CS348 Project Report

**A)**

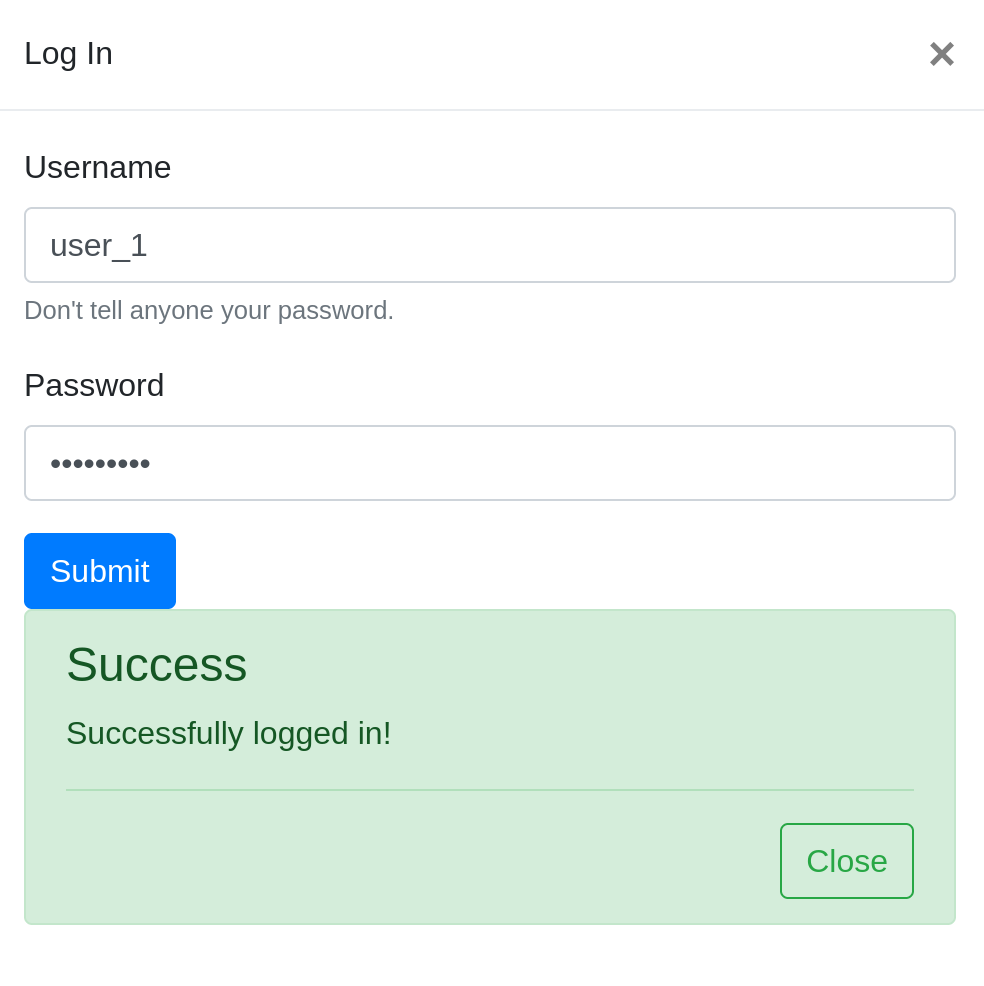
