Angular

By Chandrasekaran Janardhanan

Contents

Session pre-requisite

- ❖ Basic understanding of HTML and CSS is required.
- ❖ Programming with JavaScript exposure is required.

Install and Setup Node JS, Angular and Visual Studio Code

Check Angular and Node JS installation

- Run the following commands in command prompt.
- Node JS

```
node -v
```

- Angular

```
ng --version
```

- If any one of it is already installed. It is recommended to install the new version after uninstalling Node JS.

Install Node JS

- Go to https://nodejs.org/en/, download the "Recommended For Most Users" version and install it.
- Open command prompt and run the command "node -v" to check if it is installed correctly

Install Angular

- Open command prompt
- Execute the following command to install angular. This will take a while. Await till it completes.

```
npm install -g @angular/cli
```

Steps to follow if Angular or Node JS is already installed

- Open command prompt
- Execute the following command to install angular. This will take a while. Await till it completes.

```
npm install -g @angular/cli
```

Launch application in Angular

- Open command prompt.
- Create a folder named angular which will be used as the root folder for angular learning.
- In Windows Explore, go to this new folder and click on the empty area of the folder bar.
- Type "cmd' and press enter.
- This will open command prompt in this new folder.
- Execute the following command to create the app named "angular-learn". We will use this angular application to learn all the concepts related to angular. This command will take a while, please await the completion.

```
ng new angular-learn
```

- Execute the following commands to change the directory to the root folder of the angular app created.

```
cd angular-learn
```

- Execute the following command to start the application:

```
ng serve --open
```

- The above command will automatically execute the app and open the browser with running app, which means angular is installed successfully.

Install Visual Studio Code

- Go to https://code.visualstudio.com/
- Click on download link
- Execute the downloaded executable to install Visual Studio Code.
- Launch the application and check if it opens up correctly.

Understanding what is a Single Page Application (SPA)

Activity

Multi-page Application

- Go to https://www.thehindu.com/
- Click on a news article
- Observe that the browser page goes blank and then the news article page is reloaded

Single Page Application

- Go to https://translate.google.com/
- Try translating few words
- Try changing the language in either side and translate few words
- Notice that the page is never reloaded

Explanation

Schematic representation of single and multi-page application.

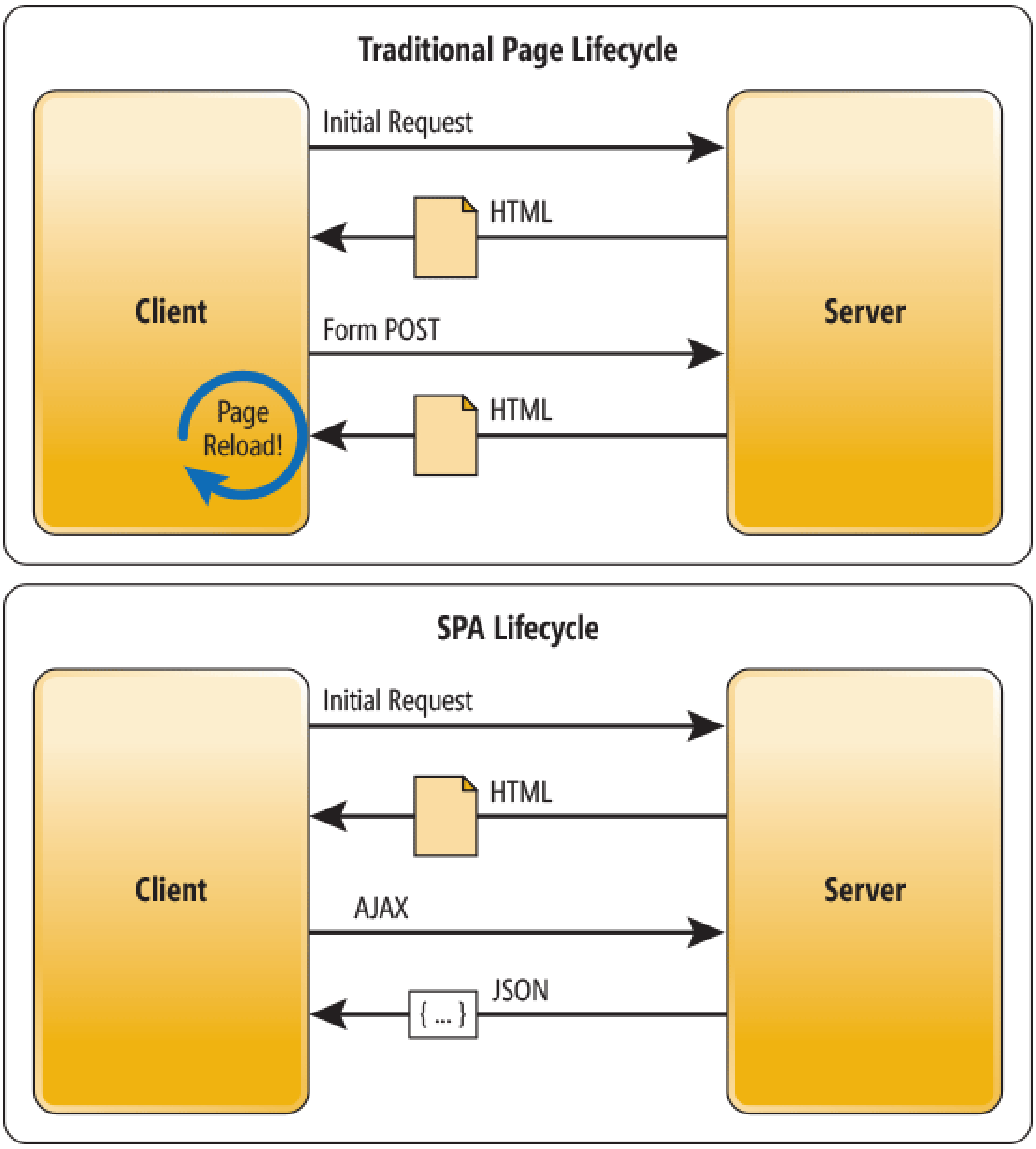


Image Courtesy: https://www.goconqr.com/c/74455/course_modules/113576-single-page-vs-multi-page-architecture#

Multi-page Application

- Traditional applications are mostly multi-page applications
- Multi page applications does not use AJAX.
- The HTTP request send receives a fresh HTML document and the entire page is cleared and freshly rendered by the browser.
- The disadvantage of multi-page application is that there is large data transferred every time a page is accessed, which puts more load on the server.
- Technologies for building multi-page application:
    - Microsoft – ASP, ASP.NET
    - Java – JSP, Servlet, Struts, Spring MVC

Single Page Application (SPA)

- Single Page Application uses JavaScript and AJAX for implementation
- When the URL of Single Page Application is accessed, all the pages of the applications are loaded in a single request and response.
- After the initial load, screen navigation is done based on hiding and showing DOM elements.
- Pages that require fresh data will be using AJAX in the background to get the data from the server.
- One disadvantage of Single Page Application is that, if the website has lots of screens and images, then the initial load time of the application will be high.
- There are various JavaScript frameworks such as Angular, ReactJS and VueJS has evolved which helps in building Single Page Applications.
- These frameworks do not address the server-side part of the implementation, but only helps in building the front-end part.
- These frameworks use JavaScript and AJAX for implementation of their frameworks.

About Angular

- Angular JS version 1.0 was released in 2012.
- Angular JS team realized that it becomes difficult to implement and manage moderately complex applications.
- Based on the concepts of Angular JS a completely revamped framework was developed named as Angular 2, which makes Angular JS completely incompatible with Angular.
- Angular 2 was launched on 14 Sep 2016.
- Angular is an application design framework and development platform for creating single-page apps.
- The version of Angular as of April 2020 is version 9.
- Over these years, Angular has gained tremendous popularity.
- The Angular project is developed and managed by Google.
- Angular depends on TypeScript, which is a Typed scripting language for JavaScript.
- Even as Angular has moved till version 9, the basic conceptual implementation remains the same from Angular 2 to Angular 9. Many enhancements and improvements are made and released, but the core conceptual aspects remain intact.
- Angular web site: https://angular.io/

Get a glimpse of Angular

- The user list display done using jQuery earlier had been implemented using Angular and is available in the link below:

  https://stackblitz.com/edit/angular-fchphp?file=src%2Fapp%2Fapp.component.html
- Open this link and it is implemented in angular
- Click on "user.service.ts" to understand how the AJAX call is made.
- Stack Blitz is a web site for building angular applications online. The examples and tutorial related coding available in https://angular.io is available in Stack Blitz.

Questions

- What is multi-page application?
- What is single page application?
- Multi-page application loads only once. (True / False)
- What are the advantages of Single Page Application?
- Are there any disadvantage of Single Page Application, if so, what is it?
- List few technology frameworks available to implement multi-page application.
- What are the popular JavaScript frameworks available to implement Single Page Application?
- What technologies does implementation of Single Page Application depends on?
- SPA can be implemented without AJAX. (True / False)
- What is Angular used for?
- What is the difference between Angular JS and Angular?
- What is the scripting language used to implement Angular?
- Which company manages the development of Angular?

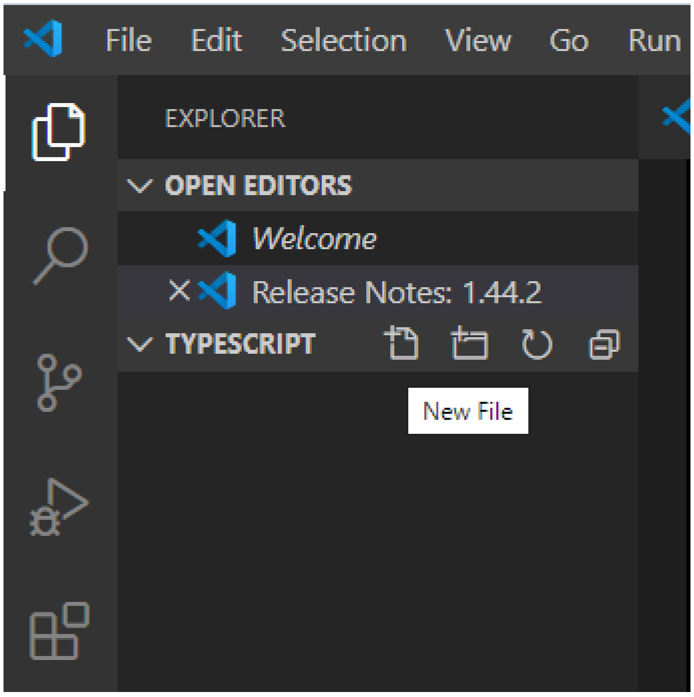
Display "Hello World" using TypeScript

Activity

Write a TypeScript program to display "Hello World" in an HTML page.

Solution

- Create a new folder named "typescript-learn" in D: to hold all the files related to this topic.
- Open Visual Studio Code
- Use "Open Folder" option and select the "typescript-learn" folder
- Create new file named "hello-world.ts" using the "New File" option.



- Include the following line in "hello-world.ts", which displays "Hello World" within the HTML document

```
document.body.textContent = "Hello World";
```

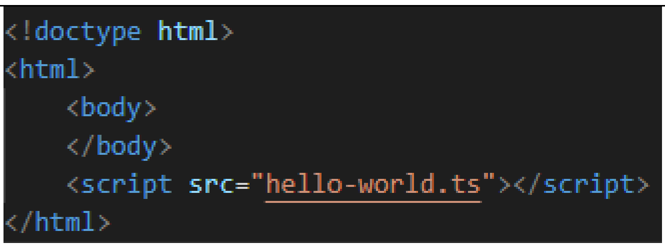
- Save hello-world.ts
- Open command prompt in D:\typescript-learn folder
- Install typescript by executing the following command:

```
npm install -g typescript
```

- Execute the following command to compile:

```
tsc hello-world.ts
```

- This will generate a new file called hello-world.js
- Open this file in Visual Studio Code and check the content, which will have same content as hello-world.ts.
- Create a new file hello.html, include doctype, body and script tag. In the script tag include hello-world.js.



```
<!doctype html>
<html>
    <body>
    </body>
    <script src="hello-world.ts"></script>
</html>
```

- ❖ Open hello-world.html in browser and check if "Hello World" is displayed.
- ❖ Open hello-world.ts in Visual Studio Code
- ❖ Modify the existing code minor error as specified below. textContent is misspelled here:

```
document.body.textConten = "Hello World";
```

- ❖ Now try compiling the program using tsc command. This should result in a compilation error.

About Visual Studio Code

- ❖ Visual Studio Code is a source code editor from Microsoft.
- ❖ In 2019 Developer Survey of Stack Overflow was ranked as popular development environment, with 50.7 respondents claiming to use it.
- ❖ This was first released in Nov 2015.
- ❖ This editor can be used for various programming languages like Java, JavaScript, Go, Node.js and C++.
- ❖ This will be editor we will be using to develop Angular applications.

About TypeScript

- ❖ Open source programming language.
- ❖ It is developed and maintained by Microsoft.
- ❖ TypeScript is a Typed superset of JavaScript
- ❖ TypeScript compiles to plain JavaScript
- ❖ JavaScript errors are not known during development.
- ❖ JavaScript is not strongly typed (a variable can be used to store any data type).
- ❖ With TypeScript strong type checking can be implemented.
- ❖ TypeScript files are stored with extension ".ts"
- ❖ For the above reasons TypeScript had been adopted as the language for Angular.
- ❖ All the source files in angular are written in TypeScript.
- ❖ All built-in library of Angular is implemented in TypeScript.

Check your understanding

- What is TypeScript?
- What is the output of TypeScript file compilation?
- What are the advantages of using TypeScript over JavaScript?
- What is the command to compile a TypeScript file?
- What is the extension of a TypeScript source file?

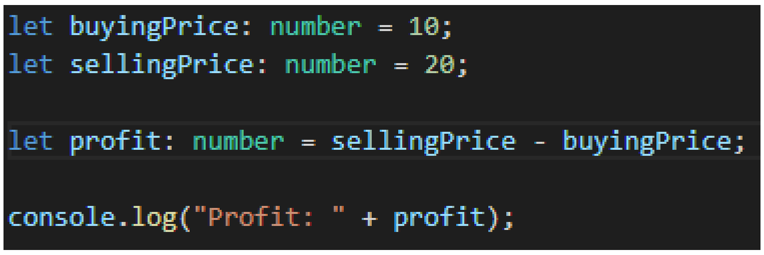
Calculate profit based on buying price and selling price

Problem

Calculate profit based on buying price and selling price.

Solution

- Create a new file profit.ts
- Include the code below in this file.



```
let buyingPrice: number = 10;
let sellingPrice: number = 20;

let profit: number = sellingPrice - buyingPrice;

console.log("Profit: " + profit);
```

- Open command line within Visual Studio Code following the steps below:

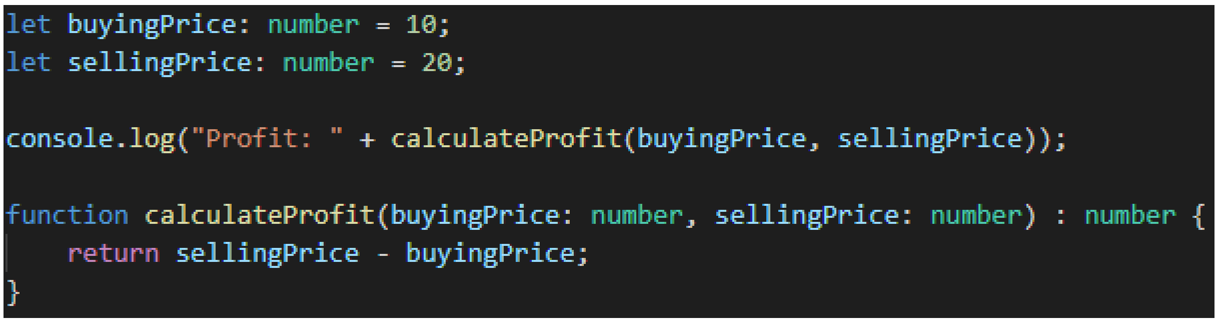
  Visual Studio Code > Menu > View > Terminal
- Execute the following command in the terminal to compile:

```
tsc profit.ts
```

- Execute the following command to execute the JavaScript:

```
node profit.js
```

- Implement the below code solve this problem using function:



```
let buyingPrice: number = 10;
let sellingPrice: number = 20;

console.log("Profit: " + calculateProfit(buyingPrice, sellingPrice));

function calculateProfit(buyingPrice: number, sellingPrice: number) : number {
    return sellingPrice - buyingPrice;
}
```

Explanation

- Define data type of a variable after colon.
- The data type number is used to represent numeric value.
- Following are the other data types:
  - Boolean

    ```
    let isDone: boolean = false;
    ```
  - String

    ```
    let color: string = "blue";
    ```
  - Array

    ```
    let list: number[] = [1, 2, 3];
    ```
  - Tuple – For having an array of different types:

```
// Declare a tuple type
let x: [string, number];
// Initialize it
x = ["hello", 10]; // OK
```

- Enum – User friendly names for a set of numeric values

```
enum Color {Red, Green, Blue}
let c: Color = Color.Green;
```

- Any – Can be used to store any data type.

```
let notSure: any = 4;
notSure = "maybe a string instead";
notSure = false; // okay, definitely a boolean
```

- Void – Used in a function, if it does not return any value
- Null and Undefined – Similar to how it is defined in JavaScript
- Object – Refers data types other than whatever is defined above.

❖ The operators and expressions remain the same syntax as JavaScript.

❖ Make note how function parameters are defined with data type.

❖ Also note how return type of a function is defined.

Check your understanding

❖ What is the syntax to define a variable in TypeScript?

❖ What are the various data types available in TypeScript?

❖ What is the purpose of Tuple?

❖ What is the purpose of Enum?

❖ What is the purpose of Any?

Strongly defining types for Objects using interface

Activity

We created JavaScript object Employee, Department, Skill objects when learning JavaScript. Implement this in TypeScript using interface.

Solution

- Create new file employee-test.ts
- Open employee.html file created during JavaScript learning.
- Copy and paste the employee variable definition code from employee.html to employee-test.ts:

```
var employee = {
    firstName: "John",
    lastName: "Smith",
    salary: 5000,
    permanentStaff: true,
    department: {
        id: 1,
        name: "HR"
    },
    skills:
    [
        new Skill(1, "HTML"),
        new Skill(2, "CSS"),
        new Skill(3, "JavaScript")
    ],
    fullName: function () {
        return this.firstName + " " + this.lastName;
    }
};
```

- Remove the fullName function.
- Modify skill data as specified below.

```
{ id: 1, name: "HTML" },
{ id: 2, name: "CSS" },
{ id: 3, name: "JavaScript" }
```

- Modify the employee variable definition line as specified below:

```
let employee : Employee = {
```

- Copy and paste the console.log() part from employee.html into employee-test.ts:

```
/*console.log("First Name: " + employee.firstName);
console.log("Last Name: " + employee.lastName);*/
console.log("Name: " + employee.fullName());
console.log("Salary: " + employee.salary);
console.log("Permanent Staff: " + employee.permanentStaff);
console.log("Department ID: " + employee.department.id);
console.log("Department Name: " + employee.department.name);
console.log("Skill 1: " + employee.skills[0].id + ", " + employee.skills[0].name);
console.log("Skill 2: " + employee.skills[1].id + ", " + employee.skills[1].name);
console.log("Skill 3: " + employee.skills[2].id + ", " + employee.skills[2].name);
```

- Uncomment firstName and lastName console logs.
- Remove the console log that uses fullName().
- Include below interface definitions after the console logs:

```
interface Employee {
    firstName: string,
    lastName: string,
    salary: number,
    permanentStaff: boolean,
    department: Department,
    skills: Skill[]
}

interface Department {
    id: number,
    name: string
}

interface Skill {
    id: number,
    name: string
}
```

- Compile and run the program
- Check if the employee details are printed accordingly.
- Notice that compilation works fine only when the properties defined in interface and actual object matches.

Solution (using different source files)

- Create new file employee-test.ts and include employee variable definition and console logs.
- Create new file department.ts and include the department interface definition in this.
- Create new file skill.ts and include the skill interface definition in this.
- Compile employee-test.ts
- It will fail with error that Employee is not defined.
- Export employee interface in employee.ts

```
export interface Employee {
```

- Import employee in employee-test.ts

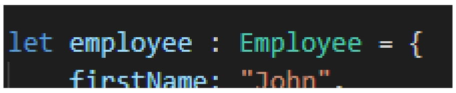
```
import { Employee } from "./employee";
```

- Now compilation of employee.ts fails that it cannot find Department and Skill.
- Apply appropriate export and import.
- Compile and run the program.

Explanation

- Interface helps in defining a type contract of an object.
- Property name and the data type are separated by colon when defining a property.

- Each property in interface is separated by comma.
- When defining the employee JavaScript object, we are associating the object with an interface.



```
let employee : Employee = {
    firstName: "John",
```

- This enforces the user to define appropriate property names and data type as per the interface.
- This creates a strong type checking between employee object and interface.
- A group of TypeScript source files can come under a module.
- The module definition happens when the keywords import and export is used.
- The export keyword makes the interface definition available to other TypeScript files.
- The import keyword helps creating a reference for other TypeScript source files, so that objects available in the source file can be used.
- Suffix the property of an interface with "?" to make it as an optional parameter.
- Try suffixing any parameter in one interface and check the compilation behavior.

Check your understanding

- What is the purpose of interface in TypeScript?
- What is the syntax for defining a property in interface?
- What is the purpose of export keyword?
- What is the purpose of import keyword?
- What is a module in TypeScript?

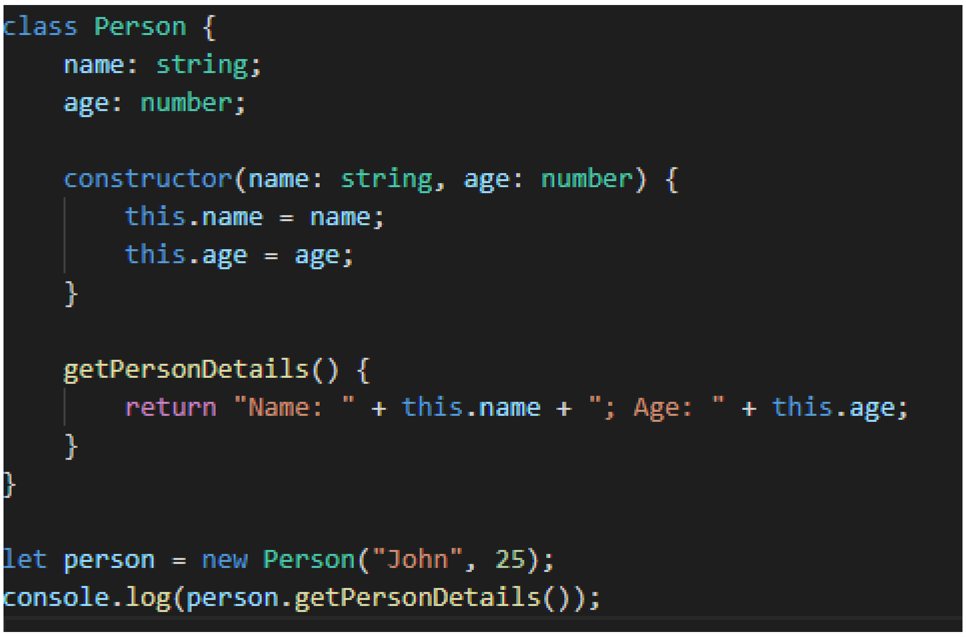
Display person details using Class

Activity

Display person details using class with properties name and age.

