myFlix Angular Client

Building a movie web app
with Angular and REST APIs

myFlix Angular Client

Overview

This project was developed as part of my Full-Stack Web Development coursework and served as the Angular front-end client for the Movie API I previously built.

The application enables users to browse a catalog of movies, view detailed descriptions, register an account, log in, and manage their favorite movies. While the functionality was similar to an earlier React-based version, building this Angular client required learning an entirely new framework and rethinking how to integrate the existing backend without introducing new issues.

Tech Stack:

Angular, Angular Material, TypeScript, Node.js/Express (API), MongoDB, GitHub Pages, Render

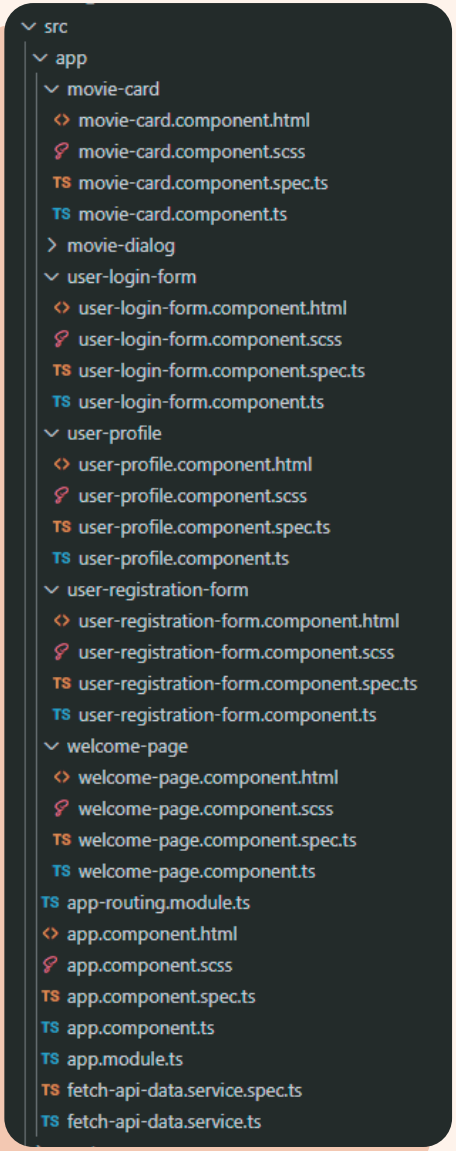
Data Model & API Integration

To start, I designed Angular services to connect to my existing Movie API.
This included endpoints for:

- **User registration and login**
- **Managing favorite movies**
- **Retrieving movie details**
- **Updating or deleting user profiles**

Challenge:
Since the same API was also powering my React client, I had to carefully test each change to ensure I didn't break existing functionality in the React project.

Solution:
I created separate environment configurations and tested each client independently in Postman and in-browser.



Building the Core Application

I used Angular CLI to scaffold the project and created key components:

Welcome View

Registration and login forms

Movie Card Grid

Displaying all movies

Movie Details Catalog

Showing synopsis, genre, and directors

User Profile

Managing account settings and favorites

Routing and lazy loading were set up to create a smooth single-page experience.

Styling with Angular Material

For UI styling, I used Angular Material's prebuilt components, including cards, dialogs, and form controls.

```
@use '@angular/material' as mat;

@include mat.core();

$myFlix-Angular-client-primary: mat.define-palette(mat.$deep-purple-palette, 500, 300, 700);
$myFlix-Angular-client-accent: mat.define-palette(mat.$blue-grey-palette, A200, A100, A400);

$myFlix-Angular-client-warn: mat.define-palette(mat.$red-palette);

$myFlix-Angular-client-theme: mat.define-dark-theme((
  color: (
    primary: $myFlix-Angular-client-primary,
    accent: $myFlix-Angular-client-accent,
    warn: $myFlix-Angular-client-warn,
  )
));

@include mat.all-component-themes($myFlix-Angular-client-theme);
```

Snippet of custom SCSS theme configuration for Angular Material components.

