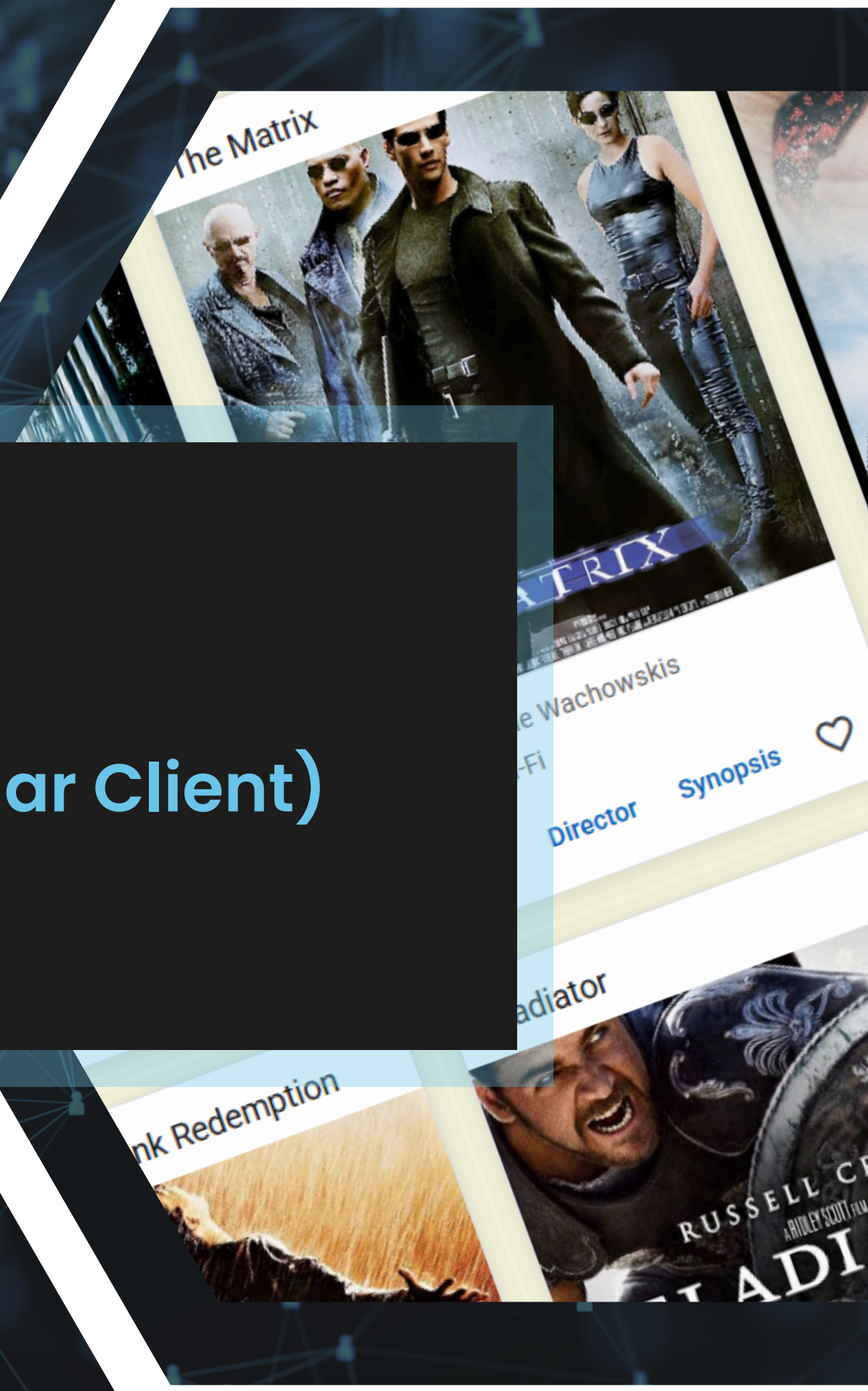
CASE STUDY

myFlix

Full-Stack Movie Database (Angular Client)

