Adobe XD proved of very useful, allowing for the efficient creation and modification of early mockups of the application structure as well as user interaction flow by leveraging its prototyping service.

7.1.1 LAYOUT DESIGN TESTING

With the functional requirements set, I had clear guidelines to follow for the creation of the initial version of the user interface design. Using Adobe XD, I started the construction of the possible layouts of each page. Early-stage UI development allowed me to show my mockups to potential users and get valuable feedback, after which the user interface mockups were adjusted accordingly. The combination of early-stage UI development and user-feedback allowed for a fine-tuned page layout that could efficiently be implemented later on, during the coding phase of the project. Figure 6 shows a high-precision mockup of the profile page which was iteratively updated until it sufficiently addressed user feedback. Figure 7 is the later implementation of the mockup which was also examined by a different set of users in-order to obtain feedback on if the final implementation abides by the mockup interface features.

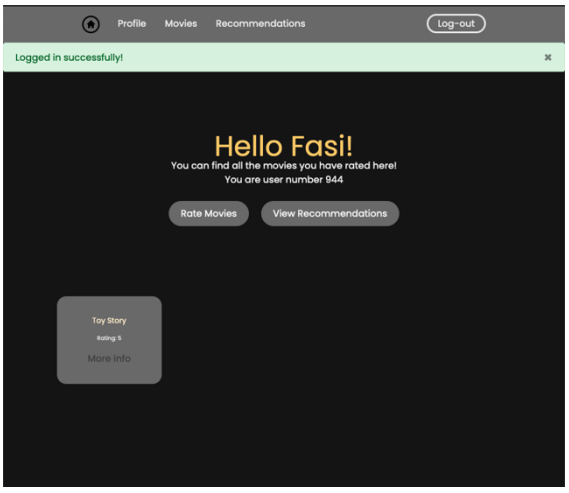

Figure 6 Final implementation of profile page

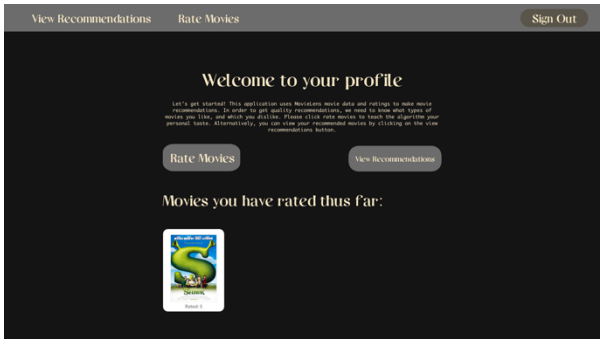

Figure 7 Final Adobe XD Mockup of profile page

7.1.2 USER INTERACTION FLOW TESTING

Another powerful feature of Adobe XD is prototyping. Prototyping in Adobe XD allows for the simulation of the user-interaction flow of the webpage at the early design phase. Being able to simulate examples of navigation allowed for the involvement of users at an even deeper level of the final product design. By allowing the users to navigate through a dynamic mockup of the application, I was able to gain feedback on the flow of the application early on. Through this

process, it was decided that the implementation of an introductory welcome page would be of benefit to new users more so than a hinderance. This form of user-based testing further aided the design process by allowing me to develop the best possible navigation paths based on user feedback.

7.1.3 Responsiveness testing

One of the major requirements set for the application was to have the user-interface be usable at any screen size. The development of a web application's user interface is a long and tedious process, that can fundamentally only be tested through trial and error. However, by using Google Chrome as my testing browser, I was able to quickly simulate screen sizes of various popular devices as well as adjust CSS code on the fly using the in-built Chrome DevTools. However, to ultimately be sure that the user interface is viewable on various devices I decided it was best to test it on different devices. To do this, I configured my Heroku deployment platform to publish a testing branch of the GitHub repository, which assigned a domain name for the application that was accessible publicly. This process ensured that the user interface of the application was designed in a manner that accommodates various screen sizes.

