Stage 6:

CS411 Final Project Report

WEROCK007:

FLIXRECC

A MOVIE RECOMMENDATION GENERATOR AND TRACKER

TA Name: Kevin Pei
Aarushi Biswas (abiswas7)
Kartik Mehra (kartikm3)
Nikhil Sharma (nsharm40)
Yubin Koh (yubink2)

1. Please list out changes in directions of your project if the final project is different from your original proposal (based on your stage 1 proposal submission).
    a) First, we originally proposed to provide a Movies and TV shows recommendation based on the user's preferences of actors, genres, ratings, languages, and release date. We changed this functionality to be split up by two parts: filter by genre preference and filter by actor/actress name. The user can choose one of the eight genres, and the filtered list of movies and tv shows will be added to the user's "movies/tv shows recommended for you" list. Also, in order to provide the best recommendation, after the filtration based on the user preference, we added 5 best movies/tv shows according to the ratings.
    b) In our proposal, we were debating between two different datasets for our web application: IMDb and TMDb. We decided to go with IMDb, because it was more straightforward to download the datasets.
    c) We wanted to provide a drop down list in the search bar in a way that as the user types in the name of an actor/actress, we provide a list of possible names. We decided not to implement this, and instead, we used the search bar to look up a movie or tv show title name. Also, the user must type in the title name correctly in order for our application to fetch the data correctly.
    d) Our UI design changed a lot from the original proposal, mainly due to the implementation of Flutter. Also, we ended up not providing a movie poster as well as its description as it was not available in our database. Our overall web-application is text-based.

2. Discuss what you think your application achieved or failed to achieve regarding its usefulness.
    a) Our application was successful in providing a personalized recommendation list for Movies and Tv shows. We were able to perform a filtration based on specific user preferences - such as genres, ratings, and actors - and save it to each of the user accounts. This way, users can always log back in and update their recommendation lists. Furthermore, when an user creates an account, one must provide age. Based on the age, we also filter out the rated movies and shows,

because we wouldn't want to recommend rated shows to an underage user. Considering these functionalities, our application provides a personalized recommendation list successfully.

b) However, we believe that it would have been more entertaining and useful to provide a movie poster or trailer to the user before one can add it to the "to watched list." Since our application is all text-based, it has limitations on introducing what the movie or tv show is about.

3. Discuss if you changed the schema or source of the data for your application
   a) Our source of the data didn't change a lot except for the fact that we decided to stick with IMDb instead of TMDb. Our entities didn't change a lot either. As we originally proposed, we downloaded all of the datasets from IMDb, because they all contained entities that we needed for our own schema. However, as our UML diagram changed a lot (as described in the next question), it directly affected the schemas. Mainly because we had to get rid of three tables that we were considering to be our entity tables and add another entity table. Other than that, we did not change any of the pre-existing schemas by much. Our relational table schemas changed to the following:

RecommendedList:

RecommendedList(UserId : VARCHAR(50) [FK to UserProfile.UserId], EntertainmentId_: VARCHAR(50) [FK to Movies.Movie_ID or Tv.Tv_ID])

ToWatchList:

ToWatchList(UserId : VARCHAR(50) [FK to UserProfile.UserId], EntertainmentId_: VARCHAR(50) [FK to Movies.Movie_ID or Tv..Tv_ID])

AlreadyWatchedList:

AlreadyWatchedList(UserId : VARCHAR(50) [FK to UserProfile.UserId], EntertainmentId : VARCHAR(50) [FK to Movies.Movie_ID or Tv.Tv_ID])

And UserProfile schema also changed based on the suggestion of our TA to the following:

UserProfile:

UserProfile(UserId : VARCHAR(50) [PK], UserName : VARCHAR(50) [FK to UserLogin.UserName]])

The schema of the TV Shows table that we added is the following:

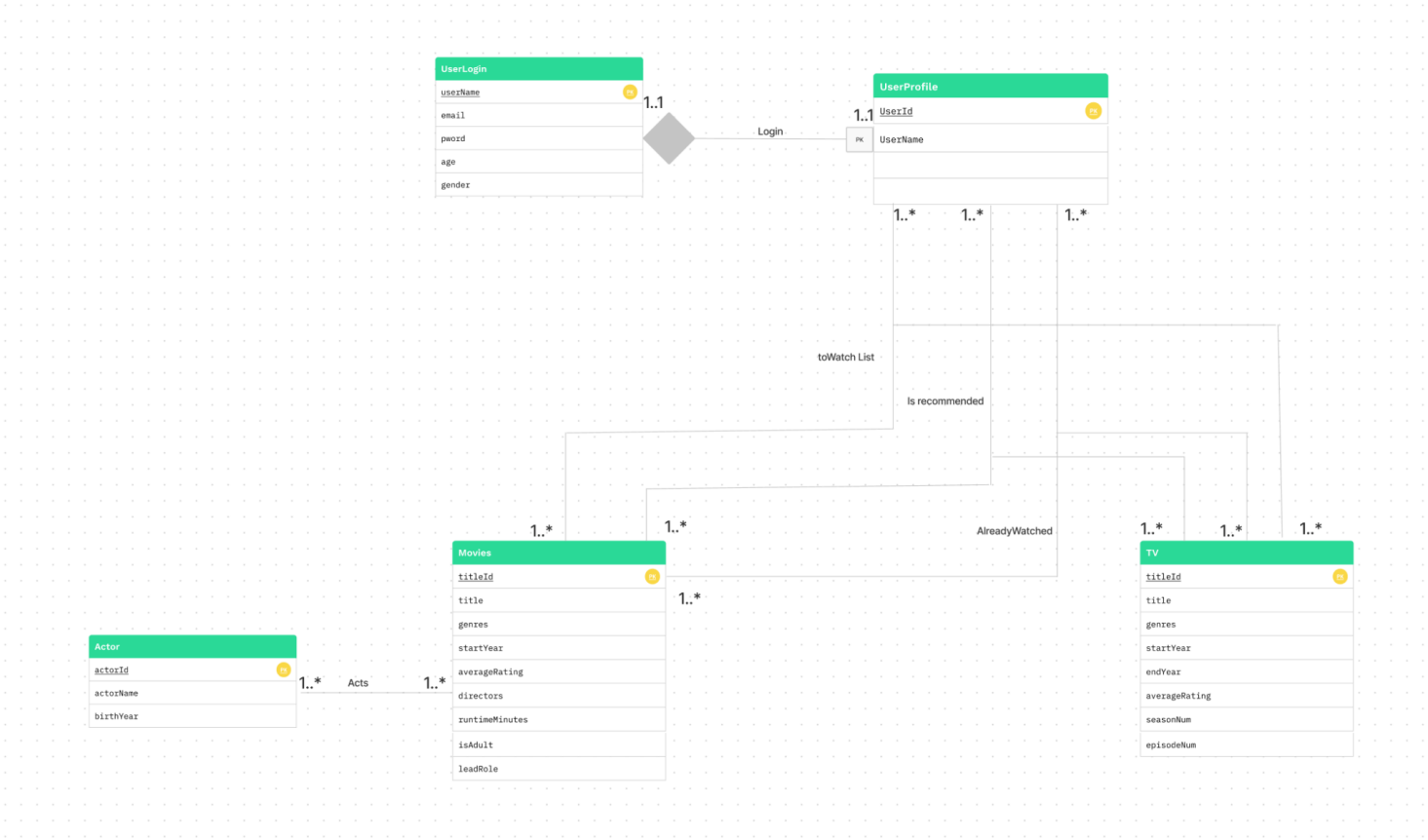
TV(titleId : INT [PK], title : VARCHAR(50), genres : VARCHAR(20), startYear : VARCHAR(20), endYear : VARCHAR(20), averageRating : DOUBLE, seasonNum : VARCHAR(20), episodeNum : VARCHAR(20))

The Acts Table which is the relational table between the Movies table and the Actors table

Acts(actorId : INT [PK], EntertainmentId : INT [FK to Movies.Movie_ID or Tv.Tv_ID])

4. Discuss what you change to your ER diagram and/or your table implementations. What are some differences between the original design and the final design? Why? What do you think is a more suitable design?
    a) We made major changes to our UML diagram from what we originally proposed. This was mainly due to the fact that our original diagram was not meeting the criteria or some of its implementation was incorrect. For example, we originally proposed to create a separate table for each of the toWatchList, isRecommended, and AlreadyWatched. However, there should be relations between UserProfile and Movies, and single attribute entities should not be entities. Also, because they're many-to-one relationships, they should not have their own tables. Furthermore, we added a TV table along with our original Movie table, because we were not meeting the requirements for the number of entities in our UML Diagram.