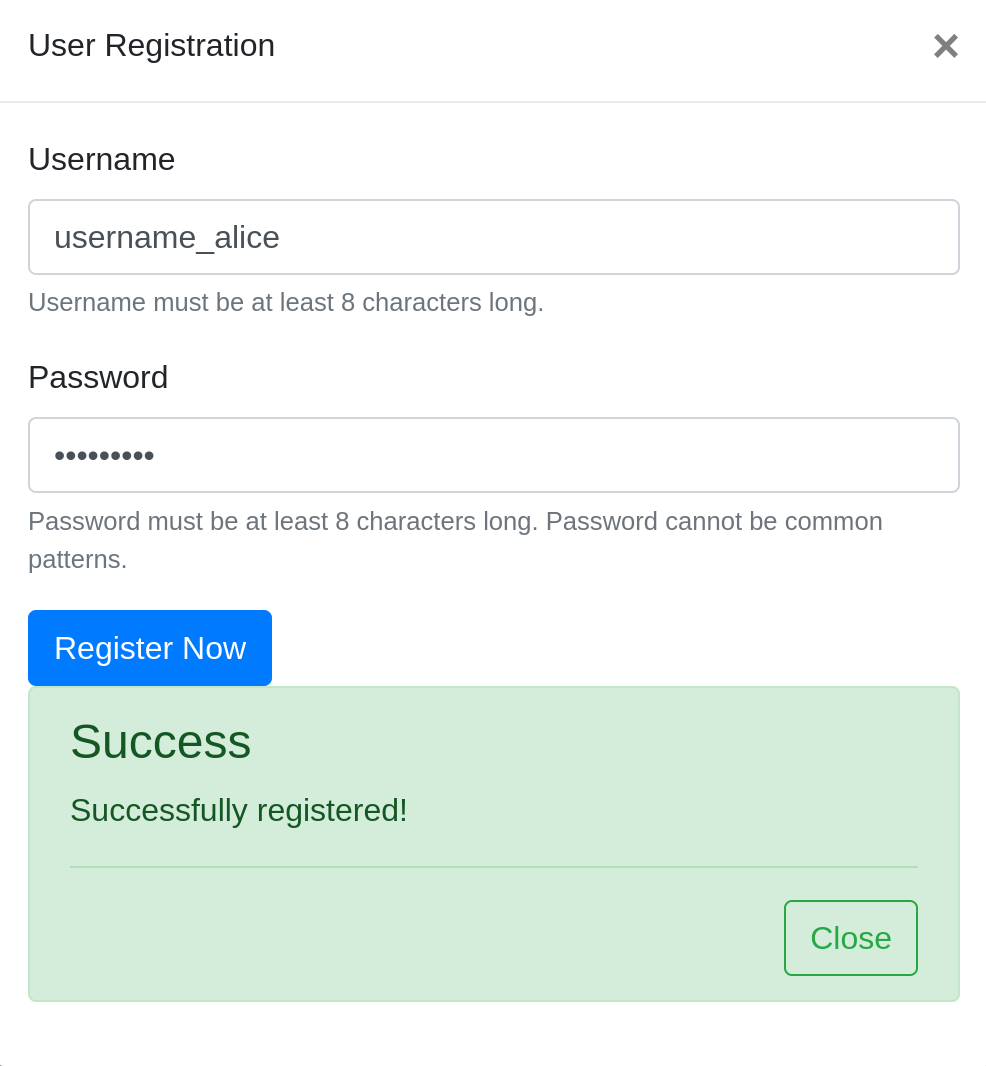
**A brief description of our application**