7.2 Recommender engine testing

Throughout the years of development of recommender engines, offline metrics have been used to indicate various features of these systems, such as accuracy, diversity and novelty of the generated predictions. It is important to preface this by highlighting that offline metrics simply act as indicators, while the real evaluator of a recommendation system's suggestions is the user. However, testing plays a major role in the recommender system pipeline and must not be ignored if one hopes to develop a usable system. For Curatorem's recommender system, I chose to use the root mean square error evaluation metric. RMSE is a type of error calculation that allows one to numerically gauge how close two values are to each other. For a recommender system these two values are the predicted rating for a movie and the actual user rating. RMSE is a good indicator of the accuracy of the predictions generated by the system. The closer the RMSE value to 0 the more accurate the predictions of a model are. In my testing I utilized a function that built the model multiple times with different parameters, and then calculated the RMSE value of each model. This allowed me to compare different model parameters and decide which one produced the lowest RMSE value.

7.3 Overall system testing

System testing examines the overall functionality of an integrated system to ensure that it meets the functional specifications. The process of testing a completed system requires using the system in a manner that is as close as possible to how the user would. As Curatorem is required to be used on different devices, it was important for me to utilize different hardware to test it. As stated in section 7.1.3, I configured my Heroku deployment platform to deploy a test branch of the GitHub repository to be able to access early versions of the application on different devices. This allowed for a thorough system testing process to take place, whereby a different navigation paths were followed on a desktop browser and a mobile browser. By iteratively doing this I was able to locate any issues in the functionality of the completed application and address them.

8 PROJECT EVALUATION AND CONCLUSION

After finishing Curatorem, it was critical to review the final product. Throughout the project, self-reflection and user feedback were combined to achieve evaluation at each stage of the development lifecycle. Naturally the final product of any development cycle may differ slightly from the original idea of the product. This may be attributed to a variety of internal and external factors; therefore, it is important to discuss and explain any deviations from the original project specification and their causes.

Having clarified this, I believe it is accurate to say that the final version of Curatorem reasonably satisfies all the functionalities stated in the above project specification. Involving potential users at the design stages and later on in the final stages of development has allowed for an end-product that is functional and serves its original purpose in a manner that provides the user with a good experience. This section follows the structure of Section 4: Evaluation criteria, describing the final state of each component with an added sub-section for potential improvements.

8.1 LOGIN/REGISTRATION

The login/registration system serves two main purposes: to authenticate users and to separate users in a means that the computer understands. When a new user would like to access the services of Curatorem, they must first register. The registration page is a minimal form requiring 3 pieces of information: an email, a username and a password. The most important being the email and the password. The login page follows the same design cues as the sign-up page, in where it is two simple input fields that ask the user for their email address and password. If the entered credentials are correct, the user is able to access their specific profile and the rest of Curatorem's functionality. The final implementation of the login/registration system serves its purposes in a manner that allows the application as a whole to function appropriately.

8.2 RECOMMENDATION SYSTEM

Accessing the recommendations page shows the user their respective movie recommendations if they have rated enough movies. The recommendation system follows a collaborative filtering through matrix factorization approach that utilizes the alternating least squares algorithm. The evaluation criteria set in section 4 states that the recommender system must provide relevant suggestions with a low RMSE. The final model RMSE value was around 0.9, which after looking at the suggested movies for a variety of different users seems to be recommending movies appropriately, fulfilling the evaluation criteria in this respect. The other set criterion was that the system must generate different suggestions for each user, this has also been achieved successfully by utilizing the `recommendForAllUsers()` PySpark function.

8.3 USABILITY

The usability criteria were set as the following: The graphical user interface of the application must prove to be easy to use. The interaction flow must be smooth and intuitive. The application should be free of major bugs. The application must be responsive and functional on varying screen sizes. When logging into Curatorem, the user is greeted with their profile page that clearly presents all

of the movies they have rated. The navigation bar or the 'Rate Movies' & 'View Recommendations' buttons allow them to quickly access all of the other functionality of the application in one click. This creates simple, direct and smooth flowing navigation of the application, fulfilling the first two criteria. As of final testing, I have not discovered any major application breaking bugs, and the application functioned as expected. I utilized CSS viewport breakpoints to adapt my application's user interface to different screen sizes, and through final testing I deemed it to be sufficiently responsive.