Updated UML Diagram



5. Discuss what functionalities you added or removed. Why?

   Before going into detail about the functionalities changed/added/removed from our original proposal, we would like to run through the functionalities of our current webpage. Our web page starts off with the login page where, as the name suggests, the user can login. The user is also presented with three other options related to their account – Create account, Update password, Delete Account. Once an user logins in, their username is stored in order to provide them with their respective entertainment lists – Recommended List, To Watch List and Already Watched List. On logging in, the user is presented with the homepage. There are 4 main elements to the homepage – Search, User Profile, Filter System and the most Liked Movies and TV. On clicking the

search button, the user is given the option to write the name of any movie they want to search and is returned the information about the movie such as its runtime, release date, whether it is adult or not, etc. Along with this, the user can also add that particular movie to their To Watch List by clicking an additional button. Next, when the user clicks on the user profile button, they are presented with buttons to access their 3 lists that were mentioned earlier. When viewing their recommended list, the user can add one of the movies to their To Watch List. Similarly, when viewing their To Watch List, the user can add one of the movies to their Already Watched List which will also be deleted from their To Watch List. The Recommendation List is populated by the Filter System available to the user in the homepage. There are two options – to filter by genre after viewing each genre's average runtime, or to filter movies and TV Shows by their favorite actor/actress names. The last element on the homepage is the most liked movies and TV shows which shows the users a ranking of the top 5 most watched movies/ TV shows with a badge (one of FIRE, MID, STANK) assigned to them depending on their popularity. There were quite a few functionalities added to our project which were not mentioned in our proposal. Firstly, the delete account, update password and search options were given to the user partially to take care of the CRUD applications and also partially because they make the webpage more realistic considering most movie recommendation generators do also have a search functionalities and most web pages who are storing your login information have a delete + update option. The idea of the lists and filtration were mentioned in our proposal with minor changes which we have talked about earlier. The last functionality that we added was the most liked/watched tables. We found the idea of presenting the top most popular movies to the user very alluring along with the fact that it would again make our web page more realistic. Furthermore, we incorporated our stored procedure into this specific functionality which worked out very well in the end.

6. Explain how you think your advanced database programs complement your application.
    a) Stored Procedure:

        Our application had many statistics about the movies and tv shows in our

databases from how long they run to how highly rated they are. This made us think of a metric which represents how well certain movies and tv shows are doing among our users. We then decided to use a stored procedure for this as these metrics will be the same for every user and will need to loop over all the users to get the values we need. We wanted to see how many times movies and tv shows were in people's 'towatch list', this would give us an idea of what is popular amongst our users and give others another set of movies and tvs to recommend. We gave a certain label to if the count was above a certain threshold which were 'FIRE', 'MID', 'STANK'. The way our stored procedure worked was that we had two cursors where one went over the movies and the other over the Tv shows. This looped over the towatch list and counted the amount of times that title was in the towatch list table and then gave them a label and stored it in the respective table we created within the stored procedure.

Our stored Procedure is as following:

```
DELIMITER //

CREATE PROCEDURE mostPopular()
BEGIN

DECLARE varMovieID varchar(256) ;
DECLARE varMovieName varchar(256);
DECLARE varMovieCount INT;
DECLARE varMoviePop varchar(256);

DECLARE varTvID varchar(256) ;
DECLARE varTvName varchar(256);
DECLARE varTvCount INT;
DECLARE varTvPop varchar(256);

DECLARE movieCursor CURSOR FOR
(SELECT m.titleId, m.title, COUNT(tw.UserId) as MovieCount
FROM ToWatchList tw JOIN Movies m ON m.titleId = tw.EntertainmentId
GROUP BY m.titleId);
```

```
DECLARE tvCursor CURSOR FOR
(SELECT t.titleId, t.title, COUNT(tw.UserId) as TvCount
FROM ToWatchList tw JOIN TV t ON t.titleId = tw.EntertainmentId
GROUP BY t.titleId);

DROP TABLE IF EXISTS moviePopTable;
DROP TABLE IF EXISTS tvPopTable;

CREATE TABLE moviePopTable (
movieName varchar(256) Primary Key,
movieCount INT,
moviePop varchar(256)
);

CREATE TABLE tvPopTable (
tvName varchar(256) Primary Key,
 tvCount INT,
 tvPop varchar(256)
);

OPEN movieCursor;
BEGIN

DECLARE loop_exit BOOLEAN DEFAULT FALSE;
DECLARE CONTINUE HANDLER FOR NOT FOUND SET loop_exit = TRUE;
cloop: LOOP

FETCH movieCursor INTO varMovieID, varMovieName, varMovieCount;

IF varMovieID = NULL THEN
LEAVE cloop;
ELSEIF loop_exit THEN
LEAVE cloop;
END IF;

IF varMovieCount > 4 THEN
SET varMoviePop = "FIRE";
ELSEIF varMovieCount >= 3 THEN
SET varMoviePop = "MID";
ELSE
```

```
SET varMoviePop = "STANK";
END IF;

INSERT IGNORE INTO moviePopTable VALUES(varMovieName, varMovieCount,
varMoviePop);
END LOOP cloop;

END;
CLOSE movieCursor;

OPEN tvCursor;
BEGIN

DECLARE loop_exit BOOLEAN DEFAULT FALSE;
DECLARE CONTINUE HANDLER FOR NOT FOUND SET loop_exit = TRUE;
cloop: LOOP

FETCH tvCursor INTO varTvID, varTvName, varTvCount;

IF varTvID = NULL THEN
LEAVE cloop;
ELSEIF loop_exit THEN
LEAVE cloop;
END IF;

IF varTvCount > 4 THEN
SET varTvPop = "FIRE";
ELSEIF varTvCount >= 3 THEN
SET varTvPop = "MID";
ELSE
SET varTvPop = "STANK";
END IF;

INSERT IGNORE INTO tvPopTable VALUES(varTvName, varTvCount, varTvPop);
END LOOP cloop;

END;
CLOSE tvCursor;

END;
```