Solution

❖ Create new file person.ts and include the code below, then compile, run and check result:



```
class Person {
    name: string;
    age: number;

    constructor(name: string, age: number) {
        this.name = name;
        this.age = age;
    }

    getPersonDetails() {
        return "Name: " + this.name + "; Age: " + this.age;
    }
}

let person = new Person("John", 25);
console.log(person.getPersonDetails());
```

Explanation

❖ A class is used to define an object and its behavior.
❖ A class can contain a set of properties. In the above class name and age are properties.
❖ A class can be related to real world objects.
❖ The constructor helps in initializing the values of the properties.
❖ A new instance of Person class can be created using the new keyword.
❖ The expression 'new Person("John", 25)' calls the constructor.
❖ Once the instance is created, then the methods in the class can be called using the variable.
❖ The 'this' keyword represents the current object instance.
❖ When there is a need to use a property of within a class, it is mandatory to use the 'this' keyword.
❖ Similar to interface we can define properties of a class which are of another type of class.

Check your understanding

❖ What is a class?
❖ What is a constructor and what is its purpose?
❖ What is the syntax to define a constructor?
❖ How to create a new object of a class?

Display Hello World in Angular

Activity

Display “Hello World” in angular-learn application.

Solution

- Open “angular-learn” folder in Visual Studio Code.
- In the terminal window execute the following command to start the application:

  ```
  ng serve
  ```
- Then open http://localhost:4200 in the browser to launch the angular application.
- If the above command fails with following error:

  ```
  ng : File C:\Users\< username >\AppData\Roaming\npm\ng.ps1 cannot be loaded
  because running scripts is disabled on this system.
  ```

  Then execute the command in the terminal window to enable script execution in Power Shell.

  ```
  Set-ExecutionPolicy -ExecutionPolicy RemoteSigned -Scope CurrentUser
  ```

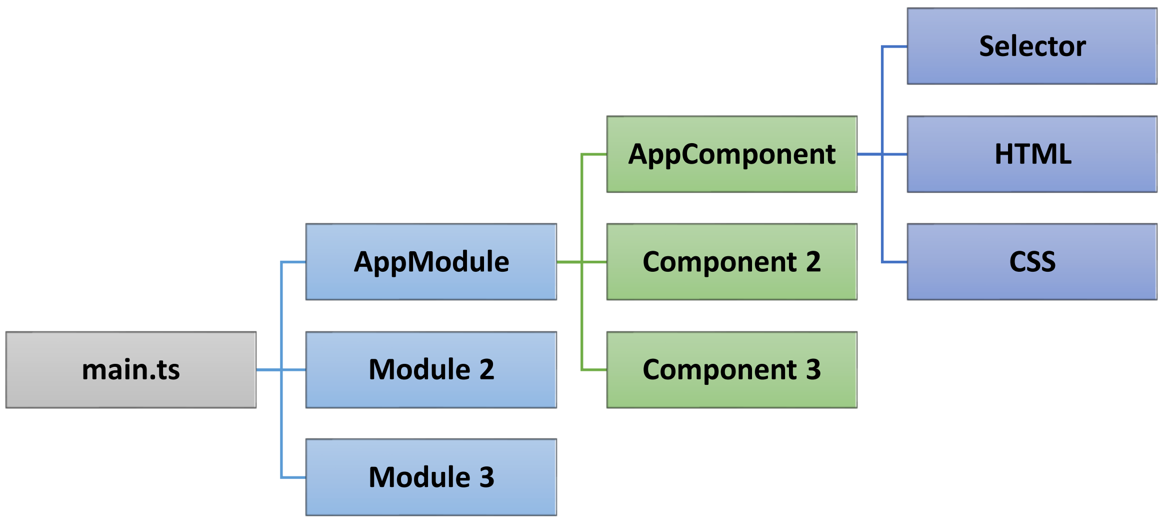
  Then try executing the ng serve command again.
- Open src/app/app.component.html
- Remove the entire code
- Include Hello World withing <h1> tag
- Save the file
- Check in browser if “Hello World” is displayed.

Explanation

- When the angular application starts, the following are the steps that will occur.
  - The entire code is compiled and relevant JavaScript files are generated.
  - First the compiled JavaScript file of src/main.ts will be executed.
  - This loads the AppModule, which is imported from src/app/app.module.ts
  - In app.module.ts,
    - AppModule class is defined
    - This class is defined with a decorator called @NgModule
    - The NgModule defines an object with four arrays declarations, imports, providers and bootstrap.
    - The bootstrap property loads the AppComponent, which is imported from src/app/app.component.ts
    - The Ng suffix represents Angular.
  - In app.component.ts
    - AppComponent class is defined
    - This class is defined with the decorator @Component
    - The @Component decorator defines three properties selector, templateUrl and styleUrls.
    - A component is a reusable part of angular application.

        - The 'selector' defines a name through which it can be referenced in the application web page.
        - Here the selector is defined as app-root
        - The 'templateUrl" property defines the HTML required for this component.
        - This refers to app.component.html, in which we have included the "Hello World"
        - Based on the selector this component gets loaded into src/index.html
        - styleUrls property helps define component specific styles.
    - In index.html, the <app-root></app-root> is defined within the body which includes the app-component based on the selector.
    - Based on this app.component.html is processed and included in page.
    - AppComponent is the parent most component in angular.
- An angular application can have one or more modules, by default only one module is created which is the AppModule.
- Each Module in Angular can contain one or more components. Components are the actual representation of view to the user of the application.
- Components depends on services for getting a functionality implemented. Service is represented as a class.
- If any other change is made in app.component.html and saved, angular compiles the code and automatically reloads the browser page automatically.

Putting things together



Check your understanding

- What is the first file that is executed as part of starting an angular application?
- What does main.ts do?
- What is the default angular module that is loaded when starting an angular application?
- What is the default component that is loaded when starting angular application?
- What is the purpose of the @NgModule decorator?
- What are the properties of @Component decorator?
- What is the purpose of 'selector' property of @Component decorator?

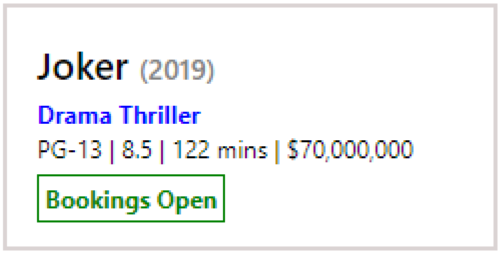
❖ What is the purpose of ‘templateUrl’ property of @Component decorator?

Display Movie details using Interpolation and Structural Directives

Problem

Display move details similar to the layout below. Use the below link to get the movie details:

Expected Layout



Joker (2019)

Drama Thriller

PG-13 | 8.5 | 122 mins | $70,000,000

Bookings Open

Solution (Step #1 – Creation of new component)

- ❖ Open command prompt or terminal window in "angular-learn" folder.
- ❖ Execute the following command to create a new component.

```
ng generate component movie
```

Explanation

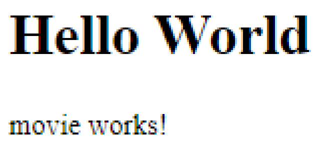
- ❖ The above command creates a new folder "movie" in the "app" folder.
- ❖ In this new folder, four new files are created:
    - o movie.component.css
    - o movie.component.html
    - o movie.component.spec.ts
    - o movie.component.ts
- ❖ Above files comprise the set of files for the new component.
- ❖ Open src/app/app.module.ts and notice the following changes:
    - o Import of MovieComponent
    - o Inclusion of MovieComponent in declarations section of NgModule decorator.

Solution (Step #2 – Include movie component in app component)

- ❖ Open app.component.html
- ❖ Include the line below after the <h1> tag.

```
<h1>Hello World</h1>
<app-movie></app-movie>
```

- ❖ Check if the output looks something like the below:

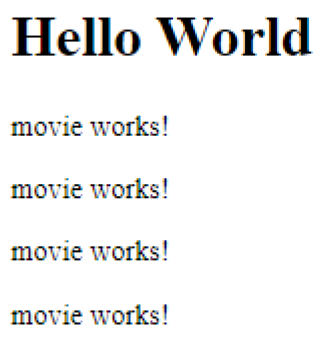


Hello World

movie works!

Explanation

- ❖ Open src/app/movie/movie.component.html
- ❖ Check if the content "movie works!" is present in this file. This is what is displayed in the place of <app-movie></app-movie> defined in app.component.html file.
- ❖ The tag <app-movie> links to the selector defined in movie.component.ts
- ❖ Now app component becomes the parent component and movie component becomes the child component.
- ❖ Copy and paste the <app-movie></app-movie> multiple times in app.component.html and check the result.



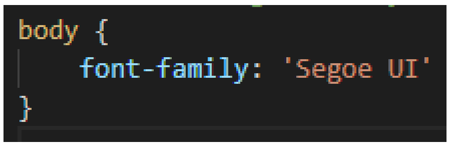
```
Hello World

movie works!

movie works!

movie works!

movie works!
```

- ❖ This demonstrates how quickly a component can be reused in multiple places.
- ❖ Remove the <h1> tag and multiple <app-movie> tag, finally have only on <app-movie> tag.

Solution (Step #3 – Display movie title)

src/styles.css

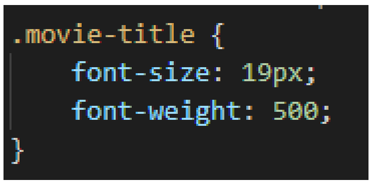
- ❖ Include the following style, so that font can be applied globally:



```
body {
    font-family: 'Segoe UI'
}
```

src/app/movie/movie.component.css

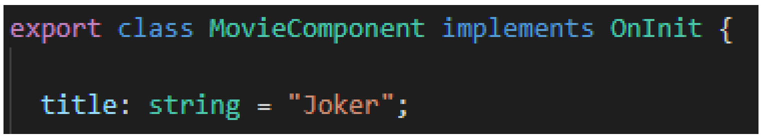
- ❖ Open styles.css implemented during bootstrap.
- ❖ Copy movie-title class from styles.css
- ❖ Paste the style definition in movie.component.css



```
.movie-title {
    font-size: 19px;
    font-weight: 500;
}
```

src/app/movie/movie.component.ts

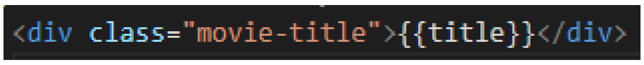
- ❖ Include a new property "title"



```
export class MovieComponent implements OnInit {

  title: string = "Joker";
```

src/app/movie/movie.component.html

- ❖ Modify the content to display the title.



```
<div class="movie-title">{{title}}</div>
```

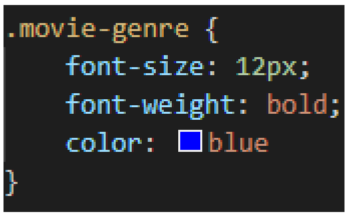
Explanation

- The styles.css file in src folder is the global css file.
- The file movie.component.css is the style for movie component.
- The property "title" is defined for movie component as string type and the value is assigned as "Joker"
- This data is read in movie.component.html using **interpolation** which is nothing but enclosure of the variable with double curly braces.

Solution (Step #4 – Display Genre)

src/app/movie/movie.component.css

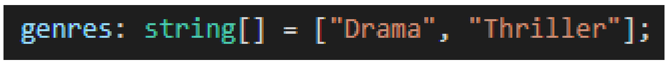
- Include movie-genre class



```css
.movie-genre {
    font-size: 12px;
    font-weight: bold;
    color: blue
}
```

src/app/movie/movie.component.ts

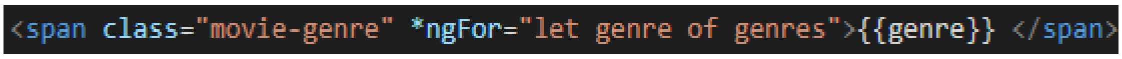
- Include a new property "genres"



```ts
genres: string[] = ["Drama", "Thriller"];
```

src/app/movie/movie.component.html

- Modify the content to display the title.



```html
<span class="movie-genre" *ngFor="let genre of genres">{{genre}} </span>
```

Explanation

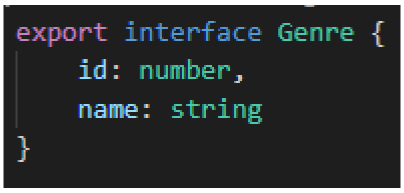
- The genres property is defined as a string array in movie.component.ts
- In movie.component.html, the genres are displayed using ngFor directive.
- Structural Directives
  - Structural Directives is a concept within angular which reshapes the DOM's structure by adding, removing or changing HTML elements.
  - *ngFor is one of the Structural Directive available in angular.
  - It defines a variable 'genre' for array 'genres'
  - In this example, the <span> tag is repeated as many times based on the size of the array provide.
  - The 'genre' variable available in ngFor is used as an interpolation to read each item from the array.

Solution (Step #4 – Organize movie data into appropriate model classes)

src/app/movie/model/movie.ts

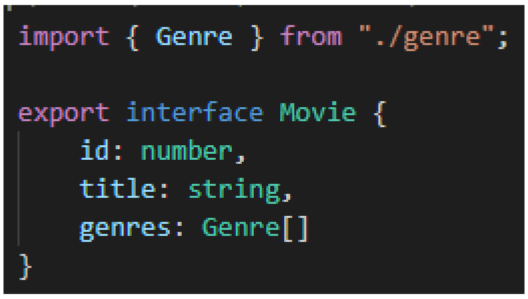
- Create a new folder "model" in "movie" folder.

- Click in the model folder in the left-hand side file navigator of Visual Studio code.
- Click “New File”
- Name the file as “genre.ts” and define the interface:



```
export interface Genre {
    id: number,
    name: string
}
```

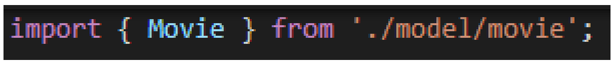
- Create new file as “movie.ts” and define the interface:



```
import { Genre } from "./genre";

export interface Movie {
    id: number,
    title: string,
    genres: Genre[]
}
```

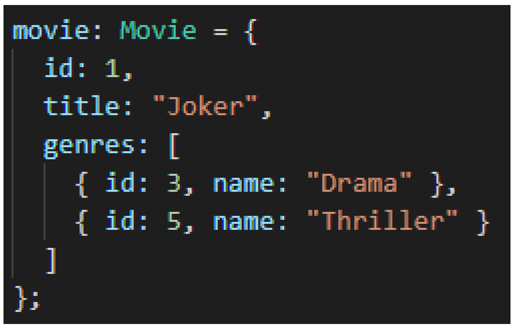
src/app/movie/movie.component.ts

- Import movie



```
import { Movie } from './model/movie';
```

- Include movie property

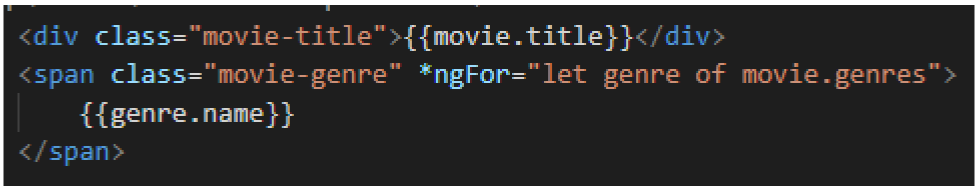


```
movie: Movie = {
  id: 1,
  title: "Joker",
  genres: [
    { id: 3, name: "Drama" },
    { id: 5, name: "Thriller" }
  ]
};
```

- Remove title and genres

src/app/movie/movie.component.html

- Modify to refer movie



```
<div class="movie-title">{{movie.title}}</div>
<span class="movie-genre" *ngFor="let genre of movie.genres">
    {{genre.name}}
</span>
```

Solution (Step #4 – Display Release Date)

src/app/movie/model/movie.ts

- Include release date.

```
export interface Movie {
    id: number,
    title: string,
    genres: Genre[],
    releaseDate: Date
}
```

src/app/movie/movie.component.ts

- ❖ Include release date for movie property

```
releaseDate: new Date(2019, 7, 31)
```

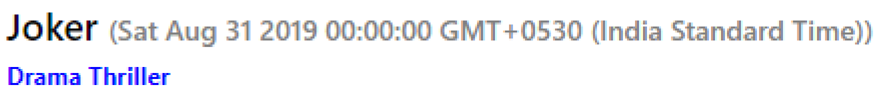
NOTE: In JavaScript Date object, month is defined as index number starting from 0 and ending with 11, hence August is represented as 7.

src/app/movie/movie.component.html

- ❖ Include release date

```
<div class="movie-title">
    {{movie.title}}
    <span class="movie-year">({{movie.releaseDate}})</span>
</div>
```

- ❖ The result should look like below:



Joker (Sat Aug 31 2019 00:00:00 GMT+0530 (India Standard Time))
Drama Thriller

- ❖ Modify the release date to get converted into release year

```
>({{movie.releaseDate | date: 'yyyy'}})<
```

Pipes

- ❖ Pipes take data as input and transforms to a desired output
- ❖ Following are the built-in pipes available in angular:
  - Date Pipe
  - Upper Case Pipe
  - Lower Case Pipe
  - Currency Pipe
  - Percent Pipe
- ❖ Necessary formatting can be done based on format provided after colon.

Solution (Step #4 – Display MPAA Rating, Rating, Duration and Budget)

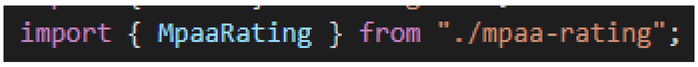
src/app/movie/model/mpaa-rating.ts

- ❖ Create this new file with the content as specified below:

```
export interface MpaaRating {
    id: number,
    name: string
}
```

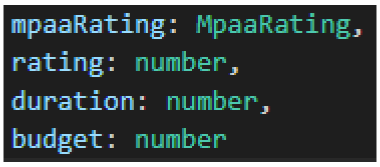
src/app/movie/model/movie.ts

- Import MpaaRating



```
import { MpaaRating } from "./mpaa-rating";
```

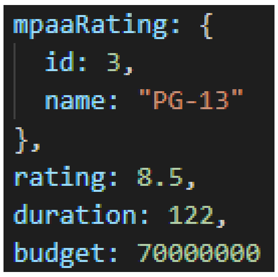
- Modify to include MPAA rating, Rating, Duration and Budget



```
mpaaRating: MpaaRating,
rating: number,
duration: number,
budget: number
```

src/app/movie/movie.component.ts

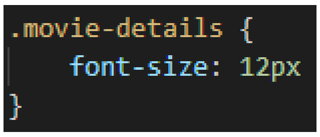
- Include the new properties



```
mpaaRating: {
  id: 3,
  name: "PG-13"
},
rating: 8.5,
duration: 122,
budget: 70000000
```

src/app/movie/movie.component.css

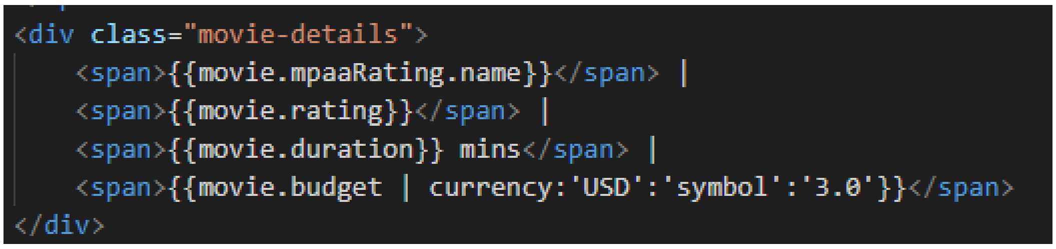
- Include style



```
.movie-details {
    font-size: 12px
}
```

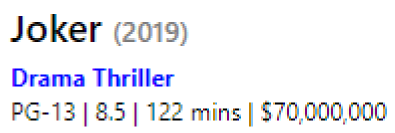
src/app/movie/movie.component.html

- Include the respective property values. Currency pipe is used here to define the currency and remove the decimal places.



```
<div class="movie-details">
    <span>{{movie.mpaaRating.name}}</span> |
    <span>{{movie.rating}}</span> |
    <span>{{movie.duration}} mins</span> |
    <span>{{movie.budget | currency:'USD':'symbol':'3.0'}}</span>
</div>
```

- The result should look like below:

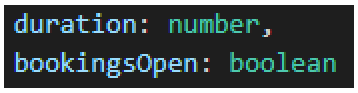


Joker (2019)
Drama Thriller
PG-13 | 8.5 | 122 mins | $70,000,000

Solution (Step #5 – Display booking status)

src/app/movie/model/movie.ts

- Include new property for booking status:



```
duration: number,
bookingsOpen: boolean
```

src/app/movie/movie.component.ts

- Include the new property

```
duration: 122,
bookingsOpen: true
```

src/app/movie/movie.component.css

- ❖ Copy and paste the styles below:

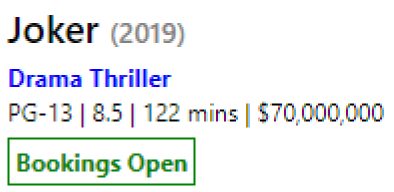
```css
.booking-open {
    font-size: 12px;
    border: 1px solid green;
    color: green;
    padding: 3px;
    font-weight: bold
}

.booking-closed {
    font-size: 12px;
    border: 1px solid red;
    color: red;
    padding: 3px;
    font-weight: bold
}
```

src/app/movie/movie.component.html

- ❖ Display booking status as specified below:

```html
<span *ngIf="movie.bookingsOpen" class="booking-open">
   Bookings Open
</span>
<span *ngIf="!movie.bookingsOpen" class="booking-closed">
   Bookings Closed
</span>
```

- ❖ The result should look like below:



Joker (2019)
Drama Thriller
PG-13 | 8.5 | 122 mins | $70,000,000
Bookings Open

Include overall border

- ❖ Enclose the entire content in movie.component.html within a div with class as "movie-card"
- ❖ Copy and paste the styles below in movie.component.css:

```css
.movie-card {
    padding: 15px;
    border: 2px solid rgb(218, 211, 211);
    width: 25%;
    float: left;
    margin: 15px
}
```

- ❖ Save and check if the expected result is achieved.

Explanation for ngIf

'ngIf' is a structural directive. If the boolean expression is true then the HTML element will be displayed else it is not displayed.

Summary

- **Interpolation**
  - Using interpolation include computed text between HTML tags.
  - Double curly braces are the delimiter for interpolation
  - The component property is included within the double curly braces.
  - Examples:
    - `<h3>Current customer: {{ currentCustomer }}</h3>`
    - `<div><img src="{{itemImageUrl}}"></div>`
    - `<p>The sum of 1 + 1 is {{1 + 1}}.</p>`
    - `<p>The sum of 1 + 1 is not {{1 + 1 + getVal()}}.</p>`
- **Pipes**
  - A pipe takes in data as input and transforms it to desired output.
  - Following are the built-in pipes available in angular:
    - Date Pipe
    - Upper Case Pipe
    - Lower Case Pipe
    - Currency Pipe
    - Percent Pipe
  - Pipes can be combined to have useful combinations. Refer example below. This is called as Pipe Chaining.

    ```
    {{ birthday | date | uppercase}}
    ```
- **Structural Directives**
  - Structural Directives is a concept within angular which reshapes the DOM's structure by adding, removing or changing HTML elements.
  - *ngIf
    - ngIf helps to decide whether an HTML element needs to be shown or not.
    - The value part of ngIf should contain a Boolean expression.
    - If the Boolean expression is true, then it shows the element.
    - If the Boolean expression is false, it does not show the element.
  - *ngFor
    - ngFor helps in repeating a HTML element based on the list of items passed to it as input.
    - The value of ngFor contains the definition of the array to use and the variable definition for each array item.

Check your understanding

- What is interpolation?
- What is the syntax to use interpolation?
- What does the content withing the curly braces of interpolation refers to?
- Is it possible to have expressions within interpolation? (True / False)
- What are pipes?
- What is the syntax to use pipes?
- What are the various built-in pipes available in angular?
- What is pipe chaining?
- What is a Structural Directive?
- What is the purpose of *ngIf directive?
- What is the syntax of *ngIf?
- What is the purpose of *ngFor directive?
- What is the syntax of *ngFor?

