**CS673 Software Engineering** 

**Team 4 - RecFlix**

**Software Design Document**

| Team Member | Role(s) | Signature | Date |
| --- | --- | --- | --- |
| Alex Kolbin | Configuration Leader | *AK* | 05/28/2022 |
| Rui Mao | Design and Implementation Leader | *RM* | 05/29/2022 |
| Ellie Nerney | Project Manager | *EN* | 05/30/2022 |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

**Revision history**

| **Version** | **Author** | **Date** | **Change** |
| --- | --- | --- | --- |
| **1** | **Alex Kolbin** | **05/28/2022** | **Cloud Architecture + Infrastructure Security** |
| **2** | **Rui Mao** | **05/29/2022** | **Introduction + Matrix Factorization Algorithm + Neural Network Architecture** |
| **2** | **Ellie Nerney** | **05/30/2022** | **Database Design** |
| **3** | **Ellie Nerney** | **06/13/2022** | **Introduction** |

[Introduction](#_87t9hln2vjz0)

[Software Architecture](#_3ipvmjgn6clp)

[Database Design (if applied)](#_ky60nv8suxxm)

[Security Design](#_x18fj36s1121)

[Design Patterns](#_8r4agmh10eya)

[Key Algorithms](#_mtfbusfb0eq3)

[UI Design](#_7ucksmkf6rzx)

[Classes and Methods](#_pupof9lqa5t8)

[References](#_15tmymhipvdv)

[Glossary](#_8n34lvocupub)

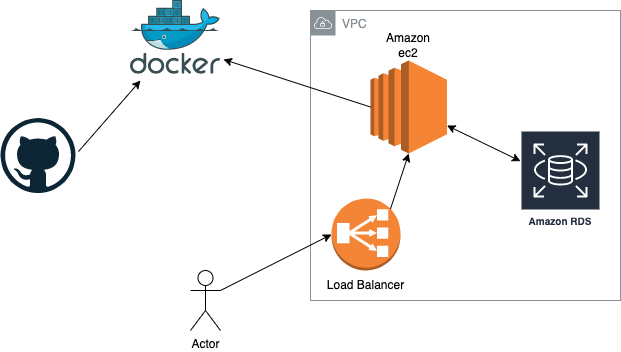
# Introduction

Our goal is to create a Movie Recommendation system which utilizes numerous technologies. Hosted on AWS, with an Angular front-end, our system allows a user to give a small sample input, which gets relayed to the Django back-end for the Neural Network (written in Python) to make a prediction on what other movies the user would enjoy.

# Software Architecture

In this section, you will describe the decomposition of your software system, which includes each component (which may be in terms of package or folder) and the relationship between components. You shall have at least one diagram to show the whole architecture of . The interface of each component and dependency between components should also be described. If any framework is used, it shall be defined here too.

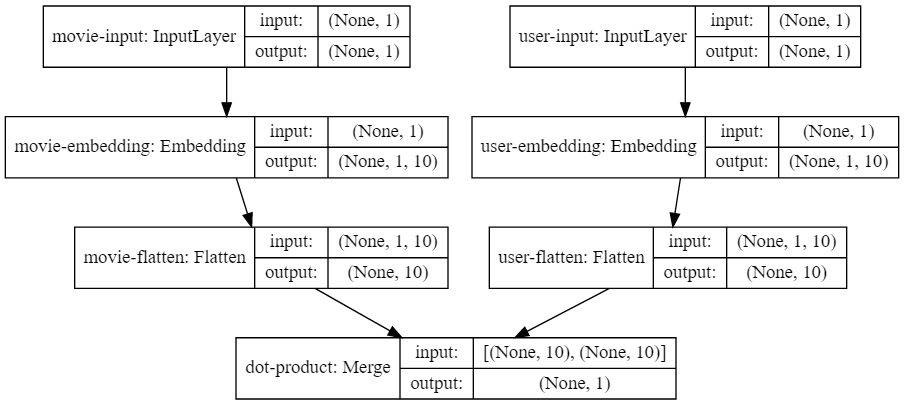
* + **Cloud Architecture:**The following diagram outlines the cloud architecture of our application, and how each component is connected. In summary, our application code is stored in GitHub. Anytime a PR is created, security scans will run on the code. When the code is merged into main, a new Docker image will be created using the updated code. The image is sent to Dockerhub where it is then pulled by our application server and runs on a new container which gets exposed to the public internet. The running application interacts with our RDS PostgreSQL database which exists in the same VPC as the web server. Users reach our web server by reaching the load balancer DNS. The load balancer redirects their traffic to the web server where the user can interact with the application. Behind the scenes, the server will be interacting with the RDS database using REST API requests.

