COMPSCI 235 Assignment 2

For this assignment, I implemented two additional features (a User Watchlist and OMDb images).

New Feature One: User Watchlist

Description:

The first feature I implemented in this assignment was a watchlist. This feature allows logged in users (if they are not logged in, pressing the “Add to watchlist” button redirects them to the login page) to browse movies and add it to their watchlist. After adding to their watchlist, the user is able to continue browsing through movies but can check their watchlist anytime by accessing the watchlist page from the navigation menu.

Key Design Features / Design Patterns:

The watchlist feature works by having a list of movies (initially empty) in the memory repository. When a user presses the “Add to watchlist” button, this triggers the add_to_watchlist() blueprint with the movie rank being passed as an argument. In this blueprint, the function add_to_watchlist() from services is called, and this results in the movie with the corresponding rank to be appended to the watchlist in the repository. As well as the movie rank being passed as an argument, the cursors / view review variables are also passed. This is so that after the user has pressed the button, the blueprint is able to use these parameters to redirect them to the same page they were on previously. This allows the user to continue browsing through movies without having to navigate through previously visited pages.
In the watchlist page itself, the number of movies per page is limited to 5. If a user has over 5 movies in their watchlist, a cursor is used to navigate through first/next and last/previous pages. As movies are already added to the user’s watchlist, the “Add to watchlist” button is removed from this page but “Post review” and “View reviews” are still available.

One design pattern used by this feature is the Repository pattern. This feature takes advantage of this pattern by using the repository in order to retrieve objects (the watchlist list) from the memory repository. In order for the blueprint to add a movie to the watchlist, it calls the add_to_watchlist() function from the service layer. This function in the service layer is then dependent on the abstract repository so that it can call the add_to_watchlist() function from the memory repository. Using the Repository pattern makes it much easier to test specific parts of code and allows for flexibility in the application.
Similarly, this feature also uses the design principle of Dependency Inversion. Rather than allowing a high-level module (i.e. the movies in the Service layer) to be dependent on low-level modules (such as the database itself), it instead depends on abstractions (an abstract repository). This design principle allows for changes to be more easily made and prevents having to alter code in multiple areas which could potentially lead to the program breaking frequently.

Benefits to the project:

A watchlist provides additional functionality and purpose to registered / logged in users. Users are able to easily keep track of certain movies they may be interested in as opposed to having to use alternative methods such as bookmarking the webpage.

New Feature Two: OMDb API

Description:

The second feature I implemented was using an open-source database (OMDb) to be able to display the posters of each movie on the page where possible. If a poster for a specific movie is available from the database, this feature allows that image to be displayed on the browse movies pages. Otherwise, no URL exists and it just displays a broken image link.

Key Design Features / Design Principles :

I tried to implement this feature via two ways; for the first method I attempted to display an image through the use of JavaScript/Jinja in the HTML file and for the second, I used the python OMDb module. Originally, I only wanted a search method to allow users to search for movies by title. This would then return a page with a single corresponding movie along with the poster, title, description e.t.c. This was where I attempted to challenge myself to use a script in my HTML file. The JavaScript function takes the title of the movie (passed as a Jinja variable) in order to get a string containing the URL of the movie poster. The image source of the first corresponding movie poster is then returned to be displayed in a div on my movies page.
Then I later decided that I wanted to display posters for all of the movies in my browse movie pages. Because I wasn't too familiar with JavaScript, I used the OMDb module instead. To implement this, I created an additional field in my Movie class (in my domain folder) which stores the string for the poster URL. I then created a new function in the repository which generates the URL for the poster using the title of a movie and then assigns this to the "poster" field of the Movie class . In my movies.py file, I am then able to call the get_posters() function from the service layer in order to get the poster link and be able to display it on my HTML by using Jinja's dictionary format {{ movie.poster }}.

Similar to the watchlist feature, I used the Repository pattern and Dependency Inversion in order to implement the OMDb module in my application. As much as possible (while trying to avoid duplication of code), I was attempting to adhere to the Single Responsibility principle. For each function I wrote, I tried to ensure that they were only written to aim to perform/return one thing so that if something goes wrong, I have a clear idea of where the problem may be lying.

Benefits to the project:

As opposed to my first feature which offers extra functionality to my website, this feature instead offers something to the visual aspect of my webpage. The posters allows movies to be more readily and easily identifiable, rather than relying on the user to depend on movie titles and descriptions to recognise the films. It results in greater similarity with my project and an actual movie database.