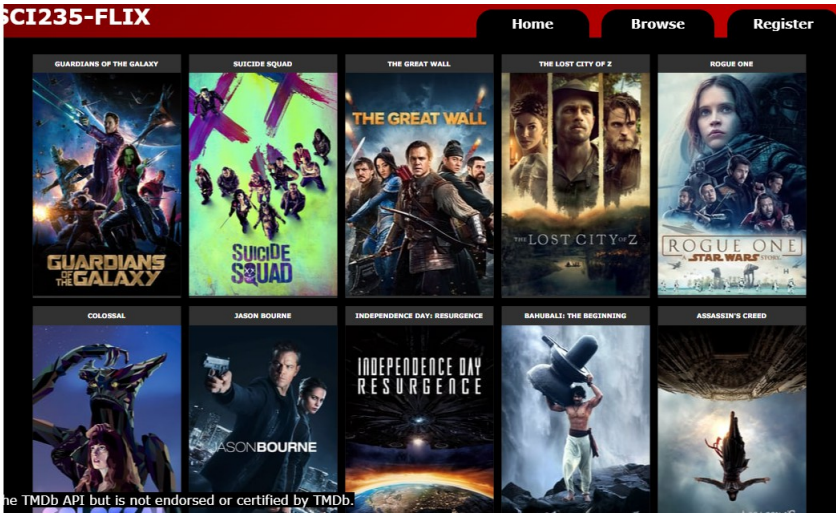
CompSci235Flix – CS235 Assignment 2
By Toby Tomkinson

Feature(s) Implemented

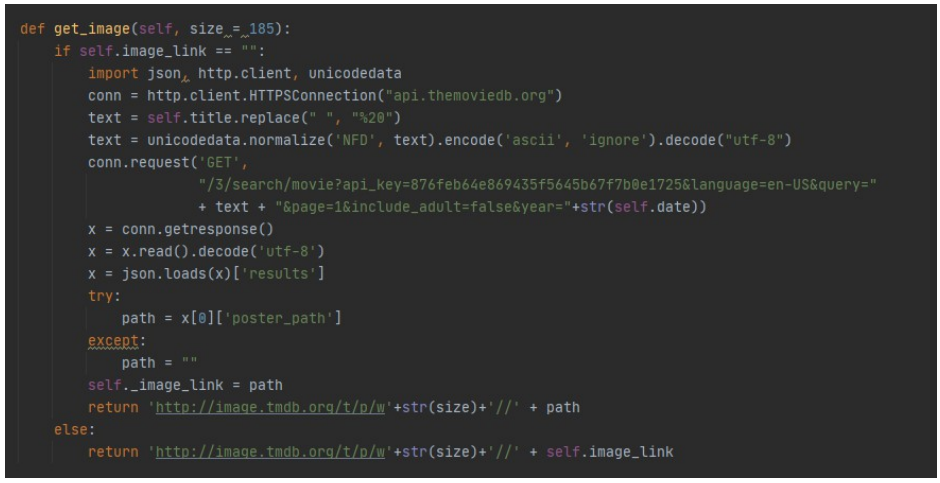
So the main new feature I implemented was use of The Movie Database (TMDb)to add images to the movies for browsing (and on the homepage). It's quick and easy to set up an account and the images are available in various resolutions, which I make use of – when browsing, the images are 185 pixels wide (w185) and on the webpage the images are 300px wide (w300). This means that the images look clear even on larger screen while using less data, resulting in a better experience with faster loading times. As is visible at the bottom of the picture, I have "This product uses the TMDb API but is not endorsed or certified by TMDb" visible as per the TMDb terms of use, linked below:



https://www.themoviedb.org/documentation/api/terms-of-use
Specifically, term 1a requires any user of TMDb to follow term 3, which states that "*You shall place the following notice prominently on your application: This product uses the TMDb API but is not endorsed or certified by TMDb*". I also wrote a small .bat file for testing convenience but it's only a minor QoL feature.

Design Principles and Patterns

The single design principle was one I adhered to for most, if not all, of the code I wrote for this assignment. Each class clearly manages a different function of the application and said function is easily identifiable. For example, the exact code I implemented for the new feature, adding images, is part of the Article class in the model. A benefit of using the single design principle is that the code is fairly self contained and could easily be moved between classes to be adapted for getting pictures of actors and/or directors in possible future revisions of this program.



```
def get_image(self, size = 185):
    if self.image_link == "":
        import json, http.client, unicodedata
        conn = http.client.HTTPSConnection("api.themoviedb.org")
        text = self.title.replace(" ", "%20")
        text = unicodedata.normalize('NFD', text).encode('ascii', 'ignore').decode("utf-8")
        conn.request('GET',
                     "/3/search/movie?api_key=876feb64e869435f5645b67f7b0e1725&language=en-US&query="
                     + text + "&page=1&include_adult=false&year="+str(self.date))
        x = conn.getresponse()
        x = x.read().decode('utf-8')
        x = json.loads(x)['results']
        try:
            path = x[0]['poster_path']
        except:
            path = ""
        self._image_link = path
        return 'http://image.tmdb.org/t/p/w'+str(size)+'//' + path
    else:
        return 'http://image.tmdb.org/t/p/w'+str(size)+'//' + self.image_link
```

The code itself is fairly straightforward – it checks to see if the url is registered in the Article class. If it is, then it returns it. If it's not, it requests the relevant movie's image URL from the TMDb server and registers it in the Article class.