Best Bus Case Study implementation

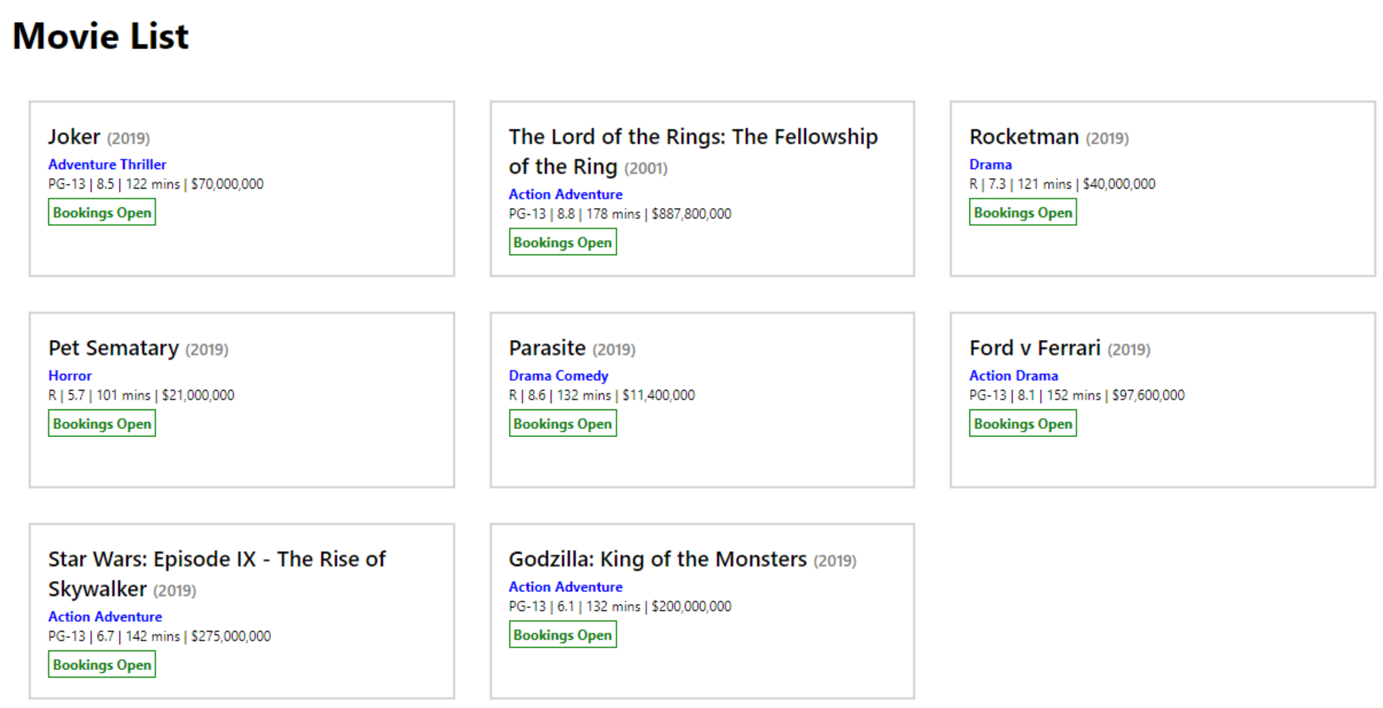
- Implement the Best Bus Case Study to display one Bus Schedule
- High level steps to get this implemented:
  - Create a new angular application named “best-bus” in a different folder.
  - Identify the components to be created
  - Identity the interfaces to be created
  - Implement the component to display bus schedule using interpolation

Display Movie List with inter component communication

Problem

Based on the layout that displayed a single movie detail. Display the list of movies.

Expected Layout



Movie List

Joker (2019)
Adventure Thriller
PG-13 | 8.5 | 122 mins | $70,000,000
Bookings Open

The Lord of the Rings: The Fellowship of the Ring (2001)
Action Adventure
PG-13 | 8.8 | 178 mins | $887,800,000
Bookings Open

Rocketman (2019)
Drama
R | 7.3 | 121 mins | $40,000,000
Bookings Open

Pet Sematary (2019)
Horror
R | 5.7 | 101 mins | $21,000,000
Bookings Open

Parasite (2019)
Drama Comedy
R | 8.6 | 132 mins | $11,400,000
Bookings Open

Ford v Ferrari (2019)
Action Drama
PG-13 | 8.1 | 152 mins | $97,600,000
Bookings Open

Star Wars: Episode IX - The Rise of Skywalker (2019)
Action Adventure
PG-13 | 6.7 | 142 mins | $275,000,000
Bookings Open

Godzilla: King of the Monsters (2019)
Action Adventure
PG-13 | 6.1 | 132 mins | $200,000,000
Bookings Open

Solution (Using movie component)

src/app/data.ts

- Create a new file data.ts in src/app folder.
- Open the URL https://raw.githubusercontent.com/jjchandru/angular-learn/master/src/app/data.ts in browser.
- Copy the code in the browser and paste it in data.ts file.

src/app/movie/model/movie.component.ts

- Import the data file

```
import { movies } from '../data';
```

- Remove the existing movie variable definition code and include the below code in it's place.

```
movie: Movie = movies[0];
```

- This should display the Joker movie details in the browser.
- Modify the movie array index value to 3. This should display the movie details of Rocketman.
- Create a new movies array variable, referring the data from data.ts.

```
movies: Movie[] = movies;
```

src/app/movie/movie.component.html

- Modify the enclosing div with ngFor to display all movies.

```
<div class="movie-card" *ngFor="let movie of movies">
```

Solution (Using different components for listing and movie detail)

Create new movie-list component

- ❖ Execute the command below in console to generate a new component.

```
ng generate component movie-list
```

src/app/app.component.html

- ❖ Modify to include the new movie list component instead of movie component.

```
<app-movie-list></app-movie-list>
```

src/app/movie-list/movie-list.component.ts

- ❖ Include below imports:

```
import { Movie } from '../movie/model/movie';
import { movies } from '../data';
```

- ❖ Include movies property and the movie list data from data.ts.

```
export class MovieListComponent implements OnInit {

  movies: Movie[] = movies;

  constructor() { }

  ngOnInit(): void {
  }

}
```

src/app/movie-list/movie-list.component.html

- ❖ Modify the content as specified below:

```
<h1>Movie List</h1>
<app-movie [movie]="movie" *ngFor="let movie of movies"></app-movie>
```

src/app/movie/movie.component.ts

- ❖ Modify the property movie with @Input and comment out the previous definitions:

```
//movie: Movie = movies[0];
//movies: Movie[] = movies;
@Input() movie: Movie;
```

src/app/movie/movie.component.html

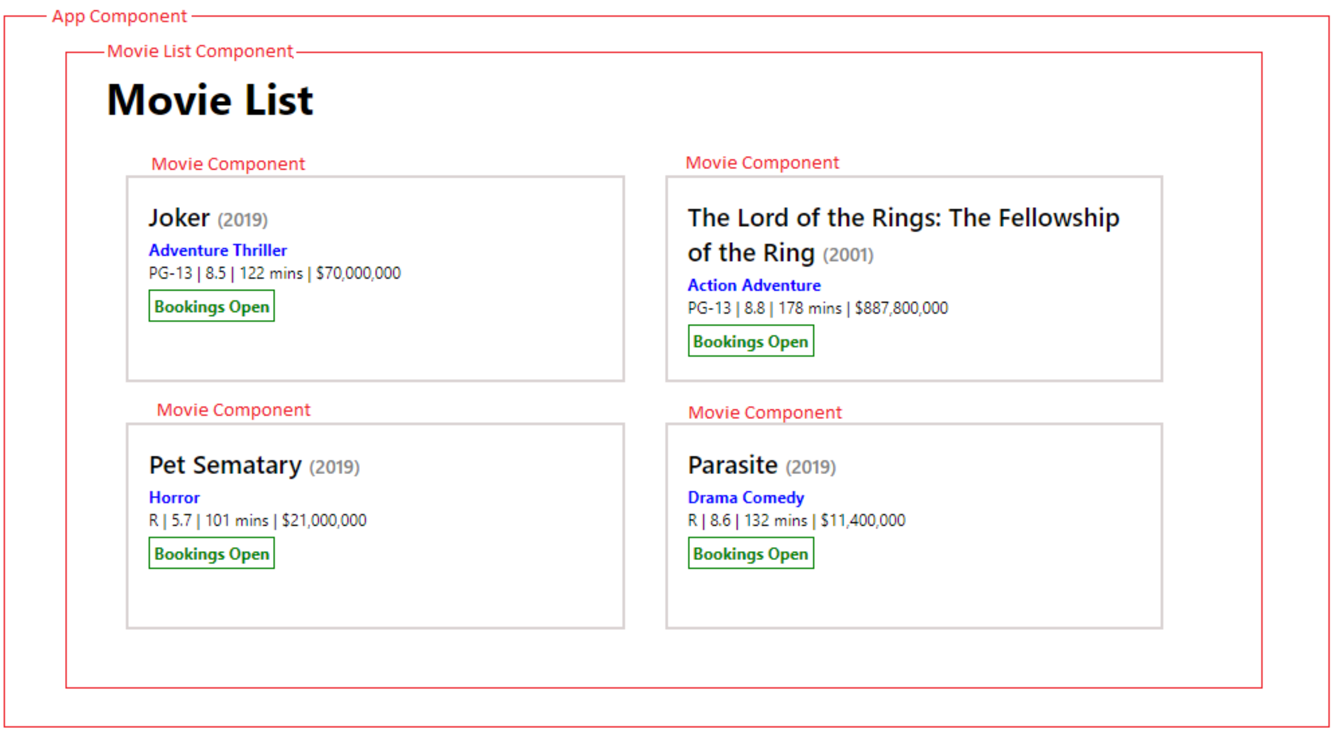
- ❖ Remove the *ngFor definition in movie-card div.

```
<div class="movie-card">
```

- ❖ Save all files and check if the expected result is achieved.

Explanation

❖ Find below the structure of components applied in our current implementation.



Movie List

Joker (2019)
Adventure Thriller
PG-13 | 8.5 | 122 mins | $70,000,000
Bookings Open

The Lord of the Rings: The Fellowship of the Ring (2001)
Action Adventure
PG-13 | 8.8 | 178 mins | $887,800,000
Bookings Open

Pet Sematary (2019)
Horror
R | 5.7 | 101 mins | $21,000,000
Bookings Open

Parasite (2019)
Drama Comedy
R | 8.6 | 132 mins | $11,400,000
Bookings Open

❖ The parent most component is app component.

❖ The movie list component is placed under app component using <app-movie-list> selector in app.component.html.

❖ The movie component is placed under movie-list component using <app-movie> selector in movie-list.component.html.

❖ This structure makes it easier to use movie component in any other component.

❖ In data.ts the movie list is defined for various movies as JavaScript object aligned with the respective interface.

❖ In movie list component the movie list from data.ts is referenced as a component property.

❖ Refer the code below that includes the movie component:

```
<app-movie [movie]="movie" *ngFor="let movie of movies"></app-movie>
```

❖ The *ngFor makes the <app-movie> selector displayed multiple times based on the number of items in the movies array.

❖ In the attribute definition [movie]="movie", the value "movie" refers to the movie defined in *ngFor.

❖ This way of binding an html element attribute with component property is called Property Binding.

❖ To understand Property Binding, let us include the following line of code within the movie-card div in movie.component.html.

```
<div class="movie-card">
    <input type="text" name="title" value="{{movie.title}}">
```

❖ This will display a text box with movie title. Here we are using interpolation.

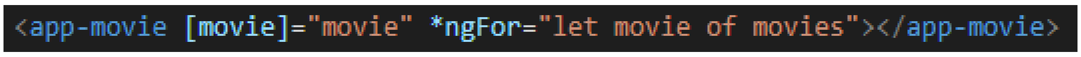
❖ The same result can be achieved by value definition as specified below.

```
[value]="movie.title">
```

- This approach of using square brackets around the property of html element is called Property Binding.
- The square brackets defined in HTML Template of angular represents the data communication from component to the view.
- Definition of square brackets withing TypeScript or JavaScript code represents an array.
- HTML Template is the name used in angular terminology to represent component.html. This is also called as view.
- In the case of movie-list.component.html the property binding is applied on a component selector, when means the <app-movie> tag that points to movie component.
- Hence the data gets transferred to the movie.component.ts. The movie within square brackets represents the 'movie' property defined in the movie.component.ts.
- Remove the input text box from movie.component.html.

Summary for Component Interaction

- So far, we have created two components.
- Each component can have its own set of properties, with its own scope.
- Properties in one component cannot be directly accessed in another component.
- In this example, we passed movie variable from movie-list component to movie component using Parameter Binding.



```
<app-movie [movie]="movie" *ngFor="let movie of movies"></app-movie>
```

- This example also conveys how to communicate from Parent Component to Child Component.
- The Parent Component sends the data using Parameter Binding.
- The Child Component receives the data using @Input decorator.
- Based on the passed movie data from movie-list component, each movie card is displayed in movie component.

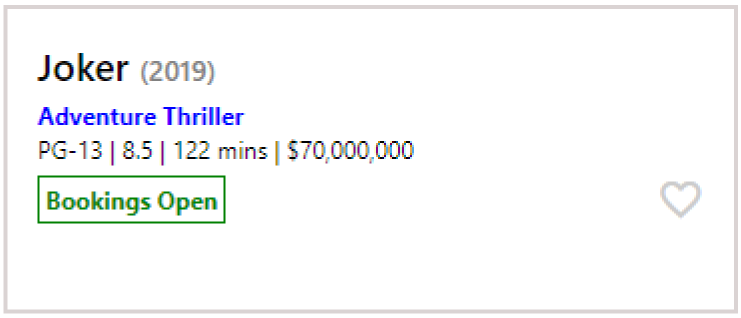
Include a Favorite icon for each Movie to demonstrate @Output decorator

Problem

Include a favorite icon in each movie, so that the user can select favorite movie.

Expected Output

Before favorite selection



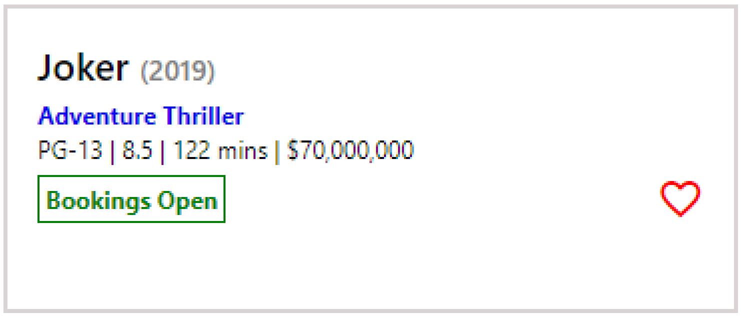
Joker (2019)

Adventure Thriller

PG-13 | 8.5 | 122 mins | $70,000,000

Bookings Open

On favorite icon hover



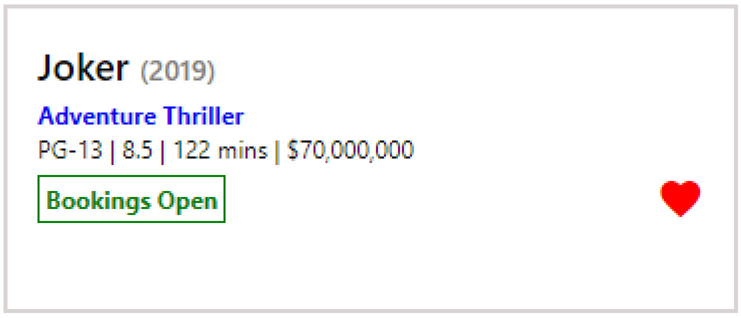
Joker (2019)

Adventure Thriller

PG-13 | 8.5 | 122 mins | $70,000,000

Bookings Open

After favorite selection



Joker (2019)

Adventure Thriller

PG-13 | 8.5 | 122 mins | $70,000,000

Bookings Open

On hovering over the icon, the mouse icon should get changed to hand.

Solution

src/index.html

- Copy and paste the google material icon into the <head> tag.

```
  <link href="https://fonts.googleapis.com/icon?family=Material+Icons"
      rel="stylesheet">
```

src/app/movie/movie.component.css

- Copy and paste styles for favorite.

```
.favorite {
    color: #CCCCCC;
    cursor: pointer;
    float:right
}

.favorite:hover {
    color: red;
    cursor: pointer;
}

.favorite-selected {
    color: red;
    float: right;
    cursor: pointer;
}
```

src/app/movie/model/movie.ts

- Include new property named favorite of type boolean.

src/app/data.ts

- Include favorite attribute in each movie object with value as false.

src/app/movie/movie.component.ts

- Import Output and EventEmitter.

```
import { Component, OnInit, Input, Output, EventEmitter } from '@angular/core';
```

- Include @Output event after @Input

```
@Output() favoriteSelected: EventEmitter<boolean> = new EventEmitter();
```

- Include the favorite click handler after ngOnInit() method.

```
ngOnInit(): void {
}

selectFavorite() {
  this.favoriteSelected.emit(true);
  this.movie.favorite = !this.movie.favorite;
}
```

src/app/movie/movie.component.html

- Display favorite icon.

```html
<span *ngIf="!movie.favorite" class="favorite" (click)="selectFavorite()">
    <i class="material-icons">favorite_border</i>
</span>
<span *ngIf="movie.favorite" class="favorite-selected" (click)="selectFavorite()">
    <i class="material-icons">favorite</i>
</span>
```

src/app/movie/movie-list.component.html

❖ Modify to receive the event from movie component.

```html
<app-movie
    [movie]="movie"
    (favoriteSelected)="toggleFavorite()"
    *ngFor="let movie of movies">
</app-movie>
```

src/app/movie/movie-list.component.ts

❖ Include toggleFavorite() method.

```typescript
toggleFavorite() {
  console.log("toggleFavorite() called");
}
```

❖ Save all files and test if it works as expected.

❖ Check the console logs whether the toggleFavorite() method in movie-list component is called.

Explanation

❖ The @Output decorator is used to send messages to the parent component using the built-in EventEmitter class.

❖ The click event on the favorite icon is captured using the event handling mechanism of (click). The method defined in (click) needs to be implemented in the component, which will be invoked when the respective HTML element is clicked.

```html
(click)="selectFavorite()">
```

❖ The event to parent component is initiated using the emit method.

```typescript
this.favoriteSelected.emit(true);
```

❖ The event goes to the event specified in movie-list.component.html, the toggleFavorite() function is called. The favoriteSelected specified here refers to the property defined in movie.component.ts.

```html
<app-movie
    [movie]="movie"
    (favoriteSelected)="toggleFavorite()"
    *ngFor="let movie of movies">
</app-movie>
```

❖ Square brackets in HTML template refers to message from component to view (HTML).

❖ Normal brackets in HTML template refers to message from view to component.

Check your understanding

- What options are available to implement interaction between parent and child components?
- List the steps to implement sending data from parent component to child component.
- List the steps to implement sending event from child component to parent component.
- How to handle events triggered from HTML to component?

Implement movie search using Event Handling

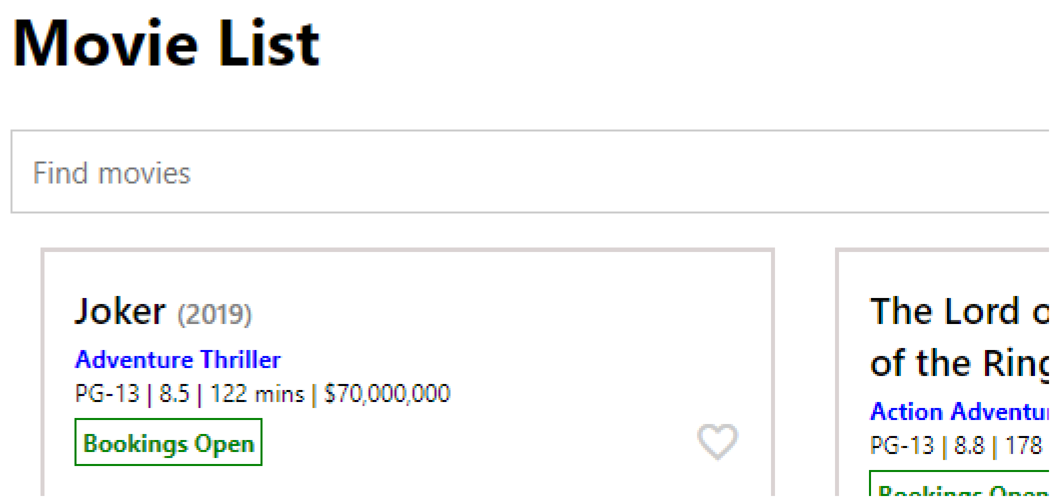
Problem

Implement a search text box to find movies. Following are the expected features:

- Include a text box for user to enter keyword to search.
- The text box should have a placeholder within the text box that displays "Find movies". Refer screen layout below.
- There is no search button, but the search should happen when user types the search keyword.

Expected Output

With search text box



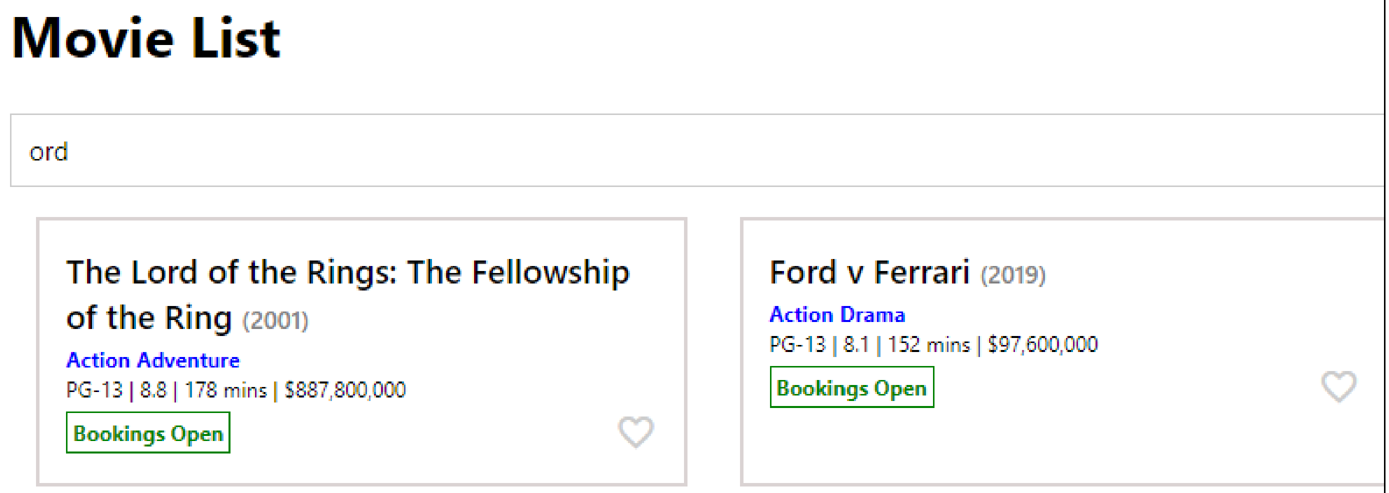
Movie List

Find movies

Joker (2019)
Adventure Thriller
PG-13 | 8.5 | 122 mins | $70,000,000
Bookings Open

The Lord o
of the Ring
Action Adventu
PG-13 | 8.8 | 178

Results after searching



Movie List

ord

The Lord of the Rings: The Fellowship of the Ring (2001)
Action Adventure
PG-13 | 8.8 | 178 mins | $887,800,000
Bookings Open

Ford v Ferrari (2019)
Action Drama
PG-13 | 8.1 | 152 mins | $97,600,000
Bookings Open

Solution

src/styles.css

- Include style for text box. Copy and paste the style below.

```css
input[type=text] {
    border: 1px solid #CCCCCC;
    padding: 10px;
    width: 95%;
    margin-top: 2px;
    margin-bottom: 2px;
    font-family: 'Segoe UI';
    font-size: 15px
}
```

src/app/app.module.ts

- Import FormsModule.

```typescript
import { FormsModule } from '@angular/forms';
```

- Include forms module in imports:

```typescript
imports: [
  BrowserModule,
  AppRoutingModule,
  FormsModule
],
```

src/app/movie-list/movie-list.component.ts

- Include new properties:

```typescript
movies: Movie[] = movies;
searchKey: string = "";
```

src/app/movie-list/movie-list.component.html

- Include input text box with code as specified below after the Movie List h1 tag.

```html
<input [(ngModel)]="searchKey" type="text" placeholder="Find movies">
<div>{{searchKey}}</div>
```

- Save all files.
- In browser, enter a search keyword in the input text box.
- Whatever is entered in the text box should get displayed in the bottom of the text box as we type. This is to just check if the search text box and component is implemented correctly.
- Once this works, remove the searchKey interpolation, but include (keyup) attribute.

```html
<input [(ngModel)]="searchKey" type="text"
    placeholder="Find movies" (keyup)="findMovie()">
```

src/app/movie-list/movie-list.component.ts

- Include new function with console log test if this function is called, when text is typed in the text box.

```
findMovie() {
  console.log("Search: " + this.searchKey);
}
```

- Save all files and check if this function is called in the console log of the browser.
- Modify the function with the coding as specified below:

```
findMovie() {
  if (this.searchKey == "") {
    this.movies = movies;
    return;
  }
  this.movies = [];
  for (let i = 0; i < movies.length; i++) {
    let title = movies[i].title.toUpperCase();
    let searchKeyUpper = this.searchKey.toUpperCase();
    if (title.includes(searchKeyUpper)) {
      this.movies.push(movies[i]);
    }
  }
}
```

- Save all files and check the result.

