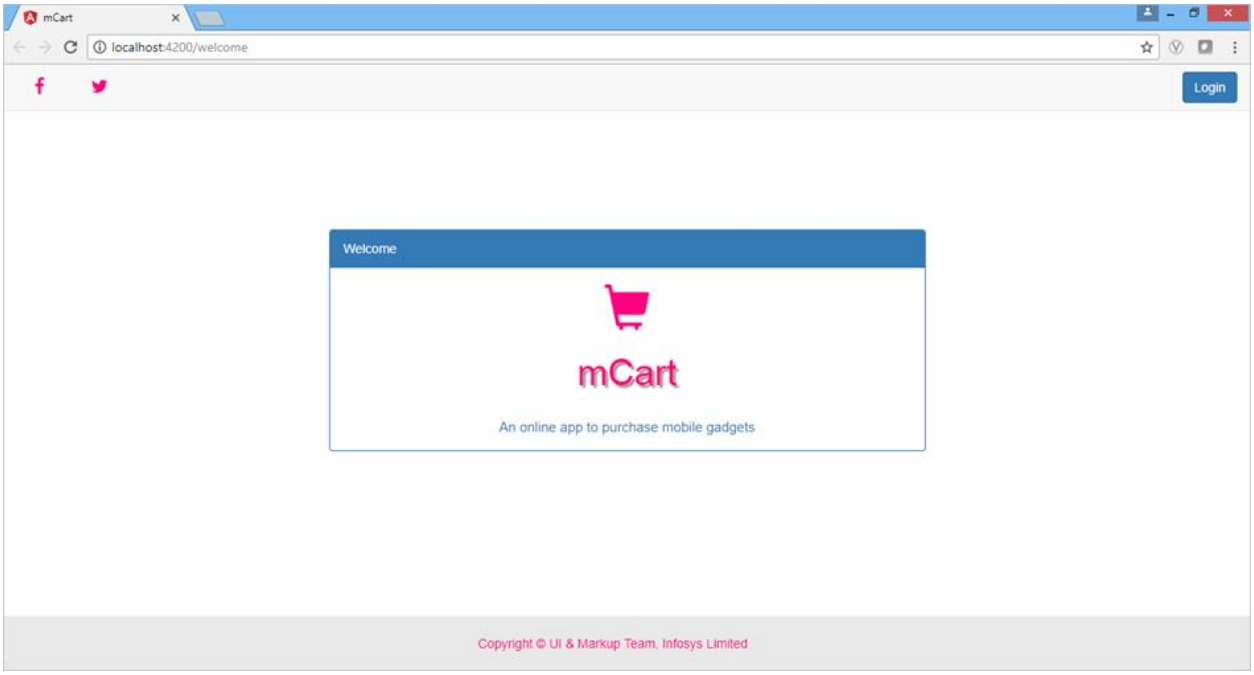
Exercise -1 : Explore mCart Case Study

Time Limit: 15 Minutes

Problem Statement

Observe the link http://localhost:4200/welcome on which mCart is running. Perform below activities to understand the features of the application



1. Click on Login button at the top right corner and observe the url

2. Login with different credentials and see the message displayed

3. Login with credentials (admin,admin) and check how the redirection is happening by observing the url

4. Click on the two tabs (Tablets, Mobiles) which displays tablet and mobile devices

5. Click on any product name and see the product detail page getting displayed

6. Click on Add to Cart button and add multiple products to the cart (selection count and total price will be displayed on the second navigation bar)

7. Click on the cart link on the second navigation bar and observe the cart page which displays the selected products

8. Click on Checkout button and observe the page displayed. Click Back button and observe navigation

9. Click on sort dropdown and observe sort functionality based on the options mentioned

10. Click on filter dropdown and observe filtering functionality based on the options mentioned

11. In search text box placed on second navigation bar, type the manufacturer name like samsung, apple etc. and observe the search functionality

12. Click on logout button at the top right corner and observe the redirection happening

Demo 2- Creating A Component

Highlights:

- Creating a component using Angular CLI
- Exploring the files created

Demo Steps:

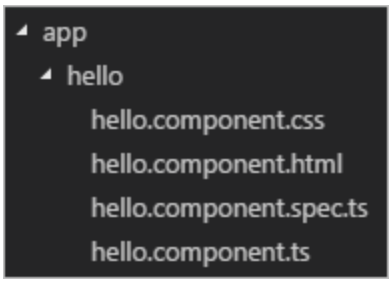
Problem Statement: Creating a new component called hello and rendering Hello Angular on the page as shown below

Hello Angular

1. In the same MyApp application created earlier, create a new component called hello using the following CLI command

   D:\MyApp>ng generate component hello

2. This command will create a new folder with name hello with the following files placed inside it



3. Open hello.component.ts file and create a property called courseName of type string and initialize it to "Angular" as shown below in Line number 8

```
1.  import { Component, OnInit } from '@angular/core';
2.  @Component({
3.          selector: 'app-hello',
4.          templateUrl: './hello.component.html',
5.          styleUrls: ['./hello.component.css']
6.    })
7.  export class HelloComponent implements OnInit {
8.         courseName: string = "Angular";
9.         constructor() { }
10.          ngOnInit() {
            }
           }
```

4. Open hello.component.html and display the courseName as shown below in Line 2

```
1.    <p>
2.      Hello {{ courseName }}
3.    </p>
```

5. Open hello.component.css and add the following styles for paragraph element

```
1.  p {
2.   color:blue;
3.  font-size:20px;
4.  }
```

6. Open app.module.ts file and add HelloComponent to bootstrap property as shown below in Line 9 to load it for execution

```
1.    import { NgModule } from '@angular/core';
2.    import { BrowserModule } from '@angular/platform-browser';
3.    import { AppComponent } from './app.component';
4.    import { HelloComponent } from './hello/hello.component';
5.    @NgModule({
6.        imports: [BrowserModule],
7.        declarations: [AppComponent, HelloComponent],
8.        providers: [],
9.        bootstrap: [HelloComponent]
10.    })
11.    export class AppModule { }
```

7. Open index.html and load hello component by using its selector name i.e., app-hello as shown below in Line 12

```
1.  <!doctype html>
2.      <html lang="en">
3.        <head>
4.            <meta charset="utf-8">
5.            <title>MyApp</title>
6.            <base href="/">
7.             <meta name="viewport" content="width=device-width, initial-scale=1">
8.             <link rel="icon" type="image/x-icon" href="favicon.ico">
9.        </head>
10.       <body>
11.           <app-hello> </app-hello>
12.       </body>
13.     </html>
```

8. Now run the application by giving the following command

```
D:\>ng serve –open
```

Demo 3 – **Templates**

Highlights:

- Using inline template
- Exploring the syntax of inline template

Demo Steps:

Consider HelloComponent created in the previous demo. Angular CLI has used external template option. Now let us use inline template option.

Problem Statement: Moving html code of HelloComponent to component class using inline template which should display the following output

Hello Angular

1. Copy the code from hello.component.html and paste it in hello.component.ts (Line 4 to 6)

```
1.  import { Component, OnInit } from '@angular/core';
2.      @Component({
3.          selector: 'app-hello',
4.          template:`
5.                      <p>Hello {{ courseName }}</p>
6.                  `,
7.           styleUrls: ['./hello.component.css']
8.                      })
9.    export class HelloComponent implements OnInit {
10.         courseName: string = "Angular";
11.         constructor() { }
12.         ngOnInit() { }
13.       }
```

2. Save the file and observe the output in the browser

Demo 4 – **Elements of Templates**

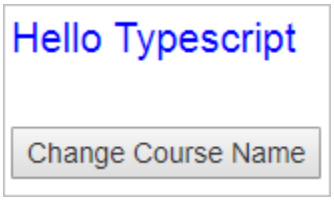
Highlights:

- Understanding the template elements
- Responding to user actions

Demo Steps:

Problem Statement: Adding a button to hello component template and when it is clicked, it should change the courseName as shown below

When button is clicked, it changes the course name Angular to Typescript as shown below



1. Open hello.component.ts, add a method called changeName() as shown below in Line 9-11

```
1.  import { Component } from '@angular/core';
2.      @Component({
3.          selector: 'app-hello',
4.          templateUrl: './hello.component.html',
5.          styleUrls: ['./hello.component.css']
6.      })
7.     export class HelloComponent implements OnInit {
8.         courseName: string = "Angular";
9.         changeName() {
10.            this.courseName = "TypeScript";
11.        }
12. }
```

2. Open hello.component.html and add a button and bind it with changeName() method as shown in Line 5

```
1. <p>
2. Hello {{ courseName }}
3. </p>
4. <br/>
5. <button (click)="changeName()">Change Course Name</button>
```