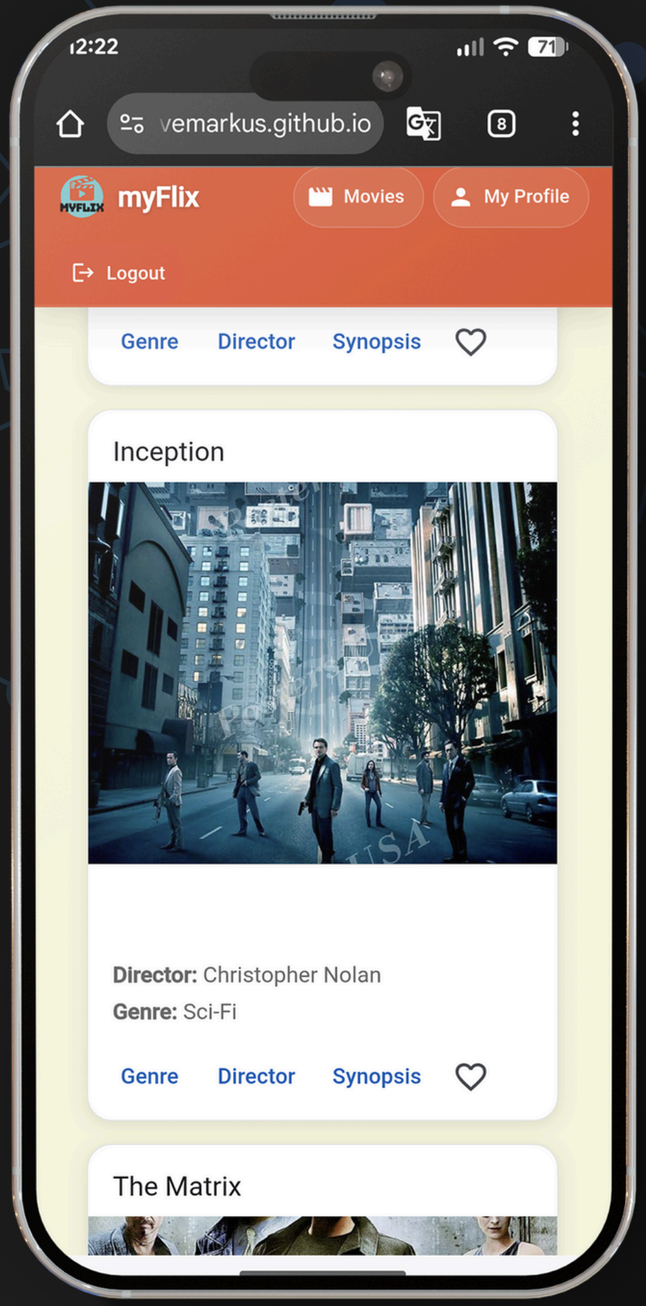
MARK FRANCIS



Overview

myFlix is a full-featured, single-page web application built with Angular 17+ that allows users to explore a movie database, view detailed information about films, directors, and genres, and manage a personalized list of favorite movies. Users can register, authenticate securely, update their profiles, and curate their own favorites collection. The application communicates with a live backend API and is deployed as a production-ready client on GitHub Pages. The project emphasizes clean architecture, scalability, documentation, and professional UI/UX using Angular Material.