The purpose of this application is to provide users with the tools needed to review movies or choose the next movie to watch. The tools include commenting/rating movies, searching by genre, and discovering the most popular movies, along with a few other features. Currently, websites like IMDB do not provide ways to discuss movies as a community via comments and discussions. We want the average user to submit their ratings, but also enable them to request new movie entries, and share their experiences of the movie. We planned to use the data from IMDb. Everyone who wants to see reviews of movies and/or contributes new reviews to movies could be a potential user. We are not planning to expose the database system directly to any users. But website administrators can manipulate the data via the admin/management page.

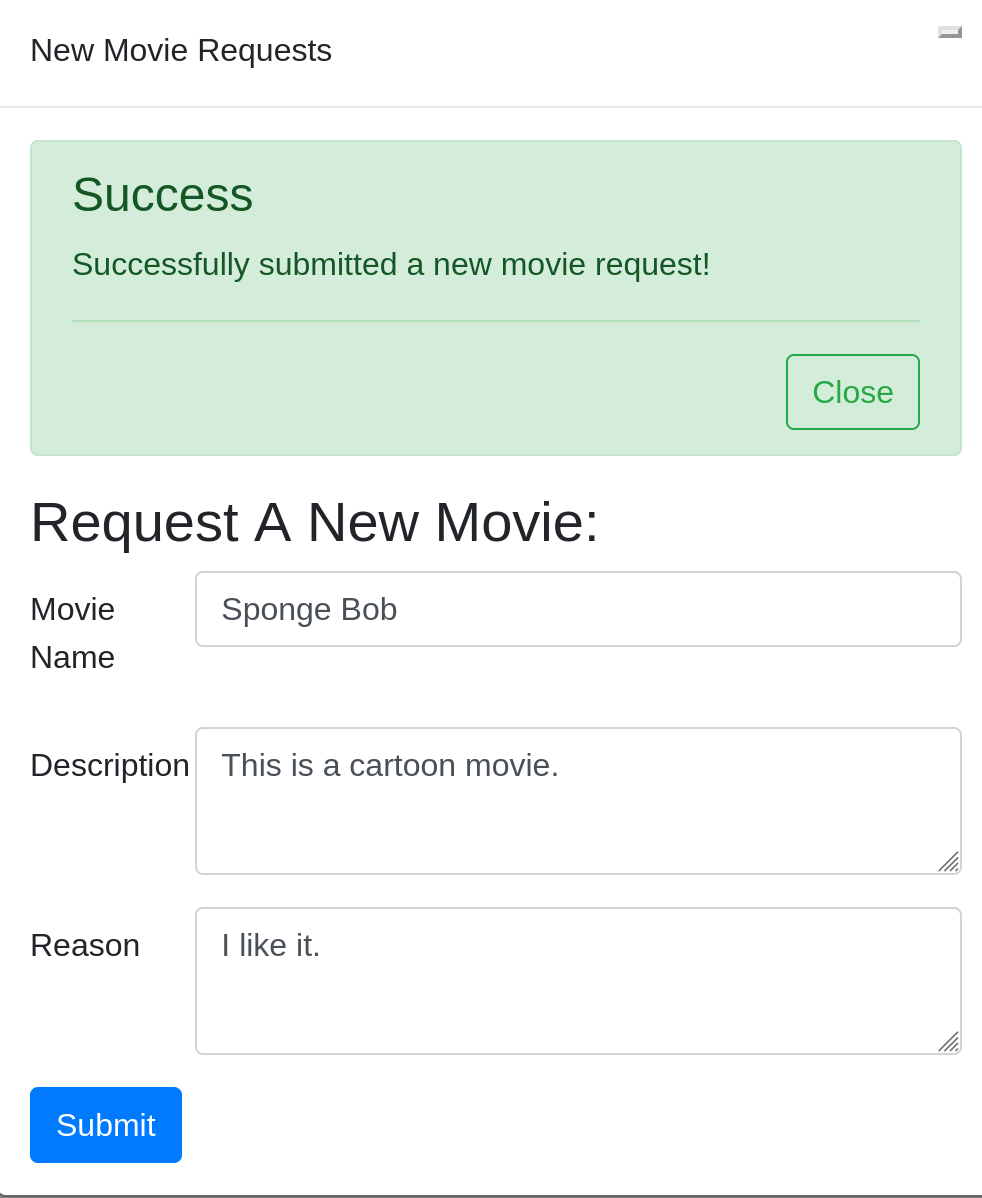
Initially, we planned to support these features: For a standard user, one can review comments of movies, add new comments, delete their own comments, and propose to add new movies. For an administrator, besides everything a standard user can do, one can also delete any comments, accept new movie proposals, add/delete movies, freeze/unfreeze/delete users. In addition to these initial features, we will also allow users to obtain the average rating of a movie, obtain movies by genres, and discover the most popular movies.