3. Save the files and check the output in the browser

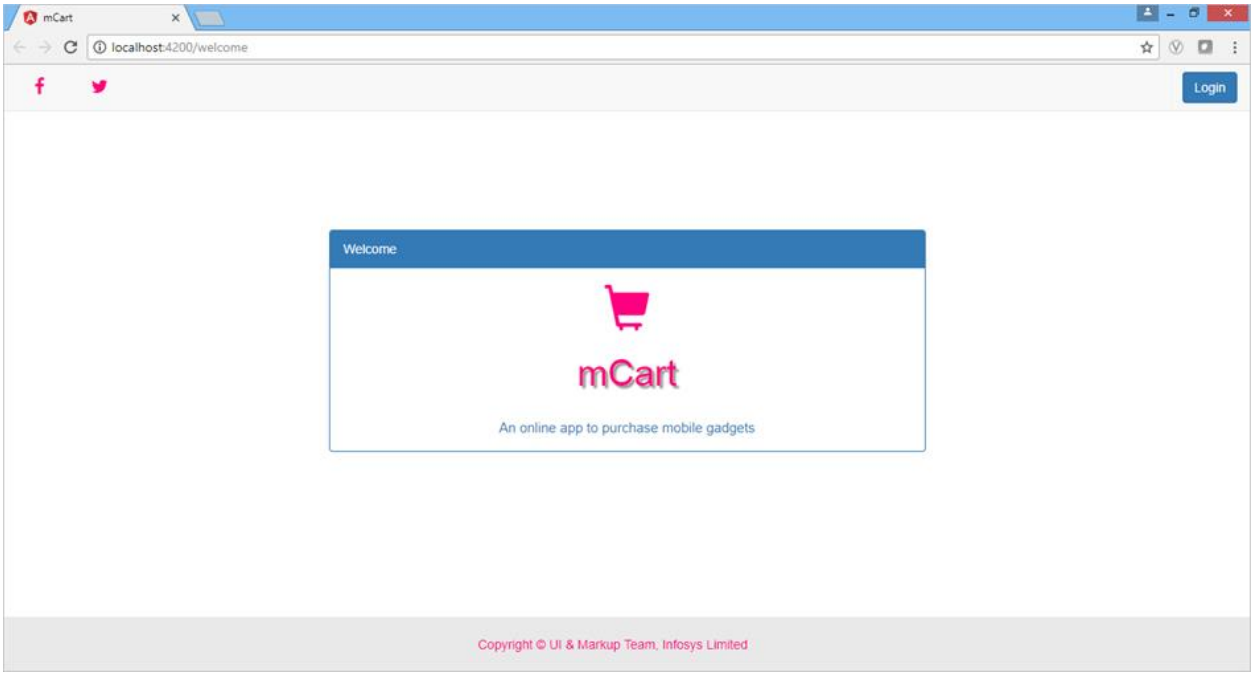
<u>Demo 5 – Welcome Component in mCart</u>

Highlights:

- Exploring the WelcomeComponent in mCart application
- Understanding the template and styling for WelcomeComponent

Demo Steps:

In our mCart application, we have a welcome screen as shown here,



- This welcome screen is created in the WelcomeComponent. Let us explore and understand the WelcomeComponent of our mCart application.
- We can find the files related with WelcomeComponent in welcome folder present inside the app folder (src --> app --> welcome).

1. Code for WelcomeComponent is present in the file welcome.component.ts, let us understand the below code present in the WelcomeComponent.

```
1  import { Component } from '@angular/core';
2
3  @Component({
4      templateUrl: 'welcome.component.html',
5      styleUrls: ['welcome.component.css']
6  })
7  export class WelcomeComponent {
8      public pageTitle: string = "Welcome";
9
10     constructor() {
11         document.getElementById("login").style.display = "";
12     }
13 }
```

Line 3-6: @Component to mark class as component and the component is binded with template and css file using templateUrl and styleUrls properties respectively
Line 8: We have created a property called pageTitle and initialized it to “welcome”
Line 11: This statement displays the login button at the top right corner of the page.

2. Code for WelcomeComponent template is present in our welcome.component.html file. Let us understand the template code given below.

```
1  <div class="container container-styles">
2      <div class="panel panel-primary">
3          <div class="panel-heading">{{pageTitle}}</div>
4          <div class="panel-body">
5              <div class="row">
6                  <span class="img-responsive center-block logo-styles"> <span
7                      class="glyphicon glyphicon-shopping-cart"> </span>
8                  </span>
9                  <div id="div1" class="shadow title-styles">
10                     mCart</div>
11             </div>
12             <br />
13             <div class="row">
14                 <div class="text-center text-styles">An online app to purchase mobile gadgets</div>
15             </div>
16         </div>
17     </div>
18 </div
```

Line 3 : We are rendering pageTitle property using interpolation
Line 6-8: Displays shopping cart symbol. We have used bootstrap CSS classes for this.
Line 9-10: Displays mCart title

3. Code for the styling of WelcomeComponent is present in the welcome.component.css file. Let us understand the code present below.

```
1  .shadow {
2      text-shadow: 3px 3px 2px rgba(150, 150, 150, 1);
3  }
4
5  .logo-styles{
6      width: 50px;
7      font-size: 50px;
8      color: #ff0080
9  }
10
11 .title-styles{
12     text-align: center;
13     color: #ff0080;
14     font-size: 40px
15 }
16
17 .text-styles{
18     color:#337ab7;
19     font-size: 15px
20 }
21
22 .container-styles{
23     position: relative;
24     top: 180px;
25     width:50%
26 }
```

Line 1-3: shadow class applies shadow effect to mCart text
Line 5-9: logo-styles class applies width, font-size and color properties to the shopping cart logo
Line 11-15: title-styles class applies the mentioned css properties to mcart text
Line 17-20: text-styles class applies the mentioned css properties to the description text rendered at the bottom of welcome component
Line 22-26: container-styles class applies the mentioned css properties to the entire container

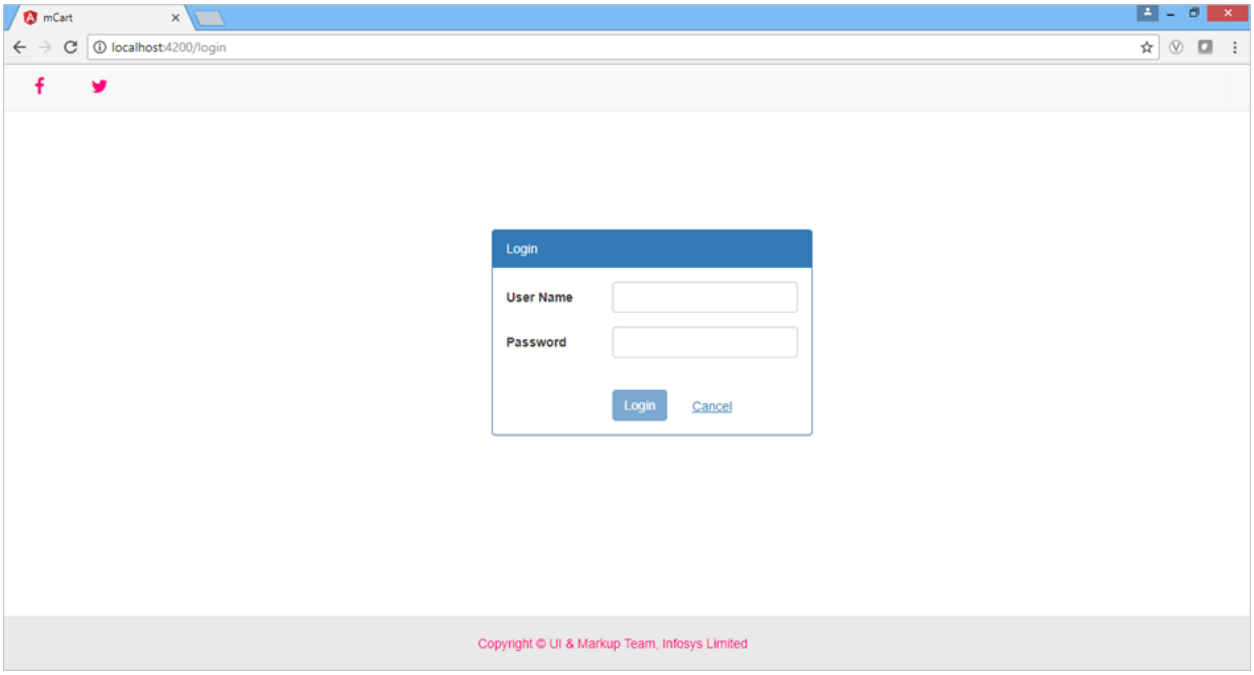
Demo 6 : LoginComponent in mCart

Highlights:

- Exploring the LoginComponent in mCart application
- Binding the components to root module of our application

Demo Steps:

- In our mCart application, we have a login screen as shown here



- This login screen is created in the LoginComponent. Let us explore and understand the LoginComponent of our mCart application.
- We can find the files related with LoginComponent in login folder present inside the app folder (src --> app --> login).

