**Modules in Angular:**

1) Create a Module named HRMS.

2) Add following two components in HRMS module:

a) Employee

b) HR

3) Export only Employee component from HRMS module.

4) Add Employee component to AppComponent.

**Data Binding in Angular:**

1) Create a variable named message in a component class as shown below:

message:string = “Message of the Day”;

Display the value of the message variable in component HTML (Hint: Use String Interpolation).

Sample output is shown below:



2) Create a variable named todo in a component class as shown below:

todo:string = “Enter the message of the day”;

Create a TextBox in component HTML. Assign the value of todo variable to the placeholder attribute of this TextBox.

Sample output is shown below:



3) Create a button in the component HTML as shown below:



When the user clicks on Change Message button, the value of message variable should change to the “Tip of the Day”.

4) Display the value of message variable in the TextBox. Enter a new message in the TextBox. When a user clicks on the Change Message button, the value of message variable should be updated with whatever value user has entered in the TextBox.

**Component Communication:**

**Parent Child Communication:**

1) Declare a string variable in App component class and assign some value to it.

2) Create a new component named AppChild. Display the value of a variable declared in Step 1 in the HTML of AppChild component.

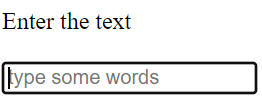
3) Declare a string variable in AppChild component class and assign some value to it. Write a method in AppChild component. This method should emit a value of the string variable.

4) Add SendData button to the AppChild Component. Call the method written in Step3 on Click event of SendData button.

5) Display the value of a string variable declared in Step 3 in AppComponent.

**Sibling Communication:**

6) Create an AppSearch component. Add a TextBox in the HTML of this component. AppSearch Component should look as shown below:



7) Add AppSearch component to the AppComponent.

8) Add a string variable in AppComponent class.

9) Add a method in AppComponent class. This method should capture the keyword keyed in the TextBox in the AppSearch component. Use the string variable declared in Step 8 to capture this value.

10) Create an AppTable component.

11) Declare an Array in AppTable component. The array members should have following properties:

a) name

b) age

c) post

Add some members to this array (Hardcoded).

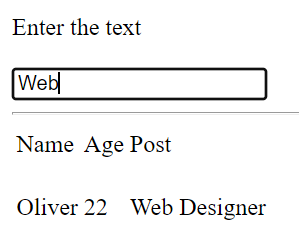
12) Add AppTable component to the AppComponent.

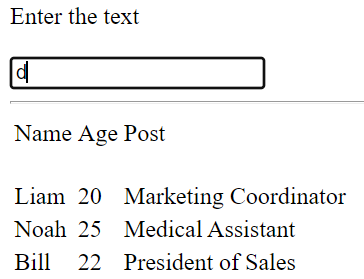
12) Pass the value captured in Step 10 as an input to AppTable component.

13) Based on the keyword keyed in the TextBox, AppTable component should display the matching components from the array.

Sample screenshots are shown below:







**Content Projection:**

1) Create a component named StylishButton.

2) Add StylishButton component to the AppComponent. AppComponent should pass the HTML to be displayed on StylishButton component.

Sample User Interface for StylishButton component is shown below:



3) Create a component named Technologies.

Add the following CSS to Technology component:

.card { min-width: 280px;  margin: 5px;  float:left  }

.header { color: blue}

4) Technology component should have one Parent div and three Child divs. Child divs should be named header, content and footer.

5) Add Technology component to AppComponent. AppComponent should pass the HTML to be displayed on Technology component.

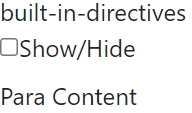
Sample User Interface for Technology component is shown below:



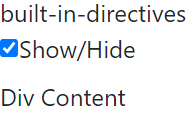
**Angular Directives:**

1) Create an Angular component. Please add Checkbox to that component with a text Show/Hide.

When the Checkbox is unchecked, component should display Para Content.



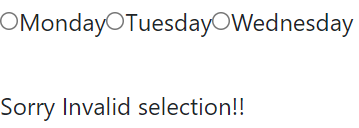
When the Checkbox is checked, component should display Div Content.



2) Create an Angular component. The component should have three Radio Buttons:

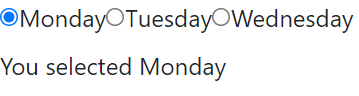


When none of the Radio Button is selected, the component should display Sorry Invalid Selection!!



When user selects a Radio Button, the component should display the message You Selected along with Radio Button text.

