E-mail:wen.r@husky.neu.edu Name: Rui Wen

Final Project Report

__ Movietime website

Problem statement:

The problem I want to solve is to make a simple and small website for people to find the movies they enjoy.

There are several purposes I want to achieve.

First, help people search the movies they want more information, nowadays new movies come to us just like a tsunami and we are easily to get lost in this movie world. Then we need more information about the movies.

Second, as a super movie fan, I always feel that I need a private space to manage all the movies I wish to watch in the near future. Then a “like” or “wish to see” function is vital for a movie website. The like action can also become part of the reviews and it would be a recommendation to other users.

Third, the movie fans are really easily to become friends, so offer them some ways to communicate with each other would be a good decision, some review and reply would be very important.

Proposed solution:

To solve the three problems mentioned above, I made three corresponding design.

First, create a search function in the top of my website, this can help users to search the movies they want and if you like the movie, you can click on the like button and this movie would be save into your favorite movie database.

Second, the favorite movie list is a good design for movie fans, so I add a lot of like button in movie detail pages, these buttons would help users build their own libraries of great movies.

Third, the whole movie page is built by adding all the users' movie libraries on this website, so we can easily find the fashion and classical movie. I also made a comment part under every movie, and people can discuss in this part.

Architecture:

My project contains six main parts:

The first part is the home page which holds the basic information and interface of the website, all the function must start with homepage.

The second part is the register and login part, in this part, using this part, we can register as users and we can achieve more features on this website, if we did not login as a user, we would not be able to use many other parts in this website.

The third part is the users' favorite movies list, here you can find all the movies you have liked before, in this part you can also get the detail information of the movies you like.

The fourth part is the whole movie list part, it is made up by all the users' favorite

movie list, so this list can reflect the taste of the website users.

The fifth part is the comment part. Under every movies' detail information, there is a comment part, every user can comment there and each comment would be displayed to all the users browse the same movie.

The last part is a half-done message part, if finished, users can use this part to send a message to the user he/she wants to talk to for this website knows the e-mail address of every user.

APIs:

To get the data of movies I need, I found a API called the Rotten Tomatoes API, it is a RESTful web service API, I also considered the Fandango API but it is a jQuery API and harder to use than the Rotten Tomatoes API, so I choose this one.

The Rotten Tomatoes API using an URL like this: http://api.rottentomatoes.com/api/public/v1.0/movies.json?apikey = [your api key]&q=?&page_limit=?

In this link, we can change the way we get movie data, once we insert the rottentomatoesid into our movie database, we would not need to do this search action every time we need detailed information about this movie, so this is a good choice.

This API can also be used to search and would return a list of movies match the request, the number of movies we get can also be modified.

Technology stack that I used:

I used several technology to persist this project. The first thing I use is JPA, which is used to create all the tables in database. I use the onetomany and manytoone and manytomany annotations to create the relationship between different tables.

When checking the user identification, I use the http sessions to keep this status, it worked well for small number of pages but when I travel a lot pages and for a long time, some problems encountered.

When registering and logging in, I use JDBC to transfer the data, while when retrieve the movie detail data, I choose to use ajax, which is more friendly to the http connection and make the webpage more dynamic.

I also made a new RESTful API for my own project, which is important if some other developer want to reuse the database created by this website.

Use cases:

The use cases I achieve is listed below:

Use Case 1: User browses movies by category [browse by category]

Description: User wants to browse all the movies in a specific category.

Precondition: None.

Steps:

Actor actions:

User choose a category to browse

System Response:

Display the combination of movie name and poster in this category

Post condition: All the movies in this category are displayed. (In form of name and poster)

Use Case 2: User comments on a movie [comment on movie]

Description: User wants to write his own comment on a specific movie and gives it a rating level from one star to five stars.

Precondition: User is at the movie page, user authenticated.

Steps:

Actor actions

User starts writing comment on a movie and gives a rate level

User submits a comment

System Responses

Comment draft displayed to this user, but still keeps invisible to other users Comment and rate level displayed to all users

Post condition: New comment adds to the movie comment table and is displayed to all users.

Alternate path one:

Actor actions:

User starts writing comment on a movie without authentication

System Responses:

Ask user to login first and director him/her to login page

Post conditions: User enter the login page and after logging in is redirected back.

Alternate path two:

Actor actions:

User starts writing comment on a movie

User leaves the comment page without submitting

System Responses:

Comment draft displayed to this user, but still invisible to other users Comment and rate level saved as draft

Post condition: New comment draft adds to the comment table.

