Feras Albaroudi
falb418

COMPSCI 235: Assignment 2

GitHub

https://github.com/NeedsSoySauce/COMPSCI-235-A2

Extension

The extension for this project was:

- A watchlist which allows users to track movies they have watched and want to watch.
- The following user account actions:
    - Change username
    - Change password
    - Delete account and all associated data
- A small change (which goes against the assignment specification a bit) was to allow unauthenticated (anonymous) users to post reviews.

Key design decisions

Watchlist

A key functionality I wanted was the ability to add an ‘I want to watch” and I’ve watched this’ button anywhere I like.

- To support this I decided to create POST and DELETE methods to add a movie and remove a movie from a user’s watchlist. These methods return a plain text response indicating whether or not they were successful.
- This created a problem, as I couldn’t make a DELETE request using a form, so I opted to use a small amount of client-side javascript to make the calls using fetch and then reload the page.

At this point buttons could be added anywhere and all I needed to do was make a call to the web api endpoints. Regarding the design of how the endpoints would update a user's watchlist.

- To achieve a separation of concerns, I moved any method calls on the domain models into the service layer to keep things organized and decouple the view from the domain models (thus allowing changes to be made to the domain models without impacting the view).

Account actions

Challenges I faced in implementing these were:

- I wanted a single page that didn’t require authentication to view to have multiple forms that did require authentication for their actions
- I wanted users to be able to change their username, which meant a user would need another way to identify them that never changes (a user id)

The first challenge was resolved by creating multiple views that themselves required authentication that returned the unauthenticated ‘user’ view. From there I was able to determine what action to take based on the request’s path and method.

The second challenge was more difficult. I needed to make changes to my repository as well as user domain model to support this, and I needed to add methods to my service layer to abstract away the account actions I wanted (as these actions would affect domain models as well as the repository, but I wanted to keep separation between the view layer and the repository and domain models).

- The repository was updated to:
    - Map a username to a user id and keep that mapping updated.
    - Map a user id to the reviews posted by that user.
    - Delete a user and any reviews linked to them.
- The user domain model was updated to:
    - Include an id parameter in it’s constructor that once set was protected from being set again using python’s property decorator.
- The auth service was updated to:
    - Include services to update a user’s domain model and services to execute methods on the repository

The repository pattern was especially useful here as being able to abstract away the action to delete a user meant that in the future if the domain models or method of modeling the relationships between reviews and users changes the rest of the application wouldn’t be affected.