1. Code for the LoginComponent template is present in the **login.component.html** file. Let us understand the below given code of the LoginComponent template.

```html
1  <div class="container" style="position: relative; top: 180px;width:50%">
2      <div class="col-xs-7 col-xs-offset-3">
3          <div class="panel panel-primary">
4              <div class="panel-heading">Login</div>
5              <div class="panel-body padding">
6                  <form class="form-horizontal" #loginForm="ngForm">
7                      <div class="form-group">
8                          <label for="name" class="col-xs-4 control-label">User Name</label>
9                          <div class="col-xs-7">
10                             <input type="text" class="form-control" name="username" [(ngModel)]="login.userName" #user
11                             <div class="alert alert-danger" *ngIf="username.touched && !username.valid">UserName is re
12                         </div>
13                     </div>
14                     <div class="form-group">
15                         <label for="password" class="col-xs-4 control-label">Password</label>
16                         <div class="col-xs-7">
17                             <input type="password" class="form-control" name="password" #password="ngModel" [(ngModel)
18                             <div *ngIf="password.touched && !password.valid" class="alert alert-danger">Password is re
19                         </div>
20                     </div>
21
22                     <div *ngIf="!valid" class="error">Invalid Credentials...Please try again...</div>
23                     <br />
24
25                     <div class="form-group">
```

```html
25                     <div class="form-group">
26                         <span class="col-xs-4"></span>
27                         <div class="col-xs-3">
28                             <button (click)="onSubmit()" class="btn btn-primary" [disabled]="!loginForm.form.valid">Lo
29                         </div>
30                         <span class="col-xs-5" style="top:8px">
31                             <a [routerLink]="['/welcome']" style="color:#337ab7;text-decoration: underline;">Cancel</a
32                             </span>
33                     </div>
34                 </form>
35             </div>
36         </div>
37     </div>
38 </div>
```

Line 6-20: We have created a form with two labels and two textboxes for username and password

Note: In the code snippet (Line nos: 10,11,17 and 18), we have some additional code for data binding and validations which will be explained in subsequent demos.

Line 28: A login button is created and click event is binded with onSubmit() method. onSubmit() has the code to check the credentials validity.

Line 31: A hyperlink for cancelling the operation and navigating back to welcome component

Note: In the code snippet (Line no: 31), we have some additional code for routing which will be explained in subsequent demos.

2. LoginComponent has a model class coded in the login.ts, below is the code for it.

```
1  export class Login {
2          public userName: string;
3          public password: string;
4  }
```

Line 1-4: Login class has two properties username and password to store the values entered by the user in Login form.

3. Code for LoginComponent is coded in the file login.component.ts, let us understand the below code present for the LoginComponent.

```
1  import { Component } from '@angular/core';
2  import { Router } from '@angular/router';
3
4  import { Login } from './Login';
5  import { LoginService } from './login.service';
6
7  @Component({
8      templateUrl: 'login.component.html',
9      styleUrls: ['./login.component.css'],
10     providers: [LoginService]
11 })
12 export class LoginComponent {
13
14     login = new Login();
15
16     ...
17     onSubmit() {
18         //logic for validation
19     }
20 }
```

Line 4: Importing Login class from login.ts

Line 7-8: Component decorator marks the class as component and templateUrl to bind html page to Login component

Line 14: We have created an instance of Login class

4. Finally we need to bind both the components of Welcome and Login in the root module.

5. This is done in the module file of our application i.e. app.module.ts file, it contains the below given code. Let us try to understand it.

```
1  import { NgModule } from '@angular/core';
2  import { BrowserModule } from '@angular/platform-browser';
3  import { HttpClientModule } from '@angular/common/http';
4  import { FormsModule } from '@angular/forms';
5
6  import { AppComponent } from './app.component';
7  import { AppRoutingModule } from './app-routing.module';
8  import { WelcomeComponent } from './welcome/welcome.component';
9  import { LoginComponent } from './login/login.component';
10
11
12 @NgModule({
13     imports: [BrowserModule, HttpClientModule, FormsModule,  AppRoutingModule],
14     declarations: [AppComponent, WelcomeComponent,  LoginComponent],
15     providers: [],
16     bootstrap: [AppComponent]
17 })
18 export class AppModule { }
```

Line 8-9: Import WelcomeComponent and LoginComponent classes

Line 14: Include them in the declarations property of NgModule decorator to make them available in the entire module

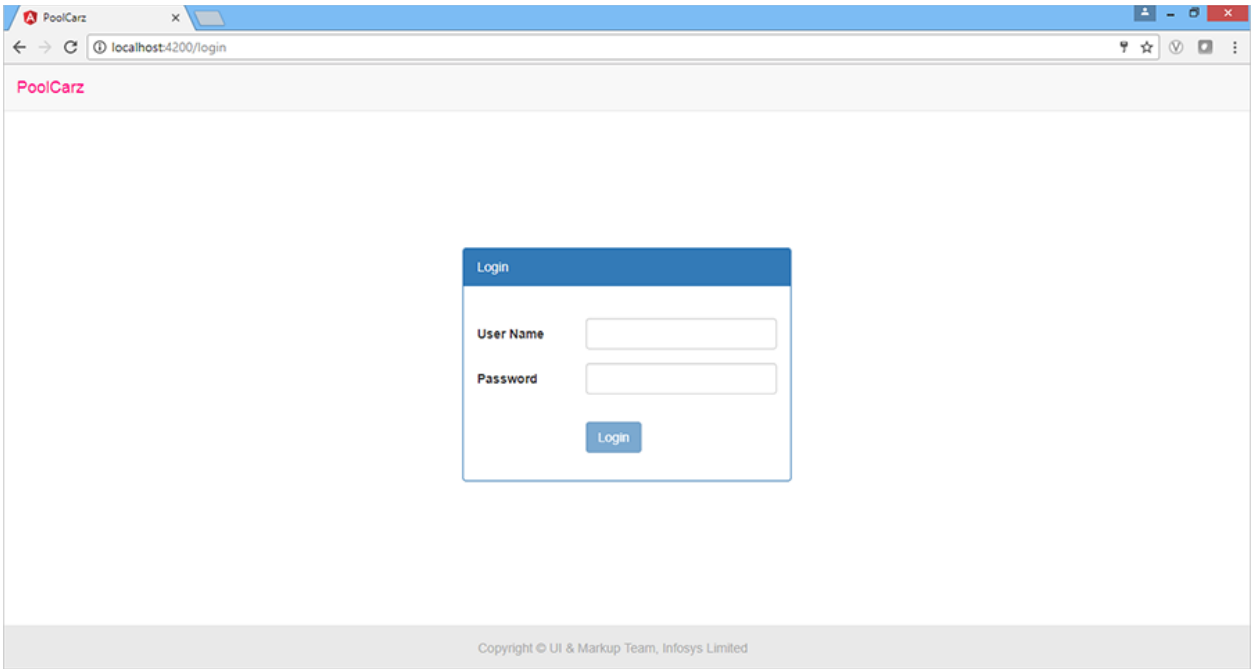
<u>Exercise 2</u> – **<u>Components in Poolcarz application</u>**

Time Limit: 20 Minutes

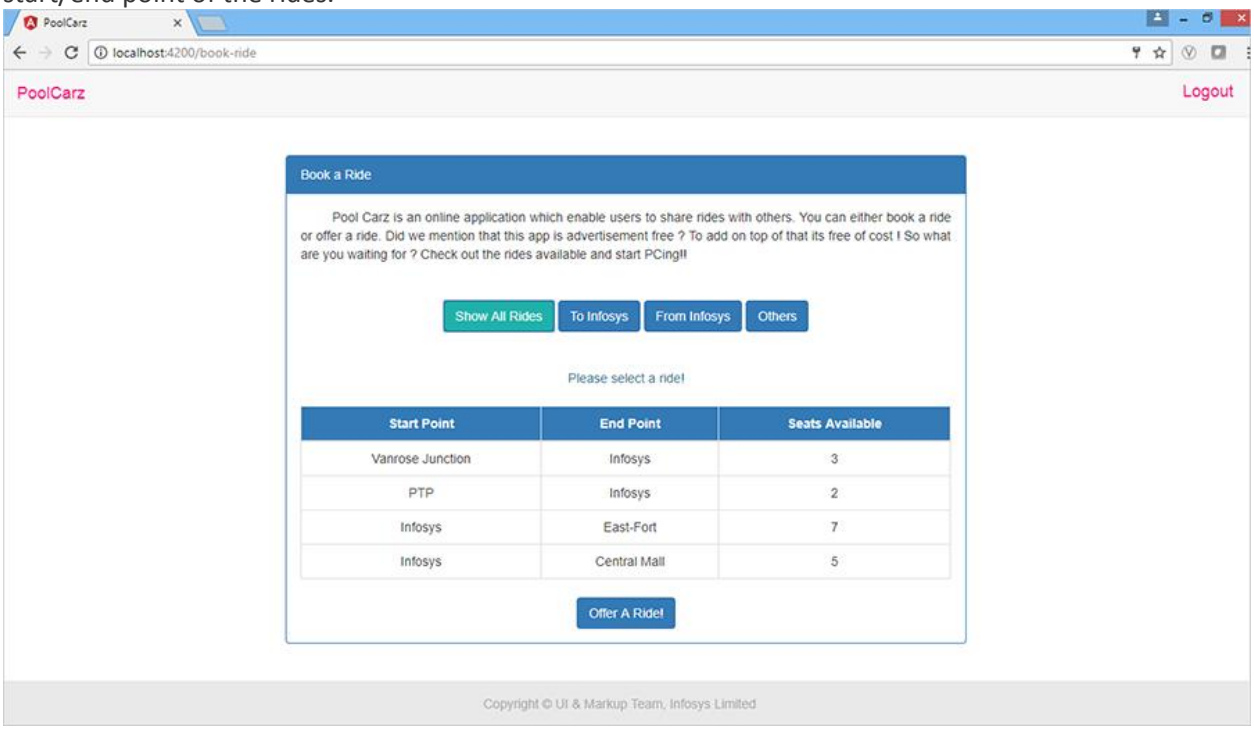
Problem Statement

- We will be progressively building Pool Carz application throughout this course.
- PoolCarz is a web application for car pooling. The application allows users to share ride with others. User can either book a ride or offer a ride.
- Use Cases:
    - Login
    - Book a ride
    - Ride details
    - Offer Ride
    - Logout