E.g. – If user selects Monday, then the component should display the message You Selected Monday.



3) Create an Angular component. Define the following array in the component:

employees: any[] = [

    {

        code: 'emp1', name: 'Karthik', gender: 'Male',

        annualSalary: 5500, dateOfBirth: '25/6/1988'

    },

    {

        code: 'emp2', name: 'sachin', gender: 'Male',

        annualSalary: 5700.95, dateOfBirth: '9/6/1982'

    },

    {

        code: 'emp3', name: 'rahul', gender: 'Male',

        annualSalary: 5900, dateOfBirth: '12/8/1979'

    },

    {

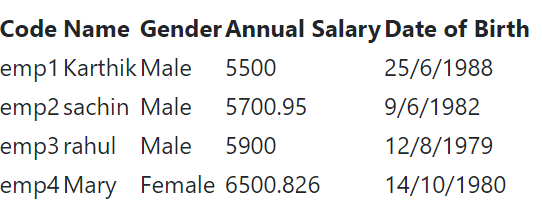
        code: 'emp4', name: 'Mary', gender: 'Female',

        annualSalary: 6500.826, dateOfBirth: '14/10/1980'

    },

  ];

The component should display the following output:



4) Include bootstrap in your Angular application. Create an Angular component. Define the following array in the component:

people: any[] = [

    {

      "name": "Douglas  Pace",

      "country": 'UK'

    },

    {

      "name": "Mcleod  Mueller",

      "country": 'USA'

    },

    {

      "name": "Day  Meyers",

      "country": 'HK'

    },

    {

      "name": "Aguirre  Ellis",

      "country": 'UK'

    },

    {

      "name": "Cook  Tyson",

      "country": 'USA'

    }

  ];

The component should display the following output twice once using ngStyle and once using ngClass:



Use the following bootstrap classes while displaying output using ngClass:

For UK : text-success

For USA : text-primary

For HK : text-danger

Write a method in a component that returns the color depending on the country. The method should return the following colors:

For UK : Green

For USA : Blue

For HK : Red

Call this method while displaying the output using ngStyle.

5) Create a custom directive that behaves like ngIf.

Add a div tag in App component with some text. Apply the custom directive you created to this div tag.

Add a Checkbox in App component. Div tag with custom directive should be displayed when this Checkbox is checked and should not be displayed when the Checkbox is unchecked.

Sample output is shown below:





6) Create a custom directive that behaves like ngClass. Add a button in App component. Apply the custom directive to this button.

Sample output is shown below:



**Pipes in Angular:**

1) Create an Angular application.

2) Declare following variables in App component class:

  content:string="Pipes in Angular";

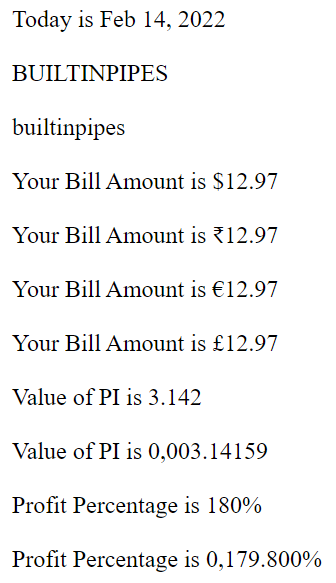
  dob:number = Date.now();

  amount:number = 12.97;

  pi: number = 3.14159265359;

  profitPer:number =  1.798;

3) Write the necessary code to produce the following output:



**CUSTOM PIPE**

1. Create a component named **FilterComponent.**
2. Create an array of 50 strings *(hard-coded)*. The array must contain some people/employee/person names.
3. Create a custom pipe named **FilterPipe** which should be applicable to the array created.
4. The pipe must accept a condition as an argument *(the filter criteria)* & must return the names matching the criteria.
5. Using an **ngFor** directive iterate the array inside the template so that only the names matching the filter criteria are displayed. The sample code inside the template might look like this:

*<div \*ngFor=”let name of names | filter: ……>*

*{{ name }}*

*</div>*

**@ViewChild & @ViewChildren DECORATORS**

1. Create two components named **NumberListComponent & NumberComponent** respectively.
2. The **NumberComponent** must be a child of **NumberList.**
3. The **NumberListComponent** has an array of 10 numbers, which it iterates through using **ngFor** & on each iteration, it renders a **NumbersComponent** and passes the number being iterated to it as an input property. Example:

*<app-number [number]=”n”></app-number>*

1. In the **NumberComponent,** display the value of *number* input property with a checkbox next to it. So, now the **NumbersComponent** must have a list of numbers with a checkbox next to it, something like this:

[ ] 10

[ ] 20

[ ] 30

[ ] 40

….