**Features**

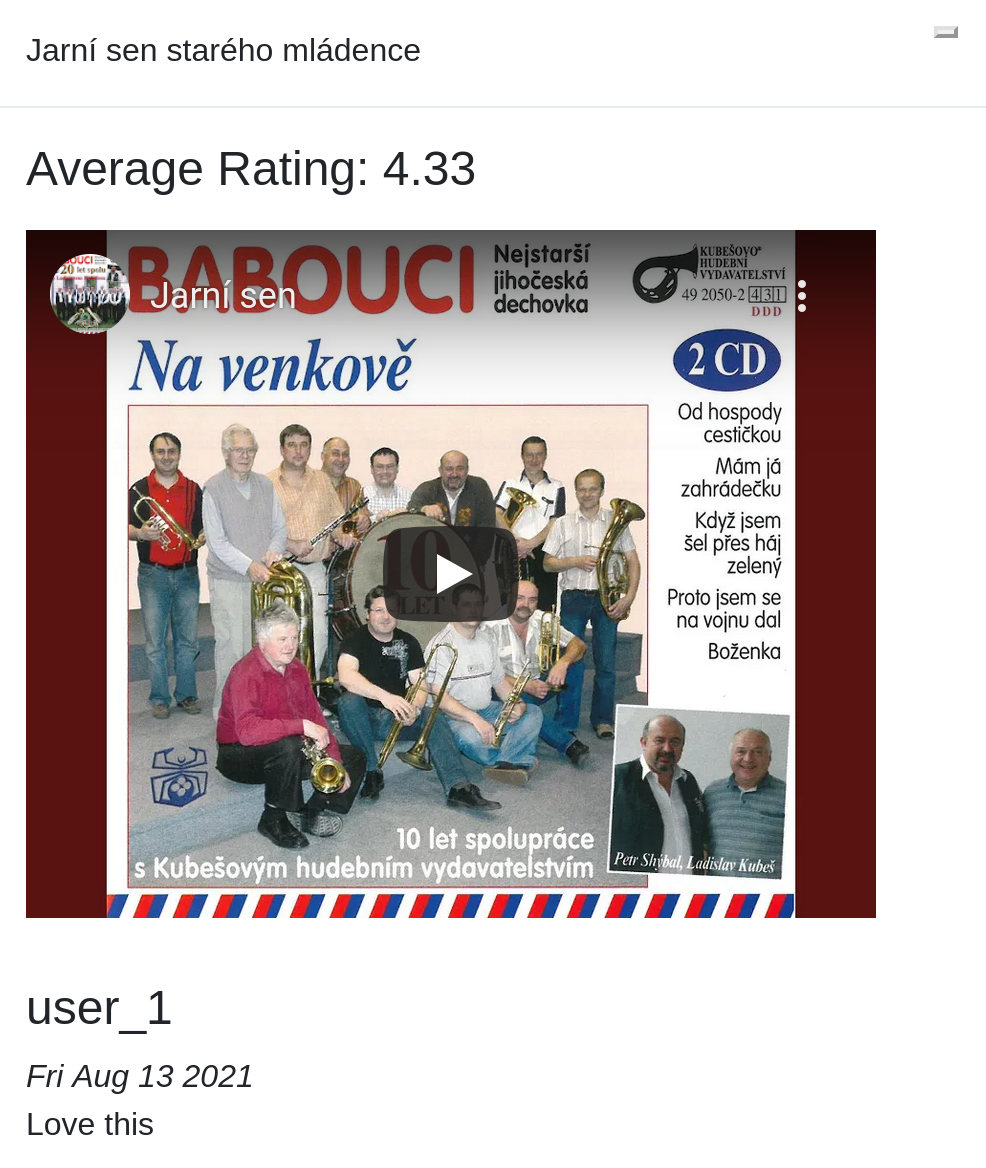
1. User Registration + Log in/ Log out: For user registration, a user should be entering a username and a password in a text field on the website, and then press the ‘submit’ button to submit the user registration, then the password is hashed, and a new entry is stored in the database. There is a login button on every page. After clicking it, a login page is shown for the user. After entering the username and password, the user clicks the ‘sign in’ button to sign in. For logout, there is a ‘logout’ button on every page, too. After clicking it, the user is logged out. Motivation for this feature is that we want to have users submit new movie requests, add/edit comments, and rate movies. Since comments and rating are important for our project, we want users to register first, so that we know who makes changes.