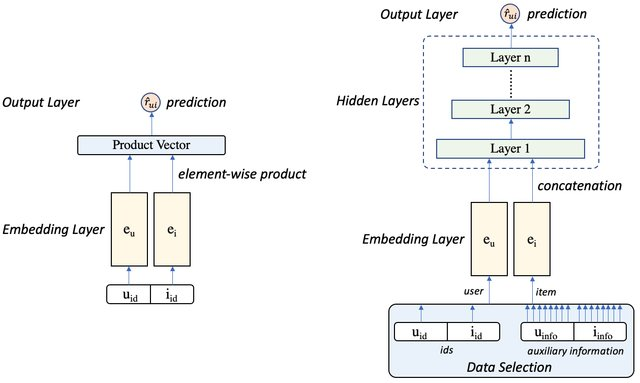


* + **Neural Network Architecture:**

The neural network model has a layered architecture. We use a feed-forward neural network to handle both collaborative filtering and auxiliary information.

Since the recommender system expects inputs from both users and movies, the model flattens both inputs to layers with appropriate shapes and concatenates them. Then the model uses the neural collaborative filtering layers to find the correlations between the user latent vector and the movie latent vector.

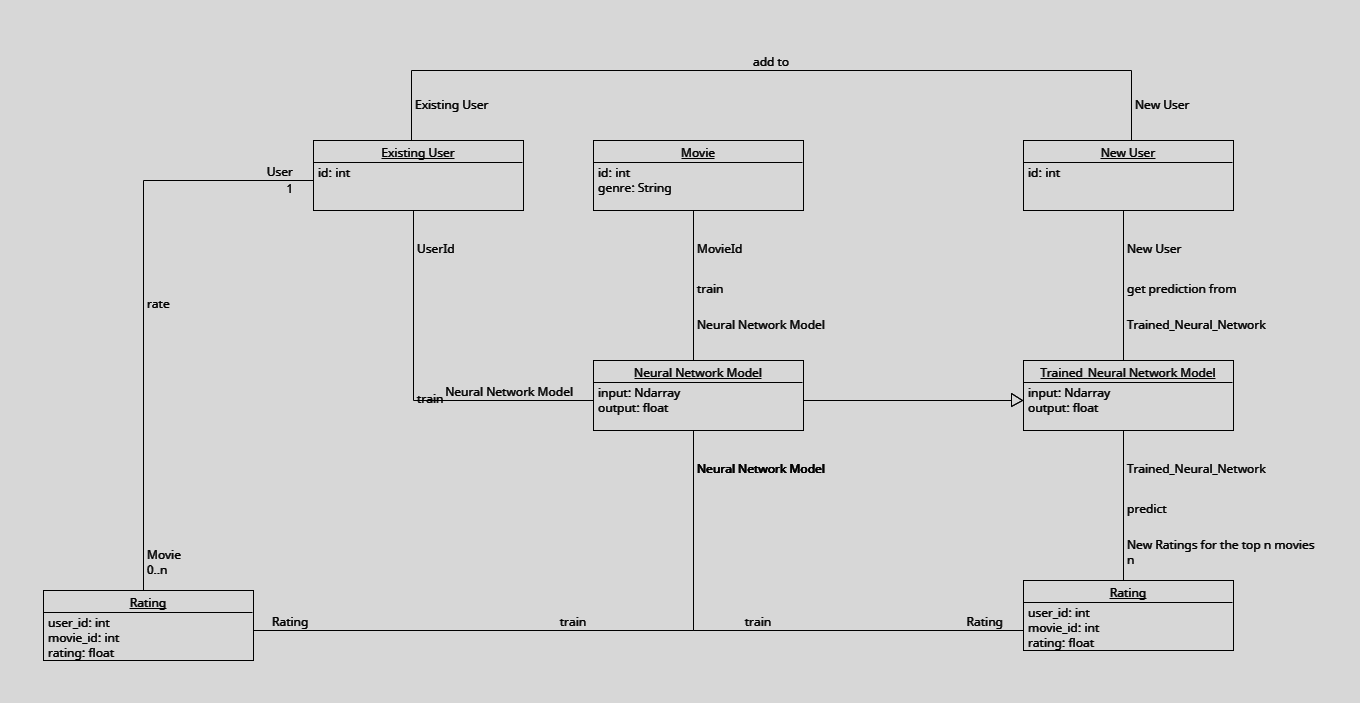


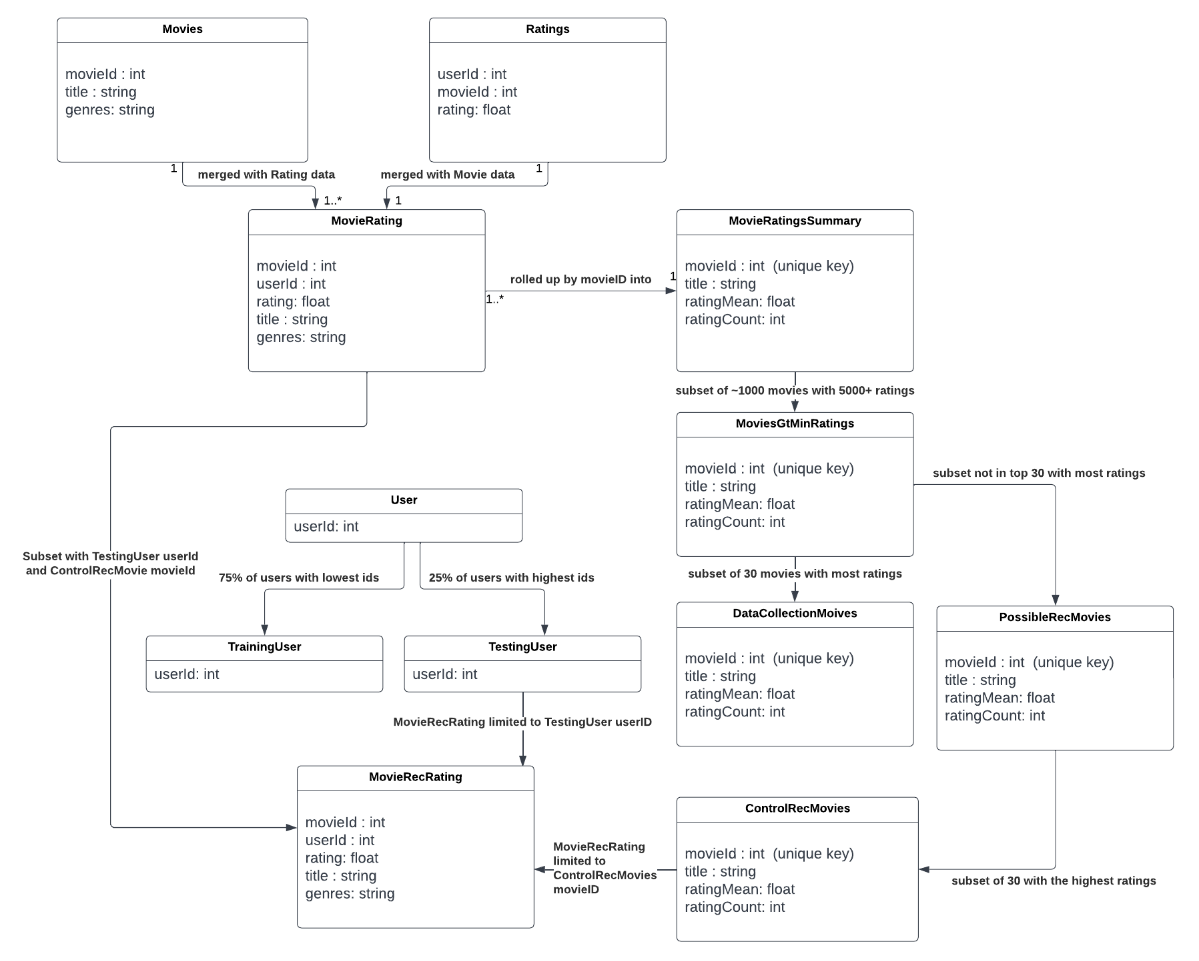
****

# Class Diagram

In this section, you will provide a detailed description of each component (or package) and use one or multiple class diagrams to show the main classes and their relationships in each component.

