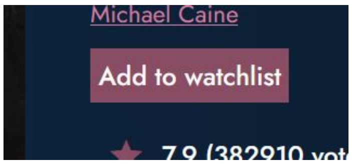
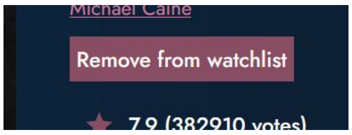
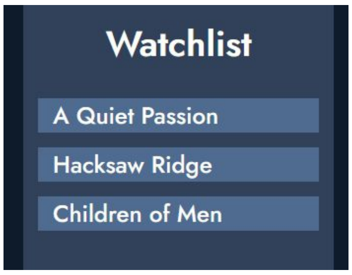
I implemented 2 (and a half) cool new exciting and awesome features in this project:

- Watchlist
- ODMB movie posters
- Popular movie recommendations

Watchlist







Since I had already implemented a watchlist in part 1 of this assignment, I saw no reason not to include it in this part.

On a movie's page, there is an 'add to watchlist' button in the information section. Upon pressing this button, the movie will be added to the user's watchlist, if the user is not logged in the will be prompted to.

If the movie is already in the users watchlist, the button will say 'remove from watchlist', and will do that.

Users can see their watchlist on the sidebar, it's not particularly useful since you can't watch movies on this site, but you can click on each movie to take you to its page.

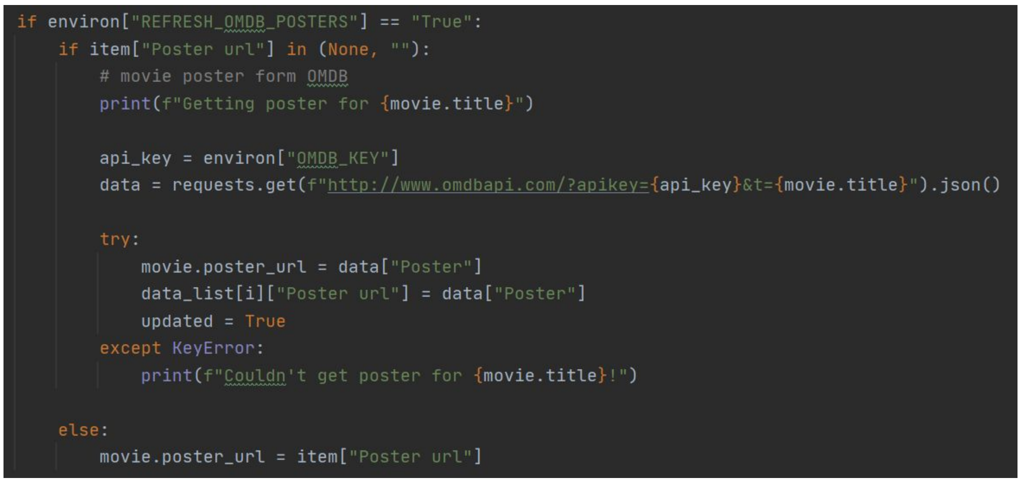
Each watchlist is stored in its User object in the repository upon initialisation, a list of movie ids is read for each user to create their watchlist.

Using OMDB API

In order to get the posters for the movies, I used the OMDB API. I found that querying the API every time I wanted a poster was quite slow, so I just made it collect them all during initialisation and write them to the movies CSV file.

Side note: Were we supposed to use the Data1000Movies.csv file from part 1? I couldn't find any mention of where we were supposed to get the data from, so that's what I used. Hope that's correct or I just lost a bunch of marks.

This way all the poster links (that the api could get, it failed to get quite a few) are all in the database upon every subsequent initialisation the code to query the API is still in the init_repo() function, it can be enabled by setting the environment variable REFRESH_OMDB_POSTERS to True



```python
if environ["REFRESH_OMDB_POSTERS"] == "True":
    if item["Poster url"] in (None, ""):
        # movie poster form OMDB
        print(f"Getting poster for {movie.title}")

        api_key = environ["OMDB_KEY"]
        data = requests.get(f"http://www.omdbapi.com/?apikey={api_key}&t={movie.title}").json()

        try:
            movie.poster_url = data["Poster"]
            data_list[i]["Poster url"] = data["Poster"]
            updated = True
        except KeyError:
            print(f"Couldn't get poster for {movie.title}!")

    else:
        movie.poster_url = item["Poster url"]
```

CS235Flix/adapters/memory_repository.py line 211

The poster URLs can be obtained from a movies poster_url property, this way, if may have to be loaded, the API does not have to be used and thus the user does not have to wait for all the urls to be gotten, the website can load them on the fly.

(Sort of feature): Popular movie recommendations

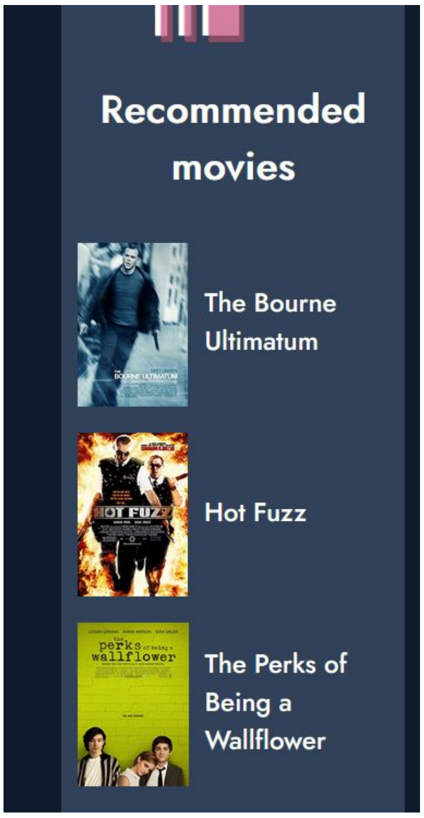
This is similar to the recommendation of movies to a user based on their preferences, however I am recommending to non-users.

I needed something to put on the sidebar if a user was not logged in. I had planned to put a recommended movies section on the home page, but the sidebar seemed to fit just as well.

The way it decides which movies to pick is that it orders the movies in the repository by their external rating multiplied by their metascore. It then chooses 3 random movies from this list and displays them, so it is different every time the page loads.

Obviously if the database wasn't a static csv file, this algorithm would be different, most likely a combination of date, score, revenue and ratings.

However, in this implementation, the repository simply has a get_popular_movies() method which selects them.


Recommended movies
The Bourne Ultimatum
Hot Fuzz
The Perks of Being a Wallflower