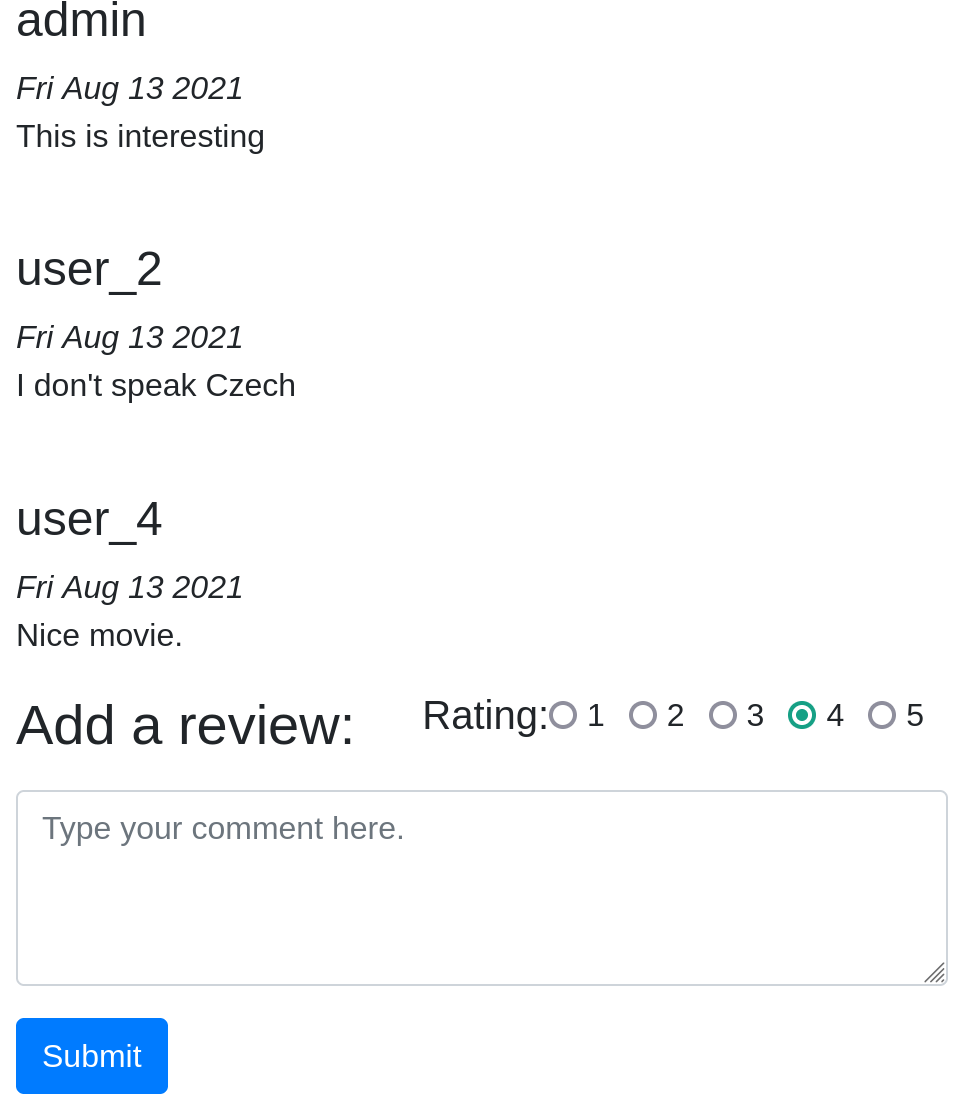
1. Request to add new movies: We should have a ‘request new movie’ page. In that page, a user can enter the movie they want to add, the description of the movie, and the reason for adding the movie. After pressing the ‘submit’ button, the request is stored in the database. Since we cannot cover all the movies in the world, and there will be new movies produced after hosting this website, we need users to contribute to the website. They can request to add a new movie if it is not in our database. An admin can approve the request and our application will add that movie to our database automatically.



