myFlix Angular Client

A Web Development Case Study

OVERVIEW

PROJECT

To use Angular to build a front-client for an API called myFlix that I had created earlier for CareerFoundry's Full Stack Web Development course. The myFlix API had been designed to provide users with information about movies, movie directors and genres, allow them to establish profiles and select movies as favorites, and update their information.

CONTEXT

I decided in 2020 to kickstart a new career for myself as a web developer by enrolling in CF's Full Stack Web Development course. This project was the crowning achievement of the curriculum, which I produced in order to expand my skill base to include Angular after completing several projects with React.

TIMEFRAME 1 month

ROLE Full Stack Web Developer

SUPPORTERS (thank you!)

- Alfredo Salazar Vélez (CF mentor)
- Jay Quach (CF tutor)

TOOLS

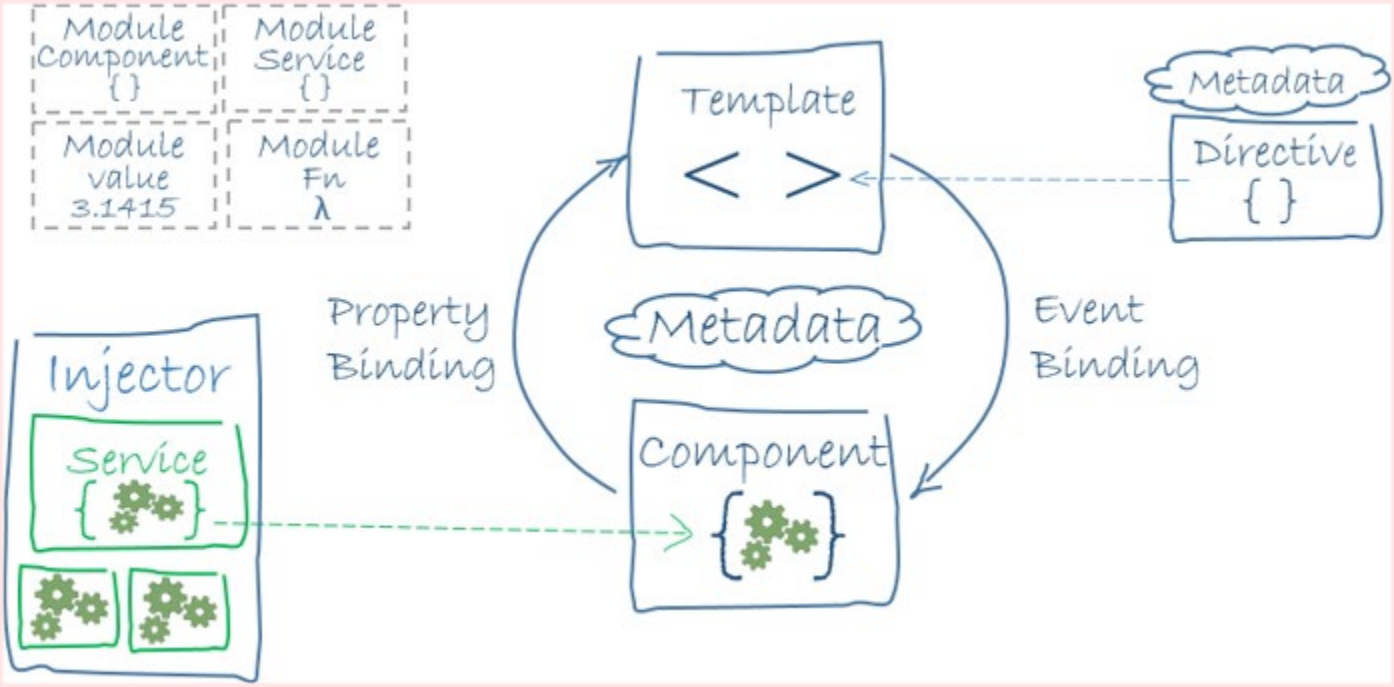
- Angular
- JavaScript

THE PROCESS

1. SETTING THINGS UP

The learning curve for Angular is **very steep**!

