CompSci 235 Assignment 2 Report

11/10/2020

My web application for CompSci 235 Assignment 2 includes a feature whereby a logged-in user can manage a list of movies for later viewing. This list exists in the order in which movies are added to it, and is unique to the user. I decided on this feature because of the prior work put into the domain model for such a feature (accomplished during Assignment 1), and my confidence in executing the addition of this feature to the application.

A logged-in user has the ability to add and remove movies from their watchlist via a button located at the bottom of a movie's information page. If the movie being displayed on the information page is not in the user's watchlist, this button will be labelled "Add to Watchlist", and will add said movie to the end of the user's watchlist. If the page's movie is in the user's watchlist, the button will be labelled "Remove from Watchlist", and will remove said movie from the user's watchlist. Should the user not be logged in, this button will direct the user to the login page.

I decided on incorporating this add/remove button because it provides an intuitive, user-friendly solution to the task of adding and removing movies from the user's watchlist. I would have liked to have added this button to the articles shown when searching for movies, so that a user would not be required to make an additional navigational step when adding to their watchlist (particularly when searching for multiple movies to add).

A logged-in user is able to traverse through the watchlist in two separate views:

- List-view: Shows the user's watchlist as a list of articles in similar fashion to the other list-displaying pages of this application. Each article directs the user to the corresponding movie's information page, acting as a transition into detail-view (see below).
- Detail-view: Shows the information page for a single movie, with buttons allowing the user to navigate to the next or previous movie in the watchlist. If no previous/next movie exists in the watchlist, the corresponding button is not shown.

A logged-in user can access list-view through a tab in the navigation panel labelled "My Watchlist". This tab will only show if the user is logged in and their watchlist contains at least one movie, as a tab which only shows a blank page or redirects the user to a page that is already linked to (i.e. login) would be redundant.

It is desirable to keep the elements of such a list short and concise, as this allows for more efficient browsing, and also allows more elements to be displayed on a single page. However, it is also desirable to access the complete set of information for each movie in the list – as well as the multiple functions supplied by the different buttons on the information page – without having to backtrack to list-view every time the user changes movies. Having these two different means of traversing a user's watchlist provides a handy solution to these contrasting issues.

I decided to add this watchlist feature to my application working in my 'movies' blueprint, as it is the responsibility of the 'movies' blueprint to handle requests concerning movies and how they are listed/displayed (adhering to the single responsibility principle). Here I added a new blueprint route for the endpoint '/watchlist'. This allowed me to separate the concerns of other list-displaying modules/endpoints from the responsibility of serving up the watchlist in list-view (once again adhering to the single responsibility principle). I also made slight modifications to the 'User' class in my domain model. These modifications included adding a field 'self.__watchlist' to the class' constructor, which is used to store a 'User' instance's particular watchlist as an instance of the 'Watchlist' class (previously defined in Assignment 1). I also added a property/getter method to the class for accessing this field.

This application allows the user to search for movies by title, director, starring actor, and genre. Searching by title shows a list of articles corresponding to movies whose titles start with or otherwise include the search term. These articles contain a movie's title, release year and description. Should the number of movies that fit the search exceed the maximum number of articles that can be shown on a single webpage (4 by my screen's estimation), a button appears at the bottom of the list labelled "Next" which directs to a page displaying the next set of articles. A button labelled "Prev" is shown on all subsequent pages which directs the user to the previous page of articles. My application's homepage displays a random list of 5 articles in the same list-of-articles format (although no page-traversal is needed for the homepage).

I decided on this 'page-by-page' approach to showing lists of movies (as opposed to alternative methods, such as limiting the number of results returned by the search) because it provides a simple, elegant solution to having a severely limited number of articles per page. This listing method allows the user to narrow down their search without being concerned about potentially interesting movies being hidden from view because other titles fit the search term slightly better (or are simply higher in the alphabet). The search is non-case-sensitive, and is not exclusive to the beginning of title strings for the same reason.

When searching by director or starring actor, a list of articles pertaining to directors or actors (rather than movies) which fit the search are displayed. Each of these articles contains the full name of the person, and when clicked directs the user to a list of movies (as with searching by title) said person directed/starred in.

My original intention was to have these search options directly display a list of movies affiliated with directors/actors who fit the search. I decided to narrow this search down by first displaying a list of directors/actors, as this allows for a more direct search for a specific director/actor's movies, rather than requiring the user to browse through a muddied list of movies that may or may not pertain to their person of interest.

The list of actors also provides a button in each article labelled "Colleagues" which directs the user to a list of the actor's colleagues (i.e. actors who have co-starred in movies with the specified actor), allowing users to peruse the database of actors by association and find their respective collection of works.

When searching by genre, no search term is required. Rather a list of all the genres in the database is displayed (on multiple pages, by the method mentioned previously). From here the user can select their preferred genre and be taken to a list of all the movies in that genre.

As the list of genres is comparatively much smaller than the list of movies, directors or actors, I felt that a search form would not be required to narrow down the list to the user's specified preferences in genre.