**JACS**

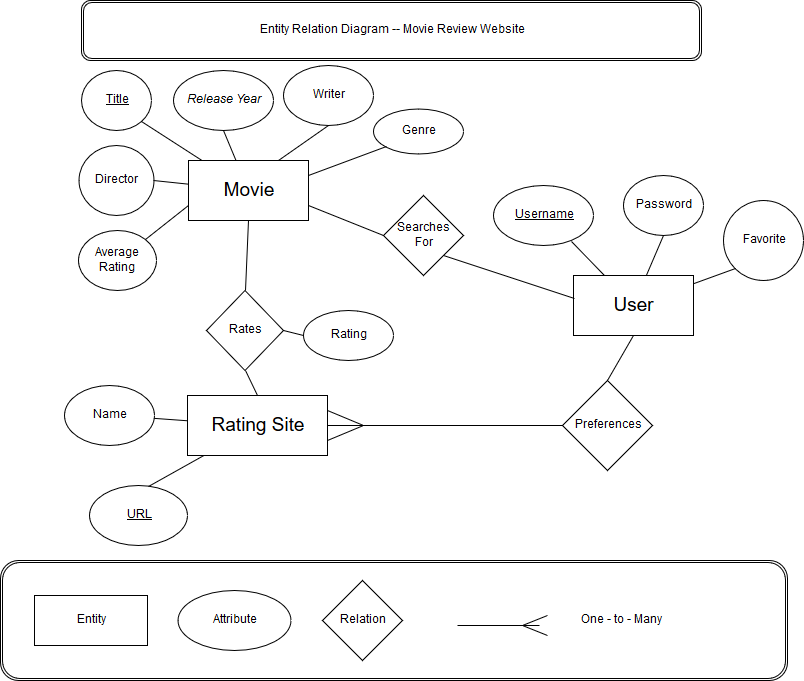
**Colin Cherry**

**Jean Bougeois-Cruz**

**Introduction:**

JACs at its heart is a system to help the users find out whether they will like a newly released or old movie. This is done by allowing a user to search for a movie, find some basic data on the film, and see the rating given to the movie by a preferred rating site, and the average rating across the common rating websites. This Project was jointly selected for the group’s common interest in movies, and after a few shared stories of going to movies rated by certain critic websites, only to find not all of them accurate to our own personal tastes and preferences. Thus, we created the concept of a database that would scrape the web for the ratings of movies from different critic review websites. Then storing that info in one place and allowing the various users in that system to pick the rating site they believe to be the most accurate to their tastes. Therefore, a user would have the ability to quickly get ratings that mirror their own values while still getting an average from the critic industry. Hopefully helping a consumer pick a movie they will then enjoy with ease. To accomplish this the project has three key components; a web interface to allow a user to interact with the system, a Flask implementation to take data from the web view to the last component, the actual database where the information is stored.

**Database Details:**

The database was designed with a logical flow of what a user would do to access and use the database, with a binary thought on how the database would get its data and eventually share this information, when requested, with the user. The main template for the database and its tables can be seen in the following E-R Diagram:

Relational Schema:

* + - Movie (movieID, title, year, writer, director, genre, averageRating)
    - User (username, password, siteURL)
    - RatingSite (siteURL, siteName)
    - Rates (siteURL, movieID, siteRating)
    - Favorite (username, movieID, userRating)

The Tables for the database were created to first store a user data into the Users table, with attached attributes of their username, password, and preferred website. Then the user could, from their logged in instance, search the table of Movie, later renamed Rates for ease of documentation, for any of its attributes listed in the above E-R diagram. Another table, RatingSite, holds the attributes pertaining to the critic websites that the movie ratings are pulled from, and can be referenced by a user’s preferences.

The functional dependencies are handled by each table’s primary key being a foreign key for other tables, allowing different tables to access the attributes of a certain table by joining along that shared key. For example, by joining the RatingSite with the Rates tables using the movieID you can find the rating of the movie by that specific critic site, just as you can also access the attributes of the movie such as the director, writer, and year. With this handling of the functional dependencies the database is in BCNF form. This is due to certain tables having the Keys of other tables as their attributes, in our database this is commonly MovieID.

**Functionality Details:**

Basic:

When a user registers with the database, they will have to give a username and a password, done via a registration page of html. After those two facts are satisfied the user’s username and password will be sent to the database and will be inserted into the database. Once there they will be able to log in normally with the normal login portion of the home page, their credentials checked against the table of users to see if username matches password or if the user is even in the database. However once logged in they would have access to the primary search page, this page too takes in input from the user in the form of a string and passes the data to be matched to an item in the database. The search page implements multiple features starting with letting the user filter (search through) the list of movies by the movies attributes; genre, year, director, or writer, as stated before this is done by trying to match the input from the user to one of these categories inside the Database. The user will also get access to another search option, the option to be displayed the best rated movie of all time (of entities from the critic source sites). This option would be created by an aggregate on the database’s stored ratings. From the back end there will be simple insert commands that would allow for simplistic and easy scalability for the database, thus giving an easy option to keep the application up to date as new movies are released and rated by the critics.

Advanced:

The application will try to closely match to the user their preferred rating sites, allowing the application to find and then use a user’s preferences to better provide a rating for the searched movie. This will be done by first asking the user what they hold to be their favored review site out of the list of primary critic sites used for the application. The application will also allow the user to ask for an aggregate of the database to return the most highly rated movie of all time according to the average of the critic websites, this will also take into account ratings of movies by users which is stored in the Favorites table.

**Implementation Details:**

The application was built using several languages and platforms, these platforms include Python for implementing the middle workings of the application, connecting the UI and the database, this would be used in conjunction with Flask, a platform for going between UI in HTML and a database while providing functionality to the application but not adding unnecessary features that might slow the application down. For the database itself, the structure of the database and its tables are produced using SQLite, used for its common conjunctive use with Flask. For the UI the implementation was created using HTML to create the web based forms to send user data and user requests to the flask and then eventually the database and display it again. This is done by an implementation of Flask, using its functions to transition data from the front end to queries to the SQLite database.

GitHub Link:

https://github.com/CJCherry/JACs

**Experiences:**

From this project, we have collectively been able to work and learn languages we had not yet had a chance to work with, and not in a sanitized lab setting like most programming assignments, but with a near real world example and implementation. The hardest problems we had to deal with was data collection and linking. The data collection was difficult because not all critic websites willing to share their information, and to make the project work the number of websites had to be reduced. The linking was problematic due to unfamiliarity with the environment we were using to link the web UI and the database. As for extending the project into more advanced and mature systems, several things could be done. First off would be a more personalized and stylized style for the UI portions of the project to give the application its own flavor and make user interface more interesting for the user. Along with more inter user functionality to see if a user relates more to a critic’s review of a movie or other users. Other features that could be added could be a way to connect to a streaming service the user subscribes to and automatically fetch ratings from there. However most paramount of the features to add before a public release would be an overhaul to the username password system to store the passwords as encrypted data to keep the user’s profiles secure.

**Implementation Difficulties:**

Some of the problems we had implementing this application is the linking of the web UI and the SQLite database back-end. The Flask middle glue code was never delivered to be used by the group to complete the full application with the disappearance of the groups third member, Aaron, from the group without any information about him being unable to complete his part of the final project.

**References:**

“HTML5 Tutorial.” *HTML Tutorial*, www.w3schools.com/html/.

Ronacher, Armin. “Welcome to Flask¶.” *Welcome to Flask — Flask Documentation (0.12)*, flask.pocoo.org/docs/0.12/.

“SQLite Home Page.” *SQLite Documentation*, www.sqlite.org/docs.html.