8.4 PERFORMANCE

Performance of the application is the last evaluation point. The criteria were set as the following: Users can navigate the application in a timely manner. The recommender system generates updated recommendations on a regular basis. By using a python web-framework, one is given the advantage of using a powerful server to do all the heavy computations and web-page rendering. The combination of a server-side loading and a simple user interface, the navigation performance of the application was sublime. The final configuration of the recommender system yielded hourly recommendation updates to all users, fulfilling the evaluation criterion.

8.5 POTENTIAL IMPROVEMENTS

While the final iteration of the project fulfills all of the evaluation criteria, some areas can be improved, and some additions can be made to increase the overall functionality.

8.5.1 BROWSING

The current movie browsing service accomplishes the fundamental function of a movie browser. However, I believe that this can be improved upon in two areas: more browsing options and a filtration system. In earlier versions of Curatorem a search function was indeed implemented and mostly functional. However, I was not satisfied with its performance. Upon further research I discovered an alternative search method that may have worked, but it was too financially straining to operate. A dedicated search function would have been a useful tool to allow users to locate specific movies in a more efficient manner than utilizing the browser's in-built search function. Another addition that I believe would have improved upon the current movie browsing system is the ability to filter the movies by various constraints.

8.5.2 ADD NEW MOVIES TO THE DATABASE

One area that I overlooked in the early conceptualization of Curatorem is the limitation created by using a fixed size dataset. The final version of the application only allows users to rate movies in the database. This is a limitation that could have been overcome by allowing users to push new movies to the database and presenting movies not limited to the movies in the database.

8.5.3 FREQUENCY OF RECOMMENDATION GENERATION

As briefly stated earlier the final configuration of the recommender system suggestion generation was set at an hourly update rate. Ideally, this frequency would be at a much higher rate. However, increasing this rate would be very financially straining as it is computationally intensive task that when utilizing a platform as a service quickly increases in financial cost.

8.6 CONCLUSION

Developing Curatorem has been a very informative and enlightening experience. It has allowed me to become familiar with the entire development process of a Python web-application. Starting from an idea and going through the process of researching around the topic of recommender systems, studying various approaches and technologies that may be used to accomplish my goals, and finally implementing my goal and publishing it as a complete application. Early on in the project lifecycle it was important for me to understand the state-of-the art of the field of recommender systems, as this would play a guide for how I would approach developing a recommender system of my own and set goals that I must meet. During the design stages of the application, I made sure to abide by the user-centered design approach by involving potential users in the creation of the user-interface through an iterative mockup process where user feedback was received and used to fine-tune the final structure of the pages as well as the navigation flow of the entire application.

Finally, I am happy to have had this experience. I believe that Curatorem has accomplished the goals I set out to achieve and fulfilled its key requirements of being a project that would allow me to delve into the world of data science and solve the problem of movie oversaturation.

References

[1] F. Ricci, L. Rokach, and B. Shapira, "Recommender Systems Handbook," in *Recommender Systems Handbook*, vol. 1–35, 2010, pp. 1–35.

[2] A. Rajaraman and J. D. Ullman, *Mining of Massive Datasets*. USA: Cambridge University Press, 2011.

[3] G. Lumpkin and G. G. Dess, "E-Business strategies and internet business models: How the internet adds value," 2004.

[4] S. S. Iyengar and M. R. Lepper, "When choice is demotivating: Can one desire too much of a good thing?," *J. Pers. Soc. Psychol.*, vol. 79, no. 6, p. 995, 2000.

[5] J. B. Schafer, J. Konstan, and J. Riedl, "Recommender systems in e-commerce," in *Proceedings of the 1st ACM conference on Electronic commerce*, 1999, pp. 158–166.