Explanation

Forms Module

- This is first time we are introducing a form field, which is the search text box.
- For implementing form related aspects, angular has a built-in module and it needs to be initiated.
- Hence we are initializing FormsModule in app.module.ts.
- The ngModel included in movie-list.component.html is dependent on FormsModule.

Two-way binding

- The [(ngModel)] in the code below ensures the following aspects:

```
<input [(ngModel)]="searchKey" type="text"
    placeholder="Find movies" (keyup)="findMovie()">
```

  - When loading the display of the component, the value for the text box comes from searchKey component property.
  - If we assign some default value in the component for searchKey then the value is pre-populated. Please give it a try and revert back the code.
  - When we type any value in the text box, the [(ngModel)] definition passes the data from the text box to the component.

    ```
    searchKey: string = "";
    ```

  - As seen earlier, square brackets represent sending data from component to view.
  - Normal brackets send data from view to component.

- In the case of [(ngModel)], since both square brackets and normal brackets are defined it takes care of both sending and receiving data.
- This is called two-way binding.

Event Handling

- The (keyup) refers to the event when user presses a key in keyboard and the time when the key goes back up after pressing.
- As always data flow from component to view is represented in normal brackets.
- This event is bound to the function findMovie() in the component.
- Hence every time the user presses a key in the text box the findMove() function is called.

How filtering works?

- The objective of findMovie() function is to modify the "movies" component property with values based on the search keyword.
- If the searchKey is empty then, it means that the user has cleared whatever he has typed.
- In this scenario we need to display all the movies. That is what the below code does:

```
if (this.searchKey == "") {
  this.movies = movies;
  return;
}
```

- In this this scenario, there is not need to do further filtering, hence a return is provided.
- As we have to freshly populate the movies array based on the new search keyword, we are first clearing the existing entries in the array.

```
this.movies = [];
```

- The for loop iterates through the movie list in data.ts.
- For each movie, it has to check if the search keyword is part of the movie title. If so, then that item is added into the this.movies array, which is what is used in the component.

```
for (let i = 0; i < movies.length; i++) {
```

```
if (title.includes(searchKeyUpper)) {
  this.movies.push(movies[i]);
}
```

- The includes() method makes a case sensitive checks.
- So, if the user enters in upper case, then we may never find a match.
- Hence both the search keyword and movie title is converted to upper case before comparison.

```
let title = movies[i].title.toUpperCase();
let searchKeyUpper = this.searchKey.toUpperCase();
```

Questions

- List the steps involved to implement form handling using ngModel.
- What is two-way binding? How can it be achieved?
- How is event handling achieved in angular? List out the steps.

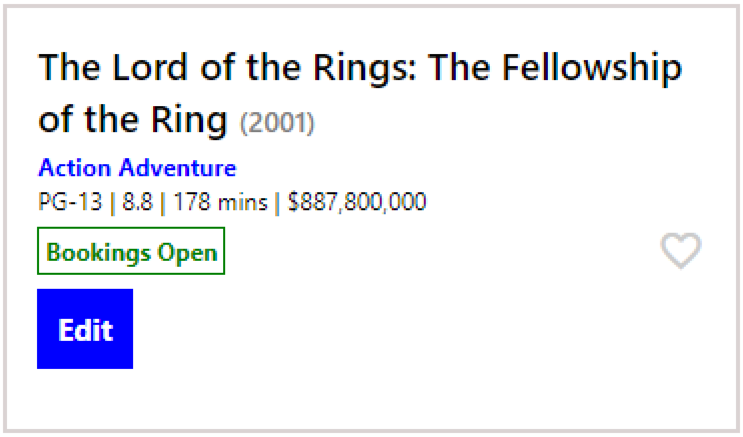
Implement Edit Movie using Routing and Template Driven Form

Problem

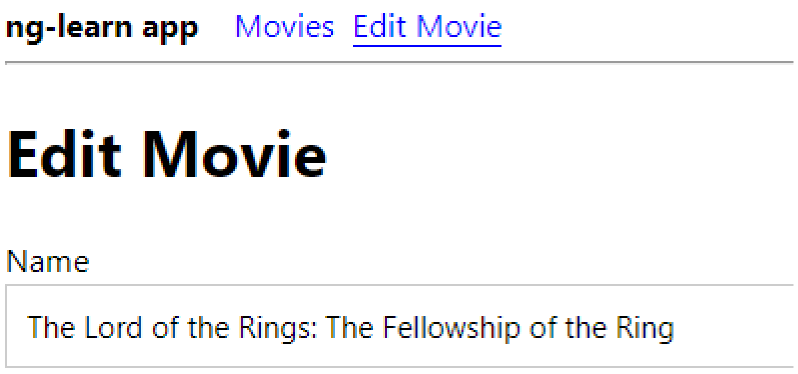
Create a form to edit movie title.

Expected Output

Movie Component


The Lord of the Rings: The Fellowship of the Ring (2001)
Action Adventure
PG-13 | 8.8 | 178 mins | $887,800,000
Bookings Open
Edit

Edit Movie Component



Edit Movie

Name

The Lord of the Rings: The Fellowship of the Ring

Solution (Step #1 – Create routes for displaying edit form)

Create edit movie component

- Open new terminal
- Run the below command to create a new component:

```
ng generate component edit-movie
```

src/app/app-routing.module.ts

- Include imports

```
import { MovieListComponent } from './movie-list/movie-list.component';
import { EditMovieComponent } from './edit-movie/edit-movie.component';
```

- Modify the routes

```
const routes: Routes = [
  { path: 'movie-list', component: MovieListComponent },
  { path: 'edit-movie', component: EditMovieComponent }
];
```

src/app/app.component.css

- Include the following styles.

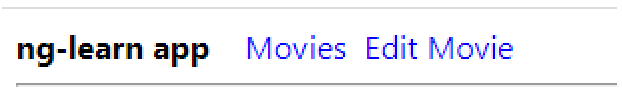
```
a:visited {
    color: blue
}

a {
    text-decoration: none;
}

.active-link {
    border-bottom: solid 1px blue
}
```

src/app/app.component.html

- ❖ Include the following code:

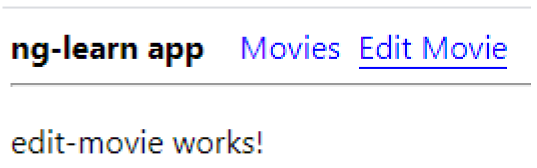
```
<strong>ng-learn app</strong>&nbsp;&nbsp;&nbsp;
<a routerLink="movie-list" routerLinkActive="active-link">Movies</a>&nbsp;
<a routerLink="edit-movie" routerLinkActive="active-link">Edit Movie</a>&nbsp;
<hr>
<router-outlet></router-outlet>
```

- ❖ Save all files
- ❖ The output should look like this:



```
ng-learn app    Movies  Edit Movie
```

- ❖ Clicking on Movies should display the movie listing component.
- ❖ Clicking on Edit Movie should display the following:



```
ng-learn app    Movies  Edit Movie

edit-movie works!
```

Explanation

- ❖ In app-routing.module.ts we defined two paths one for movie list screen and one of edit movie screen.
- ❖ In the app.component.css, we defined the following:
  - The anchor tag text decoration style ensures that the anchor tag is not underlined.
  - The a:visited ensures that there is no color change for a visited link.
  - The .active-link defines the style when a specific link or route is currently active. Once can notice that the underline is displayed when a link is clicked. This underline is displayed because of the style definition.
- ❖ In app.component.html:
  - The routerLink attribute value refers to the path defined in app-routing.module.ts
  - The routerLinkActive attribute value represents the style class. If that particular router link is active then the respective style class will be applied.
  - The <router-outlet></router-outlet> will be replaced with selector of the respective component mapped in app-routing.module.ts. For example, the "Movies" link on the web

page points to the URL /movie-list, which maps to MovieListComponent defined in "app-routing.module.ts", hence this component selector will be replaced for <router-outlet>.

Solution (Step #2 – Create router link to another component)

styles.css

- Copy and paste the styles below for buttons.

```
input[type=button] {
    padding: 10px;
    background-color: blue;
    border: 0px solid blue;
    color: white;
    margin-top: 10px;
    margin-bottom: 10px;
    font-family: 'Segoe UI';
    font-size: 15px;
    cursor: pointer;
    font-weight: bold
}

input[type=button]:hover {
    padding: 10px;
    background-color: white;
    border: 1px solid blue;
    color: blue;
    margin-top: 10px;
    margin-bottom: 10px;
    font-family: 'Segoe UI';
    font-size: 15px;
    cursor: pointer;
    font-weight: bold
}
```

- Save the file.

src/app/app-routing.module.ts

- Modify the edit movie route with parameter.

```
{ path: 'edit-movie/:id', component: EditMovieComponent }
```

src/app/movie/movie.component.html

- Include button after the favorite icon definition.

```
<div>
    <input type="button" value="Edit" (click)="showEdit()">
</div>
```

- Save and check if button is displayed. The color should get changed on hover.

src/app/movie/movie.component.ts

- Import router.

```
import { Router } from '@angular/router';
```

- Inject router through the constructor.

```
constructor(private router: Router) { }
```

- Include showEdit() method

```
showEdit() {
  console.log("Inside showEdit()");
  this.router.navigate(['/edit-movie', this.movie.id]);
}
```

- Save file.
- In movie list page click on Edit button. The following should happen:
  - The URL contains the route and the respective Movie Id.
  - The Edit Movie Component is displayed.
  - In the link "Edit Movie" link is selected.

Explanation

- The edit movie component path is modified to include passing a parameter value to the component.

```
{ path: 'edit-movie/:id', component: EditMovieComponent }
```

- The Router imported in movie.component.ts contains the routes configured in app-routing.module.ts

```
import { Router } from '@angular/router';
```

- The following statement injects the Router object instance into this component. This also makes router as one of the properties of movie component.

```
constructor(private router: Router) { }
```

- Since 'router' becomes a property it is accessed in showEdit() method as this.router.
- The router.navigate() method programmatically initiates routing to a router path. It also sends the parameter value to the router URL.

```
this.router.navigate(['/edit-movie', this.movie.id]);
```

Solution (Step #3 – Read the router parameter in edit movie component)

src/app/movie/edit-movie.component.ts

- Import ActivatedRoute

```
import { ActivatedRoute } from '@angular/router';
```

- Inject ActivatedRoute in constructor

```
constructor(private route: ActivatedRoute) { }
```

- Modify ngOnInit() method.

```
ngOnInit(): void {
  let id: string = this.route.snapshot.paramMap.get('id');
  console.log("Movie ID: " + id);
}
```

- Save the file and check if click on Edit button in movie list displays the respective movie id in the console.

Explanation

- ❖ The ActivatedRoute contains the current activated route which point to EditMovieComponent.
- ❖ Using the ActivatedRoute the parameter value can be obtained using the parameter map available in the route.

```
this.route.snapshot.paramMap.get('id')
```

Solution (Step #4 – Get the movie based on id and display)

src/app/movie/edit-movie.component.ts

- ❖ Include movie property

```
movie: Movie;
```

- ❖ Modify ngOnInit() method.

```
ngOnInit(): void {
  let id: string = this.route.snapshot.paramMap.get('id');
  for (let i: number = 0; i < movies.length; i++) {
    if (movies[i].id == parseInt(id)) {
      this.movie = movies[i];
    }
  }
}
```

- ❖ The built-in JavaScript method parseInt() method is used to convert the string to number.

src/app/movie/edit-movie.component.html

- ❖ Modify to include the code below:

```
<h1>Edit Movie</h1>
<form>
    Name<br>
    <input type="text" [(ngModel)]="movie.title" name="title">
</form>
```

- ❖ Save all files and check if the selected movie title is displayed.
- ❖ The ngModel helps applying the two-way binding between the component and view.

Include validations for movie title

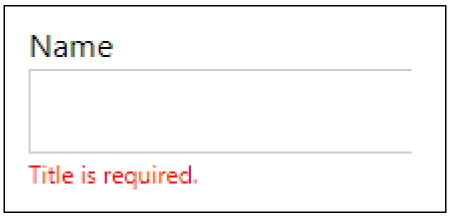
Problem

For the movie title text box, include validations to check the following conditions:

- Title fields is mandatory.
- Title should have at least 2 characters.
- Title should not exceed 30 characters.

Expected Output

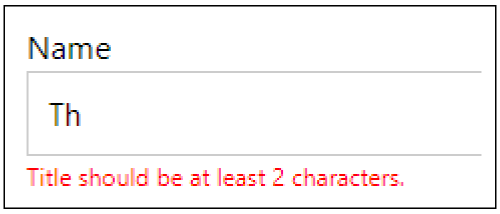
When title is not entered



Name

Title is required.

When only one character is entered.

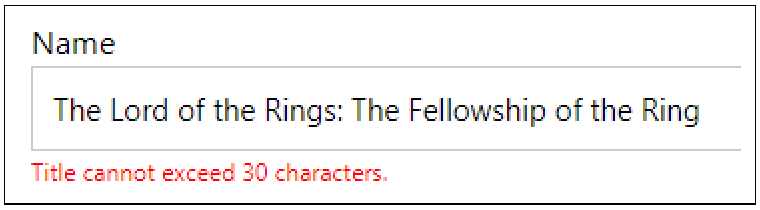


Name

Th

Title should be at least 2 characters.

When number of characters exceed 30



Name

The Lord of the Rings: The Fellowship of the Ring

Title cannot exceed 30 characters.

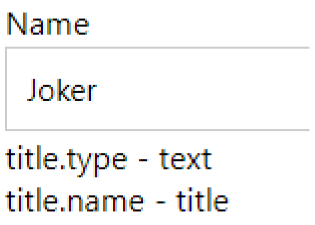
Solution (Step #1 – Understanding Template Reference Variable)

src/app/movie/edit-movie.component.html

- Include the following interpolations below the title text box.

```
<input type="text" [(ngModel)]="movie.title" name="title" #title>
<div>title.type - {{title.type}}</div>
<div>title.name - {{title.name}}</div>
```

- Save the file.
- The result should look like below:



Name

Joker

title.type - text
title.name - title

Explanation

- ❖ The #title defined in the text box is called as Template Reference Variable.
- ❖ This variable is not way related to the properties in component.
- ❖ This Template Reference Variable refers to the title text box HTML Element, which can be found from the type and name property values displayed using interpolation.

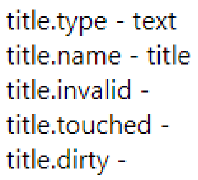
Solution (Step #2 – Error property values from ngModel)

src/app/movie/edit-movie.component.html

- ❖ Include below specified code after the display of type and name.

```
<div>title.invalid - {{title.invalid}}</div>
<div>title.touched - {{title.touched}}</div>
<div>title.dirty - {{title.dirty}}</div>
<div>title.errors.required - {{title.errors?.required}}</div>
<div>title.errors.minlength - {{title.errors?.minlength | json}}</div>
<div>title.errors.maxlength - {{title.errors?.maxlength | json}}</div>
```

- ❖ Save the file and check the result.

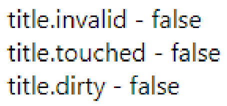


```
title.type - text
title.name - title
title.invalid -
title.touched -
title.dirty -
```

- ❖ Modify the Template Reference Variable as specified below:

```
#title="ngModel"
```

- ❖ Remove the type and name display.



```
title.invalid - false
title.touched - false
title.dirty - false
```

- ❖ Click inside the title text box. Without making any changes, click outside the text box. The title.touched value changes to true.
- ❖ In the text box remove one character. This changes values of title.dirty to true.

Explanation

- ❖ The definition of "ngModel" to Template Reference Variable #title attaches the error validation related properties to #title. After this change, title.type does not contain any value.
- ❖ "title.dirty" value is "true" when any changes are made in the title text box.
- ❖ "title.touched" value "true" denotes user had tried to use the text box, but no changes made.

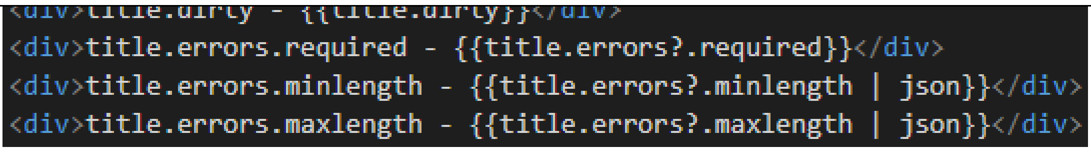
Solution (Step #3 – Display error properties from ngModel)

src/app/movie/edit-movie.component.html

- ❖ Modify the text box as specified below.

```
<input type="text" [(ngModel)]="movie.title" name="title" #title="ngModel"
    minlength="2" maxlength="30" required>
```

- ❖ Include the following code to display error related fields after the display of dirty.



```
<div>title.dirty - {{title.dirty}}</div>
<div>title.errors.required - {{title.errors?.required}}</div>
<div>title.errors.minlength - {{title.errors?.minlength | json}}</div>
<div>title.errors.maxlength - {{title.errors?.maxlength | json}}</div>
```

- Save the file and check the following in the result:
  - Click “Movies” and then click “The Lord of the Rings: The Fellowship of the Ring”

```
title.invalid - true
title.touched - true
title.dirty - false
title.errors.required -
title.errors.minlength -
title.errors.maxlength - { "requiredLength": 30, "actualLength": 49 }
```

  - Have only one character in the title text box.

```
title.invalid - true
title.touched - true
title.dirty - true
title.errors.required -
title.errors.minlength - { "requiredLength": 2, "actualLength": 1 }
title.errors.maxlength -
```

  - Clear all the values in the text box.

```
title.invalid - true
title.touched - true
title.dirty - true
title.errors.required - true
title.errors.minlength -
title.errors.maxlength -
```

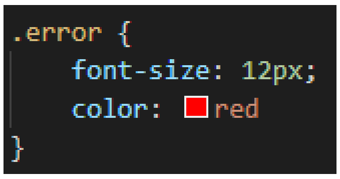
Explanation

- When user modifies the content of title text box, error object is created based on the attributes minlength, maxlength and required defined in the <input> element.
- The attribute minlength="2", creates error object if there is only one character entered in the text box. The title.invalid property is set of true.
- The attribute maxlength="30", creates error object if the number of characters exceed 30 and sets title.invalid to true.
- The attribute required, creates error object if the text box is empty.
- The attributes minlength, maxlength and required are properties of HTML.
- Angular uses these attributes as validation rules and creates error object accordingly.

Solution (Step #4 – Display error messages)

src/styles.css

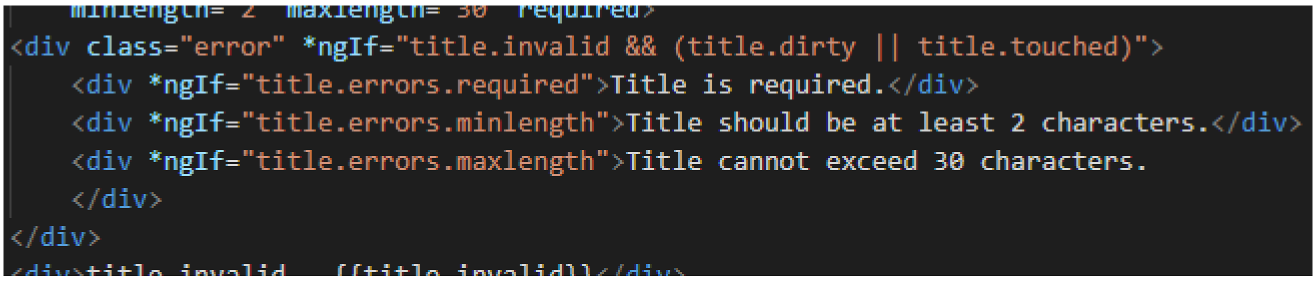
- Include below style to display validation error messages.



```
.error {
    font-size: 12px;
    color: red
}
```

src/app/movie/edit-movie.component.html

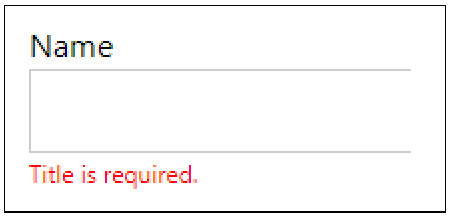
- ❖ Include below code after the title text box.



```
    minlength= 2  maxlength= 30  required>
<div class="error" *ngIf="title.invalid && (title.dirty || title.touched)">
    <div *ngIf="title.errors.required">Title is required.</div>
    <div *ngIf="title.errors.minlength">Title should be at least 2 characters.</div>
    <div *ngIf="title.errors.maxlength">Title cannot exceed 30 characters.
    </div>
</div>
<div>title.invalid - {{title.invalid}}</div>
```

- ❖ Save all files and check the result. This should match with the expected result with all validations.

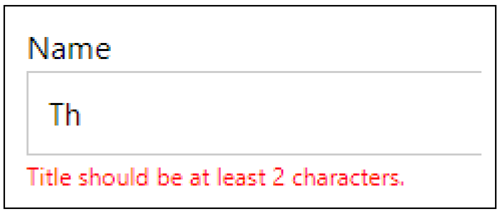
  When title is not entered



Name

Title is required.

When only one character is entered.

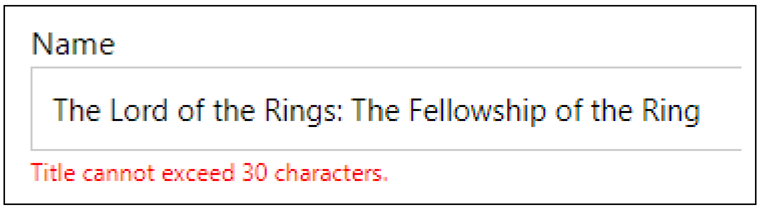


Name

Th

Title should be at least 2 characters.

When number of characters exceed 30



Name

The Lord of the Rings: The Fellowship of the Ring

Title cannot exceed 30 characters.

Explanation

- ❖ In this final step we are using the error properties in the title and structural directives to display appropriate messages.

Template Driven Form

- ❖ Angular provides two ways to manage form development:
  - Template Driven Form
  - Reactive Form
- ❖ The term 'Template' in angular always refer to the HTML part of the component.
- ❖ The coding we implemented for having a text box for title is implemented using Template Driven Form.
- ❖ The necessary coding required to implement form and validations is completely done in a Template (HTML), hence this type of methodology is called as Template Driven Form.
- ❖ Two-way data binding is used here to map the data in the component with template.