..

1. In the **NumberListComponent,** create a button titled **Select All.** When the button is clicked, all checkboxes must be checked.
2. In the **NumberListComponent,** create a button titled **Select all odd numbers.** When the button is clicked, all checkboxes with odd numbers must be checked.

**SERVICES**

**Using an Angular service, establish the following communication for two sibling components.**

1. Create two components named **Sender** & **Receiver** respectively. These two must be SIBLING COMPONENTS inside the root component.
2. Each component will have a textbox and a button titled **SEND MESSAGE**.
3. Each component will also have a paragraphelement below the textbox.
4. When something is entered in each of the textboxes in each component and the button is clicked, the text must be transmitted to the opposite component. Each component will then display the transmitted message inside their paragraph element respectively.

**ROUTING**

1. Create 3 components named **Home, Downloads** & **Friends** respectively.
2. Each component’s template can display a simple string.
3. For the application, configure the following routes:
4. / OR /home  display HomeComponent
5. /downloads  display DownloadsComponent
6. /friends  display FriendsComponent
7. Any other route, redirect to the HomeComponent

1. Display navigation links on the root component to allow the user to navigate.
2. Create an array of 10 names inside the **FriendsComponent.**
3. Display each element of the array as a link in the **FriendsComponent.**
4. When a name is clicked, display another component named **FriendComponent** inside the **FriendsComponent** which displays the friend name clicked. Configure a child route for the **FriendsComponent** as follows:

/friends/friend/<<friendname>> *(<<friendname>> is a route param)*

1. Add a route guard to the **FriendsComponent** to ensure that the route can be visited only if there are more than 10 friends. In case the number of friends is less then 10, redirect back to **HomeComponent.**

**LAZY LOADING**

1. In the Routing assignment created above, put the **FriendsComponent** inside a module named **FriendsModule** & lazy load the module when the route changes to **/friends.**

**OBSERVABLES**

1. Create a TS file named **DataSource.ts**.
2. Create a function named **GetNumbers()** which returns an Observable of numbers.
3. From the function create an observable of 10 numbers using the RxJs **from()** operator and return the observable.
4. From the root component, in the **ngOnInit()** method, call the function, subscribe to the observable & display each number in a bulleted list form in the component’s template.
5. Refactor the observable in step 3 so that it is now created using the **of()** operator.
6. Create another component & in its template subscribe to the observable using the **async** pipe and display the numbers in a bulleted list format.

1. In the assignment for SERVICES, implement sibling component communication using RxJs **Subject** & **BehaviorSubject.**

**HTTP COMMUNICATION & INTERCEPTORS**

1. Create file named **data.json** with the following data:

{

"cars": [

{

"id": 1,

"brand": "Ford",

"model": "Titanium",

"year": 2018

"type": "automatic"

},

{

"id": 2,

"brand": "Hyundai",

"model": "Accent",

"year": 2017

"type": "manual"

},

]

}

Put some more data related to cars in the same manner.

1. Host this using **json-server** as a REST API**.**

1. Create an Angular service named **CarsService** which contains the following methods:

allCars()  GET request to /api/cars

findCar(id: number)  GET request to /api/cars/id

findCarByBrand(brand: string)  GET request to /api/cars?brand=brand

addCar(car: any)  POST request to /api/cars

removeCar(id: number)  DELETE request to /api/cars/id

refactorCar(id: number, car: any)  PUT request to /api/cars/id

**Each of these methods must return an observable. The POST and PUT methods must return the newly added car and refactored car respectively.**

1. Create separate components to invoke the above mentioned endpoints. Each component must use the service via Dependency Injection to perform CRUD operations.

**Implement the error handling logic inside the service itself. How to do this? Refer the link below to get an idea:**

[**https://angular.io/guide/http#handling-request-errors**](https://angular.io/guide/http#handling-request-errors)

1. Create an HTTP interceptor which logs the request date and time to the console whenever any type of HTTP request is made.

1. Create an HTTP interceptor which adds the following custom HTTP header only when a POST request is made:

**context: IMPACT Angular Assignments**

1. Create an HTTP interceptor which converts the incoming response to a plain string. This interceptor must be applicable only to GET requests.