[6] Motion Picture Association, "2019 THEME Report," 2019. Accessed: Mar. 23, 2021. [Online]. Available: https://www.motionpictures.org/research-docs/2019-theme-report/.

[7] Netflix, Inc., "FINAL Q4-19 Shareholder Letter," 2019. Accessed: Mar. 23, 2021. [Online]. Available: https://s22.q4cdn.com/959853165/files/doc_financials/2019/q4/FINAL-Q4-19-Shareholder-Letter.pdf.

[8] The Walt Disney Company, "Global number of Disney+ subscribers 2021," *Statista*, 2021. https://www.statista.com/statistics/1095372/disney-plus-number-of-subscribers-us/ (accessed Mar. 23, 2021).

[9] C. C. Aggarwal, *Recommender systems*, vol. 1. Springer, 2016.

[10] Google LLC, "Collaborative Filtering Advantages & Disadvantages," *Google Developers*. https://developers.google.com/machine-learning/recommendation/collaborative/summary (accessed Mar. 22, 2021).

[11] B. Lika, K. Kolomvatsos, and S. Hadjiefthymiades, "Facing the cold start problem in recommender systems," *Expert Syst. Appl.*, vol. 41, no. 4, Part 2, pp. 2065–2073, Mar. 2014, doi: 10.1016/j.eswa.2013.09.005.

[12] G. Koutrika, "Modern recommender systems: from computing matrices to thinking with neurons," in *Proceedings of the 2018 International Conference on Management of Data*, 2018, pp. 1651–1654.

[13] C. Sammut and G. I. Webb, *Latent Factor Models and Matrix Factorizations*. Springer US, Boston, MA, 2010.

[14] Simon Funk, "Netflix Update: Try This at Home," 2006. https://sifter.org/~simon/journal/20061211.html (accessed Nov. 15, 2020).

[15] J. Bennett and S. Lanning, "The netflix prize," in *Proceedings of KDD cup and workshop*, 2007, vol. 2007, p. 35.

[16] E. Kangas and T. Kinnunen, "Applying user-centered design to mobile application development," *Commun. ACM*, vol. 48, no. 7, pp. 55–59, Jul. 2005.

[17] M. Kurosu and K. Kashimura, "Apparent usability vs. inherent usability experimental analysis on the determinants of the apparent usability," vol. 2, pp. 292–293, Jan. 1995.

[18] W. E. Hick, "On the Rate of Gain of Information," *Q. J. Exp. Psychol.*, vol. 4, no. 1, pp. 11–26, Mar. 1952, doi: 10.1080/17470215208416600.

[19] NNgroup, *Jakob's Law of Internet User Experience*. 2017.

[20] "The MovieLens Datasets: History and Context: ACM Transactions on Interactive Intelligent Systems: Vol 5, No 4." https://dl.acm.org/doi/10.1145/2827872 (accessed Apr. 17, 2021).
[21] S. Balaji and M. S. Murugaiyan, "Waterfall vs. V-Model vs. Agile: A comparative study on SDLC," *Int. J. Inf. Technol. Bus. Manag.*, vol. 2, no. 1, pp. 26–30, 2012.
[22] "Welcome to Flask — Flask Documentation (1.1.x)." https://flask.palletsprojects.com/en/1.1.x/ (accessed Apr. 19, 2021).
[23] P. G. D. Group, "PostgreSQL," *PostgreSQL*, Apr. 19, 2021. https://www.postgresql.org/ (accessed Apr. 19, 2021).
[24] "PySpark Documentation — PySpark 3.1.1 documentation." https://spark.apache.org/docs/latest/api/python/index.html (accessed Apr. 19, 2021).
[25] "Web Frameworks." https://www.fullstackpython.com/web-frameworks.html (accessed Apr. 19, 2021).
[26] "Deployment." https://www.fullstackpython.com/deployment.html (accessed Apr. 20, 2021).
[27] L. Kevin, "A function for ALS hyper-param tuning," *GitHub Gist*. https://gist.github.com/KevinLiao159/9f69049d6d3d8a096c0ea08dbc29591b (accessed Apr. 24, 2021).