Check your understanding

- What is a Template Driven Form?
- What is ngModel and how it works?
- What is Template Reference Variable?
- What is the syntax to define a Template Reference Variable?
- List the properties from which property we can know the specific details about a validation error?

Include Save button and display data model

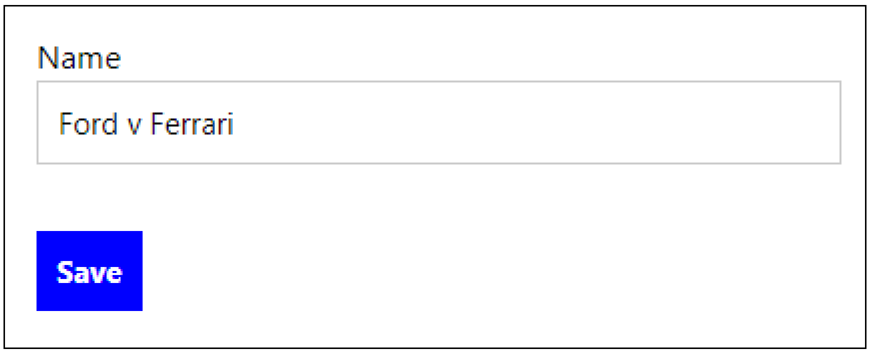
Problem

Implement the following in Edit Movie screen:

- Include Save button. The Save button should be disabled when there are validation errors.
- Display model data as JSON.

Expected Output

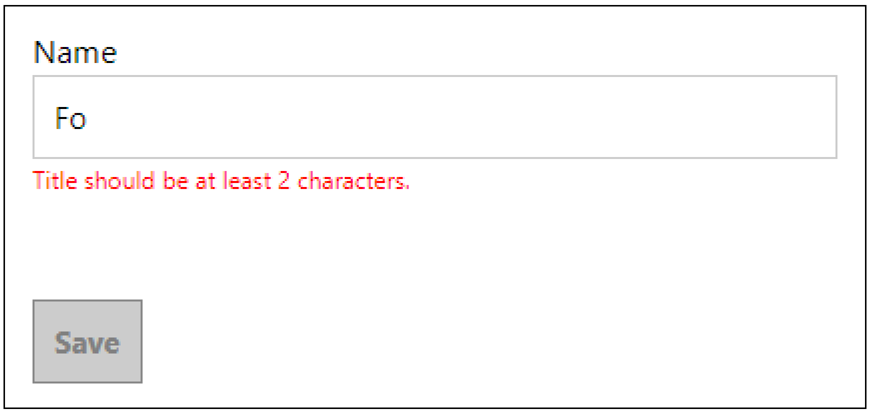
Save button display



Name

Ford v Ferrari

Save button disabled when there are validation errors

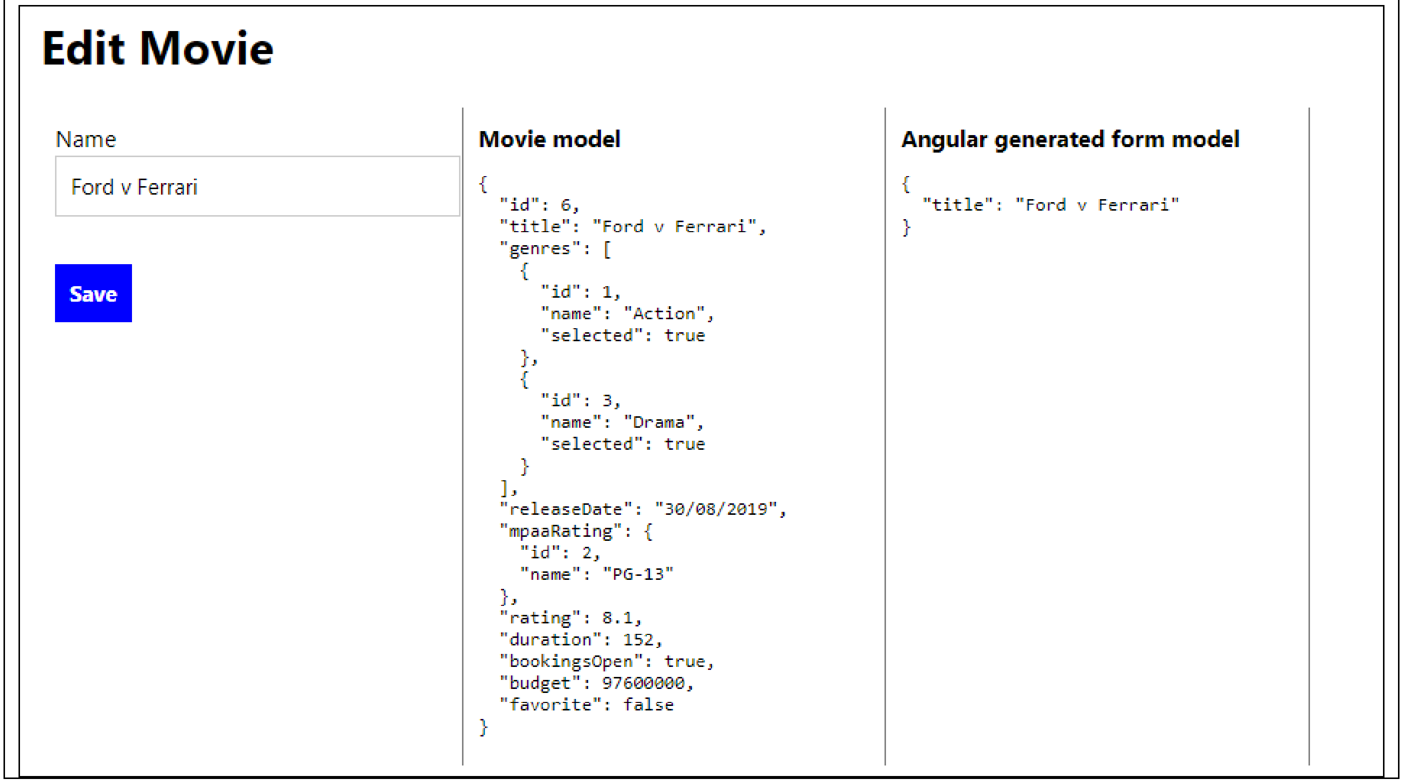


Name

Fo

Title should be at least 2 characters.

Data model display



Edit Movie

Name

Ford v Ferrari

Movie model

```
{
  "id": 6,
  "title": "Ford v Ferrari",
  "genres": [
    {
      "id": 1,
      "name": "Action",
      "selected": true
    },
    {
      "id": 3,
      "name": "Drama",
      "selected": true
    }
  ],
  "releaseDate": "30/08/2019",
  "mpaaRating": {
    "id": 2,
    "name": "PG-13"
  },
  "rating": 8.1,
  "duration": 152,
  "bookingsOpen": true,
  "budget": 97600000,
  "favorite": false
}
```

Angular generated form model

```
{
  "title": "Ford v Ferrari"
}
```

Solution (Step 1 – Include Save button)

src/style.css

- Apply button style for submit as well.

```
input[type=button], input[type=submit] {
```

```
input[type=button]:hover, input[type=submit]:hover {
```

- Copy and paste the above hover and change hover to disabled, the change the color:

```
input[type=button]:disabled, input[type=submit]:disabled {
    padding: 10px;
    background-color: #CCCCCC;
    border: 1px solid #888888;
    color: grey;
    margin-top: 10px;
    margin-bottom: 10px;
    font-family: 'Segoe UI';
    font-size: 15px;
    cursor: pointer;
    font-weight: bold
}
```

src/app/movie/edit-movie.component.html

- Modify the form as specified below:

```
<form #movieForm="ngForm" (ngSubmit)="onSubmit()">
```

- Include submit button.

```
<input type="submit" [disabled]="!movieForm.valid" value="Save">
```

src/app/movie/edit-movie.component.ts

- Include code below to handle the submit event:

```
onSubmit() {
  console.log(JSON.stringify(this.movie));
}
```

- Save all files.
- Verify the following:
  - Check if Save button is displayed, the button turns grey and disabled for validation errors.

Solution (Step 2 – Display model data)

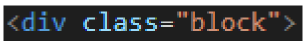
src/app/movie/edit-movie.component.css

- Copy and paste the below style.

```
.block {
    float: left;
    width: 30%;
    height: 600px;
    padding: 10px;
    border-right: 1px solid grey
}
```

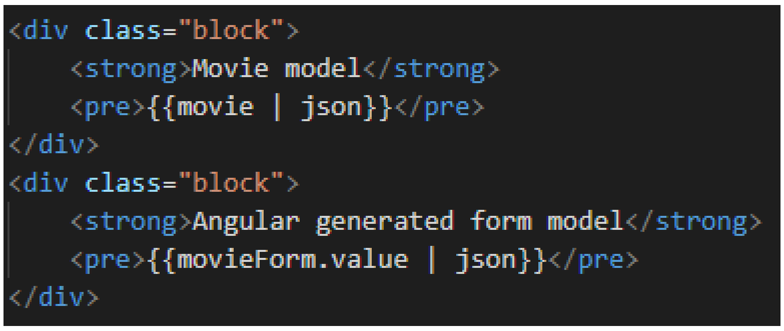
src/app/movie/edit-movie.component.html

- ❖ Enclose the entire <form> within a div and define the style.



```html
<div class="block">
```

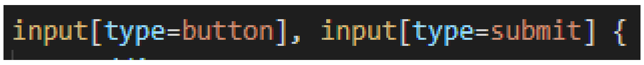
- ❖ Include the below divs after the block div.



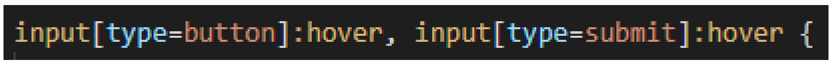
```html
<div class="block">
    <strong>Movie model</strong>
    <pre>{{movie | json}}</pre>
</div>
<div class="block">
    <strong>Angular generated form model</strong>
    <pre>{{movieForm.value | json}}</pre>
</div>
```

src/style.css

- ❖ Apply button style for submit as well.

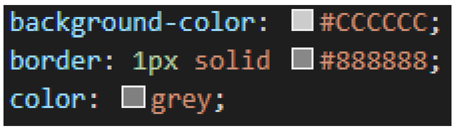


```css
input[type=button], input[type=submit] {
```



```css
input[type=button]:hover, input[type=submit]:hover {
```

- ❖ Copy and paste the above hover and change hover to disabled, the change the color:



```css
background-color: #CCCCCC;
border: 1px solid #888888;
color: grey;
```

- ❖ Save all files and verify the following:
  - The display should get divided into 3 sections.
  - The first section containing the form.
  - The second section containing the movie model data as JSON
  - The third section containing the form data as JSON

Include display of MPAA Rating drop down

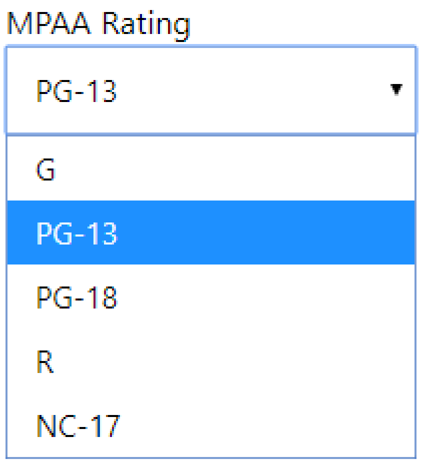
Problem

Implement the following in Edit Movie screen:

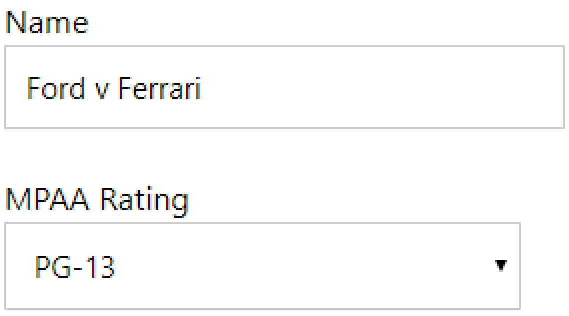
- ❖ Include MPAA Rating drop down having the list of ratings.
- ❖ The drop down should get select based on the respective rating of the movie.
- ❖ Include Save button.
- ❖ The Save button should be disabled when there is

Expected Output

MPAA Rating drop down



Drop down selected based on the rating



Solution (Step 1 – Populate the MPAA Rating list)

src/app/movie/edit-movie.component.ts

- ❖ Import MpaaRatings array from data.ts.

```
import { MpaaRating } from '../movie/model/mpaa-rating';
import { mpaaRatings } from '../data';
```

- ❖ Include property to hold the list of MPAA Ratings.

```
mpaaRatings: MpaaRating[] = mpaaRatings;
```

src/app/movie/edit-movie.component.html

- ❖ Remove all error property display divs.
- ❖ Include code below after the error div of title.

```
<br><br>
MPAA Rating<br>
<select name="mpaaRating">
    <option *ngFor="let mpaaRating of mpaaRatings">{{mpaaRating.name}}</option>
</select>
```

- ❖ This should display the drop down with list of MPAA Ratings.
- ❖ This drop down lacks the following set of features:
  - o The respective MPAA Rating is not pre-selected
  - o Two-way binding is not done, so changes will not reflect to the component.

Solution (Step 2 – Make respective rating pre-selected for MPAA Rating select box)

src/app/movie/edit-movie.component.ts

- ❖ Include property to hold the list of MPAA Ratings.

```
mpaaRatings: MpaaRating[] = mpaaRatings;
```

- ❖ Include property to hold the selected MPAA Rating.

```
selectedMpaaRating: MpaaRating;
```

- ❖ Modify onSubmit() to set the select MPAA Rating to the movie component property.

```
onSubmit() {
    this.movie.mpaaRating = this.selectedMpaaRating;
    console.log(JSON.stringify(this.movie));
}
```

src/app/movie/edit-movie.component.html

- ❖ Modify <select> with ngModel and include [ngValue] attribute for option tag.

```
<select name="mpaaRating" [(ngModel)]="selectedMpaaRating">
    <option [ngValue]="mpaaRating" *ngFor="let mpaaRating of mpaaRatings">
        {{mpaaRating.name}}
    </option>
</select>
```

src/style.css

- ❖ Apply input[type=text] style to select as well.

```
input[type=text], select {
```

- ❖ Save all files and check the result.

Explanation

- ❖ The following change binds selectedMpaaRating property with the select box.

```
<select name="mpaaRating" [(ngModel)]="selectedMpaaRating">
```

- ❖ Following is the expected structure of <option> tag:

```
<option value="1" selected>PG-13</option>
```

- ❖ The [ngValue] attribute for option tag is created based on the below definition. ngValue helps to bind the selected item with an object instead of a string:

```
<option [ngValue]="mpaaRating" *ngFor="let mpaaRating of mpaaRatings">
    {{mpaaRating.name}}
</option>
```

- The square brackets represent data from component to view. The MPAA Rating object from mpaaRatings property is assigned here.
- The change in onSubmit() ensures that we are getting the selected MPAA Rating into the movie property of the component.

Include rest of all form fields for Editing Movie

Problem

In Edit Movie page include the form fields for Rating, Duration, Budget, Release Date, Bookings Open and Favorite.

Solution

src/app/movie/model/movie.ts

- ❖ Modify the data type for release date as "string"

src/app/data.ts

- ❖ Modify releaseDate as dd/mm/yyyy format for all movies.

src/app/movie/model/genre.ts

- ❖ Include new property 'selected' to manage the checkboxes and make it optional with question mark.

src/app/movie/edit-movie.component.ts

- ❖ Include a new property for displaying the list of genres.

```
genres: Genre[] = genres;
```

- ❖ Include the second for loop in from the code below into ngOnInit() method as specified below:

```
ngOnInit(): void {
  let id: string = this.route.snapshot.paramMap.get('id');
  for (let i: number = 0; i < movies.length; i++) {
    if (movies[i].id == parseInt(id)) {
      this.movie = movies[i];
    }
  }

  // iterate all genres from data.ts
  for (let i = 0; i < genres.length; i++) {
    genres[i].selected = false;
    // iterate through genres of this movie and set selected
    for (let j = 0; j < this.movie.genres.length; j++) {
      if (genres[i].id == this.movie.genres[j].id) {
        genres[i].selected = true;
      }
    }
  }
}
```

- ❖ Copy and paste below new method to handle the event when any checkbox is checked or unchecked.

```
genreChanged(id: number) {
  for (let i = 0; i < genres.length; i++) { //Change selected flag
    if (genres[i].id == id) {
      genres[i].selected = !genres[i].selected;
    }
  }
  // clear current genres of this movie and add items based on selected flag
  let selectedGenres: Genre[] = [];
  for (let i = 0; i < this.genres.length; i++) {
      if (genres[i].selected) {
        selectedGenres.push(genres[i]);
      }
    }
    this.movie.genres = selectedGenres;
  }
```

src/app/movie/edit-movie.component.html

- Copy and paste the code below to display the fields.

```
Rating<br>
<input type="text" [(ngModel)]="movie.rating" name="rating">
<br><br>

Duration<br>
<input type="text" [(ngModel)]="movie.duration" name="duration">
<br><br>

Budget<br>
<input type="text" [(ngModel)]="movie.budget" name="budget">
<br><br>

Release Date<br>
<input type="text" [(ngModel)]="movie.releaseDate" name="releaseDate">
<br><br>

    Bookings Open<br>
<input type="radio" name="bookingsOpen" [(ngModel)]="movie.bookingsOpen" [value]="true"> Yes
<input type="radio" name="bookingsOpen" [(ngModel)]="movie.bookingsOpen" [value]="false"> No
<br><br>

Favorite <input type="checkbox" name="favorite" [(ngModel)]="movie.favorite">
<br><br>

Genres<br>
<label *ngFor="let genre of genres">
    <input type="checkbox" name="genres" [checked]="genre.selected"
        (change)="genreChanged(genre.id)" value="genre.id"> {{genre.name}}
</label>
<br><br>
```

- Save all files.
- Check if form field values are populated.
- When changes are made in the form fields, the changes should get reflected in the movie component property.

Explanation

- Displaying form fields for rating, duration, budget and release date are straight forward direct two-way binding.
- The radio buttons are selected based on the [value] property binding. If the value provided matches with the Boolean value of the item, then it will be selected.

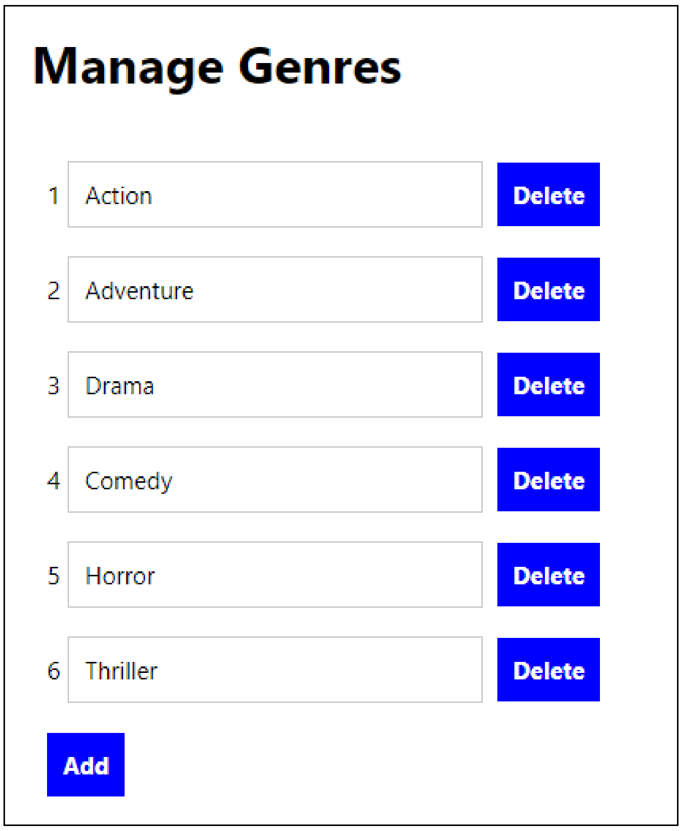
- When using two-way binding of date field, the conversion did not happen as expected and all the options available were in round about way, hence the field data type has been changed from Date to string.
- Two-way binding of checkboxes for Genres were also not possible because of the dynamic nature of checkboxes, ideally this can be easier if implemented using Reactive Forms, hence the display of checkboxes and reading the checkbox values of the form had been done manually.
    - A new attribute selected is included in genre.ts and this attribute represents whether each checkbox of genre should be checked or unchecked.
    - In ngOnInit, the selected value is set based on the items available in movie
    - The genreChanged() method is called when any one of the genre checkbox is clicked.
    - In this method the clicked item is identified and the selected Boolean value is toggled, then the genre items in the movie object is modified to reflect the selected items.

Manage Genres using Reactive Forms

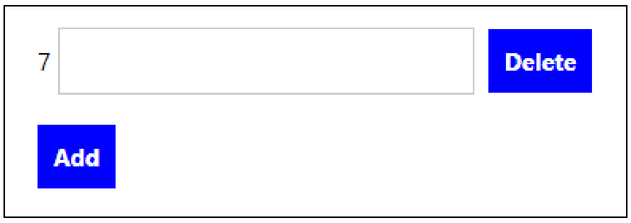
Problem

The Genres already included is not an exhaustive list. For example, genres like Fantasy, Biography, Historical are currently not part of the list. Include a page for admin to manage the list of genres.

Expected Layout



Clicking Add should add a new entry in the bottom.



Display the model data adjacent to the form

```
{
  "genres": [
    {
      "id": 1,
      "name": "Action"
    },
    {
      "id": 2,
      "name": "Adventure"
    },
    {
      "id": 3,
      "name": "Drama"
    },
    {
      "id": 4,
      "name": "Comedy"
    },
    {
      "id": 5,
      "name": "Horror"
    },
    {
      "id": 6,
      "name": "Thriller"
    }
  ]
}
```

Solution (Step #1 – Create component and add route)

- Create new component for managing genres.

```
ng generate component genres
```

app-routing.module.ts

- Import the new component

```
import { GenresComponent } from './genres/genres.component';
```

- Include path in routes

```
{ path: 'genres', component: GenresComponent }
```

app.component.html

- Include anchor tag for new component:

```
<a routerLink="genres" routerLinkActive="active-link">Genres</a>&nbsp;
```

- Save all files and check if Genres link works.

Solution (Step #2 – Include reactive form group)

app.module.ts

- Import reactive forms module.

```
import { FormsModule, ReactiveFormsModule } from '@angular/forms';
```

```
imports: [
  BrowserModule,
  AppRoutingModule,
  FormsModule,
  ReactiveFormsModule
],
```

genres.component.css

- Copy and paste below styles.

```
input[type=text] {
    border: 1px solid #CCCCCC;
    padding: 10px;
    width: 40%;
    margin-top: 2px;
    margin-bottom: 2px;
    font-family: 'Segoe UI';
    font-size: 15px
}

.block {
    float: left;
    width: 45%;
    height: 600px;
    padding: 10px;
    border-right: 1px solid grey
}
```

genres.component.ts

- Include imports

```
import { FormGroup, FormControl } from '@angular/forms';
```

- Include form property

```
genreForm: FormGroup = new FormGroup({
  name: new FormControl('')
});
```

genres.component.html

- Include code to display form:

```
<h1>Genres</h1>
<form [formGroup]="genreForm">
    <input type="text" formControlName="name">
</form>
```

- Save all files and check if form is displayed in Genres link

Explanation

- FormGroup and FormControl are built classes of angular.
- FormGroup represents a collection of form elements.
- FormControl represents a specific input element of a form.
- The attribute [formGroup] helps to assign the genre form from the component to the view.
- The attribute formControlName helps in mapping a specific form element. This should match with the property defined in FormGroup.