b) Trigger:

The trigger was created in order to more easily filter movies from underage users. When a user indicates their age (the event), the trigger creates a hash value (the action) dependent on whether or not they are under the age of 17 (the condition) and stores it in a separate table. From this, we are able to filter films that are marked as strictly for adults more efficiently, and thus movies will not be inserted into their to watch lists that are marked as adult only. This allowed for much simpler and faster filtration of films as there was no need to consistently make comparisons checking if the user's age was above the threshold to allow adult content, as we could instead fetch the hash value and directly compare the first value of the id to the isAdult field in the Movies table.

The code for the trigger is as following:

```
DELIMITER //
CREATE TRIGGER CreateUser
        AFTER INSERT ON UserLogin
        FOR EACH ROW
BEGIN
        IF (NEW.age > 17) THEN
                INSERT INTO UserProfile(UserId, UserName)
VALUES(CONCAT("1", NEW.UserName), NEW.UserName);
        ELSE
                INSERT INTO UserProfile(UserId, UserName)
VALUES(CONCAT("0", NEW.userName), NEW.UserName);
   END IF;
END;//
DELIMITER ;
```

7. Each team member should describe one technical challenge that the team encountered. This should be sufficiently detailed such that another future team could use this as helpful advice if they were to start a similar project or where to maintain your project.

    a. Yubin: We struggled a lot in the early stage of the project regarding coming up with the UML diagram, mainly because we got confused with the technical terms, such as relations and entities. For example, we originally had a separate table for each of the toWatchList, isRecommended, and AlreadyWatched. However, as previously discussed in #4, we realized that they should be relations between UserProfile and Movies, as well as UserProfile and TV, instead of a single attribute. We also didn't know that many-to-one relationships should not have their own tables. Furthermore, we originally didn't have an entity table for the Actors, but as we wanted to provide a functionality where the user can search up for all the movies and tv shows a specific actor/actress acted in, we figured out that we need to create an entity table for the Actors. As we thought of our application in more detail and updated our UML diagram accordingly, we were able to resolve our confusions with relation and entity tables.

    b. Aarushi: One of our initial struggles was in implementing the database locally on our MySQL Workbench. Our plan was to extract the data from IMDB into our local database and then use those extracted tables to make our entity tables. We spent a lot of time doing this. Initially our data was not being able to be imported into the workbench because of bad data rows and so we even wrote a python script which takes care of those bad data. Once we got that, the actual importing of the data was also taking too long. We started with the smallest table, hoping to get an idea of how long it would take to import our largest table. However, even after limiting the smallest table to 2000 rows of data, it took well over 3 hours to import the entire table. However, we could not afford to clip all tables down to only 1000 rows

because that was messing up a lot of connections between the retrieved IMDb tables which would have further affected the creation of our entity tables. We tried various methods to do this by googling ways, such as LOAD DATA. However, nothing was proving to be successful. On contacting our TA about our concern of such long importing times, he suggested that we move to GCP which made everything much easier. The import time reduced significantly and so we could import all our tables after taking care of bad data (such as NULL values) using our python script.

c. Nikhil: Initially, we struggled with our choice of the Python framework that would be used to write Flixrecc. Our first choice was to use the Django full-stack framework and connect to an HTML based front-end via the Django REST framework to build an API. However, this proved to be a challenge, as none of us had experience with Django. After struggling with Django's model and migration system to connect to the MySQL database, we switched to the Flask micro framework. We found this much easier to set up, as we found connecting to the database by the SQLAlchemy library to be much more intuitive. The front-end was then built using the Flutter development kit instead, which proved to provide a much cleaner and user-friendly interface. Overall, the change to Flask and Flutter allowed us to write the application in a much more efficient manner.

