CS 5610 Web Development Project Assignment

Project Proposal

Jason Lu
Sunny Lee
Thomas McGourty
Brandon Chung

CS 5610 Web Development Spring 2020
Professor Jose Annunziato
1/29/2020 - 2/07/2020

1.0) Team Formation and Team Project Name:

Team: RESTvengers

Project Name: FilmFinder

1.1) Problem Trying To Solve:

Streaming any form of media (movie, TV show, or even gaming systems) is expensive. We are given a vast amount of choices for users. For example, one may want to stream movies related to a particular genre [Disney for example], or another might be a sports fanatic and wants to stream movies or shows related to sports. Our overarching problem that we are trying to solve with this web development project is: users find it difficult to find a movie to watch when they don't have a movie they want to watch already. We want to reduce the time users spend searching for a movie.

a) Description of Users for the webapp:
    i) Adult User (any user 18+)
    ii) Child User (any user below 18)
    iii) Content Moderator

b) Description of Goals to be achieved in webapp:

Goals for Adult User:
- Search for movies
- Review movies
- Add movies to "watch list"
- Favorite movies
    - Receive movie recommendations based on favorites

Goals for Child User:

- Same as Adult
    - Can't see rated R movies (maybe not PG-13 either based on preferences)
    - Commenting restrictions?

Goals for Content Moderator:

- Delete inappropriate reviews
- Lock threads for commenting

1.2) Overview on how we plan to solve the problem:

We intend to solve the problem by using the following procedure:

a) Provide the movie recommendations for the user based on their preferences [tags, ratings, etc.], which will be stored in a database.
    i) We will also have a system to give a weighted score for each movie
        1) We will store some number of movies from the OMDB API in our own database, with their tags, genres, actors, etc and use this to generate the scores
        2) We will also use CRUD (create, read, update, delete) operations on the database to return to the users on the movies they would most likely to prefer
b) Search that will provide user feedback for each movie preference, based on their criteria
    i) We can reduce the time spent searching for the movies by using the OMDB API
c) We will allow users to leave ratings and reviews on individual movies
    i) Store this data in our database, fetch it upon viewing details page of a movie
    ii) Users can also comment on reviews
d) Users can showcase their favorite movies on their profile, and view other users' favorite movies on their profiles
    i) Users can also showcase reviews on their profile, and view other users' reviews on their profiles

1.3) Web API Intended To Use:
We intend to use OMDB as our main API. The reason why we chose OMDB as our API because we have tried to use other API to query for film descriptions (besides the movie, title, and genre, plus year produced), and we faced either a limitation on the number of API calls allowed or some financial constraint with making more than 100 API calls. Also, OMDB is also compatible with JSON, jQuery, and PostGres SQL, which would prove useful for compatibility reasons and querying later in the project.

1.4) Next Steps: We can clarify with the professor / TA which API to use to query for recommendations and more advanced features for our project. We will also implement a rating system for the comments, as "endorsements" (similar to that of Piazza) for the most helpful or

relevant comments. If necessary, we can also have “tags” for the comments in order to have the users select the most relevant of the movies.