Solution (Step #3 – Setting default value)

genres.component.ts

- Set value for the form control

```
genreForm: FormGroup = new FormGroup({
  name: new FormControl('test')
});
```

- Save and check if value is pre-populated in the text box.

Solution (Step #4 – Setting default values from data.ts)

genres.component.ts

- Import genres from data.ts

```
import { genres } from '../data';
```

- Modify form group to have id and name of genre:

```
genreForm: FormGroup = new FormGroup({
  id: new FormControl(''),
  name: new FormControl('')
});
```

❖ Modify ngOnInit to load data:

```
ngOnInit(): void {
  this.genreForm.patchValue(genres[2]);
}
```

genres.component.html

❖ Display form fields for genre's id and name

```
<div class="block">
    <form [formGroup]="genreForm">
        ID<br>
        <input type="text" formControlName="id"><br>
        Name<br>
        <input type="text" formControlName="name">
    </form>
</div>
<div class="block">
    <pre>{{genreForm.value | json}}</pre>
</div>
```

❖ Save all files and check if form is displayed with JSON value displayed adjacently.

Explanation

❖ The patchValue() method helps in assigning the values of the FormGroup elements in a single shot.

❖ The property genreForm.value returns the values from the form as object.

Solution (Step #5 – Simplifying the implementation using FormBuilder)

genres.component.ts

❖ Import FormBuilder

```
import { FormGroup, FormControl, FormBuilder } from '@angular/forms';
```

❖ Inject the FormBuilder through the constructor.

```
constructor(private fb: FormBuilder) { }
```

❖ Modify the genreForm definition

```
genreForm: FormGroup = this.fb.group({
  id: [''],
  name: ['']
});
```

❖ Save and check if the same result is achieved.

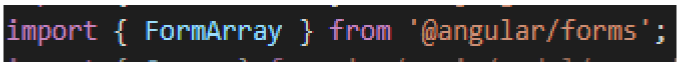
Explanation

❖ FormBuilder simplifies the way in which form controls are defined.

Solution (Step #6 – Populate genres into FormArray)

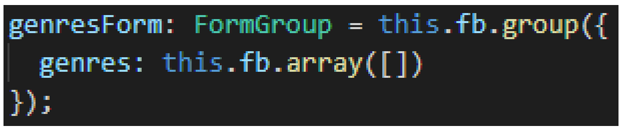
genres.component.ts

- Import FormArray



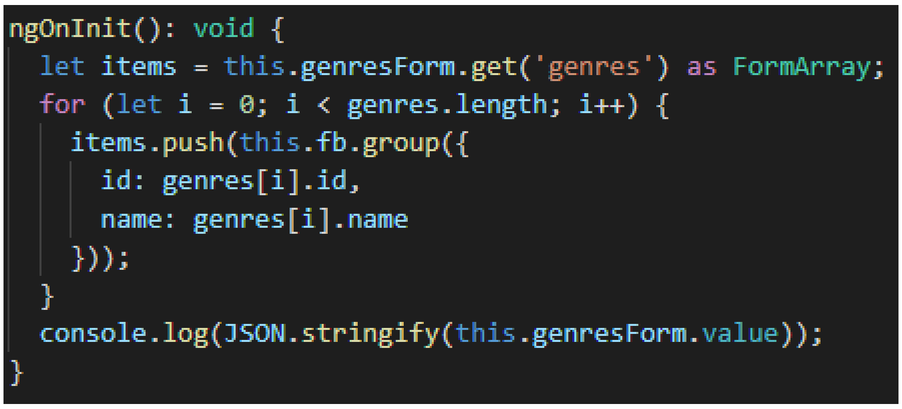
```
import { FormArray } from '@angular/forms';
```

- Initialize FormGroup



```
genresForm: FormGroup = this.fb.group({
  genres: this.fb.array([])
});
```

- Modify ngOnInit



```
ngOnInit(): void {
  let items = this.genresForm.get('genres') as FormArray;
  for (let i = 0; i < genres.length; i++) {
    items.push(this.fb.group({
      id: genres[i].id,
      name: genres[i].name
    }));
  }
  console.log(JSON.stringify(this.genresForm.value));
}
```

- Save and check if the genre details are displayed in the console log.

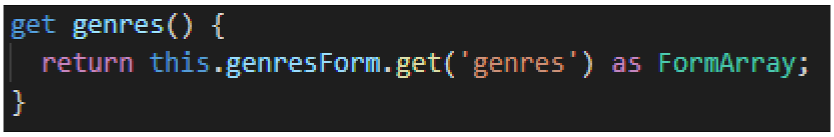
Explanation

- A new property genresForm which represents a FormGroup.
- This formGroup is defined with one attribute named “genres” which will represent the list of genres as form input elements.
- The code in ngOnInit populates the genres data from data.ts into array of controls.
- The get() method in FormGroup returns a built-in class of type AbstractControl. This is an abstract representation of any type of form element, since we know that genres is of type FormArray, we are using ‘as’ keyword to convert the AbstractControl as type of FormArray.
- The for loop iterates through each genre and adds items into FormArray with each entry having a FormGroup having two form controls, one is id and the other one is name.

Solution (Step #7 – Display the controls in the Template)

genres.component.ts

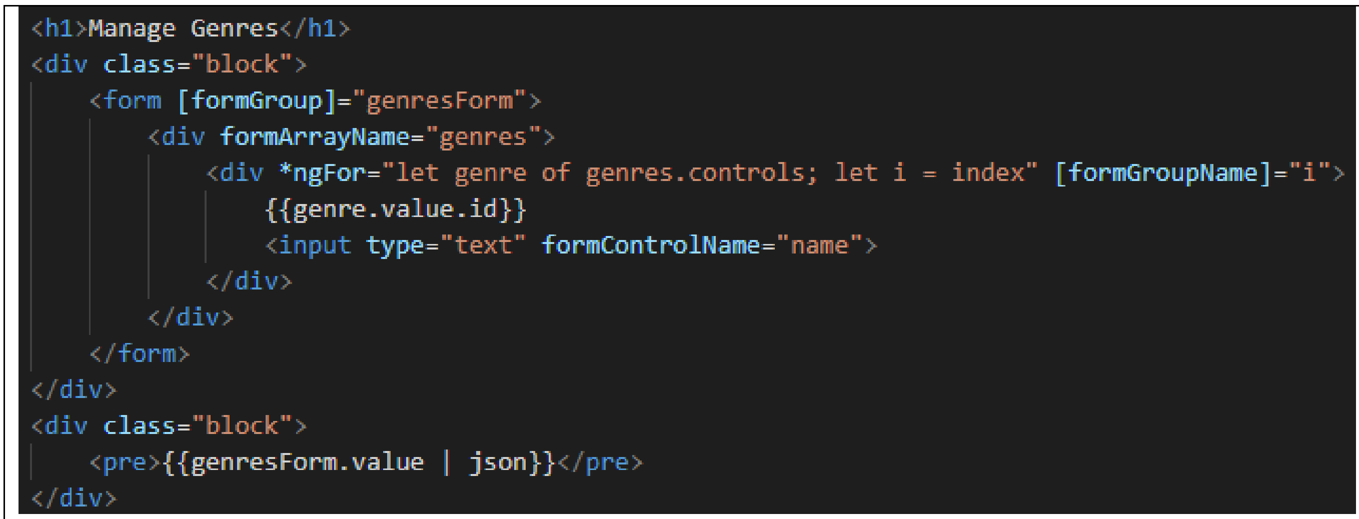
- Implement below getter method:



```
get genres() {
  return this.genresForm.get('genres') as FormArray;
}
```

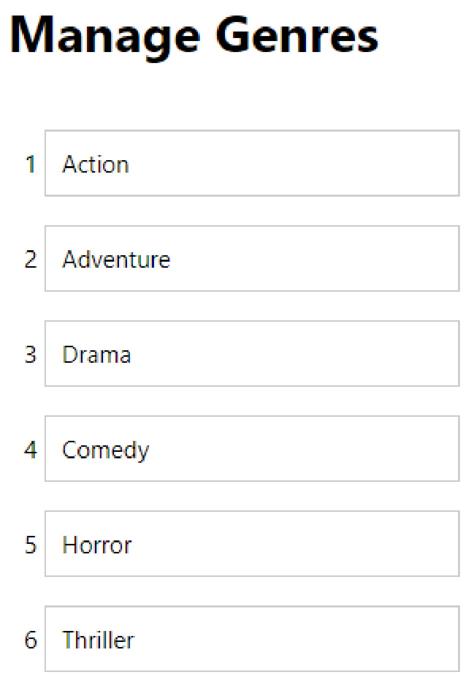
genres.component.html

- Implement coding below:



```
<h1>Manage Genres</h1>
<div class="block">
    <form [formGroup]="genresForm">
        <div formArrayName="genres">
            <div *ngFor="let genre of genres.controls; let i = index" [formGroupName]="i">
                {{genre.value.id}}
                <input type="text" formControlName="name">
            </div>
        </div>
    </form>
</div>
<div class="block">
    <pre>{{genresForm.value | json}}</pre>
</div>
```

❖ Save and check if form controls are displayed as specified below:



Manage Genres

1	Action
2	Adventure
3	Drama
4	Comedy
5	Horror
6	Thriller

Explanation

❖ [formGroup] links to the genreForm.

❖ The attribute formArrayName defines the FormArray of genres.

❖ The genres in the *ngFor refers to the getter method. The getter method helps in converting the AbstractControl to FormArray. This type conversion is not possible in Template code; hence we are using this mechanism.

❖ In TypeScript a getter method can be defined using "get" keyword and can be referred like a property in template.

❖ The interpolation within *ngFor helps in defining read only display of genre id.

❖ The reference variable "I" and formGroupName provides a reference for individual items in the array of items. This will be useful when implementing delete.

❖ The <input> tag within *ngFor helps displaying the text box for each genre.

❖ Any modification in genre should reflect in the model data.

Solution (Step #8 – Implement add genre)

genres.component.ts

- Include itemCount property to define id for new entries:

```
itemCount = 0;
```

- Initialize itemCount in ngOnInit. Inclue this as second line:

```
this.itemCount = genres.length;
```

- Implement below method:

```
addGenre() {
  let items = this.genresForm.get('genres') as FormArray;
  let id = this.itemCount + 1;
  items.push(this.fb.group({
    id: id,
    name: ''
  }));
}
```

genres.component.html

- Include add button just before the closure of the form tag:

```
<input type="button" value="Add" (click)="addGenre()">
```

- Save and check if it creates a new blank entry to type a new form.

Explanation

- The property itemCount helps in keeping track of the id to be generated for new items.
- A new FormGroup is added in the addGenre() method using push(). The id is generated and name is set blank. This automatically updates the view with a new form entry.

Solution (Step #9 – Implement delete genre)

genres.component.ts

- Include method below:

```
deleteGenre(index: number) {
  let items = this.genresForm.get('genres') as FormArray;
  items.removeAt(index);
}
```

genres.component.html

- Include delete button just after genre name control:

```
<input type="text" formControlName="name">
&nbsp;<input type="button" value="Delete" (click)="deleteGenre(i)">
```

- Save and check if it delete remove the respective entry.
- Also check if the item gets removed in the data model as well.

Explanation

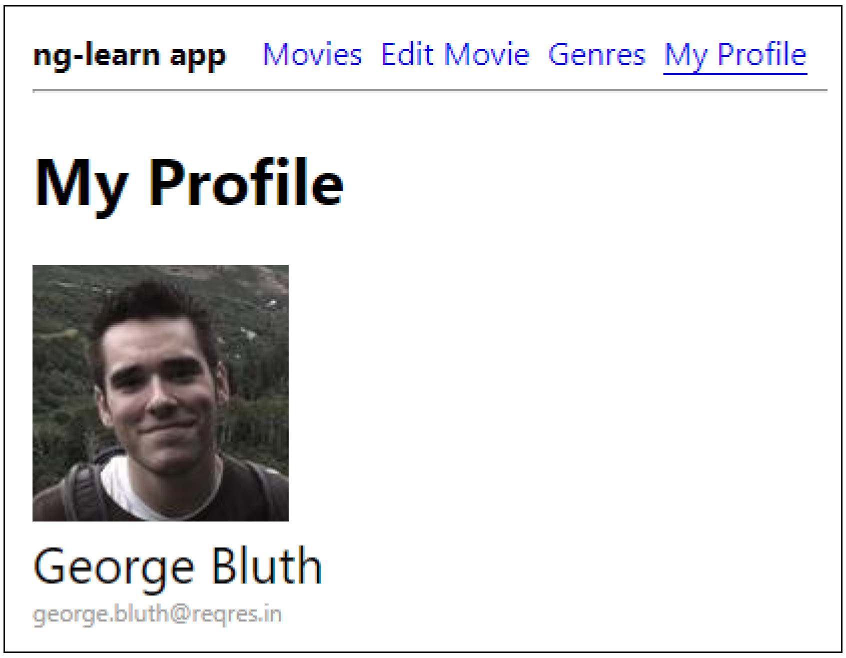
- The removeAt() method available in JavaScript remove the respective FormGroup within the FormArray, which removes the item in the display as well.

Get profile data from REST API

Problem

Provide an option to view the logged in user profile. Get the user profile from the URL https://reqres.in/api/users/1.

Expected Layout



ng-learn app Movies Edit Movie Genres My Profile

My Profile

George Bluth

george.bluth@reqres.in

Solution

- ❖ Create a new component called "profile".

```
ng generate component profile
```

- ❖ Create a new folder "model" in the profile folder.
- ❖ Create new interface user.ts in "model" folder. Copy and paste text below.

```
export interface User {
    id: number,
    email: string,
    first_name: string,
    last_name: string,
    avatar: string
}
```

- ❖ The attributes defined above has to match with the JSON response received for each user.
- ❖ Copy and paste below styles in profile.component.css:

```
.name {
    font-size: 25px;
}

.email {
    font-size: 12px;
    color: #999999
}
```

- ❖ Create a new service for profile:

```
ng generate service profile/profile
```

- Open profile.service.ts
- Include imports:

```
import { HttpClient } from '@angular/common/http';
import { Observable } from 'rxjs';
```

- Include property:

```
usersUrl: string = "https://reqres.in/api/users/1";
```

- Inject HttpClient through constructor:

```
constructor(private http: HttpClient) { }
```

- Include method:

```
getUser(): Observable<any> {
  console.log("Service - before calling get.");
  let observable: Observable<any> = this.http.get<any>(this.usersUrl);
  console.log("Service - after calling get.");
  return observable;
}
```

- Open profile.component.ts
- Import user:

```
import { User } from './model/user';
```

- Include component property:

```
user: User;
```

- Inject profile service:

```
constructor(private profileService: ProfileService) { }
```

- Update ngOnInit:

```
ngOnInit(): void {
  this.profileService.getUser().subscribe(
    data => {
      console.log("Component: Before reading data.");
      this.user = data.data;
      console.log("Compoent: After reading data.");
    }
  );
  console.log("ngOnInit execution completed.");
}
```

- Open profile.component.html. Clear existing text. Paste the code below:

```
<h1>My Profile</h1>
<img src="{{user.avatar}}">
<div class="name">{{user.first_name}} {{user.last_name}}</div>
<div class="email">{{user.email}}</div>
```

- Save all files.
- Check if the expected result is achieved.

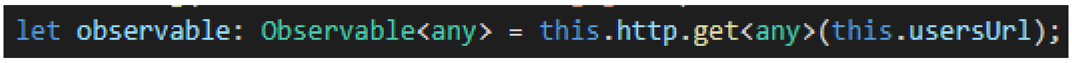
Explanation

Service

- Service in Angular encompasses a function, value or feature of an app.
- A Service has a narrow, well-defined purpose.
- Services increases modularity, where redundant code across components can be reused.
- Component's job is to enable user experience and dealing with the data related to display.
- Service should be used in the following scenario:
    - Application Logic
    - Fetching data from server.
- When creating a service notice that @Injectable decorator is added, which enables any component to inject a service through constructor.
- Angular does not enforce to use Services and it is not mandatory.
- It provides a framework to follow design principles to factor in application logic and make services available to other components through dependency injection.
- In our example, profile service is injected into the component.
- The profile service helps in fetching the data from remote server.

HttpClient

- This in-built angular library helps in making the call to REST API.



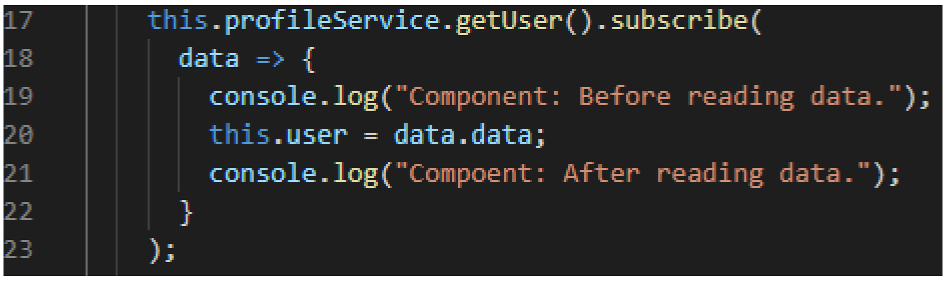
```
let observable: Observable<any> = this.http.get<any>(this.usersUrl);
```

- In the service class HttpClient object is injected through the constructor.
- The get() method makes the call to the userUrl passed as parameter.
- The mention of <any> refers to the type of data returned by the REST API.
- Since we don't have a mapping interface of the data returned, we get it as any and read the user object in the component method.

Observable

- Observable is not part of Angular library.
- It is a third-party library to implement Observable pattern.
- This follows a subscription model. Once you subscribe for an option to an Observer, the observer periodically sends subscriptions to the subscriber.
- We can also think about it for subscribing to deliver newspaper at your door steps.
- The moment you subscribe with the vendor, the vendor is going to deliver newspaper to your door steps everyday until you tell him to stop.
- This is how Observable works. This is used as a way to have asynchronous communication between two entities.
- In application development perspective consider the scenario where cricket commentary to a mobile client is displayed ball by ball. There are two solutions to this approach:
    - A JavaScript code runs every minute to get the updated. Even if there is an update or not, it will try to fetch the data.
    - In the Observer pattern, once the mobile client has subscribed, every time there is change a push will happen to the client to receive the data. This approach makes it effective since the communication happens only there is a need.
- In our example, Observer pattern is used to get the asynchronous response from the REST API.

- The component invokes the service and subscribes for the server response.

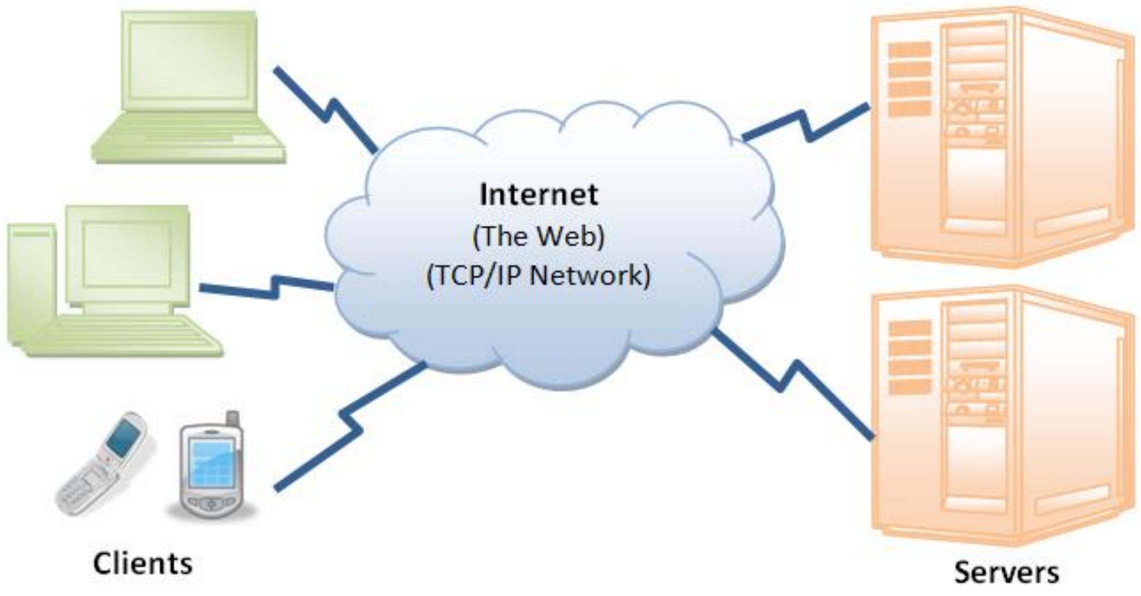


```
    this.profileService.getUser().subscribe(
      data => {
        console.log("Component: Before reading data.");
        this.user = data.data;
        console.log("Compoent: After reading data.");
      }
    );
```

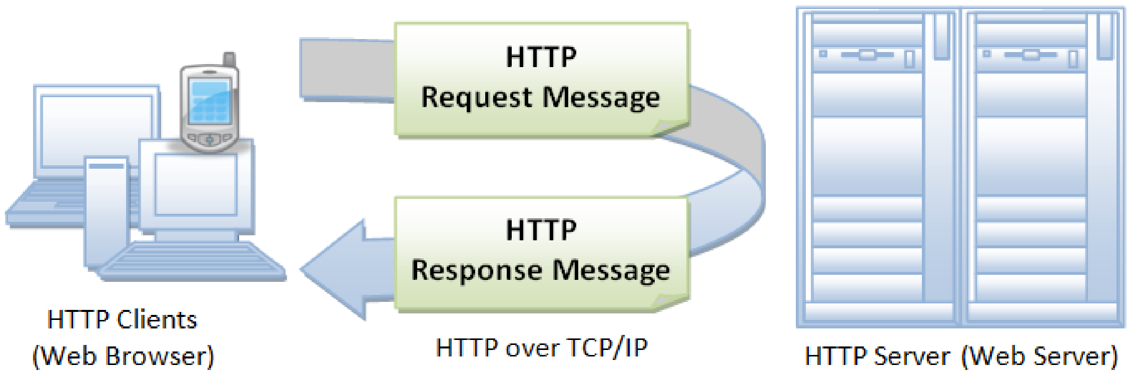
- The subscribe() function call is passed with an anonymous function.
- “data” represents the parameter for the anonymous function and it holds the data from the server.
- The user data is read from it and the component property is set.