live APP



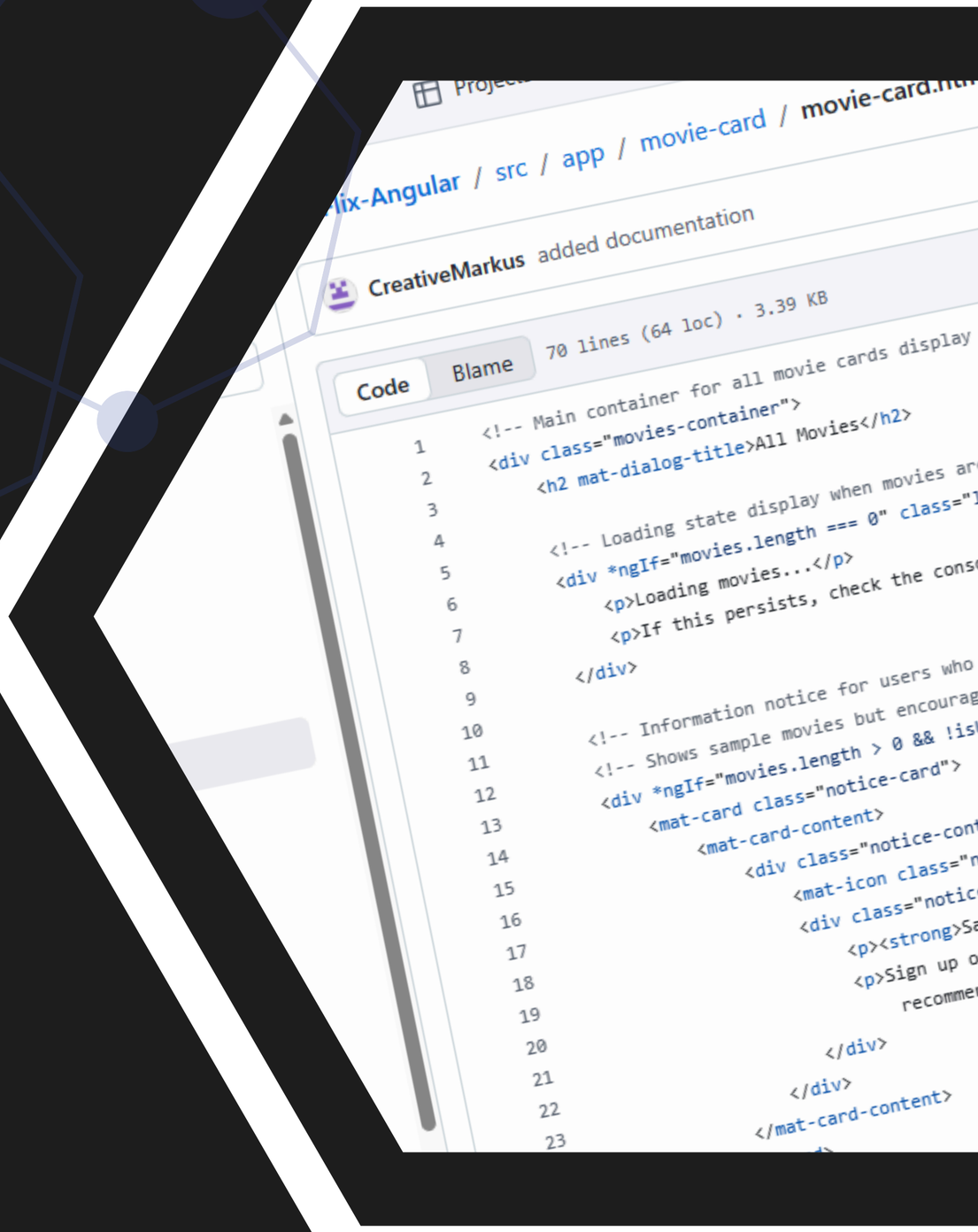
Purpose

This project was created as part of the CareerFoundry Full-Stack Web Development Program to demonstrate mastery of modern frontend development using Angular, along with real-world API integration, authentication, deployment, and documentation practices.

Objective

The primary objective of this project was to build a complete client-side application that consumes a RESTful API and demonstrates the ability to:

- Build a scalable Angular application from scratch
- Implement secure user authentication
- Manage application state and user data
- Create a professional, responsive UI
- Deploy and document a production-ready application suitable for a portfolio



Duration

The project was developed over several weeks. While the core functionality was implemented early, additional time was spent refining the architecture, improving error handling, enhancing UX with Angular Material, and generating comprehensive documentation using TypeDoc.

Credits

- Lead Developer: Mark Francis
- Program: CareerFoundry Full-Stack Web Development
- Role: Sole developer (design, development, documentation, deployment)

Methodologies & Tools

- Angular 17+ (NgModule-based architecture)
- Angular Material
- TypeScript
- RxJS
- Angular Router
- SCSS
- JWT Authentication
- TypeDoc
- GitHub Pages
- Heroku (Backend API)

Server-Side Integration

The client application connects to a pre-existing RESTful API hosted on Heroku. The API provides endpoints for user authentication, profile management, and movie-related data, including directors, genres, and favorites.

All data exchange occurs in JSON format and follows REST principles using standard HTTP methods such as GET, POST, PUT, and DELETE.

To ensure reliability, the application includes fallback mechanisms using localStorage, allowing certain user data (such as favorites) to persist even if the API is temporarily unavailable.

API Service:

FetchApiDataService centralizes all HTTP communication, improving maintainability and scalability.

Backend URL:

https://movieapi1-40cbbcb4b0ea.herokuapp.com/

Client-Side Architecture

After completing the API integration, I focused on building a clean, modular Angular client using NgModules and reusable components.