Use case 3: User browses roles of a movie [browse role list]

Description: User wants to browse all the roles in a particular movie.

Precondition: User browses some movies on the screen.

Steps:

Actor actions:

User enters the movie page

User enters the role page

System Response:

Display the basic information of the movie

Display the whole role list

Post conditions: All the role in this movie are listed to the user.

Use case 4: User likes a movie and wish to see it. [like a movie]

Description: When browsing, user finds a movie he/she likes and wants to see it in the future.

Precondition: User is in a movie browsing page or movie information page, user authenticated, wish list exists.

Steps:

Actor actions:

User likes a movie

User choose which wish list he/she wants to put the movie in

System Response:

Show all the wish list user has created before

Display the movie in the wish list

Post condition: The movie is displayed in one of the users' wish list.

Alternate path one:

Actor actions:

User likes a movie without authentication

System Response:

Ask user to log in first and direct him/her to login page

Post condition: User enter the login page and after logging in is redirected back.

Alternate path two:

Actor actions:

User likes a movie with no wish list created before

User creates a wish list and submit

System Response:

Ask user to create a wish list first and leads he/she to wish list creating page

Display the movie in the wish list just created

Post condition: A wish list is created by the user and the movie is displayed in it.

Use Case 5: User reply to a comment written by another user. [reply to comment] (This is the case where Self Reference is used)

Description: Users can reply to other users' comments to interact with each other.

Preconditions: User is in a comment page, user authenticated.

Steps:

Actor actions

User stars writing comment replying to a comment

User submits a comment

System Responses

Comment draft displayed to this user, but still keeps invisible to other users Comment displayed to all users and a notification is sent to the writer of previous comment

Post condition: New comment adds to the comment replying table and is displayed to all users.

Alternate path one:

Actor actions:

User stars writing comment replying to a comment without authentication

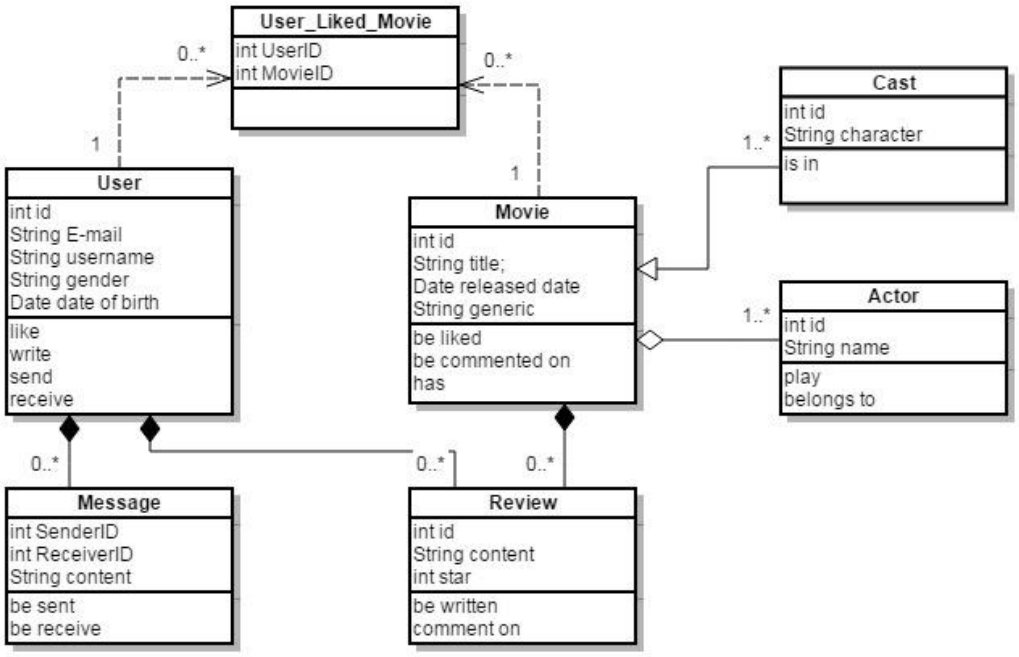
System Responses:

Ask user to login first and director him/her to the login page

Post conditions: User enter the login page and after logging in is redirected back.

Data model:

The data UML is as follows:



Here I use totally seven tables. There are actually six objects. They are user, movie, actor, cast, message and review. The User_Liked_Movie is an associated table to help handle the many to many relationship between users and movies. The user and reviews have a one to many relationship and it is the same with the movie and actors. The cast and actor has a one to one relationship which is very useful to make a form.