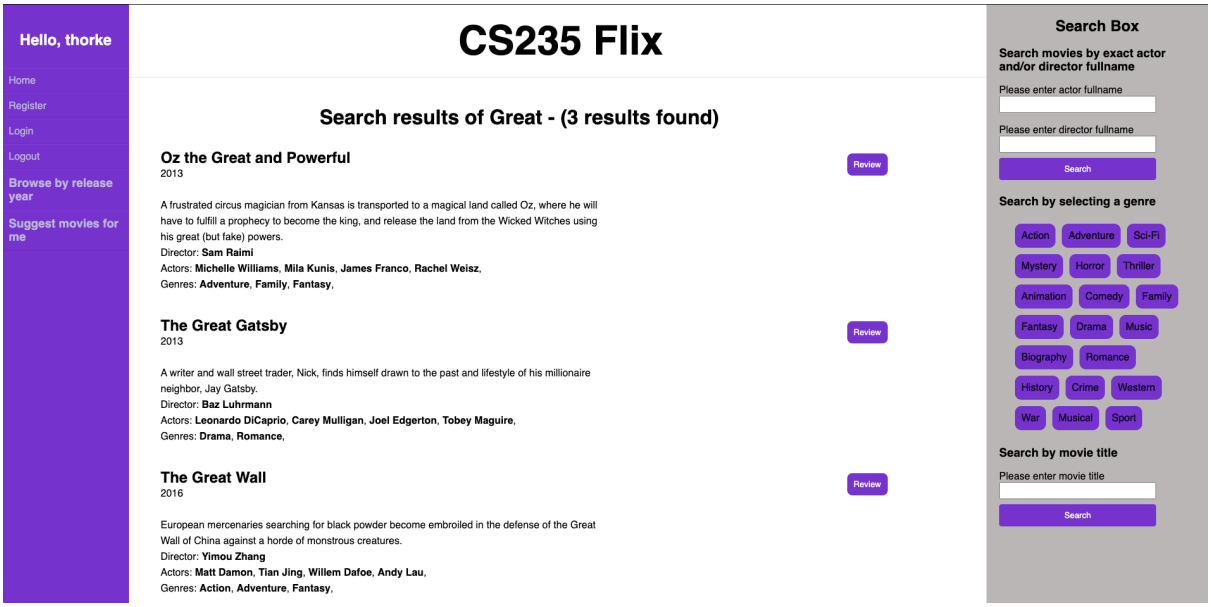
Executive Summary

CS235 Flix is a web application that allows people to view and search movies using a web browser. CS235 provides basic movie browsing features such as by release year and by genre. CS235 also provides basic features such as allowing a user to register an account, logging in and out a user, searching movies by actor and/or director full name, and allowing logged in users to review movies. However, in addition to these basic features, the author also implemented the following "new cool" features, including searching movie by title with a fuzzy search, retrieving movie cover image from a web API, and two other movie recommendation features. This journal describes these new features and the key design decisions the author made in developing this project.

Feature 1 – Advance searching features – fuzzy search by title



Feature Description of Feature 1:

In addition to searching movies by actor and/or director full name, CS235 also provides the functionality of fuzzy searching a movie by title. For example, when a user types in a keyword, "Great", the web application will search its repository and return all movies that have a title which contains the keyword "Great". For the current version of web application which uses "Data1000Movies.csv" as its repository, will return three movies: "Oz the Great and Powerful", "The Great Gatsby", and "The Great Wall".

Design Decision and Justification of Feature 1:

The search by title feature is implement on the service layer inside the movies blueprint, where a WTForms is used to obtain input from users. The implementation of this features follows closely to the Single Responsibility Principle, where two separated routes are used to handle a user's request. The first route ("/search_by_title") validates the WTForm and, if the form is valid, directs the user to the second route("/search_movies_by_title"), which uses service layer functions to retrieve movies that have titles similar to the search keyword. The second route then renders the movie template and with the help from jinja templates, the web application will display the search results back to the user.

The development of this feature also follows closely to the **Dependency Inversion Principle** introduced in the course, where functions used in the service layers were initially defined in the abstract repository pattern and developed in the memory repository layer. Unit testing and integration testing have been conducted to ensure the web application functions as expected. With the application of these design principles and patterns, the project is easy to maintain and easy for future upgrade.

Feature 2 – Interacting with a web API (IMDb py) to receive cover image for movies

Feature Description of Feature 2:

The author chose to use the free IMDb API instead of the paid OMDB API to retrieve cover image URLs for movies. The IMDb API allows the author to search movies based on a movie title and returns information of any matched movies as a list of Python dictionary objects, in which the author could extract the movie's cover image URL through the field, ['cover url']. This URL will then be passed to the browser through a jinja template and eventually be displayed on the home page of the web application.