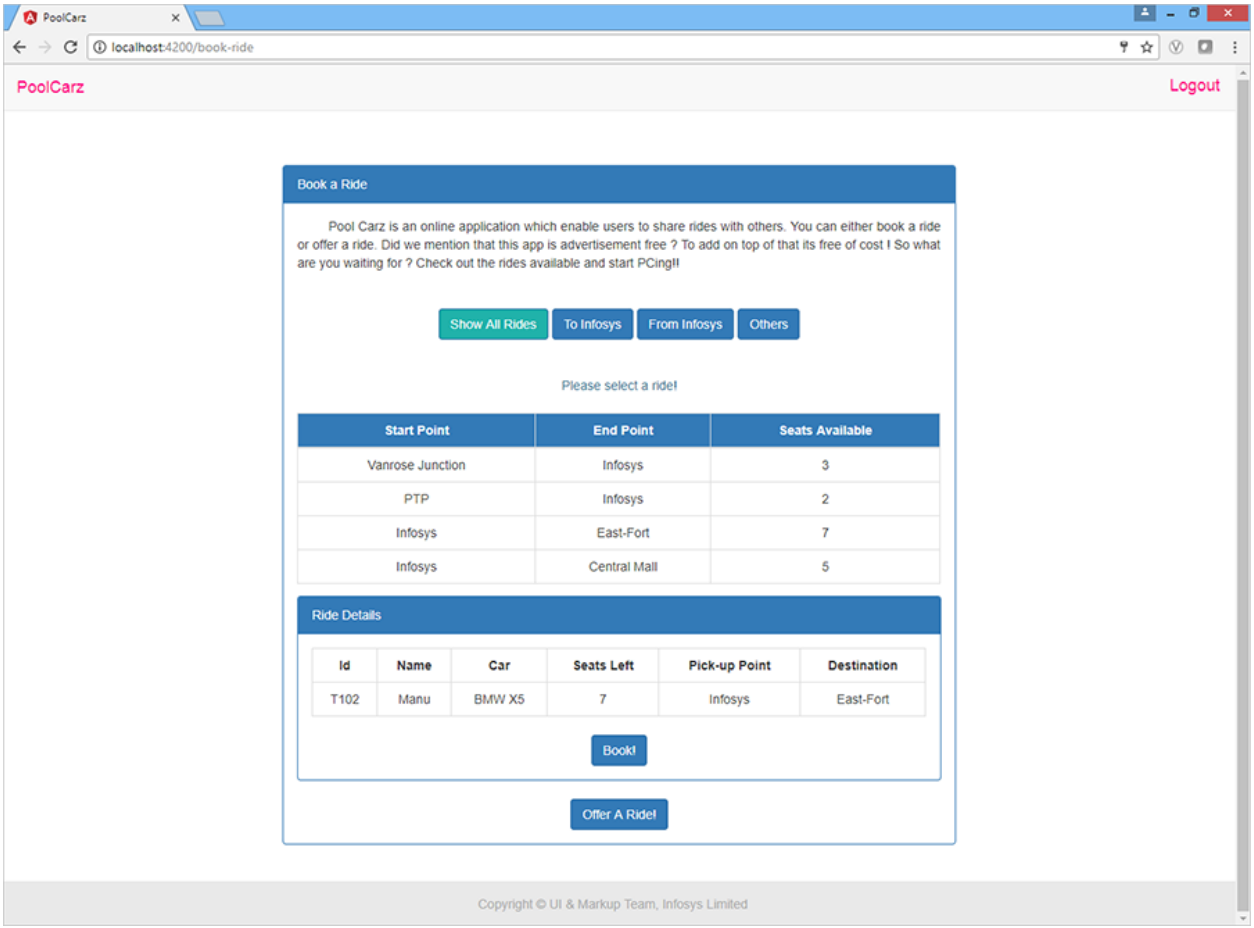
1. **Login**: Login to the application to view ride details



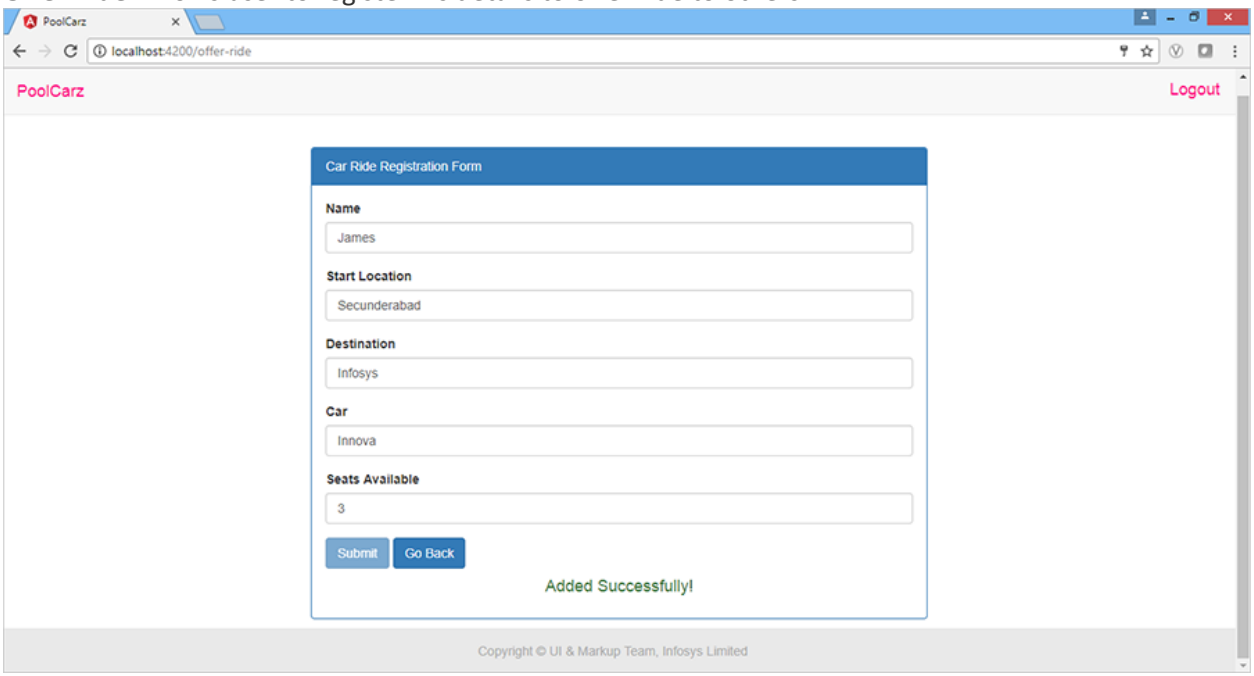
2. **Book a ride**: Renders the rides available and also allows to filter the details based on the start/end point of the rides.