1. Categorize movies by Genre: A movie can have multiple genres. A user selects a genre from a pull-down menu and presses ‘update’ button. Then a list of movies is shown on the same page. For the UI part, a user should be able to click on the movie to navigate to the movie. The motivation behind this feature is that users can find movies faster if they want a specific genre, such as Sci-fi.
2. Ordering by Rating: A user clicks on the ‘highest rating’ button. On the same page, a list of movies are displayed with their average rating, where the movies are ordered by rating. Some users like to watch movies with high ratings. Hence this feature helps them to find the top movies with high ratings.
3. Show Popular movies: A user clicks on the ‘most popular’ button. On the same page, a list of movies are displayed with their ‘heat’ value, which is counted by the number of new comments of the movies in 30 days. The movies are ordered by the ‘heat’ value. This feature helps users to catch up with the current trend. For example, a new movie is currently playing in cinemas and has many discussions on our website, a user can find the movie using the ‘most popular’ filter.
4. Display all comments for a movie: In the page for a movie, in the bottom of the page, any one (registered or not) can see the comments for this movie with the username who wrote these comments. The comments are ordered by when they are created. Very basic feature for our application. Users can see the latest comments for a movie.



1. Add a comment: A user can submit a comment under a movie by entering text in a textbox and click ‘submit’. This step requires the user to be logged in. If not, then a warning message is shown on the page, indicating that the user needs to sign up and sign in. Very basic feature. Users should be able to add new comments for a movie.



(Note: See test-production.sql in code.zip for underlying SQL queries for these features)

**Populate database**

To get a large selection of movies for the production database we utilize data from the IMDB data dump found in the tsv file in the zip from at <https://datasets.imdbws.com/title.basics.tsv.gz>.We simply move the data.tsv file to the location ./imdb\_movies\_to\_db/title.basic.tsv in the code.zip archive we submitted.

Then we can use the script contained in the code.zip archive as follows:

python .\imdb\_movies\_to\_db\movie\_tsv\_to\_db.py

That will move up to 50000 movies from the data.tsv file obtained from IMDB into our MySQL database.

This will serve as our production database by getting extra external data from the IMDB data dump and populating some sample data such as users for our specific application using the .\imdb\_movies\_to\_db\movie\_tsv\_to\_db.py and test-production.sql scripts respectively.

**Describe the system support**

The programming framework used for this project is Django (Python). With this framework, we have broken up our api into Serializers, Models, and Views. The rest API is currently running locally on our machines (Unix). We have chosen MySql for our DBMS, also currently running on our local machines. We chose MySql because it integrates nicely with Django and provides the relational queries and structure along with ample documentation needed for our application.

**B) Database Schema Design**

**Assumptions for our data and schema**

* Database named popcorn.

Comment

* cid is unique, not null, and auto incremented
* uid is not null
* mid is not null
* content is not null, and at most 200 units long
* created is not null, and default to CURRENT\_TIMESTAMP
* lastUpdated is not null, and default to CURRENT\_TIMESTAMP
* isDeleted is not null, and default to 0 (False

Genre

* gid is unique, not null, and auto incremented
* genre is unique, not null, and at most 45 units long

Movie

* mid is unique, not null, and auto incremented
* name is not null, and at most 150 units long
* description is not null, and at most 500 units long

MovieGenre

* mid is not null
* gid is not null

MovieRating

* rid is unique, not null, and auto incremented
* stars is not null
* uid is not null
* mid is not null
* isDeleted is not null, default to 0 (False)