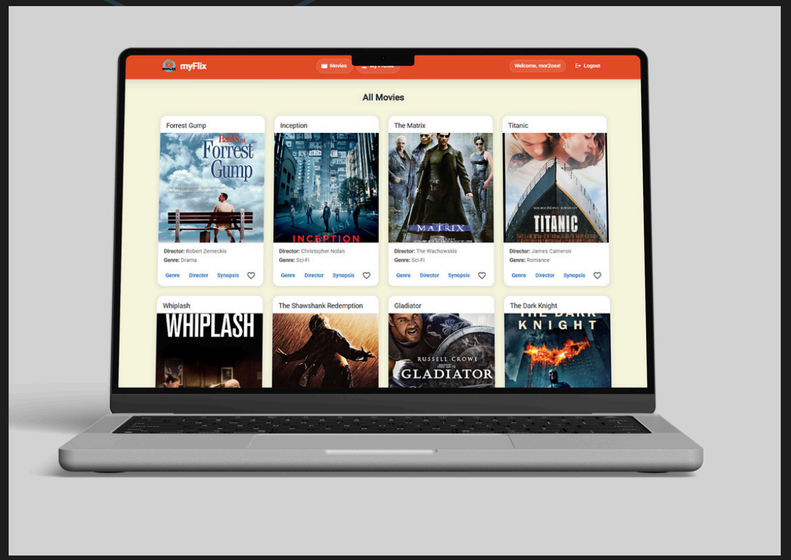
The application is structured around clearly defined responsibilities:

Core components
manage layout, navigation, and routing

Feature components
handle movies, user profiles, and authentication

Dialog components
provide focused views for movie details, directors, genres, and synopses

Services
handle all API communication and state persistence



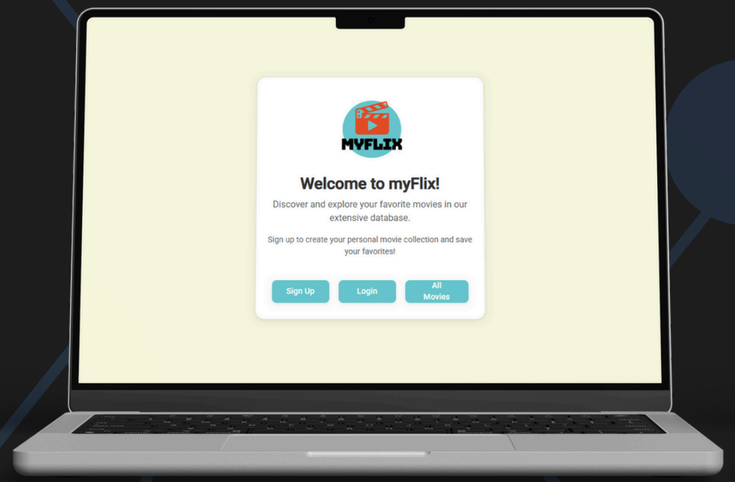
This separation of concerns ensures that the application is easy to extend and maintain.