Explanation for Web Architecture and HTTP

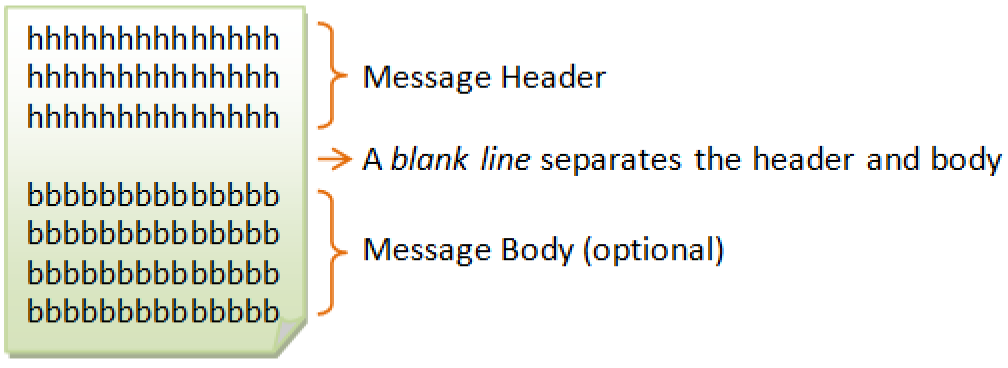
- Client/Server based Architecture
- Client initiates the request
- Server receives, processes and gives a response to the Client
- Client consumes the Server’s response and converts it into display or can use the response for processing.
- Clients
  - Hardware - Mobile Phone, Laptop, Desktop, Tablet
  - Software – Browsers (Chrome, Firefox, Edge), Developer Tools (Postman, CURL), Testing Tools (Selenium, Protractor)
- Servers
  - Hardware – Servers from various vendors (Dell, HP, IBM) running on operating systems Windows Server, Linux and Unix
  - Software – IIS Server, Apache HTTP Server, Nginx, Apache Tomcat



- In the activity we did, Chrome Browser our desktop is the client, Tomcat is the server
- Web Architecture Works on a Request / Response paradigm

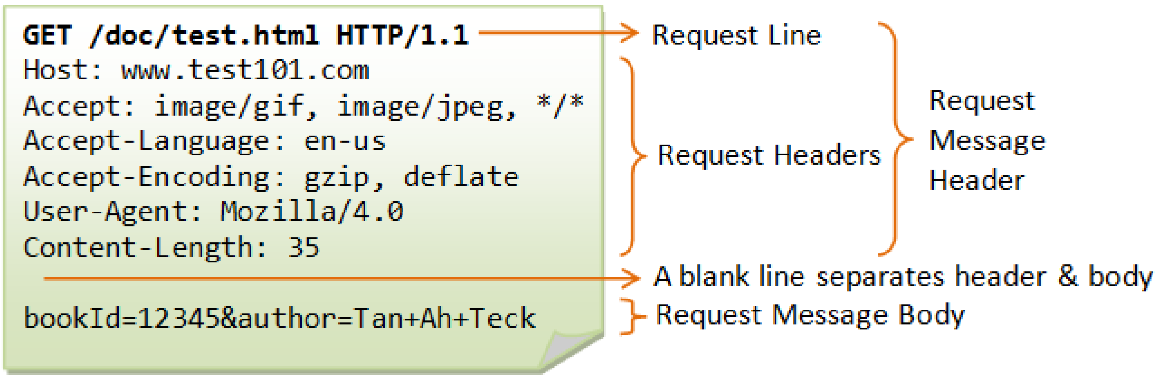


- HTTP stands for Hyper Text Transfer Protocol
- HTTP was developed by various standards organizations like CERN, IETF and W3C
- HTTP is an Application Layer protocol for transmitting documents
- HTTP is a stateless protocol
- Composition of a HTTP request or response



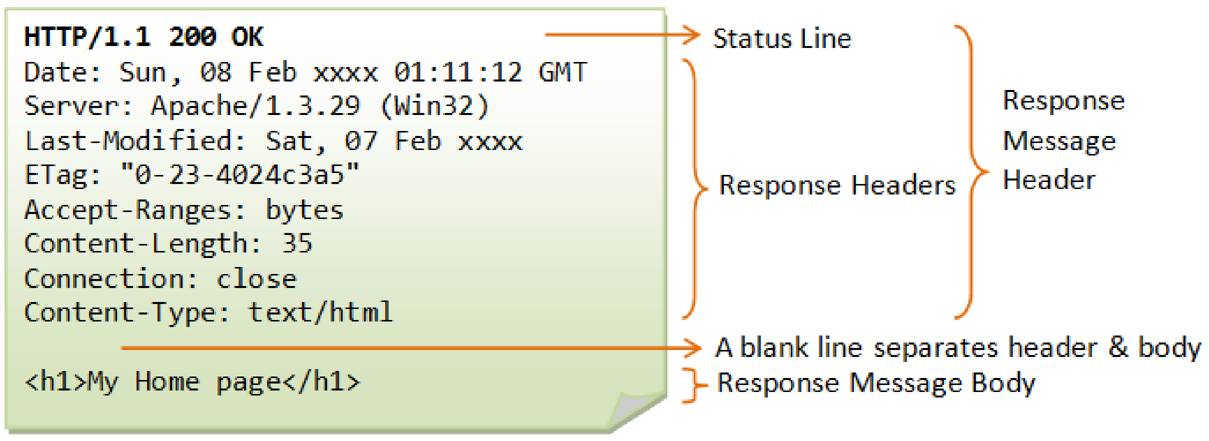
HTTP Messages

- Sample HTTP request



```
GET /doc/test.html HTTP/1.1
Host: www.test101.com
Accept: image/gif, image/jpeg, */*
Accept-Language: en-us
Accept-Encoding: gzip, deflate
User-Agent: Mozilla/4.0
Content-Length: 35

bookId=12345&author=Tan+Ah+Teck
```

- View request of getting user:
  - In Chrome Browser > Click Menu > More tools > Developer tools
  - Click Network
  - Click "My Profile" link
  - Click "1" in the network window
  - View the content under "Request Headers" section
- Sample HTTP response



```
HTTP/1.1 200 OK
Date: Sun, 08 Feb xxxx 01:11:12 GMT
Server: Apache/1.3.29 (Win32)
Last-Modified: Sat, 07 Feb xxxx
ETag: "0-23-4024c3a5"
Accept-Ranges: bytes
Content-Length: 35
Connection: close
Content-Type: text/html

<h1>My Home page</h1>
```

- ❖ Check “Response Headers” section in Network window of Chrome browser.
- ❖ View the HTTP Response Status in “General” section

All image Courtesy: https://www.ntu.edu.sg/home/ehchua/programming/webprogramming/HTTP_Basics.html

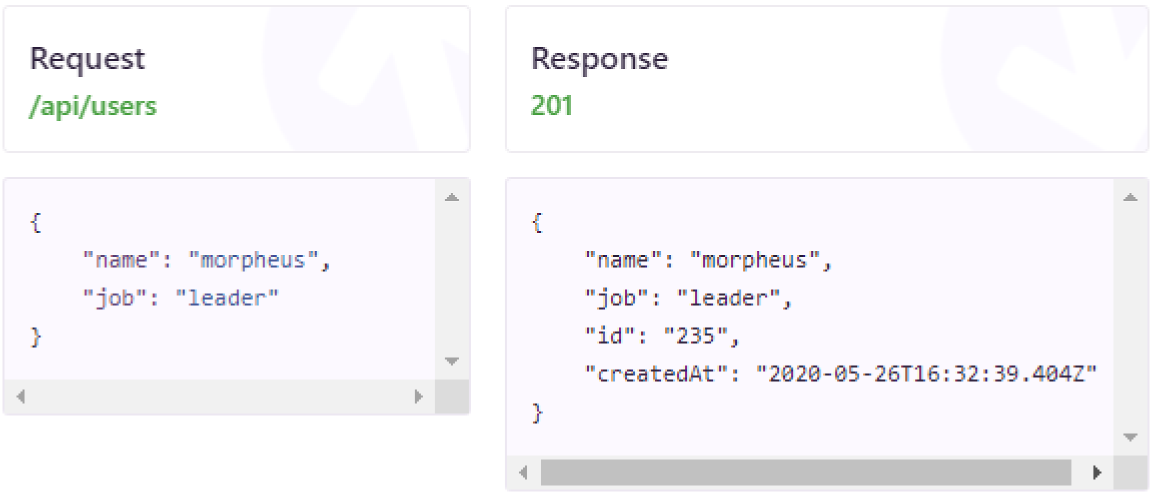
Demonstrate REST API invocation for POST, PUT and DELETE

Problem

Demonstrate invocation of REST API method types POST, PUT and DELETE.

Request Messages

POST



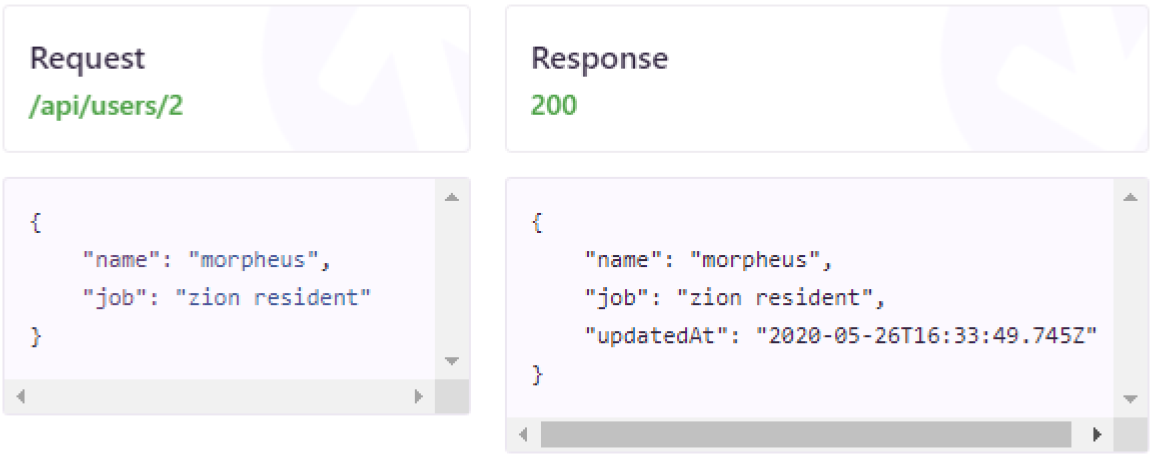
Request
/api/users

```
{
    "name": "morpheus",
    "job": "leader"
}
```

Response
201

```
{
    "name": "morpheus",
    "job": "leader",
    "id": "235",
    "createdAt": "2020-05-26T16:32:39.404Z"
}
```

PUT



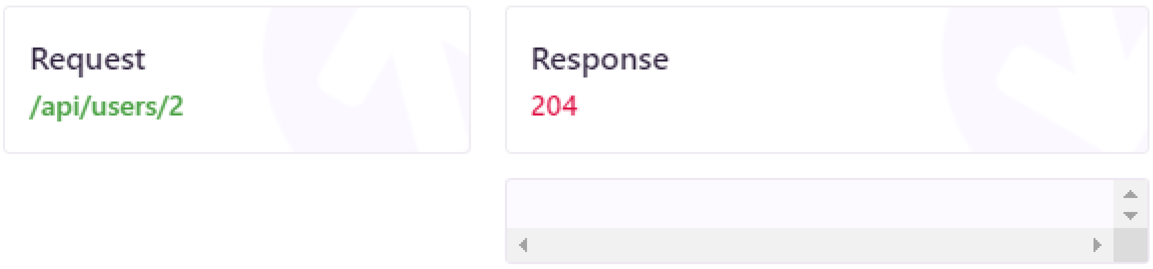
Request
/api/users/2

```
{
    "name": "morpheus",
    "job": "zion resident"
}
```

Response
200

```
{
    "name": "morpheus",
    "job": "zion resident",
    "updatedAt": "2020-05-26T16:33:49.745Z"
}
```

DELETE

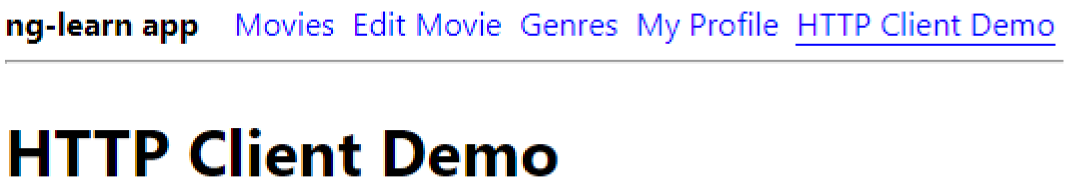


Request
/api/users/2

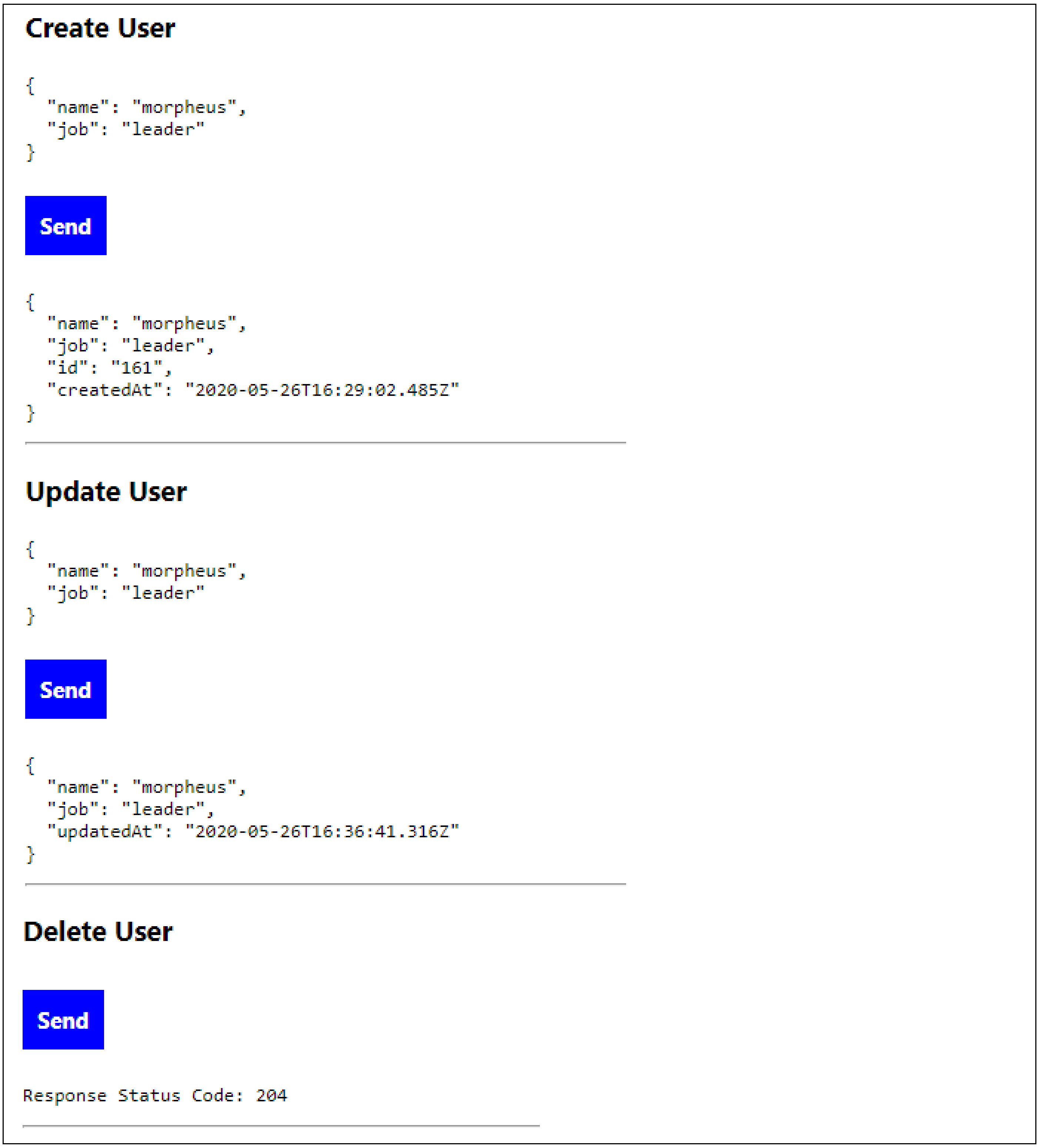
Response
204

Expected Layout

HTTP Demo Component



HTTP Client Demo


Create User

```
{
  "name": "morpheus",
  "job": "leader"
}
```

```
{
  "name": "morpheus",
  "job": "leader",
  "id": "161",
  "createdAt": "2020-05-26T16:29:02.485Z"
}
```

Update User

```
{
  "name": "morpheus",
  "job": "leader"
}
```

```
{
  "name": "morpheus",
  "job": "leader",
  "updatedAt": "2020-05-26T16:36:41.316Z"
}
```

Delete User

```
Response Status Code: 204
```

Solution (Step #1 – Create User)

- ❖ Create new component for HTTP Demo

```
ng generate component http
```

- ❖ Include routing and link in the app.component.html with title as “HTTP Client Demo”

profile.service.ts

- ❖ Include properties:

```
baseUrl: string = "https://reqres.in/api";
api: string = "/users";
```

❖ Include method

```
addUser(user: string): Observable<any> {
  let header : HttpHeaders = new HttpHeaders();
  header = header.set('Content-Type', 'application/json');
  return this.http.post<any>(
    this.baseUrl + this.api,
    user,
    {headers: header}
  );
}
```

http.component.ts

❖ Include properties:

```
user = {
  name: "morpheus",
  job: "leader"
};
createResponse: any;
```

❖ Inject service:

```
constructor(private profileService: ProfileService) { }
```

❖ Include method:

```
addUser() {
  let json: string = JSON.stringify(this.user);
  this.profileService.addUser(json).subscribe(
    data => {
      this.createResponse = data;
    }
  );
}
```

http.component.html

❖ Include below template:

```
<h1>HTTP Client Demo</h1>
<h3>Create User</h3>
<pre>{{user | json}}</pre>
<input type="button" value="Send" (click)="addUser()">
<pre>{{createResponse | json}}</pre>
<hr>
```

❖ Save all files and check if the Create User part is displayed.

❖ Click on the Send button to check if the response is received.

❖ Open network tools in the browser and check if the request is sent and the response is received.

Explanation

❖ The sample request provided in 'reqres.in' accepts an object with properties name and job as input.

❖ Let us assume that there were form fields for name and job and the data from this form is converted into JSON and passing it to the REST API service.

- A user object is created in http.component.ts and this gets converted into JSON before calling the REST API in profile.service.ts.
- A HTTP POST request is used when there is a need to create a new resource.
- The data for the new user needs to be sent in the HTTP Response Body, this is done by passing the user object as the second parameter of post() method in profile.service.ts.
- The POST request needs to be specified with header "Content-Type" having the value as "application/json", as the new user details are sent as JSON in the HTTP Request Body.

Solution (Step #2 – Update User)

profile.service.ts

- Include method

```
updateUser(user: string): Observable<any> {
  let header : HttpHeaders = new HttpHeaders();
  header = header.set('Content-Type', 'application/json');
  return this.http.put<any>(
    this.baseUrl + this.api + "/2",
    user,
    {headers: header}
  );
}
```

http.component.ts

- Include properties:

```
updateResponse: any;
```

- Include method:

```
updateUser() {
  let json: string = JSON.stringify(this.user);
  this.profileService.updateUser(json).subscribe(
    data => {
      this.updateResponse = data;
    }
  );
}
```

http.component.html

- Include below template in the end:

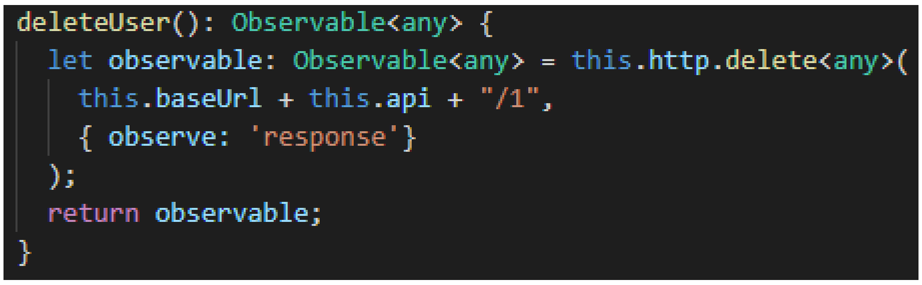
```
<h3>Update User</h3>
<pre>{{user | json}}</pre>
<input type="button" value="Send" (click)="updateUser()">
<pre>{{updateResponse | json}}</pre>
<hr>
```

- Save all files and check if the Update User part is displayed.
- Click on the Send button to check if the response is received.
- Open network tools in the browser and check if the request is sent and the response is received.

Solution (Step #3 – Delete User)

profile.service.ts

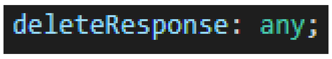
- ❖ Include method



```
deleteUser(): Observable<any> {
  let observable: Observable<any> = this.http.delete<any>(
    this.baseUrl + this.api + "/1",
    { observe: 'response'}
  );
  return observable;
}
```

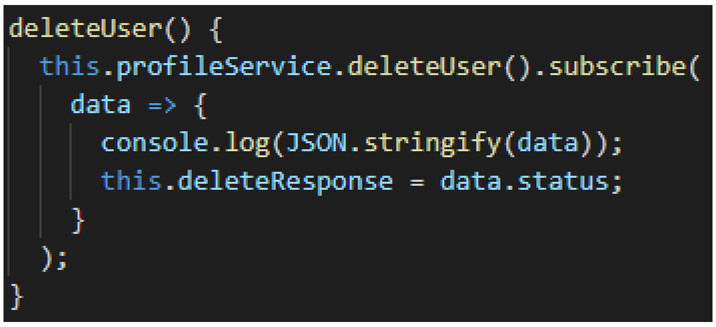
http.component.ts

- ❖ Include properties:



```
deleteResponse: any;
```

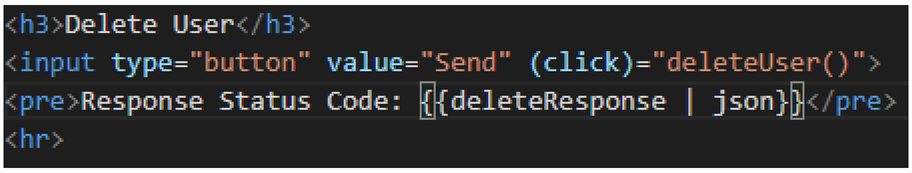
- ❖ Include method:



```
deleteUser() {
  this.profileService.deleteUser().subscribe(
    data => {
      console.log(JSON.stringify(data));
      this.deleteResponse = data.status;
    }
  );
}
```

http.component.html

- ❖ Include below template in the end:



```
<h3>Delete User</h3>
<input type="button" value="Send" (click)="deleteUser()">
<pre>Response Status Code: {{deleteResponse | json}}</pre>
<hr>
```

- ❖ Save all files and check if the Delete User part is displayed.
- ❖ Click on the Send button to check if the response is received.
- ❖ Open network tools in the browser and check if the request is sent and the response is received.

Integrate Movie features with REST API Service

Problem

- The movie data is managed statically using data.ts
- In a production environment, this will not be the scenario, there will be a REST API and database available which is the ideal end to end working scenario.
- In this exercise, Movie components will be made work end to end by implementing the following steps:
  - Install JDK, as the REST API is going to run in a Java environment
  - Launch the REST API service
  - Install MySQL server and start the database server
  - Create Movie database

Install JDK

Download JDK

- Go to https://jdk.java.net/
- Click the JDK under the “Ready for use” section
- Click on the link “zip” adjacent to windows/x64

Install JDK

- Open the downloaded zip file
- Right click on the folder starting with “jdk” and select “Copy”
- Paste the folder to D:
- Open Windows File Explore and navigate to the folder D:\jdk-[ver]\bin
- Click on the address bar and copy the folder location

Setup JDK

- Click “Start” button in windows
- Type “environment” and select “Edit the system environment variables”
- Click “Environment Variables”
- In “System Variables” section scroll down and find the variable named “Path”
- Select and click “Edit”
- Click “New”
- Paste the path copied after installation
- Move the path to the top using “Move Up” button
- Click OK three times on the respective window
- Click “Start” button and type “cmd”
- Select the Command Prompt option and open the command line window
- Type command “javac -version” and press enter
- If version number is displayed then JDK is successfully installed

Download MySQL

- In browser go to https://dev.mysql.com/downloads/mysql/
- Click the download button adjacent to “Windows (x86, 64-bit) ZIP Archive”
- Extract the root folder “mysql-8.0.20-winx64” into D:

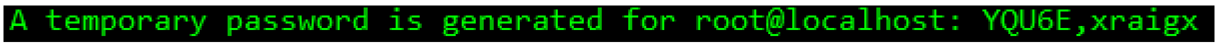
Setup Database

- Create new folder named “data” in D:\mysql-8.0.20-winx64
- In windows file explorer go to D:\mysql-8.0.20-winx64\bin folder
- Click in the empty space next to “bin” folder on the top of the window.
- Type “cmd” and press enter.
- This open command prompt in the bin folder of mysql.
- Execute the following command:

```
mysqld --initialize --console --basedir=D:\mysql-8.0.20-winx64 --datadir=D:\mysql-
8.0.20-winx64\data
```

  IMPORTANT NOTE: If the folder or version number is different, then ensure appropriate folder name.