Challenge:
Customizing Angular Material styles proved difficult. Many components relied on deeply nested classes, and I often had to override styles using the ::ng-deep selector or adjust global stylesheets.

Solution:
I created a reference sheet mapping Angular Material class names to their corresponding SCSS selectors. I also experimented with Angular's theming system to apply consistent colors and typography.

Authentication & User Management

Integrating authentication required careful attention to token management and session handling:

User credentials were sent via HTTPS to the backend.

The API returned a JWT token stored in localStorage.

Subsequent requests used the token for authorization headers.

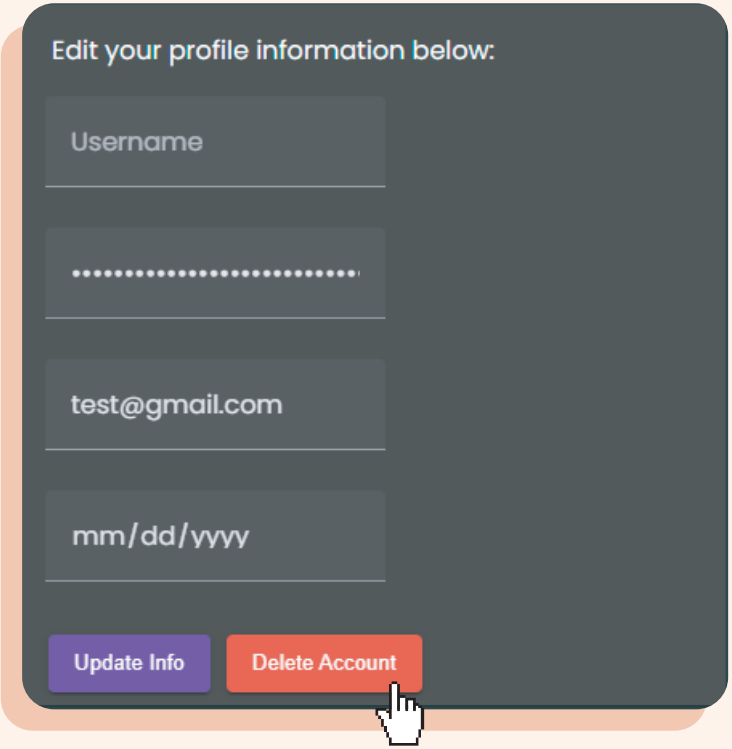
Challenge:

Because my API was also in use by my React client, I had to navigate token validation without conflicting with existing logic. I encountered issues where refreshing the Angular app would sometimes clear the stored token unexpectedly, requiring the user to log in again.

Solution:

I implemented an Angular AuthService to consistently manage login state and token storage, and wrote helper functions to validate the presence of the token on app initialization.

Deleting Accounts & Logout Flow



One specific challenge was allowing users to delete their accounts securely.

Initially, the API call would remove the account from MongoDB, but the Angular app continued to show the user as logged in.

This created a potential security risk where a deleted account could still access protected routes.

Solution:

I updated the deleteUser() function to clear localStorage and force a redirect to the Welcome view upon successful deletion. This ensured immediate logout and session termination.

Results & Reflection

The myFlix Angular Client successfully delivered a polished, responsive single-page application that integrated seamlessly with my Movie API.

Wins

- Learned Angular fundamentals, including components, services, and routing.
- Gained experience customizing Angular Material.
- Built a robust authentication flow using JWT tokens.

Lessons Learned

- When reusing a backend across multiple clients, it's important to carefully test all integrations before deploying.
- Angular Material offers powerful UI components, but customizing them requires patience and experimentation.
- Small gaps in session management (like not logging out a deleted user) can create significant security risks.

Final Thoughts

This began as a familiar project in concept and turned into a meaningful learning experience in adapting existing APIs to a new frontend framework. Completing this Angular client improved my confidence working with TypeScript, Angular services, and advanced UI libraries.