3. **Ride Details**: when user selects a particular ride, it renders complete details about that ride and allows user to book that ride

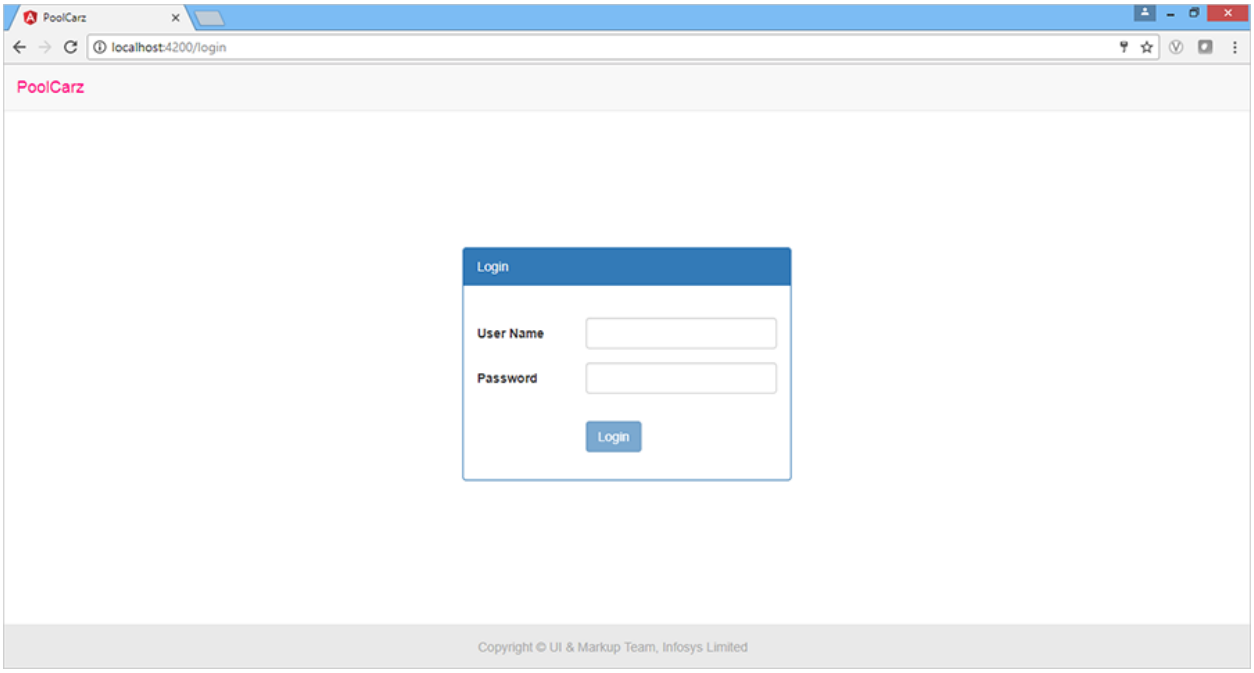


4. **Offer Ride**: Allows user to register his details to offer ride to others



- As a first step, create the folder structure for the application.
- Create a new application named 'PoolCarz' using Angular CLI tool

- Create Login component using Angular CLI
- Write the login template code in login.component.html to render the below screen



Demo 7 – ngIf

Highlights:

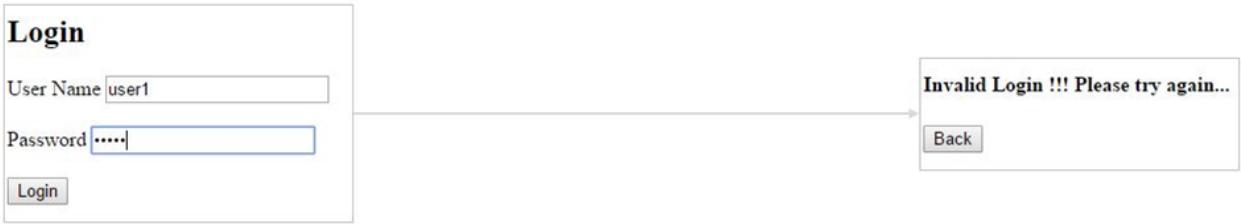
- Understanding ngIf directive
- Invoking a directive on user's action

Demo Steps:

Problem Statement: Create a login form with username and password fields. If user enters correct credentials, it should render "Welcome <<username>>" message otherwise it should render "Invalid Login!!!Please try again..." message as shown below

After entering correct credentials and clicking on Login button

After entering incorrect credentials and clicking on Login button



1. Let us write code in app.component.ts. Open app.component.ts and write the following code:-

```
import { Component } from '@angular/core';
@Component({
        selector: 'app-root',
        templateUrl: './app.component.html',
        styleUrls: ['./app.component.css']
})
export class AppComponent {
        isAuthenticated: boolean;
        submitted: boolean = false;
        userName: string;
        onSubmit(name: string, password: string) {
            this.submitted = true;
            this.userName = name;
            if (name === "admin" && password === "admin")
              this.isAuthenticated = true;
            else
              this.isAuthenticated = false;
  } }
```

2. Write the below given code in app.component.html

```
<div *ngIf="!submitted">
    <form>
      <label>User Name</label>
          <input type="text" #username><br/><br/>
       <label for="password">Password</label>
           <input type="password" name="password" #password><br/>
    </form>
    <button (click)="onSubmit(username.value,password.value)">Login</button>
</div>
<div *ngIf="submitted">
     <div *ngIf="isAuthenticated; else failureMsg">
          <h4> Welcome {{userName}} </h4>
</div>
<ng-template #failureMsg>
        <h4> Invalid Login !!! Please try again...</h4>
</ng-template>
<button type="button" (click)="submitted=false">Back</button>
```

18.

3. Add AppComponent to the bootstrap property in the root module file i.e., app.module.ts

```
import { NgModule } from '@angular/core';

import { BrowserModule } from '@angular/platform-browser';

import { AppComponent } from './app.component';

    @NgModule({

        imports: [BrowserModule],

        declarations: [AppComponent],

        providers: [],

        bootstrap: [AppComponent]

    })

  export class AppModule { }
```

4. Save the files and check the output in the browser

<u>Demo 8</u> : **<u>ngFor</u>**

Highlights:

- Understanding ngFor directive
- Iteration of an array using ngFor directive

Demo Steps:

Problem Statement: Creating a courses array and rendering it in the template using ngFor direcctive in a list format as shown below

- 0 - TypeScript
- 1 - Angular
- 2 - Node JS
- 3 - TypeScript

1. Write the below given code in app.component.ts

```
import { Component } from '@angular/core';

@Component({
 selector: 'app-root',
 templateUrl: './app.component.html',
 styleUrls: ['./app.component.css']
})
```

```
export class AppComponent {
  courses: any[] = [
    { id: 1, name: "TypeScript" },
    { id: 2, name: "Angular" },
    { id: 3, name: "Node JS" },
    { id: 1, name: "TypeScript" }
  ];
}
```

2. Write the below given code in app.component.html

```
<ul>
  <li *ngFor="let course of courses;  let i = index">
      {{i}} - {{ course.name }}
  </li>
</ul>
```

3. Save the files and check the output in the browser

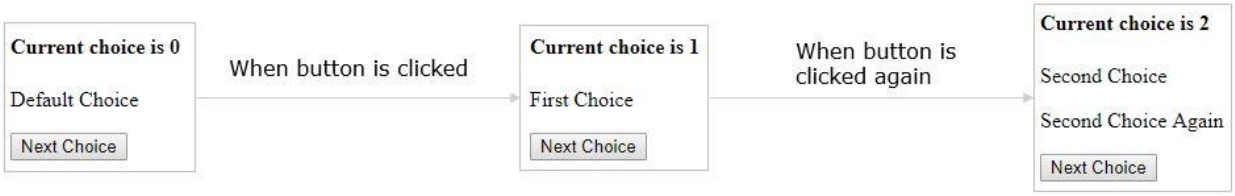
<u>Demo 9 – ngSwitch</u>

Highlights:

- Understanding ngSwitch directive
- Invoking a directive on user's action

Demo Steps:

Problem Statement: Displaying the correct option based on the value passed to ngSwitch directive as shown below



1. Write the below given code in app.component.ts

```
import { Component } from '@angular/core';

@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent {
  value: number = 0;
```

```
  nextChoice() {
    this.value++;
  }
}
```

2. Write the below given code in app.component.html

```
<h4>
  Current choice is {{ value }}
</h4>

<div [ngSwitch]="value">
  <p *ngSwitchCase="1">First Choice</p>
  <p *ngSwitchCase="2">Second Choice</p>
  <p *ngSwitchCase="3">Third Choice</p>
  <p *ngSwitchCase="2">Second Choice Again</p>
  <p *ngSwitchDefault>Default Choice</p>
</div>

<div>
  <button (click)="nextChoice()">
        Next Choice
    </button>
</div>
```

3. Save the files and check the output in the browser

<u>Demo 10 : Custom Structural Directive</u>

Highlights:

- Understanding custom directives
- Creating and using a custom directive

Demo Steps:

Problem Statement: Create a custom structural directive called 'repeat' which should repeat the element given number of times as shown below

Structural Directive

I am being repeated...

I am being repeated...

I am being repeated...

I am being repeated...

I am being repeated...