Class Diagram for the Neural Network Model:

Class Diagram for Control Model and Neural Network Testing:



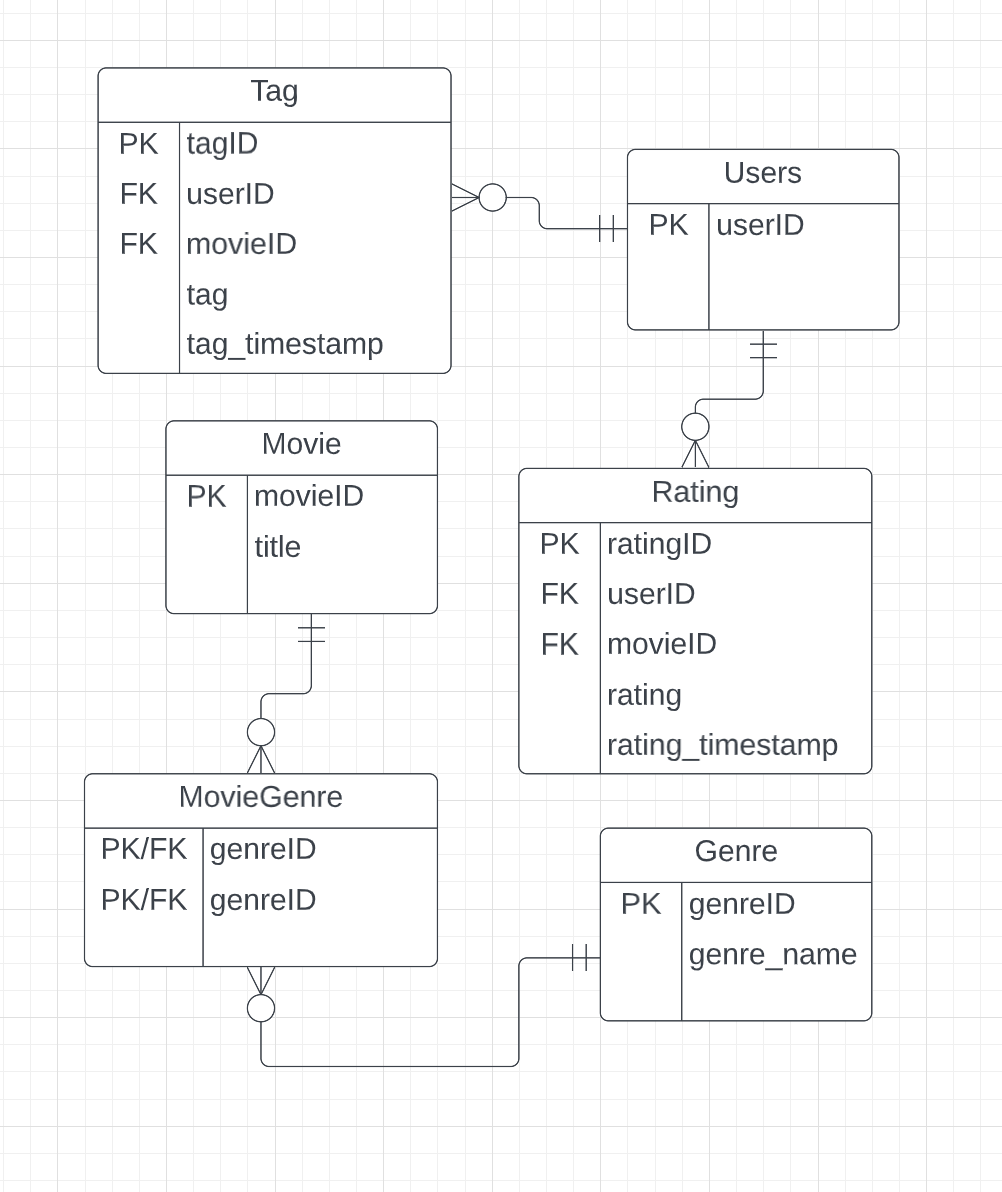
# UI Design

To be addressed in the next iteration.

# Database Design

The Database we’re using is written in PostgreSQL, and is hosted on our AWS Server. Currently there are 6 tables, just enough to house the data from our starting dataset from Kaggle and nothing more. More and more tables and relations will be added as necessary or requested.