- Make note of the password generated in the console, look for the line below:



```
A temporary password is generated for root@localhost: YQU6E,xraigx
```

- Select the password and press enter.
- Open Notepad and press Ctrl+V which will paste the password.
- If the MySQL server installation fails or gives error for some reasons, the use the below link to download MySQL server zip file:
  https://drive.google.com/file/d/1CRRJ-xUtDapgFIZPK2dJ3qh1IHJCeLH6/view?usp=sharing

Change password for ‘root’ user

- Now execute the following command to start the server.

```
mysqld --console
```

- Wait till it displays the message “ready for connections”
- Open a new command prompt window in mysql\bin folder.
- Run the following command:

```
mysql -u root -p
```

- Copy the password from notepad
- Open the command prompt window and right click, which will paste the password.
- Press enter and providing the password.
- It will open a mysql> prompt
- Execute the following command:

```
alter user 'root'@'localhost' identified by 'password';
```

- Execute the following command and press enter

```
exit
```

- Run the following command:

```
mysql -u root -p
```

- Enter the password as “password”
- It should not login in the new password.

Stopping MySQL Server

- Open the command window were the server is running
- Press Ctrl+C to stop the server.

Starting MySQL Server and MySQL Client

- Open command prompt in mysql bin folder.
- Execute the following command to start the server:

```
mysqld --console
```

- Open another command prompt in mysql bin folder.
- Execute the following to start the client. Enter ‘password’ as password when prompted.

```
mysqld -u root -p
```

Setup moviedb

- Create a new folder moviedb in D: using windows explorer.
- Open the URL
  https://drive.google.com/file/d/1RitXaYt-usQlWc3tOkEmJj100XqIHxKe/view?usp=sharing
- Download moviedb.sql into D:\moviedb folder
- Go to MySQL client command prompt and execute the following command.

```
source d:\moviedb\moviedb.sql
```

- This will create the tables and data required for movie database.
- Use the below command to change to the moviedb database

```
use moviedb;
```

- Run the following queries one by one to check if train data is available correctly.

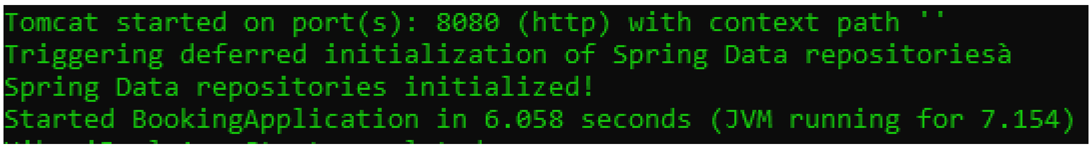
```
select * from movie;
select * from genre;
select * from mpaa_rating;
select * from movie_genre;
select * from user;
```

Download REST API jar and start the service

- Click on the link below and download the jar file.
  https://drive.google.com/file/d/1ChVacdDCuqYpEs-b4sHtXfDA_D5KPzsy/view?usp=sharing
- Move the jar file to a convenient folder.
- Open command prompt in this the folder where the jar file is moved.
- Run the following command

```
java -jar movie-rest-api.jar
```

- This should start the tomcat server with the movie service running. If the log contains the following message it means that the REST API Service is successfully started.



```
Tomcat started on port(s): 8080 (http) with context path ''
Triggering deferred initialization of Spring Data repositoriesà
Spring Data repositories initialized!
Started BookingApplication in 6.058 seconds (JVM running for 7.154)
```

Get Movie List from REST API and Database

- Create a movie service using a new terminal in Visual Studio Code.

```
ng generate service movie/movie
```

- Copy and paste the code below into movie.service.ts

```
import { Injectable } from '@angular/core';
import { HttpClient, HttpHeaders } from '@angular/common/http';
import { Observable } from 'rxjs';
import { Movie } from './model/movie';
import { Genre } from './model/genre';
import { MpaaRating } from './model/mpaa-rating';

const httpOptions = {
  headers: new HttpHeaders({
    'Content-Type': 'application/json'
  })
};

@Injectable({
  providedIn: 'root'
})
export class MovieService {
  baseUrl: string = 'http://localhost:8080/booking/api';
  movieApi: string = '/movies';
  genreApi: string = '/genres';
  mpaaRatingApi: string = "/mpaa-ratings";

  constructor(private http: HttpClient) { }

  getMovies(): Observable<Movie[]> {
    return this.http.get<any>(this.baseUrl + this.movieApi);
  }

  getMovie(movieId: number): Observable<Movie> {
    return this.http.get<Movie>(this.baseUrl + this.movieApi + '/' + movieId);
  }

  getGenres(): Observable<Genre[]> {
    return this.http.get<Genre[]>(this.baseUrl + this.genreApi);
  }

  getMpaaRatings(): Observable<MpaaRating[]> {
    return this.http.get<MpaaRating[]>(this.baseUrl + this.mpaaRatingApi);
  }

  saveMovie(movie: Movie): Observable<any> {
    return this.http.put<Movie>(this.baseUrl + this.movieApi, movie, httpOptions);
  }
}
```

movie-list.component.ts

- ❖ Remove initialization of values from data.ts by commenting the existing movies property.

```
  //movies: Movie[] = movies;
  movies: Movie[];
```

- ❖ Modify ngOnInit() method with the code below:

```
  ngOnInit(): void {
    this.movieService.getMovies().subscribe(
      data => {
        this.movies = data;
        console.log(JSON.stringify(this.movies));
      }
    );
  }
```

- ❖ The above changes should make movie list page get data from database.
- ❖ Modify a movie name in the database and check if the changes are reflected in the movie list page.

edit-movie.component.ts

- ❖ Remove imports from data.ts.

~~import { movies, mpaaRatings, genres } from '../data';~~

- ❖ Remove initialization of values from data.ts for mpaaRatings property.

```
mpaaRatings: MpaaRating[] = mpaaRatings;
```

- ❖ Inject the movieService and router.

```
  constructor(
    private route: ActivatedRoute,
    private movieService: MovieService,
    private router: Router
  ) { }
```

- ❖ Modify ngOnInit() method with below code:

```
  ngOnInit(): void {
    let id: string = this.route.snapshot.paramMap.get('id');
    this.movieService.getMovie(parseInt(id)).subscribe(
      data => {
        this.movie = data;
        this.movieService.getGenres().subscribe(
          data => {
            this.genres = data;
            // iterate all genres
            for (let i = 0; i < this.genres.length; i++) {
              // iterate through genres of this movie and set selected
              for (let j = 0; j < this.movie.genres.length; j++) {
                if (this.genres[i].id == this.movie.genres[j].id) {
                  this.genres[i].selected = true;
                }
              }
            }
          }
        );

        this.movieService.getMpaaRatings().subscribe(
          data => {
            this.mpaaRatings = data;
          }
        );
      }
    );
  }
```

- ❖ Modify genreChanged() method with code below:

```
genreChanged(id: number) {
  for (let i = 0; i < this.genres.length; i++) {
    //Change selected flag
    if (this.genres[i].id == id) {
      this.genres[i].selected = !this.genres[i].selected;
    }
  }

  // clear current genres of this movie and add items based on selected flag
  let selectedGenres: Genre[] = [];
  for (let i = 0; i < this.genres.length; i++) {
    if (this.genres[i].selected) {
      selectedGenres.push(this.genres[i]);
    }
  }
  this.movie.genres = selectedGenres;
}
```

- ❖ Modify onSubmit() with the code below:

```
onSubmit() {
  this.movieService.saveMovie(this.movie).subscribe(
    data => {
      this.router.navigate(['/movie-list']);
    }
  );
}
```

- ❖ Test the display of edit movie form with respective data population
- ❖ Test saving movie details, if it is getting saved in the database.

Protect pages using Login and Guard

Problem

- Implement login using a login form
- The login form should display message 'Invalid Username / Password" if the credentials or not correct.
- Currently hard code the username and password as admin/pwd.
- All the existing links should be accessible only after login
- Provide logout link only when logged in
- Clicking logout should navigate to login page and user now be able to access only after login

Solution (Step #1 – Create login component)

- Create new component for login

```
ng generate component login
```

app.routing.module.ts

- Import login component.

```
import { LoginComponent } from './login/login.component';
```

- Include below paths:

```
{ path: 'login', component: LoginComponent },
{ path: '', redirectTo: "/login", pathMatch: 'full'}
```

app.component.html

- Include login as first link.

```
<a routerLink="login" routerLinkActive="active-link">Login</a>&nbsp;
```

- Save and check if login component is displayed by default.

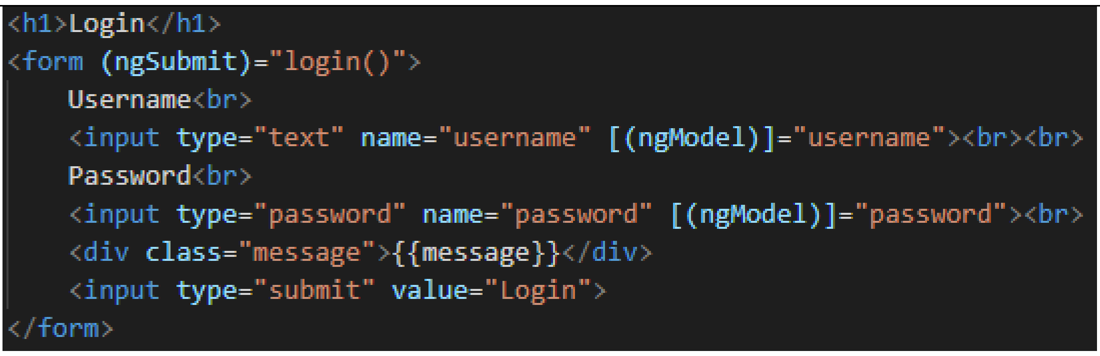
login.component.css

- Copy and paste style.

```
input[type=text], input[type=password] {
    border: 1px solid #CCCCCC;
    padding: 10px;
    width: 25%;
    margin-top: 2px;
    margin-bottom: 2px;
    font-family: 'Segoe UI';
    font-size: 15px
}

.message {
    color: red;
    font-weight: bold
}
```

login.component.html

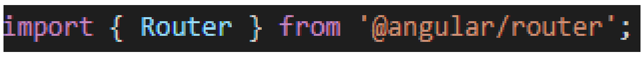
- Copy and paste html content.



```
<h1>Login</h1>
<form (ngSubmit)="login()">
    Username<br>
    <input type="text" name="username" [(ngModel)]="username"><br><br>
    Password<br>
    <input type="password" name="password" [(ngModel)]="password"><br>
    <div class="message">{{message}}</div>
    <input type="submit" value="Login">
</form>
```

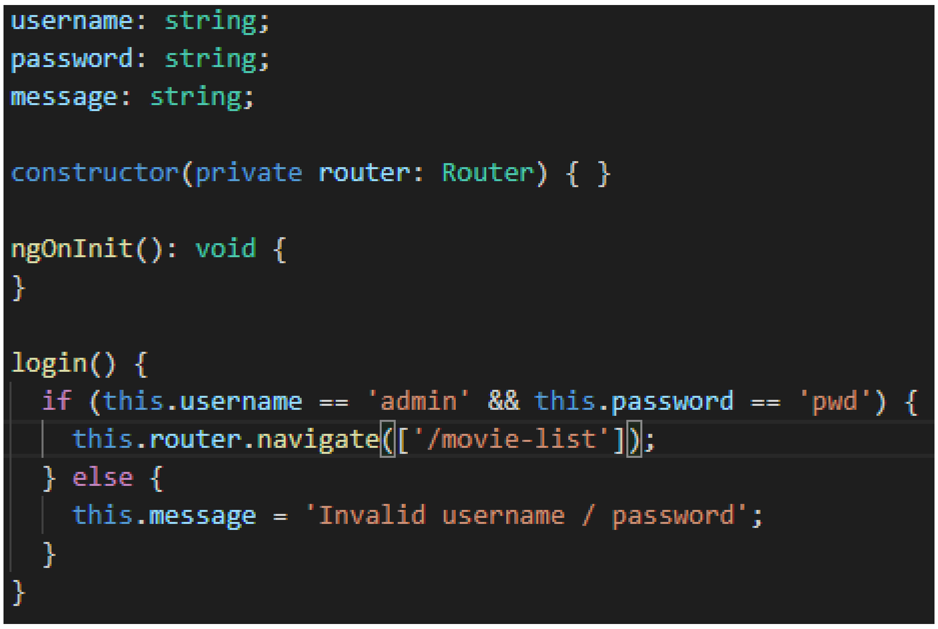
login.component.ts

- Import



```
import { Router } from '@angular/router';
```

- Implement coding as below.

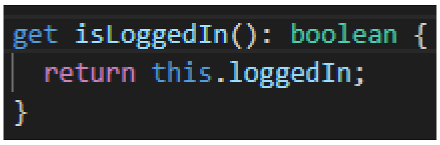


```
username: string;
password: string;
message: string;

constructor(private router: Router) { }

ngOnInit(): void {
}

login() {
  if (this.username == 'admin' && this.password == 'pwd') {
    this.router.navigate(['/movie-list']);
  } else {
    this.message = 'Invalid username / password';
  }
}
```

- Save and check if login with admin/pwd is successful. For other credentials it should fail with message.

Solution (Step #2 – Create authentication service to track logged in status)

- Create new service named “auth”
- Include property ‘loggedIn’
- Implement a new method named login() that sets the loggedIn property as true.
- Implement a new method named logout() that sets the loggedIn property as false.
- Implement getter method to expose the loggedIn property to other components.

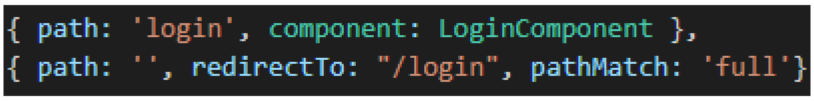


```
get isLoggedIn(): boolean {
  return this.loggedIn;
}
```

- Inject AuthService in login.component.ts constructor
- Then invoke authService.login() method to change the login status as true.

Explanation

- ❖ The empty path definition helps in defining a default path.



```
{ path: 'login', component: LoginComponent },
{ path: '', redirectTo: "/login", pathMatch: 'full'}
```

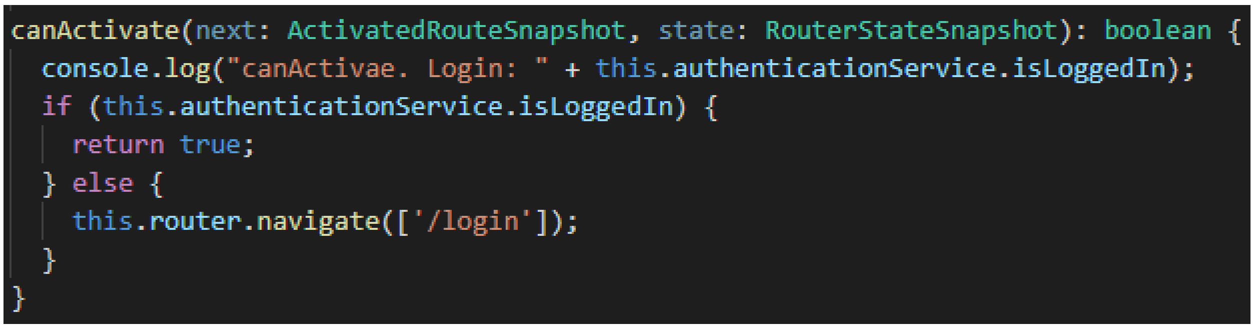
- ❖ This definition ensures that login component is displayed on start of the application.
- ❖ The authentication service helps various components to know if the user is logged in or not.

Solution (Step #3 – Create Guard to secure access to routes)

- ❖ Create new guard named “auth” in “auth” folder.

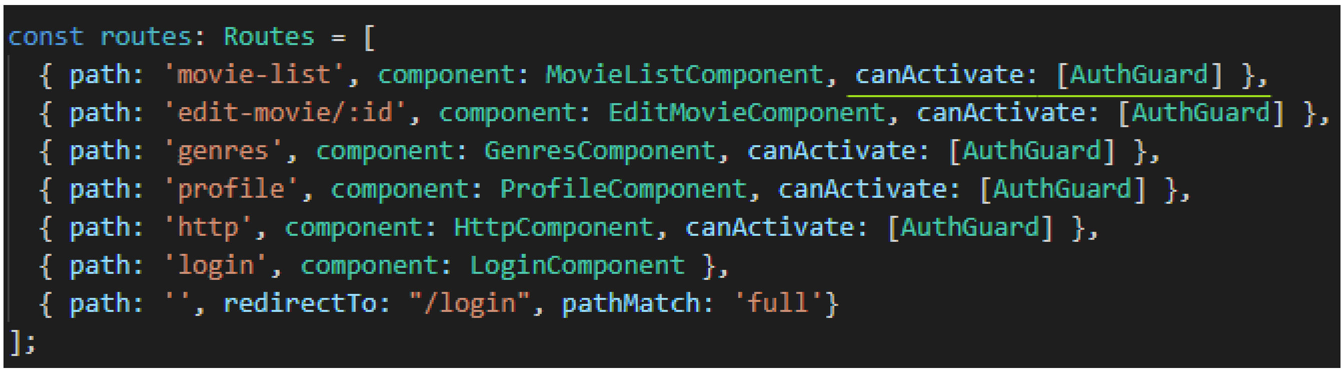
```
ng generate guard auth/auth
```

- ❖ Select canActivate() on the prompt by pressing enter.
- ❖ In auth.guard.ts inject Router and AuthService in the constructor.
- ❖ Change the return type of canActivate() method as Boolean.
- ❖ Implement the following in canActivate() method.



```
canActivate(next: ActivatedRouteSnapshot, state: RouterStateSnapshot): boolean {
  console.log("canActivae. Login: " + this.authenticationService.isLoggedIn);
  if (this.authenticationService.isLoggedIn) {
    return true;
  } else {
    this.router.navigate(['/login']);
  }
}
```

- ❖ In app.routing.module.ts update the routes with AuthGuard.



```
const routes: Routes = [
  { path: 'movie-list', component: MovieListComponent, canActivate: [AuthGuard] },
  { path: 'edit-movie/:id', component: EditMovieComponent, canActivate: [AuthGuard] },
  { path: 'genres', component: GenresComponent, canActivate: [AuthGuard] },
  { path: 'profile', component: ProfileComponent, canActivate: [AuthGuard] },
  { path: 'http', component: HttpComponent, canActivate: [AuthGuard] },
  { path: 'login', component: LoginComponent },
  { path: '', redirectTo: "/login", pathMatch: 'full'}
];
```

- ❖ Save all files.
- ❖ Check the following:
  - Login page is displayed when the URL is http://localhost:4200
  - Without login the rest of the links redirects to login.
  - After login all the pages should work.

Explanation

- ❖ In the routes where AuthGuard is defined, the routing happens only after authentication verification.

- The canActivate() method is called when ever these links are clicked.
- The canActivate() method checks if user is logged in using the authService.
- If the user is logged in it returns true so that the respective component is displayed.
- If the user is not logged in it navigates to the login page.

Solution (Step #4 – Implement logout)

- Modify app.component.css anchor style:

```
a {
    text-decoration: none;
    color: blue;
    cursor: pointer
}
```

- Implement coding below in app.component.ts

```
constructor(
  public authService: AuthenticationService,
  private router: Router) {}

logout() {
  this.authService.logout();
  this.router.navigate(['/login']);
}
```

- Include logout link in app.component.html

```
<a *ngIf="authService.isLoggedIn" (click)="logout()">Logout</a>&nbsp;
```

- Save all files and check if logout works.

Explanation

- The *ngIf definition ensures that logout is displayed only when the user is logged in.
- The logout() method changes the login status to false and navigates to login page.

Build for Production deployment

Problem

Build the angular application for production deployment.

Why we need to build for production?

- The 'ng serve' command just creates a local server for running in the desktop, it is not a production ready web server.
- For running an angular application in production, a web server is required. The web server can be any one of the following:
  - Apache Web Server
  - Nginx
  - IIS Server
  - Java Web Servers
    - Tomcat
    - JBoss
    - Web Sphere
    - Weblogic
- This is not an exhaustive list and there are numerous other servers available.
- The browser does not understand TypeScript or the folder structure of an angular project.
- Browser can only understand HTML, CSS and JavaScript.
- The build process converts all the files and angular project folder structure into HTML and JavaScript files.

Steps to Build and Deploy

Build

- Run the following command in the terminal window:

```
ng build --prod --base-href=/angular-learn/
```

- After successful execution of the above command check if a new folder 'dist' is created in the root folder of the angular application.
- The 'dist' folder should contain a folder 'angular-learn'
- The angular-learn contains an index.html file, a set of JavaScript files and few other files.
- The angular-learn folder can be copied to any web server folder to run the application.

Download JDK

- JDK is required for running Tomcat
- Go to https://jdk.java.net/
- Click the JDK under the "Ready for use" section
- Click on the link "zip" adjacent to windows/x64

Install JDK

- Open the downloaded zip file
- Right click on the folder starting with "jdk" and select "Copy"

- Paste the folder to D:
- Open Windows File Explore and navigate to the folder D:\jdk-[ver]\bin
- Click on the address bar and copy the folder location

Setup JDK

- Click "Start" button in windows
- Type "environment" and select "Edit the system environment variables"
- Click "Environment Variables"
- In "System Variables" section scroll down and find the variable named "Path"
- Select and click "Edit"
- Click "New"
- Paste the path copied after installation
- Move the path to the top using "Move Up" button
- Click OK three times on the respective window
- Click "Start" button and type "cmd"
- Select the Command Prompt option and open the command line window
- Type command "javac -version" and press enter
- If version number is displayed then JDK is successfully installed

Download Tomcat

- Go to http://tomcat.apache.org/
- Click on download of the latest version.
- Under Binary Distribution section click on zip to download.
- Open the zip file and select the folder starting with apache-tomcat
- Paste the folder in D:

Setup JAVA_HOME Environment Variable

- Click "Start"
- Type "environment"
- Select "Edit the system environment variables"
- Click "Environment Variables"
- If JAVA_HOME already exists:
    - Click on JAVA_HOME
    - Click "Edit"
    - Specify "Variable Value" as the folder "D:\jdk-[ver]\" where JDK zip was extracted
    - Click "OK" and close the windows one by one
- If JAVA_HOME does not exist:
    - Click "New"
    - Specify "Variable Name" as "JAVA_HOME"
    - Specify "Variable Value" as the folder "D:\jdk-[ver]\" where JDK zip was extracted
    - Click "OK" and close the windows one by one

Run Tomcat Server

- Open Windows Explorer and navigate to the D:\apache-tomcat-[ver]\bin folder
- Go to address bar

- Type “cmd”
- Execute the command “Catalina run”
- Open browser and specify URL http://localhost:8080
- Check if tomcat page is displays, which means that tomcat is installed successfully.
- To run the application in Tomcat server copy and paste the angular-learn folder to tomcat\webapps folder and access the application using the URL specified below:

  http://localhost:8080/angular-learn
- Test the application by logging in and navigating through the pages.
- Press “Ctrl+C” in the command line window to stop the server.

Miscellaneous Topics

Animations

https://angular.io/guide/animations

Internationalization (i18n)

https://angular.io/guide/i18n

Lazy Loading modules

https://angular.io/guide/lazy-loading-ngmodules

Interview Questions

Questions

- What is Angular used for?
- What is the TypeScript file that is loaded first when an angular application starts?
- List the steps that happen when an angular application starts.
- What is a Module?
- What is an App Component?
- What is a Component?
- What is Interpolation?
- What are Pipes?
- What is a Structural Directive?
- What is Property Binding?
- What is two-way binding?
- What is Template Reference Variable?
- A Template Reference Variable can be used in Component. (True / False)
- What are the two types of forms implementation available in Angular?
- Brief about the steps involved to implement Template Driven Form.
- Brief about the steps involved to implement Reactive Form.
- What is the purpose of a Form Builder?
- What is the purpose of a Form Array?
- What is the purpose of HttpClient module?
- What is the purpose of a Service?
- What is Dependency Injection?
- How to build an Angular application?

Answer Keys for Interview Questions

1. **question?**

   answer