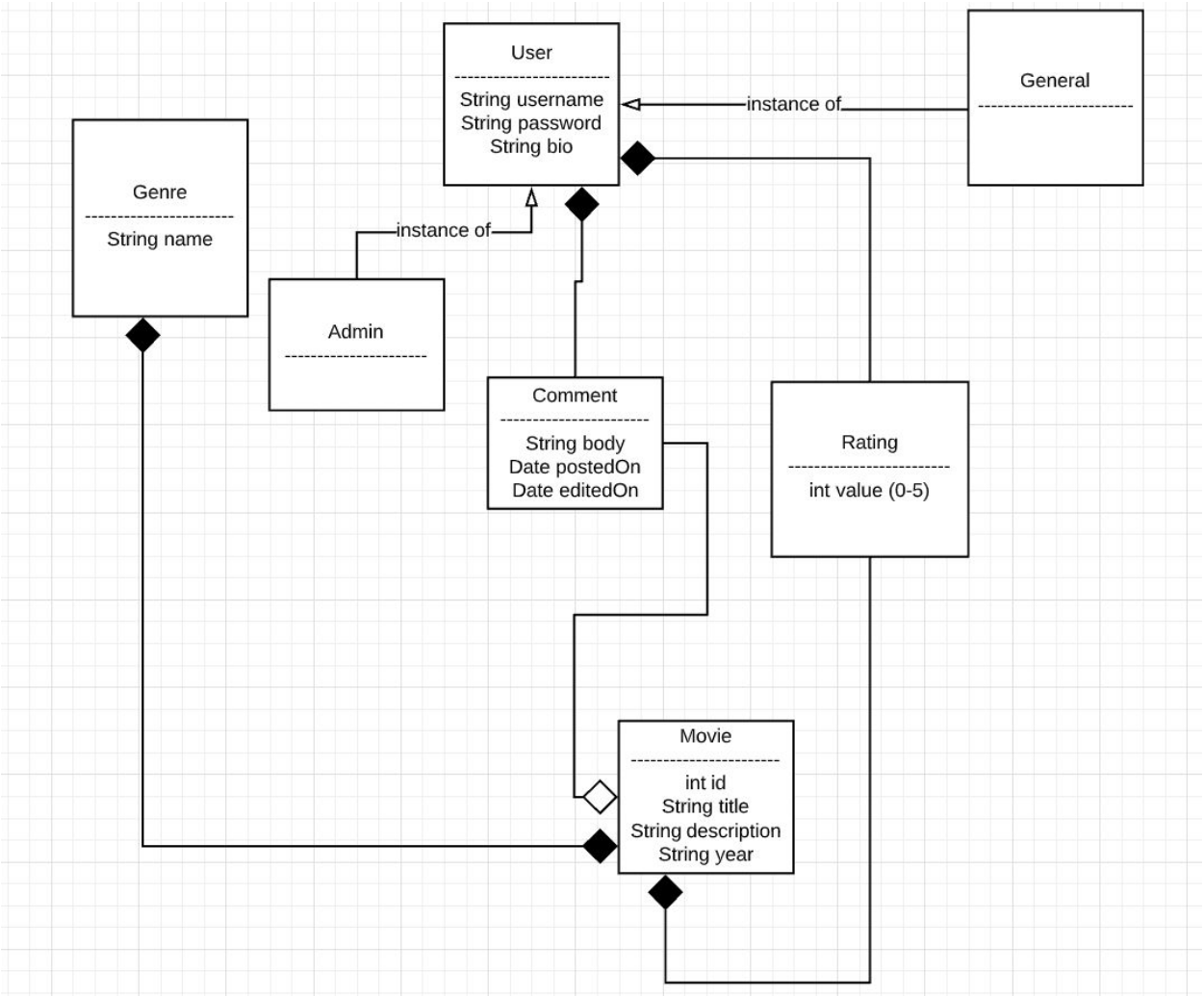
1.5) References / Sources:
http://www.omdbapi.com/

2.1) Design

Pages

1. **Landing Page - recommendations (provided login)**
2. **Register page (sign up)**
3. **User Profile (keep track of the movies, used ID defined by the server)**
4. **Movie page- comments, poster, movie data**
5. **search results- API call returns data**
6. **Recommendations page:**
    - **displays results from our recommendation algorithm (starting as a simple weighted list based on tags**
    - **could be toggled between home page and this page**
7. **FAQs (About Us for the website)**
8. **Actor page (if given time)**
9. **Rating Systems Page (random for other genres?)**
10. **Privacy Page (GDPR)**
11. **Credits Page (citing)**
12. **Details Page (user search results)**
13. **Endorsed Comments Page (the most helpful comments, determined by admin.)**
14.

2.2) UML Diagram:



2.3) RESTful URL Patterns:

Method	URL Pattern	Description
`GET`	`/api/users/${userId}`	Returns the user whose ID matches the provided ID as JSON, as well as a list of all of their movie ratings.
`GET`	`/api/users/user`	Returns the private information for the currently logged in user.
`PUT`	`/api/users/user`	Updates the currently logged in user's information with the body from the request.
`POST`	`/api/users`	Registers a new "general" (non-admin) user.
`DELETE`	`/api/users/${userId}`	Removes the user whose ID matches the

		provided. ID. Only works if the user calling the endpoint is logged in and is an admin.
GET	/api/movies	Searches for movies that match the given parameters, using the external API. Search by movie title and year, and potentially genre/actors later on.
PUT	/api/movies/${movieId}/rating	Updates the logged in user's rating for a specific movie.
POST	/api/movies/${movieId}/rating	Creates a rating for the specified movie under the logged in users.
PUT	/api/movies/${movieId}/comments/${commentId}	Updates the body of the logged in user's comment.
DELETE	/api/movies/${movieId}/comments/${commentId}	Deletes the specified comment. Only usable by the user that posted the comment or an Admin user.
POST	/api/movies/${movieId}/comments	Creates a comment from the current user for the specified movie.
GET	/api/movies/${movieId}	Retrieves all the details for the movie with the specified ID. Also retrieves all comments and ratings for the movie to display under its details.
GET	/api/users/user/recommendations	Returns a list of movies ordered by a perceived relevance to the user.
GET	/api/users/user/comments/${endorsedCommentId}	Returns a list of "endorsed" comments with the respective comment ID.