Having just completed several projects using React, it was disorienting to understand how the different elements of an Angular project (modules, templates, injectors, metadata, data binding, services) work together:



(image source: https://angular.io/guide/architecture)

My approach was to work very carefully through the process of setting up just a **basic** Angular project, **checking as I went along** how changes in one part of the project would affect its behavior elsewhere. I learned a lot at this stage, especially about how Angular components and templates interact.

2. CREATING THE WELCOME SCREEN

Problem

After implementing the logic for making API calls in a service file, I hit a snag trying to create an Angular form for users to register and login. My code generated **errors** that were very hard to interpret. Once I even got an HTTP 201 "error," **even though 201 is a success code**, indicating that a resource has been created correctly!

Solution

I finally figured out what was going wrong.

Angular receives API information and **expects** that information in the form of **an object**. But my API was also sending a success message in the form of a **string**—

> "Your account has been successfully created!"

—in addition to the important data. I had to go **back into my API code** to remove the string from the response.

Lesson learned

Check the data type of the info coming from your API!

3. BUILDING THE MAIN VIEW

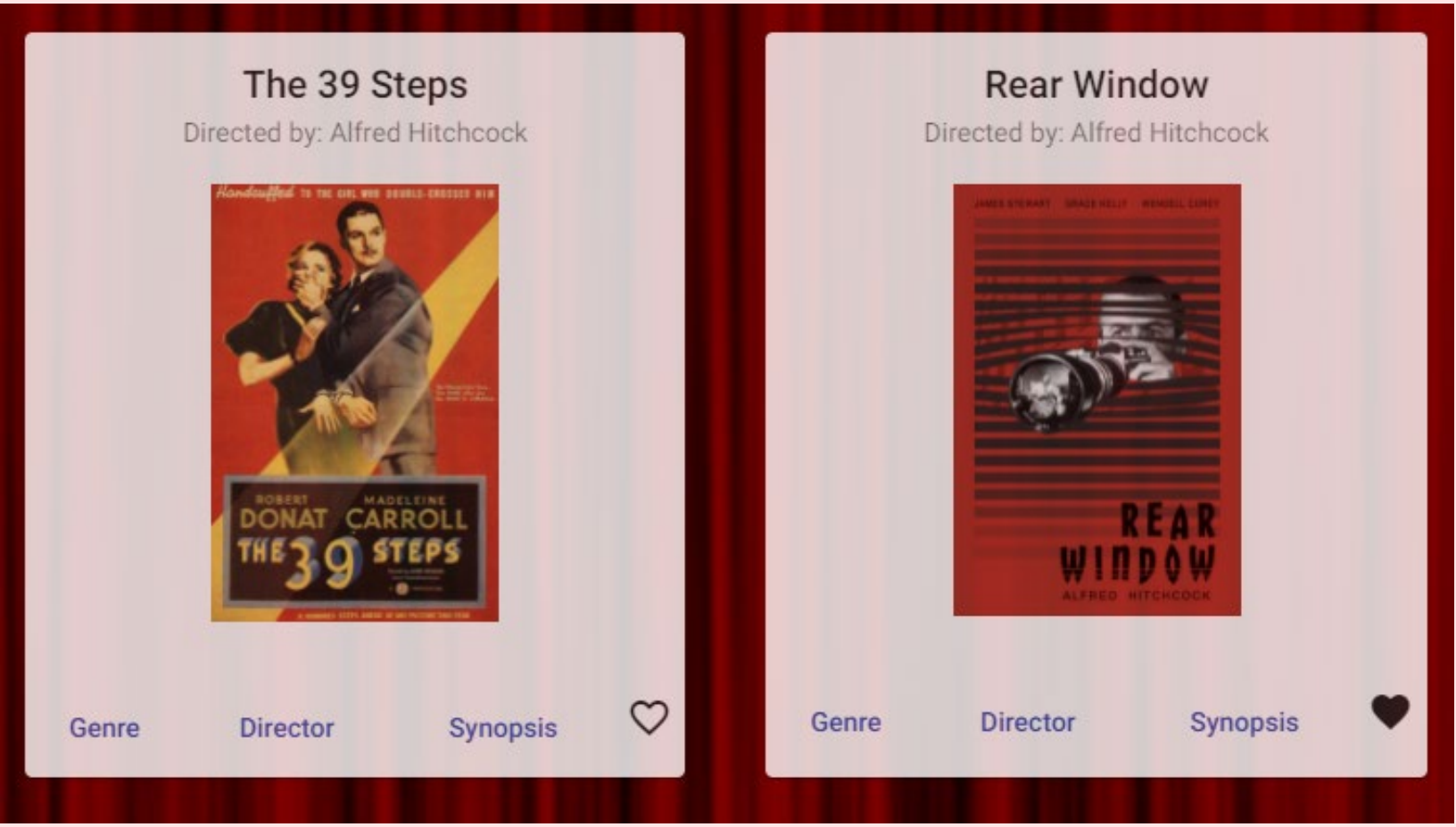
The final step was to set up the main view of the Angular client, where users can browse through movies, check out and update their profiles, and **view and edit their favorite lists**.