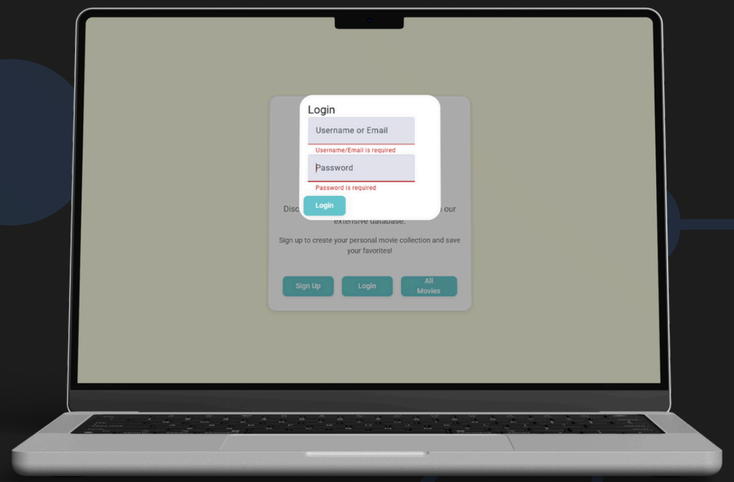
Authentication & Security

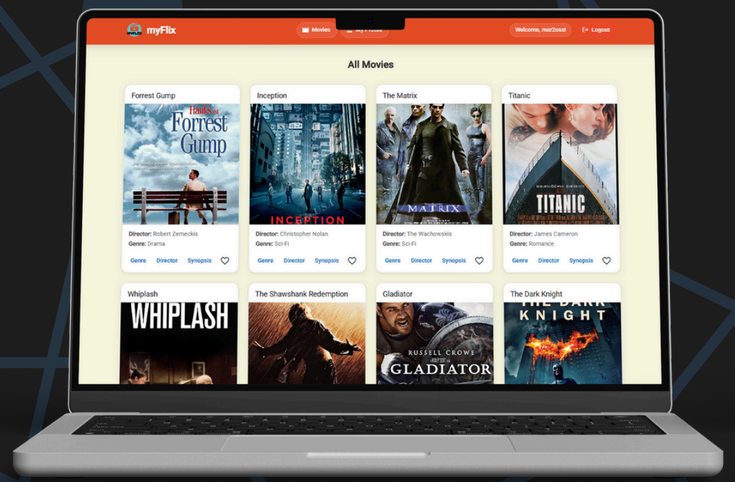
- User authentication is handled via JWT tokens, which are securely stored and attached to outgoing requests. Route guards are used to prevent unauthorized access to protected views, ensuring that only authenticated users can browse movies or manage profiles.
- Additional safeguards include:
- Form validation
- Error handling with user-friendly feedback
- Secure token management and logout handling

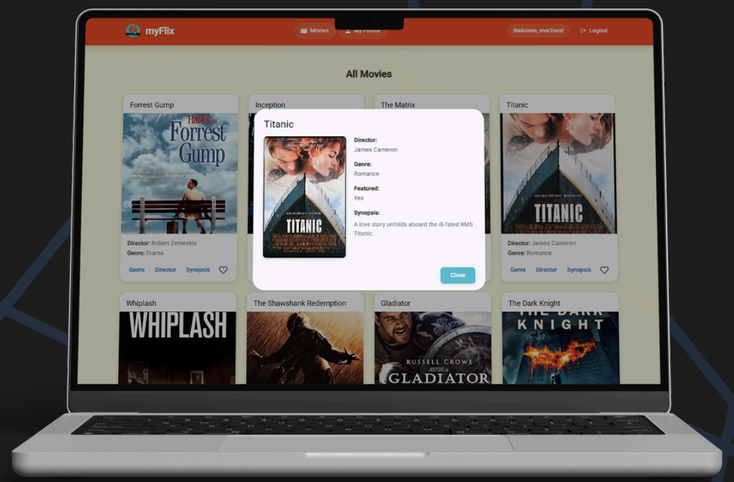
UI / UX Design

Angular Material was used to implement a consistent, professional design system across the application. Material components such as cards, dialogs, toolbars, and form controls provide a clean and accessible interface.









Page Views

- Welcome / Login View
- Registration View
- Main Movie Catalog
- Movie Detail Dialogs
- Director View
- Genre View
- User Profile View

Retrospective

What went well

The overall architecture came together cleanly. Using Angular services and RxJS allowed for a scalable and maintainable approach to data handling. I also particularly enjoyed generating comprehensive documentation with TypeDoc, as it reinforced good documentation habits and made the project feel production-ready.

What didn't go well

One of the biggest challenges was time. While working on this project, I was balancing development with childcare several days each week, which meant progress sometimes happened in shorter, more focused sessions rather than long, uninterrupted stretches.

Angular is a powerful framework, but its structure can be verbose, and keeping everything consistent across modules took more planning than I initially expected. Authentication and JWT token handling were also more challenging than anticipated, and debugging edge cases around protected routes and persisted sessions required extra time and patience.

Future Steps

In future iterations, I would like to:

- Add advanced filtering and sorting options
- Improve offline support
- Introduce pagination or infinite scrolling
- Enhance accessibility features
- Expand testing coverage

Final Thoughts

Overall, this project significantly strengthened my confidence with Angular and frontend architecture. Building a complete, deployed application with documentation and real API integration gave me a strong understanding of how production-ready Angular applications are structured and maintained. I'm proud of the final result and would comfortably extend or refactor it for real-world use.