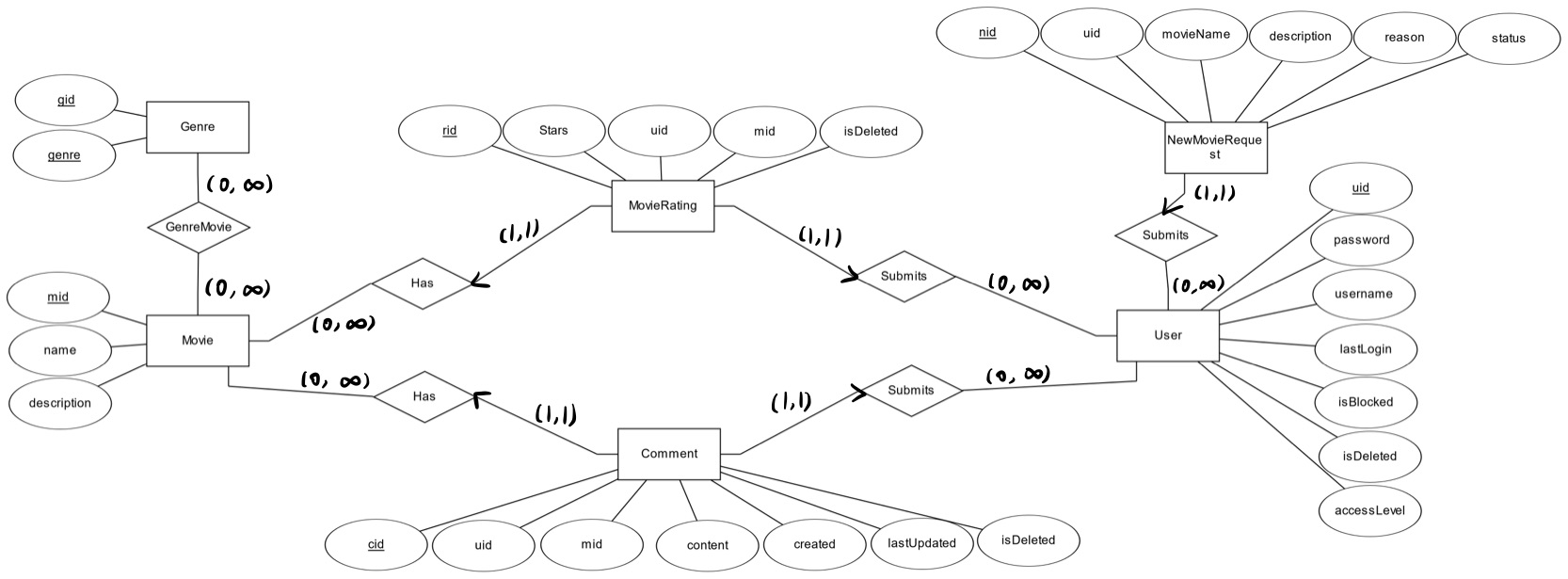
NewMovieRequest

* nid is unique, not null, and auto incremented
* uid is not null
* movieName is not null, and at most 150 units long
* description is not null, and at most 500 units long
* reason is not null, and at most 200 units long
* status is not null and default to 0 (Pending)