1. Generate a directive called 'repeat' using the following command

   D:\MyApp>ng generate directive repeat

2. Write the below given code in app.module.ts

```
import { BrowserModule } from '@angular/platform-browser';
import { NgModule } from '@angular/core';

import { AppComponent } from './app.component';
import { RepeatDirective } from './repeat.directive';

@NgModule({
  declarations: [
    AppComponent,
    RepeatDirective
  ],
  imports: [
    BrowserModule
  ],
  providers: [],
  bootstrap: [AppComponent]
})
export class AppModule { }
```

3. Open repeat.directive.ts file and add the following code

```
import { Directive, TemplateRef, ViewContainerRef, Input } from
'@angular/core';

@Directive({
  selector: '[appRepeat]'
})
export class RepeatDirective {

  constructor(private _templateRef: TemplateRef<any>, private _viewContainer:
ViewContainerRef) { }
```

```
  @Input() set appRepeat(count: number) {
    for (var i = 0; i < count; i++) {
      this._viewContainer.createEmbeddedView(this._templateRef);
    }
  }
}
```

4. Write the below given code in app.component.html

```
<h3>Structural Directive</h3>
<p *appRepeat="5">I am being repeated...</p>
```

5. Save the files and check the output in the browser

<u>Demo 11 : ngStyle</u>

Highlights:

- Understanding attribute directives
- Using ngStyle directive

Demo Steps:

Problem Statement: Apply multiple css properties to a paragraph in a component using ngStyle. Output should be as shown below

1. Write the below given code in app.component.ts

```
import { Component } from '@angular/core';


@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent {
  colorName: string = 'red';
  fontWeight: string = 'bold';
  borderStyle: string = '1px solid black';
```

```
}
```

2. Write the below given code in app.component.html

```
<p [ngStyle]="{
        color:colorName,
        'font-weight':fontWeight,
        borderBottom: borderStyle
      }">
  Demo for attribute directive ngStyle
</p>
```

<u>Demo 12 : ngClass</u>

Highlights:

- Understanding attribute directive
- Using ngClass directive to apply css classes

Demo Steps:

Problem Statement: Applying multiple css classes to the text using ngClass directive. Output should be as shown below

1. Write the below given code in app.component.ts

```
import { Component } from '@angular/core';


@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent {
```

```
  isBordered: boolean = true;
}
```

2. Write the below given code in app.component.html

```
<div [ngClass]="{bordered: isBordered}">
  Border {{ isBordered ? "ON" : "OFF" }}
</div>
```

3. In app.component.css, add the following css class

```
.bordered {
    border: 1px dashed black;
    background-color: #eee;
}
```

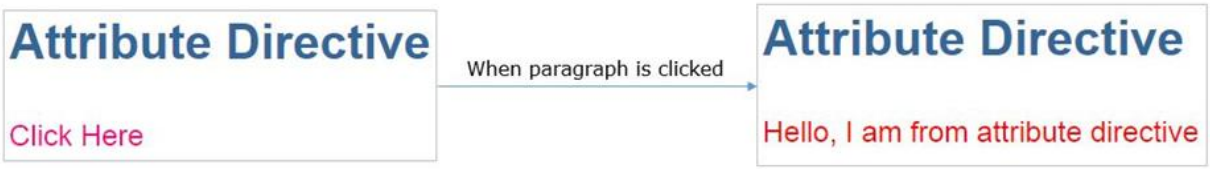
<u>Demo 13 : Custom Attribute Directive</u>

Highlights:

- Creating a custom attribute directive
- Applying custom attribute directive to the element

Demo Steps:

Problem Statement: Create an attribute directive called 'showMessage' which should display the given message in a paragraph when user clicks on it and should change the text color to red. Below is the output



1. Generate a directive called 'message' using the following command

```
D:\MyApp>ng generate directive message
```

2. Above command will add MessageDirective class to the declarations property in app.module.ts file

```
import { BrowserModule } from '@angular/platform-browser';
import { NgModule } from '@angular/core';
```

```
import { AppComponent } from './app.component';
import { MessageDirective } from './message.directive';
@NgModule({
    declarations: [
        AppComponent,
        MessageDirective
        ],
    imports: [
        BrowserModule
        ],
    providers: [],
        bootstrap: [AppComponent]
})
export class AppModule { }
```

3. Open message.directive.ts file and add the following code

```
import { Directive, ElementRef, Renderer2, HostListener, Input } from
'@angular/core';
@Directive({
  selector: '[appMessage]'
})
export class MessageDirective {
    @Input('appMessage') message: string;
    constructor(private el: ElementRef, private renderer: Renderer2) {
        renderer.setStyle(el.nativeElement, 'cursor', 'pointer');
    }
    @HostListener('click') onClick() {
        this.el.nativeElement.innerHTML = this.message;
```

```
            this.renderer.setStyle(this.el.nativeElement, 'color', 'red');
        }
    }
```

4. Write the below given code in app.component.html

```
<h3>Attribute Directive</h3>
<p [appMessage]="myMessage">Click Here</p>
```

5. Add the following CSS styles to app.component.css file

```
h3 {
  color: #369;
  font-family: Arial, Helvetica, sans-serif;
  font-size: 250%;
}
p {
  color: #ff0080;
  font-family: Arial, Helvetica, sans-serif;
  font-size: 150%;
}
```

6. Add the following code in app.component.ts

```
import { Component } from '@angular/core';
@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent {
```

```
myMessage: string = "Hello, I am from attribute directive";
}
```

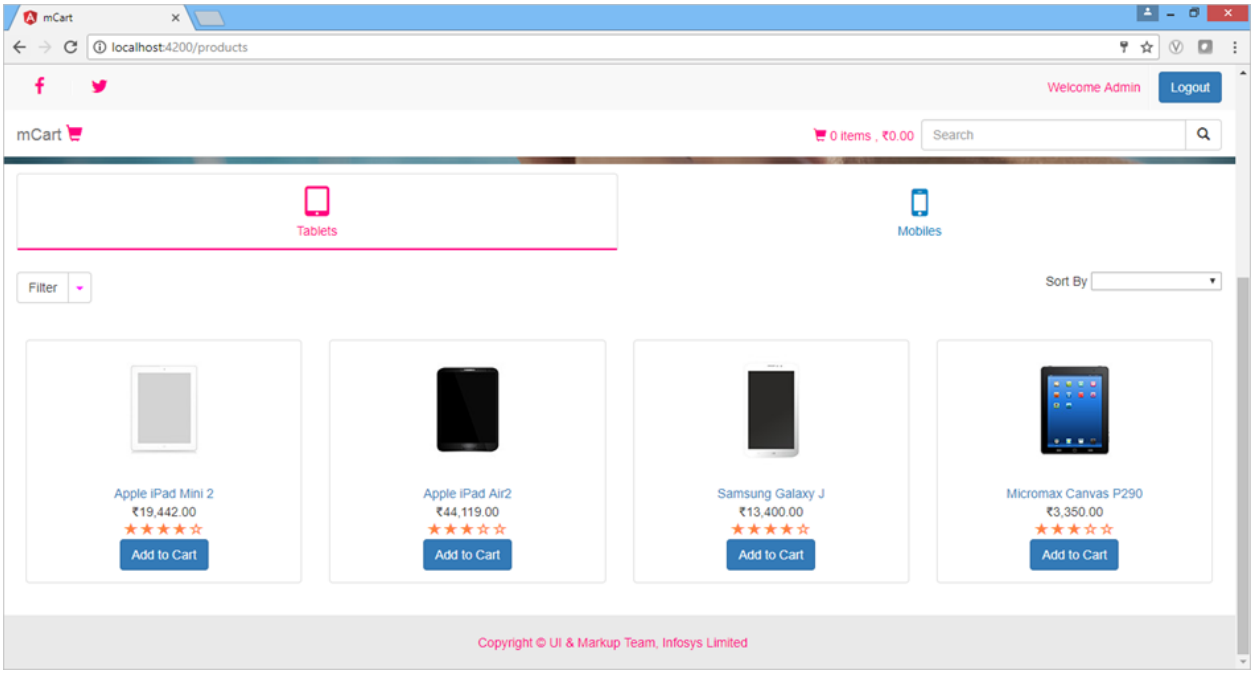
Demo 14 : Directives in mCart Application

Highlights:

- Understanding directives in mCart application
- Exploring the ProductListComponent in mCart

Demo Steps:

Now let us explore one more component called ProductListComponent, screen for which is shown here,



- Let us explore and understand the ProductListComponent of our mCart application.
- We can find the files related with ProductListComponent in product-list folder present inside the Products folder.

