TRAXSMART FRONT-END ASSIGNMENT

SUMAN MITRA

15TH OCTOBER 2020

Activity Time: **18hrs** (approximate)

Website Github

INTRODUCTION

The following report describes the complete development process for the assignment. It includes the tools, frameworks, deployment, hosting details and code overview.

TOOLS & FRAMEWORKS USED

I have chosen the following tools for developing this project. I prefer **VS Code** amongst the code editors and chosen to work with that. I've used **node version 12 on windows x64 platform and angular 10** for the framework. I have used **chrome** browser during the development process for live feedback. And lastly, for the versioning tool, I have chosen **Github**.

DEPLOYMENT & HOSTING

As the source code is deployed in Github I have chosen **Netlify** for the deployment and hosting provider. The Github integration in Netlify helped me easily create an auto-deploy pipeline. As Netlify only provides a random URL for the deployment URL I've used my domain and added a **CNAME** entry in the **Route 53 DNS Record** to point it to the random URL resource provided by Netlify. I have also used their **Let's Encrypt SSL** certificate issuance feature for the custom domain name and have a valid SSL certificate for a secure connection.

CODE OVERVIEW

I have chosen to keep the project very simple and easy to follow and thus I have refrained from creating multiple angular components as the project is relatively small and didn't require the modularity of multiple components.

The project is mainly divided into two parts i.e. the main app component and the service file. Let's have an overview of the service class first.

THE MOVIE DB API SERVICE

The Movie DB API requires an API key for operations firstly I had created a project in the developer portal of their API and acquired and API key for further operations. I have chosen the version 3 API from the available Movie DB API versions. Then I proceeded to create a few variables to hold the Base URL and the API key for reuse in the service class. The service class uses **HttpClient** module from **@angular/core** to handle the HTTP requests.

There are mainly four service methods and all are GET requests.

MOVIE SEARCH BY NAME API

The first interaction of the app is to search for movies by their name, the **`fetchMovieDataByName`** method is written for that purpose. The method takes the search term as input proceeds with the API call with the API key and returns an observable.

MOVIE DETAILS BY ID API

This method **`fetchMovieDetails`** searches the API with a given movie ID and fetches all the details available for the movie which includes a poster, backdrop image, genres, runtime, release status, release date etc.

MOVIE CREDITS BY ID API

This method **`fetchMovieCredits`** fetches all the cast and crew information available in the database by the given movie ID.

SIMILAR MOVIES BY ID API

This method **`fetchSimilarMovies`** fetches a list of similar movies that are associated with the given movie ID and similar to all the API calls returns an observable.

OTHER

Apart from these methods, the service class holds a few variables that are used as replacement images when an image path is returned as null from the API. These images are base64 encoded image strings that I have generated from a few free to use websites.

APP COMPONENT

The app component is mainly divided into three parts i.e. **HTML, CSS and TS**. Let's discuss these three parts one by one.

HTML

The HTML is categorised firstly into two parts i.e. **header** and **main**.

HEADER

The header contains only three elements i.e. name for the project, search input field and a button.

The input field has **ngModel** property associated with the form group. The button has no type assigned as by default it behaves as a submit button inside a form. The submit functionality of the form is handled by **`fetchMovies`** function written in the TS file.

MAIN

The main part of the document is subdivided into two sections i.e. **`movies-list`** and **`movie-description.`**

MOVIES-LIST

This section contains a group of cards that are dynamically loaded with a **`*ngFor`** directive. These cards hold a poster, title of the movie, release date, rating, overview and a hidden paragraph which holds the movie ID. The cards have click-through child elements and the click calls the **`showDescription`** method of the TS file and passes the event variable to the method.

MOVIE-DESCRIPTION

The description has a few parts

- A backdrop image
- Movie title
- Rating
- Genres
- Runtime
- Languages that are spoken in the movie
- Release Date
- Release Status
- IMDB link
- External link to the movie original website if a link is available
- Budget
- An overview of the synopsis
- Cast list
- Crew list
- Similar movies

The last three are arranged as card groups with the **`*ngFor`** directive. The cast card has a picture if available of the actor, their name and character name. The crew card has their picture if available and their name and the Job they had in the crew. Similar movies are also cards with the movie poster and a title and clickable. After clicking it calls the **`loadMovieDetails`** method which loads the description of this movie.

CSS

I have chosen to use plain CSS as I was advised not to use any frameworks like bootstrap or material UI. I have used media queries to deal with various screen sizes. I have kept it simple and straight forward considering the timeframe of the project. I have also used CSS Grids and Flex for two-dimensional and one-dimensional layout management. Responsive web design principles are maintained throughout.

TS

There are a few interfaces, methods and some class variables in the component typescript file. Altogether it handles the dynamicity of the application. I have chosen not to separate the interfaces for simplicity. The constructor of the class only takes a service class reference.

INTERFACE

There are four interfaces which are required to build several variables including class variables.

- **`MovieShortDescription`** holds the descriptions needed for the cards in movie-list and similar-movies.
- **`MoveDetails`** holds all the details for the movie selected to show in the description field.
- The **`Cast`** has the necessary information for the cast cards.
- The **`Crew`** has the necessary information for the crew cards.

CLASS VARIABLES

Five class variables have interface types associated with them. They are used for binding data in HTML after retrieving necessary information for API calls.

- The **`movies`** variable is of type **`MovieShortDescription`** array and holds all the information required for search result cards.

- The **details** variable holds description fetched from API for a particular movie and is of type **MoveDetails.**
- The **cast** variable holds cast details for the card.
- The **crew** variable holds crew details for the card.
- The **similar** variable holds similar movie card details.

METHODS

There are five methods in the class.

FETCH MOVIES BY SEARCH STRING AND BIND VARIABLE

This method **fetchMovies** is triggered when the user clicks the search button and the form is submitted. This method takes the form group from **ngFrom** directive. The search term is collected from form control of the input field and then passed on to the appropriate service call. After the service method returns the data is then filtered and put on the **movies** variable which is bound to cards that display the movies by that name.

FIND PARENT ELEMENT WITH SELECTOR

This method **findAncestor** searches a selector query pattern in the given elements parents. Returns the last element it has iterated through and stopped on.

FETCH ALL DETAILS OF MOVIE BY ID AND BIND VARIABLES

This method **fetchDetails** takes a movie ID and loads the details of that movie. It has three parts. Each part calls a separate API and loads the data in class variables. Let's discuss these three parts.

DESCRIPTION

This part calls the **fetchMovieDetails** service and loads the **details** class variable.

CREDITS

This part calls the **fetchMovieCredits** service and loads the **cast** and **crew** class variables.

SIMILAR MOVIES

This part calls the **fetchSimilarMovies** service and loads the **similar** class variables.

LOAD MOVIE DESCRIPTION FROM SEARCH RESULTS CARD

The method uses the **findAncestor** method to find the parent with the **movie** class and then grabs the movie ID from the hidden paragraph of that card and then calls the **fetchDetails** method with ID.

LOAD MOVIE DESCRIPTION FROM SIMILAR MOVIES CARD

The method uses the **findAncestor** method to find the parent with the **similar-movie** class and then grabs the movie ID from the hidden paragraph of that card and then calls the **fetchDetails** method with ID.

The main versatility of the project is completed through these three source files and has the necessary elements.

CONCLUSION

I have chosen simpler but effective methods for completing this project but in retrospect, this project could be made with a more modular approach. I have also chosen to not include angular routing as this is just one component and a single page. Although the project could be further improved with more easy to use frameworks as of now it could be assessed as a complete project for the given assignment.