To help users see which movies were their favorites, I decided to use Angular's built-in structural ***ngIf** directive. The *ngIf directive **adds** or **removes** an HTML element to the DOM depending on whether a specified condition obtains.

So this is what I wrote:

```
<mat-icon *ngIf="currentUsersFaves.includes(movie._id)" class="fav-icon">favorite</mat-icon>
<mat-icon *ngIf="!currentUsersFaves.includes(movie._id)" class="fav-icon">favorite_border</mat-icon>
```

The idea here was simple: if the value of the **currentUsersFaves** variable included the specified movie, then a filled-in heart would display; if not, a heart border would display instead, like so—



Problem

But it didn't work! In fact, this code made my application behave very **erratically**. Either the wrong heart icon would appear (showing favorites for non-favorites, or vice versa), or else it would just not appear at all. And depending on how I tweaked the code, my application would try to check the favorite status of each movie **dozens of times**!

Solution

It turned out there were two things I needed to do—

```
itIsAFave(movieId: string): any {
  const movieArray: any[] = this.currentUsersFaves;
  if(movieArray.some(movie => movie._id === movieId)){
    return true;
  } else {
    return false;
  }
}
```

(1) First, I linked my *ngIf directive to the result of a **function**, itIsAFave, that returns a Boolean value saying whether the given movie is a favorite or not.

(2) Second, I also needed to rewrite the functions that run when the main page loads, especially the function for **setting the currentUsersFaves variable**, which then had to be linked to the icon html element.

```
getMoviesAndFaves() {
  this.fetchApiData.getAllMovies().subscribe((res: any) => {
    this.movies = res;
    this.movies.forEach((movie) => {
      if (this.user.FavoriteMovies.includes(movie._id)) {
        this.currentUsersFaves.push(movie);
      }
    });
  });
}
```

Lesson learned

*ngIf works best when linked to a function that returns a Boolean!

THE RESULT

PROJECT OUTCOMES

The final product **fulfilled all the requirements successfully**. I created a well-constructed, bug-free Angular client for accessing my myFlix API.

A STRONGER DEVELOPER

I became a much stronger developer from this experience. I now have a solid foundation for working with Angular, and my debugging skills got some good practice working on *ngIf.

But my **biggest surprise** was the problem I ran into with the success message string from my API, since I had earlier created a React client for the same API without any such difficulty. I learned from this that because of how picky Angular is about **data types**, it pays to be careful not only about the data types within an Angular project, but also the data types that Angular receives from external sources such as an API. I will keep this in mind in future projects, not only when I am working with Angular, but also when I am creating APIs that might be accessed by Angular clients.