Code for the ProductListComponent template is present in the product-list.component.html file. Let us understand the below given code of the ProductListComponent template.

```
1. <nav class='navbar navbar-default navbar-fixed-top navbarpos'>
3. <div class='container-fluid'>
4. <a class='navbar-brand txtcolor'>{{pageTitle}} <span
5. class="glyphicon glyphicon-shopping-cart txtcolor"></span></a>
6. <div class="input-group pull-right col-md-3 searchboxpos">
7. <input type="text" class="form-control" placeholder="Search" name="q"
   [(ngModel)]="listFilter" (change)="searchtext()">
```

```
8.  <div class="input-group-btn">
9.  <button class="btn btn-default">
10. <i class="glyphicon glyphicon-search"></i>
11. </button>
12. </div>
13. </div>


14. <div class="pull-right txtcolor cartpos">
15. <span class="glyphicon glyphicon-shopping-cart"> <a
16. [routerLink]="['cart']" class="txtcolor">{{selectedItems}} items</a></span>
17. <span>, {{total | currency:'INR':'symbol':'1.2-2'}} </span>
18. </div>
19. </div>
20. </nav>
21. <br />
22. <br />
23. <div class="container" class="carouselpos">
24. <div id="carousel-example-generic" class="carousel slide carouselheight" data-ride="carousel"
    data-interval="3000">
25. <ol class="carousel-indicators">
26. <li data-target="#carousel-example-generic" data-slide-to="0" class="active"></li>
27. <li data-target="#carousel-example-generic" data-slide-to="1"></li>
28. <li data-target="#carousel-example-generic" data-slide-to="2"></li>
29. </ol>
30. <div class="carousel-inner">
31. <div class="item active">
32. <img src="assets/imgs/carousel_smart_phone.jpg" alt="First slide" style="min-
    width:100%;height:350px;">
33. </div>
34. <div class="item carouselimgpos">
35. <img src="assets/imgs/carousel1.jpg" alt="Second slide" style="min-
    width:100%;height:350px;">
36. </div>
37. <div class="item">
38. <img src="assets/imgs/tablet_blue_stylus.jpg" alt="Third slide" style="min-
    width:100%;height:350px;">
39. </div>
40. </div>
41. <a class="left carousel-control" href="#carousel-example-generic" role="button" data-
    slide="prev">
42. <span class="glyphicon glyphicon-chevron-left"></span>
43. </a>
```

```
44. <a class="right carousel-control" href="#carousel-example-generic" role="button" data-
    slide="next">
45. <span class="glyphicon glyphicon-chevron-right"></span>
46. </a>
47. </div>
48. <div class='panel with-nav-tabs panel-primary noborder'>
49. <div class='panel-heading noborder bgcolor'>
50. <ul class="nav nav-tabs noborder">
51. <li class="active tabpos"><a href="#tabprimary" (click)="tabselect('tablet')" data-
    toggle="tab"><i
52. class="fa fa-tablet fa-3x" aria-hidden="true"></i>
53. <div>Tablets</div></a></li>
54. <li class="tabpos"><a (click)="tabselect('mobile')" href="#tabprimary" data-toggle="tab"><i
55. class="fa fa-mobile fa-3x" aria-hidden="true"></i>
56. <div>Mobiles</div></a></li>
57. </ul>
58. </div>
59. <div class='panel-body'>
60. <div class="tab-content">
61. <div class="tab-pane fade in active" id="tabprimary">
62. <div class="btn-group">
63. <button type="button" class="btn btn-default">Filter</button>
64. <button type="button" class="btn btn-default dropdown-toggle" data-toggle="dropdown">
65. <span class="caret"></span> <span class="sr-only">Toggle
66. Dropdown</span>
67. </button>
68. <ul class="dropdown-menu multi-column columns-3 noclose">
69. <div class="row vdivide">
70. <div class="col-md-4">
71. <ul class="multi-column-dropdown noclose">
72. <h4>Manufacturer</h4>
73. <li *ngFor="let manufac of manufacturers">
74. <input type="checkbox" [ngModel]="manufac.checked" (change)="filter(manufac)"> <label>
75. {{manufac.id}} </label></li>
76. </ul>
77. </div>
78. <div class="col-md-4">
79. <ul class="multi-column-dropdown noclose">
80. <h4>OS</h4>
81. <li *ngFor="let ostypes of os">
82. <input type="checkbox" [ngModel]="ostypes.checked" (change)="filter(ostypes)">
83. <label> {{ostypes.id}}</label></li>
84. </ul>
85. </div>
```

```
<div class="col-md-4">
<ul class="multi-column-dropdown noclose">
<h4>Price Range</h4>
<li *ngFor="let price of price_range">
<input type="checkbox" [ngModel]="price.checked" (change)=filter(price)> <label>{{ price.id}}
</label></li>
</ul>
</div>
</div>
</ul>
</div>
<span *ngIf="chkmanosprice.length> 0"> {{products.length}}
results</span>


<div class="pull-right">
<span>Sort By </span>
<select [ngModel]="sortoption" (change)="onChange($event.target.value)">
<option value="popularity">Popularity</option>
<option value="pricelh">Price - Low to High</option>
<option value="pricehl">Price - High to Low</option>
</select>
</div>
<div *ngIf='products && products.length'>
<div class="row" *ngFor='let product of products | orderBy:sortoption ; let i = index'
[hidden]="(i%4)>0">
<div class="col-xs-3">
<span class="thumbnail text-center">
<div>
<img [src] ='product.imageUrl' [title]='product.productName'
[style.width.px]='imageWidth' [style.height.px]='imageHeight'
[style.margin.px]='imageMargin'>
</div>
<div class="caption">
<div>
<a [routerLink]="[product.productId]" >
{{product.productName}} </a>
</div>
<div>{{ product.price | currency:'INR':'symbol':'1.2-2'}}</div>
<div></div>
<ratings  class="ratingcolor" [rate]='product.rating'></ratings>
<div>
<button (click)="addCart(product.productId)"
class="btn btn-primary">Add to Cart</button>
```

```
125. </div>
126. </div>
127. </span>
128. ... //rest of the code
129. </div>
130. </div>
```

Line 1: Displays a navigation bar at the top of the page. This is the second navigation bar displayed

Line 4: Displays pageTitle property value on top of navigation bar

Line 7-8: Displays a text box for search functionality. We can search for a product based on its manufacturer

Line 16: Renders selectedItems property which holds the number of selected items to purchase

Line 17: Renders total price of the selected items

Line 23-29: Renders a carousel

Line 30-40: Adds three images to the carousel

Line 41-47: Renders left and right arrows on the carousel for navigating between the images

Line 50-57: Renders two tabs called Tablets and Mobiles to show tablet and mobile devices accordingly

Line 106: We have used built-in structural directive called ngIf to display products on the page only if products property has data inside it

Line 107: Another structural directive called ngFor is used to run a loop on products array

Line 109-127: Renders a product details such as product image, product Name as hyperlink, price, rating and a button called add to cart

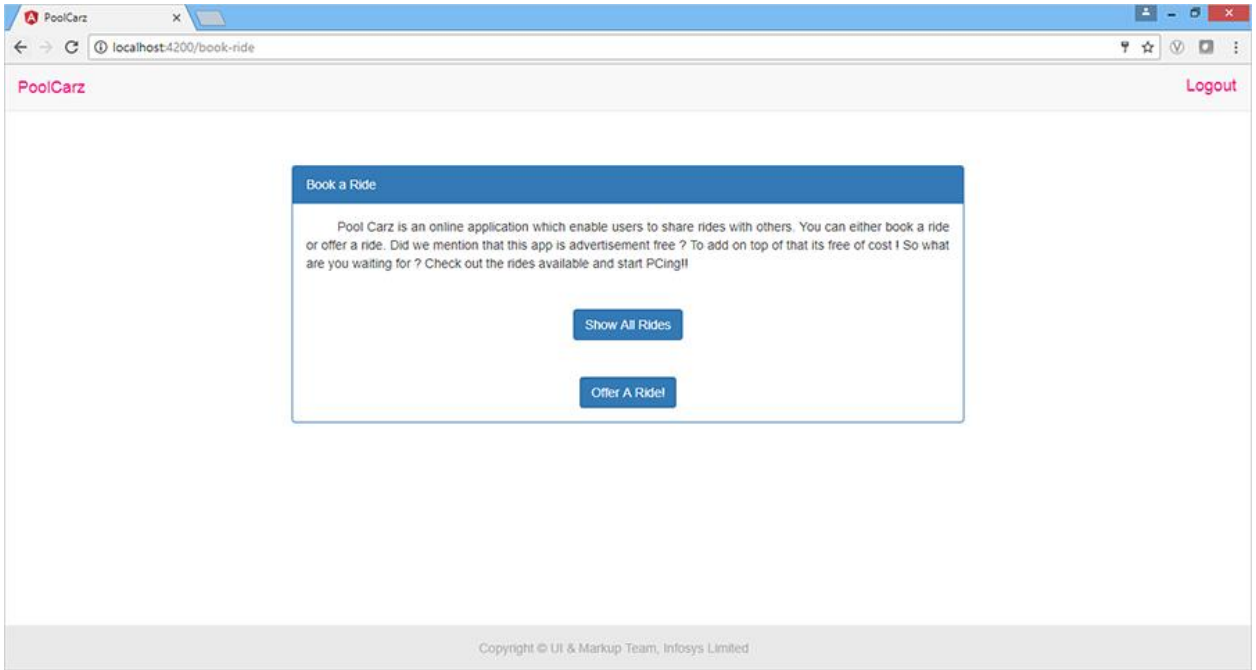
Line 112: We have used style binding to bind css properties called width, height and margin with imageWidth, imageHeight and imageMargin properties of ProdutListComponent