**ERD Design**



**Validation/Data Quality Measures**

When ingesting the source data into the Database, it is necessary for the Users table to be built out from scratch, as it was not provided, and foreign keys depend on that key existing in its respective table. This is done by inserting the data through *Stored Procedures* instead of direct insert statements. This simple line of SQL that is called at the beginning of each procedure:

INSERT into Users(userid)

VALUES (userid) ON CONFLICT DO NOTHING;

ensures that all users referenced in any Fact Table are inserted into the Users table.

In Iteration 2 this will be expanded upon to make sure that the Database is covered with comprehensive data quality measures for all current and future tables through the use of triggers and stored procedures.

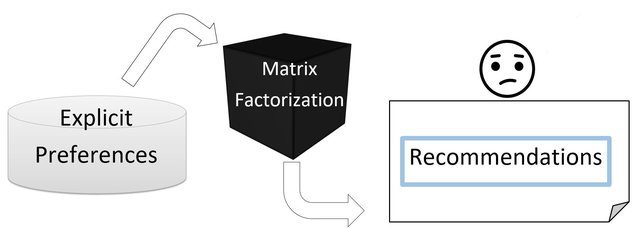
# Security Design

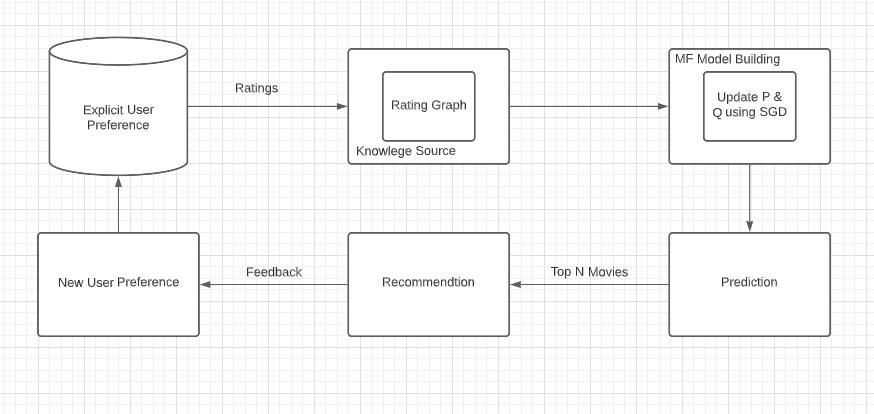
* + **Infrastructure Security:**Our GitHub repository is public, but we have enabled several security measures to protect us from both external and internal factors. There are protection rules on the main branch which disables direct pushes (unless from an administrator), and requires approval from our team for PRs. Our application is only built from the main branch, so this ensures that anything built is known by the team. Additionally, no passwords, usernames, secret keys, or endpoints are stored in plaintext. We are using GitHub Secrets to retrieve these values securely in our Actions workflows. The Docker repository and RDS database are protected by a username/password combination. The ec2 instance requires a private key to SSH. For development purposes, port 22 is open to the public. In a real production environment this would be locked down. The AWS account where the ec2 and RDS exist has root MFA enabled so that no one is able to access root via the console except for the configuration leader.

# Business Logic and/or Key Algorithms

In this section, you shall describe any key algorithms used in your software system, either in terms of pseudocode or flowchart, or sequence diagrams

The key algorithm of the recommendation system are collaborative filtering and matrix factorization. The idea is to factorize the rating matrix into lower dimensional spaces (flatten layer) using a given number of latent features such that the dot product of two latent space representations approximates the the ratings of unseen items based on the similar users’ ratings. Besides, the system also collects the user preference data to reinforce the recommender system. After obtaining the new ratings, the new user preferences will be added to the database as part of the explicit preferences and the matrix factorization will calculate the ratings based on the new user preference data.





# Design Patterns

In this section, you shall describe any design patterns used in your software system.

# Any Additional Topics you would like to include.

# References

*Alshammari, M., Nasraoui, O., & Sanders, S. (2019). Mining semantic knowledge graphs to add explainability to black box recommender systems. IEEE Access, 7, 110563–110579. https://doi.org/10.1109/access.2019.2934633*

*Yıldırım, E., Azad, P., & Öğüdücü, Ş. G. (2020). Neural hybrid recommender: Recommendation needs collaboration. New Frontiers in Mining Complex Patterns, 52–66. https://doi.org/10.1007/978-3-030-48861-1\_4*

# Glossary