d. Kartik: One issue we encountered which took us a lot of time to figure out was how to get the data the backend fetches sent to the frontend and vice versa. We were unsure of how to encode the data and also decode it so that we are able to extract the information we sent. We googled a lot about how to connect the backend and frontend using flask and flutter but to no avail. We couldn't really understand what format our data was in and what format it needed to be and hence didn't fully understand the error we faced. What really helped was using print statements to see how our data was being packaged and what format functions like jsonify, encode, decode were doing to our data. After realizing how our data was getting packaged and why the format error was coming we were able to much effectively

use stackover flow to help our case. We decided to package the data as key-value pairs where the key was the column name and tha value was the data under that column. Each dictionary represented a row and we then stored these rows in a list. It was easy to then send to our front end as our data could easily be made into a response(JSON format) which contained our data and the web server response which was ‘200’ and this is a convention which tells us that everything has gone smoothly.

8. Are there other things that changed comparing the final application with the original proposal?

a) We mentioned all the changes that we made compared to the original proposal. One thing to note is that we don’t provide any language information for each of the movies and tv shows opposed to our original proposal. This was because we didn’t realize that IMDb doesn’t provide language information. It only provides the language of the title, which we decided to remove because it was causing some issues when transforming the datasets to our own schema. So overall, the main functionalities which were changed are mentioned earlier in Q5. The changes in our entity tables and schema are mentioned under Q3 and Q4. Lastly, we did add many more SQL queries in addition to the one mentioned before, including some advanced SQL queries.

9. Describe future work that you think, other than the interface, that the application can improve on.

   For our current application, we are only using IMDb as our sole dataset. We can further improve our application by growing our current database by making use of additional datasets such as the IMDB Film Reviews data set and the Full MovieLens dataset on Kaggle to give users additional information about each movie and filters. The IMDB Film Reviews data set can be used to showcase reviews by other viewers which the users can use to decide whether they want to watch the movie or not. Furthermore, the Full MovieLens dataset can be used to provide the users with the poster of the movie as we originally had intended to but were unable to do, and also a brief description of the plot of the movie. We were also thinking that as our user base grows and our database grows, we can also segregate the movies and TV shows

according to countries or languages. Furthermore to make the user experience more friendly and adapted to their tastes, we were also thinking that it would be a good idea to make the users choose their favorite movies among a collection of choices to get an understanding of the users interests and then recommend them similar movies. This would require including a certain amount of machine learning elements in our web page.
Another few ideas we would implement in the future when our web app becomes bigger and contains more users are the following. Firstly, when we have many users from different countries we will move to our database and run our backend on the cloud and then decide to group data by countries and store data on local cloud servers in the countries so we can increase speed of the access to the data. We will also add premium services which allow certain users to pay and get more services from access to more movies/tv shows to more data about certain movies with maybe an option to view previews to even an eventual streaming service. When we decide to make these changes we might decide to use NOSQL databases because it is less rigid then SQL and will allow us to have different classes of users(depending on their subscription level) with different schemas next to each other. These users will be different from each other as certain ones will have more features than others.

10. Describe the final division of labor and how well you managed teamwork.

We believe that we had a pretty good division of labor between all the members and were able to work together smoothly. For all the checkpoints till Stage 3, we always got together to work, either in-person or on Discord. We usually did not have a lot of conflict of ideas but if either of the members ever disagreed on an idea, each would present why their idea would be suited better to our application and then we would come to a unanimous decision. After stage 3, when we started working on our application, we divided our work up between pairs. Kartik and Nikhil worked on the back-end whereas Yubin and Aarushi worked on the front-end. However, when we say divided up the work, it was mostly research divided up and writing the skeletal code. Aarushi and Yubin divided up the pages that they were supposed to work on. Kartik

and Nikhil worked on the back-end together that included getting data from our database using SQL queries that all four members had come up with together. They then worked together on packaging the data to the required format to send to the front-end. Once that was done, the entire group then got together to work on the connection between the front-end and back-end. Aarushi and Yubin worked on writing the front-end functions for unpackaging the data after understanding how the back-end works and how the data was packaged in the back-end. Similarly, there were some instances where the front-end had to send data to the back-end (such as in the create account operation) in which case Kartik and Nikhil understood how the front-end data was being packaged after which they wrote a compatible back-end. There were obviously instances where both sides had to adapt how they were packaging/unpackaging the data in order to make everything work seamlessly. However, all of this was done together, with each member and pair helping the other in doing research, figuring out errors and writing the code.