CS235 Report

There were 3 extensions added to this assignment in addition to all the basic requirements mentioned by the client.

They were:

- Recaptcha Extension to prevent bots from signing in and therefore improving the security of the application
- Maintaining and manipulating a watchlist for users only. This includes deleting and adding a movie
- Getting movie posters from the OMDB API to create a graphically pleasing website and intuitive buttons when browsing through different movies.

I. Recaptcha

This is seen when registering the user for the first time. The adjacent prompt is shown where the user must select that they are not a robot to continue. Sometimes the user will be asked to do a small task to confirm their identity:

Figure 1: Recaptcha

II. Watchlist

Every user has a managed history based on what movies they have seen (this is based on a simulation) as well as a watchlist they can add and delete from. If they watch a movie on their watchlist it will automatically be deleted from their watchlist and added to their history. This function is only available to registered users once they have logged in. This screenshot can be found in Menu->User. Adding/deleting to and from their watchlist can be done when browsing a specific movie through a toggle button as the different screenshots (Fig 3 and 4) show.

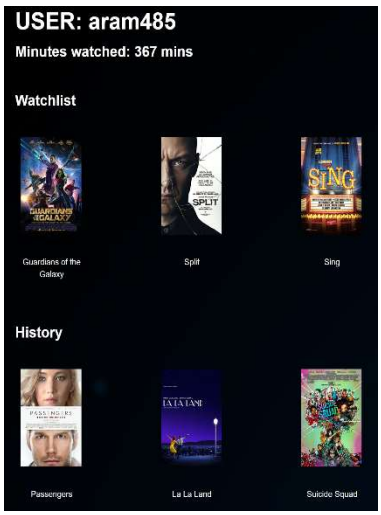


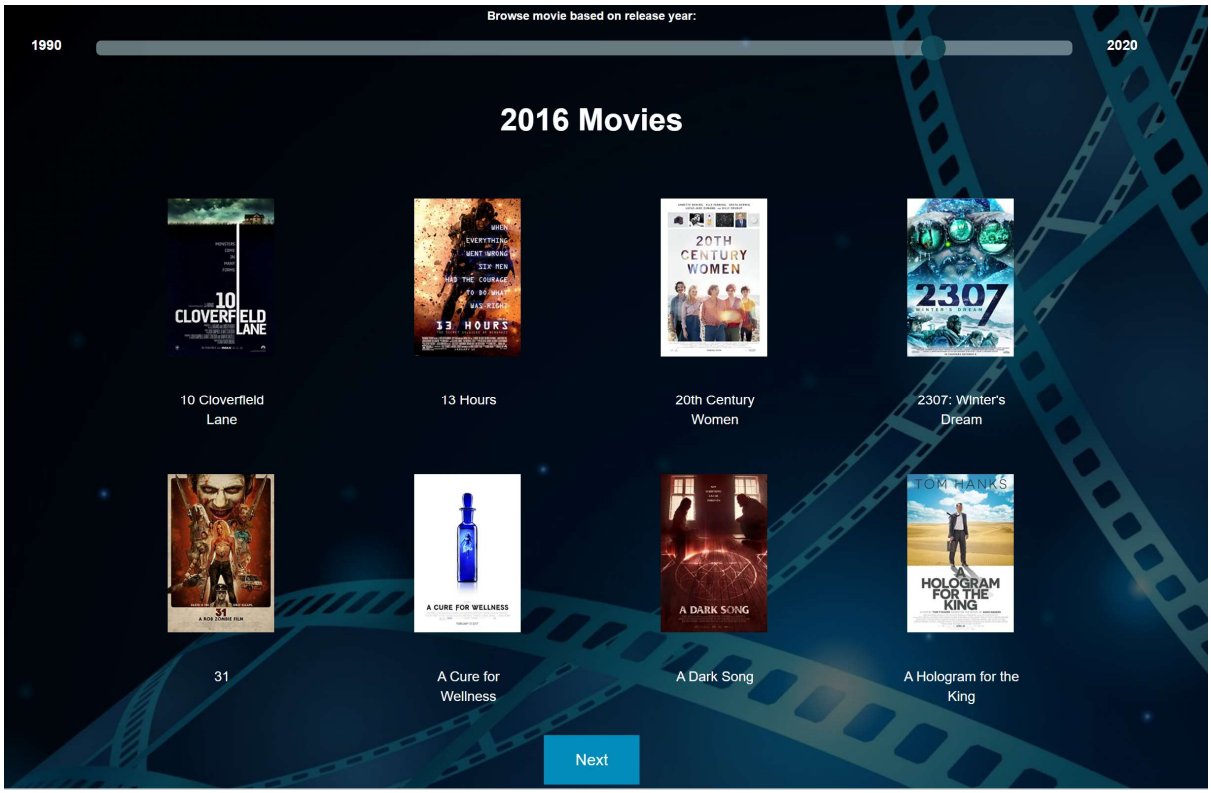
Figure 2: User homepage

Figure 3: Option to add movie to watchlist

Figure 4: Option to remove movie from watchlist

III. Movie Posters from OMDB

To allow for aesthetic appeal and visual information about the different movies the poster was taken by calling a OMDB API and using a link from that. To ensure there were minimal API calls, once the dataset was loaded the image link was added to the movie class and accessed from there henceforth. The key required to call the OMDB API is stored in plaintext in the project and there is a 10000 limit on API calls daily.



Design Decisions

To facilitate good coding practices the following design patterns were adopted:

- Repository Pattern
    - This allows us to easily change out the database because we are accessing the database through an abstract class (in this case the movie_repository). This defines certain methods to extract and manipulate specific things within the permissions of the class. E.g allowing us to add a watched movie but not remove a movie from watched movies ensuring that the classes have only restricted access to change the database.
- Blueprints to separate concerns
    - Blueprints were used to separate concerns. In this example there were 5 blueprints used; Home, User, Movie, Review and Auth for their respective tasks. Each blueprint had a view and a set of services that allowed them to interact with the repo. Again this restricted access allows us to ensure the state of the application doesn't become unpredictable while making debugging easier because each task is allocated to a certain blueprint.
- Service layer for each blueprint
    - This facilitates dependency inversion where higher modules like the view layers do not directly interact with lower layers like the repo. They do this through a service layer which essentially means both repo and view modules depend on abstractions. This means either can be interchanged easily allowing for portability. In this application the service layer was different for each blueprint. This was intentional to restrict the access of each blueprint allowing the application to be more secure and behave more predictably.
- Single Responsibility
    - This was adopted in general for all functions and classes to ensure clean code that was readable and easy to debug.