The movie cover images provided by the IMDb API are of poor resolution, but the author consider this is sufficient to demonstrate the author's learning outcome of interacting with web APIs.

Design Decision and Justification of Feature 2:

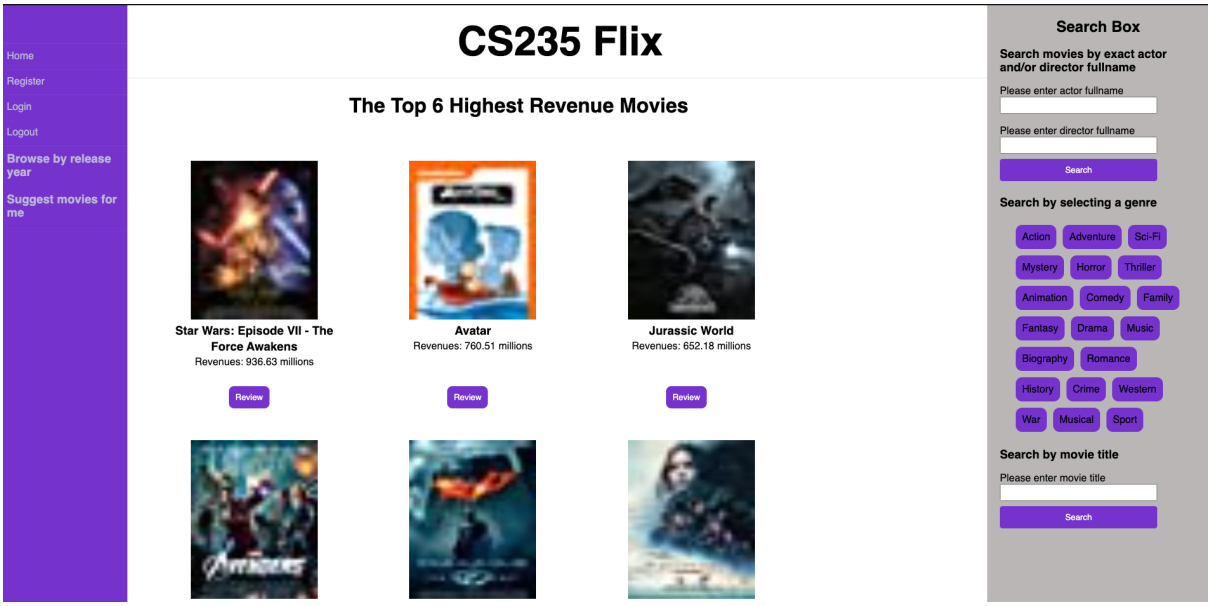
Please note that SQLALchemy is a dependency module of the IMDb API, but the author DIDN'T use any database-based repository for this Web Application, and hence should not be considered as violating the specification of this assignment.

Additionally, since the IMDb API is searching through the IMDb library, it will take some time for the API to return the search results, which means a user will need to wait for a short time to load up the home page. As such, the author decided to only receive movie images via the IMDb API for the six movies located on the home page, otherwise the web application will take too long (if not forever) to load. A much better approach would be to have the image URLs stored in the local repository and/or database, but this will violate the specification of the assignment and make this new feature more trivial.

[An important message for markers: when you start the web application, it will take a while (10 seconds for the author) to load up the home page, for the reasons described above. The long loading time occurs again when you run PyTest, as it tries to perform an integration test for the home route. All other functions of the Web Application have (almost) no loading time. If one would like to disable the API, please go to CS235Flix>home>home.py and comment out line 23 and 24 to stop extracting movie image URLs from IMDb. The author's apology to any inconvenience this may cause and thank you so much for your understanding]

This new cool feature is implemented on the service layer inside the home route to show movie cover photos for the selected movies on the home page. This new feature aims to improve the aesthetic pleasure of the web application. This new feature is considered as a "small but non-trivial" upgrade to the home route, hence, it still follows the Single Responsibility Principle and does not violate the Dependency Inversion Principle.