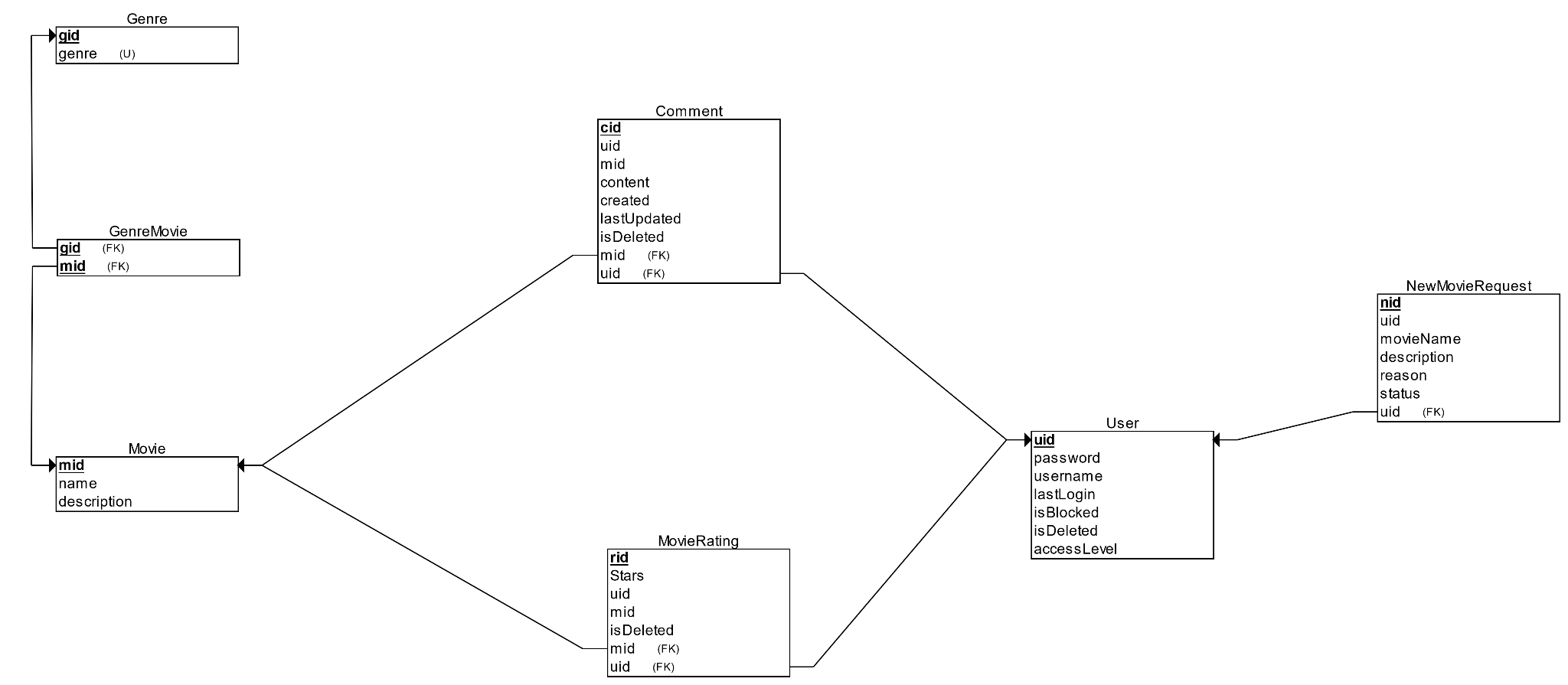
User

* uid is unique, not null, and auto incremented
* username is not null, and at most 45 units long
* password is not null
* isBlocked is not null, and default to 0 (False)
* isDeleted is not null, and default to 0 (False)
* accessLevel is not null, and default to 1
* lastLogin is not null, and default to CURRENT\_TIMESTAMP

**ER diagram**



**Relational Model**



**B) Changes you made to the database during performance tuning in Task 8**

* Add a unique index for Genre.genre
  + It improves performance on fetching sorted genres (ORDER BY Genre.genre)
* Add an index for Movie.name
  + It improves performance on filtering by movie name (WHERE Movie.name=’Kill Bill’)
* Add an index for MovieRating.stars
  + It improves performance on queries ordered by stars (ORDER BY MovieRating.stars)
* Add a unique index for NewMovieRequest.movieName
  + It improves performances on filtering by movie name (WHERE NewMovieRequest.movieName=’Kill Bill’)

**C) Member contributions**

**Hengyu Liu**: Implemented components for user login/registration and movie details page.

**Chenyang Zhang**: Added feature motivations, updated README.

**Surien Das-Giwojno**: Worked on main frontend (UI) layout. Implemented the searching and linking of the different React components. Retrieved data from our REST API.

**Vraj Patel**: Backend queries for User login (with secure passwords), Movie Requests endpoints, Return user information endpoint, Comments endpoints

**Zenong Li**: Added backend queries for searching(giving name or genre id), and paging to movie queries.

Code

Found in code.zip file.