Feature 3 – General movie recommendations to all users

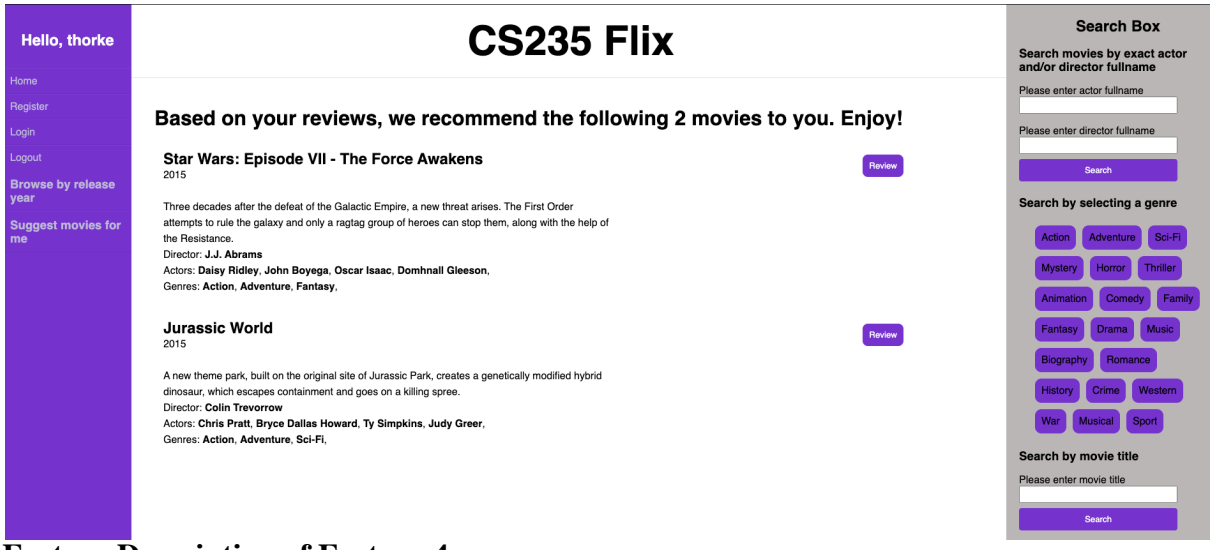


Feature Description of Feature 3:

On the home page of CS235 Flix, six movies are chosen to recommend to all users of the web application. These are the six movies with the highest revenue recorded in the memory repository. Modifications have been made to the Movie domain model to store each movie's revenue. Based on the records available in the "Data1000Movies.csv", the top six movies recommend to users are: "Star Wars: Episode VII – The Force Awakens", "Avatar", "Jurassic World", "The Avengers", "The Dark Knight", and "Rogue One". Please note that movies have "N/A" as their revenue in "Data1000Movies.csv" was considered to have 0 revenue.

The design principles will be further discussed in Feature 4.

Feature 4 – Personalized movie recommendations to logged in users



Feature Description of Feature 4:

The final new feature provides personalized movie recommendations to a logged in user based on the user's reviewed movie. An un-logged in user will be redirected to the login

page. If a logged-in user has not yet reviewed any movie, this page will show a message that encourages the user to watch and review some movies. The movies recommended to a user are based on the genres of the user-reviewed movies, where the highest revenue movie of each genre will be recommended to the user. In the example above, the user thorke has reviewed the move "Guardian of the Galaxy", which is classified as Action, Adventure and Sci-Fi, hence the highest revenue movies that are classified by any of these three genres are recommended to throke. In this case, "Star War: Episode VII – The Force Awakens" and "Jurassic World" are recommended to the throke.

Design Decision and Justification of Feature 3 and 4:

The author decides to recommend the highest revenue movies to users because higher revenue usually means higher quality and popularity of the movie, and hence more users will be interested in viewing them.

The author follows closely to the Single Responsibility Principle, where unique routes have been used for each of the two movie recommendation functions. Each of these recommendation functions have their fundamental methods developed in the repository layer, where each of them has a single responsibility. For example, a method that gets all the genres that a user is interested in based on user-reviewed movies. There is another method that integrates any necessary "single responsibility" methods to achieve the desired outcome, such as recommending movies to a user. This method will be used by the service layer to extract recommended movies to a logged in user. (Similar design approach applied to Feature 3)

This design/development approach promotes maintainability and extendibility of the Web Application, and the "single responsibility" functions could be useful to other future features. Each function has been thoroughly tested through unit testing. All routes have been tested with integration testing.

Minor changes routing decisions compared to the COVID web app

Despite the CS235 Flix web application follows closely to the design decisions made in the COVID web app, the author made the following changes to some of the routing decisions to better suit the context of the web application:

- After the user submits a review, the user will be redirected to the Search by Title page with the movie the user just reviewed being the search result. This allows the user to conveniently view the reviews he/she just made.
- After the user has successfully logged in, instead of redirecting the user to the home page, the web app will redirect the user to the "Suggest movies for me" page. This approach will allow the user to access the latest movies personalized for him/her.

Other comments:

The author follows closely to the design patterns and design principles taught in the COMPSCI 235 course during the development of CS235 Flix. The author also applied the "Visibility of system status" (one of the 10 Nielson's Heuristic, taught in COMPSCI 345) principle by providing the total number of search results to the user. Since there is a limit on the number of movies that can be displayed on a single Web page, CS235 provides information about what number of results the user is viewing to the user, for example, "you are viewing 30 of 101 results".

Conclusions

In conclusion, this assignment has been an interesting journey for the author, where he has learned quite a lot about software development and he really enjoys the process.