<u>Exercise 3 - Using Directives in Pool Carz Application</u>

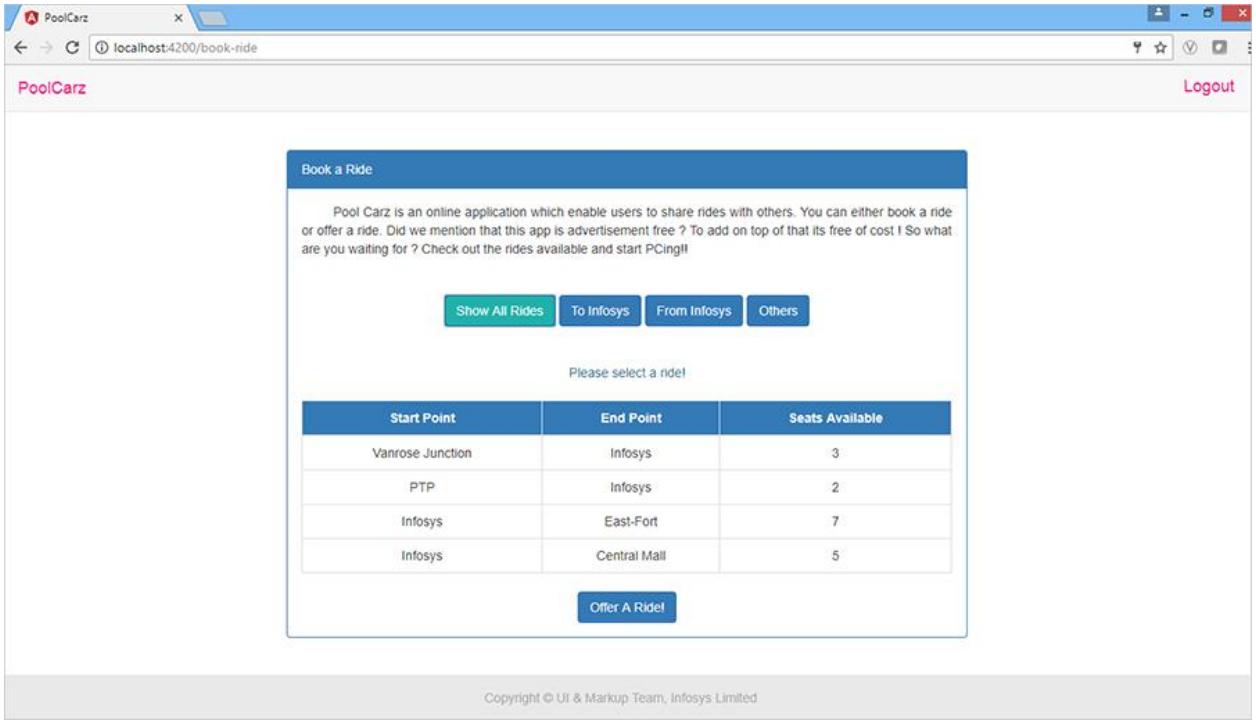
Time Limit: 20 Minutes

Problem Statement

- Creating a BookRideComponent in PoolCarz application as shown below

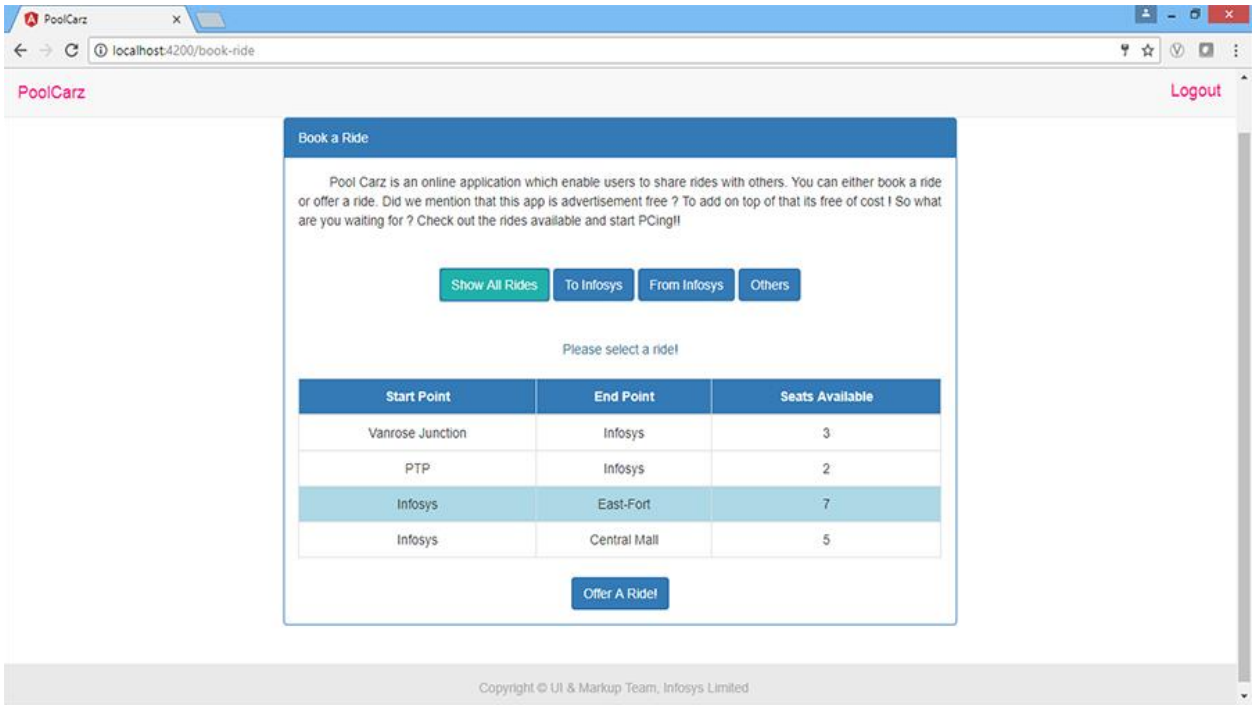


- Create BookRideComponent using Angular CLI
- Write the template code in book-ride.component.html which should display the output as shown above
- When the button "Show All Rides" is clicked, it should render the following output



- Display three buttons and a table followed by another button as shown above
- When user again clicks "Show All Rides" button, it should hide the fours buttons and table rendered. This button should act like toggle button to display and hide alternatively

- Store the ride details in an array and render it in a table as shown above. Array should contain the following fields in each object
    - id – number
    - offerId – string
    - name – string
    - car – string
    - seatsLeft – number
    - pickUp – string
    - destination – string
- Create a custom attribute directive called MouseHoverDirective and apply it on the table rows. When user hovers mouse on any row, it should highlight the row in green color and when mouse is removed from a row, the color should be removed as shown below



<u>Demo 15 : Property Binding</u>

Highlights:

- Understanding Property Binding
- Binding element property with class property

Demo Steps:

Problem Statement: Binding image with class property using property binding. Output is shown below

1. Write the following code in app.component.ts as shown below

```
import { Component } from '@angular/core';
@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent {
  imgUrl: string = 'assets/imgs/logo.jpg';
}
```

2. Write the following code in app.component.html as shown below

```
<img [src]='imgUrl' width=200 height=100>
```

3. Save the files and check the output in the browser

<u>Demo 16 : Attribute Binding</u>

Highlights:

- Understanding attribute directive
- Setting value of an attribute

Demo Steps:

Problem Statement: Binding colspan attribute of a table element to class property to display the following output

First		Second
Third	Fourth	Fifth

1. Write the below given code in app.component.ts

```
import { Component } from '@angular/core';


@Component({
```

```
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent {
  value: string ="2";
}
```

2. Write the below given code in app.component.html

```
<table border=1>
  <tr>
    <td [attr.colspan]="value"> First </td>
    <td>Second</td>
  </tr>
  <tr>
    <td>Third</td>
    <td>Fourth</td>
    <td>Fifth</td>
  </tr>
</table>
```

3. Save the files and check the output in the browser

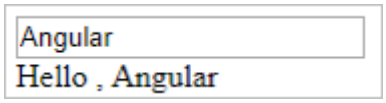
<u>Demo 17 : Two Way Data Binding</u>

Highlights:

- Working with two way data binding
- Updating the element in action

Demo Steps:

Problem Statement: Binding a textbox with a property using two way data binding. Output is as shown below



1. Write the below given code in app.component.ts

```
import { Component } from '@angular/core';


@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent {
  name: string = "Angular";
}
```

2. Write the below given code in app.component.html

```
<input type="text" [(ngModel)]="name"> <br/>
<div>Hello , {{ name }}</div>
```

3. Write the below given code in app.module.ts

```
import { BrowserModule } from '@angular/platform-browser';
import { NgModule } from '@angular/core';
import { FormsModule } from '@angular/forms';

import { AppComponent } from './app.component';
```

```
@NgModule({
  declarations: [
    AppComponent
  ],
  imports: [
    BrowserModule,
    FormsModule
  ],
  providers: [],
  bootstrap: [AppComponent]
})
export class AppModule { }
```

4. Save the files and check the output in the browser

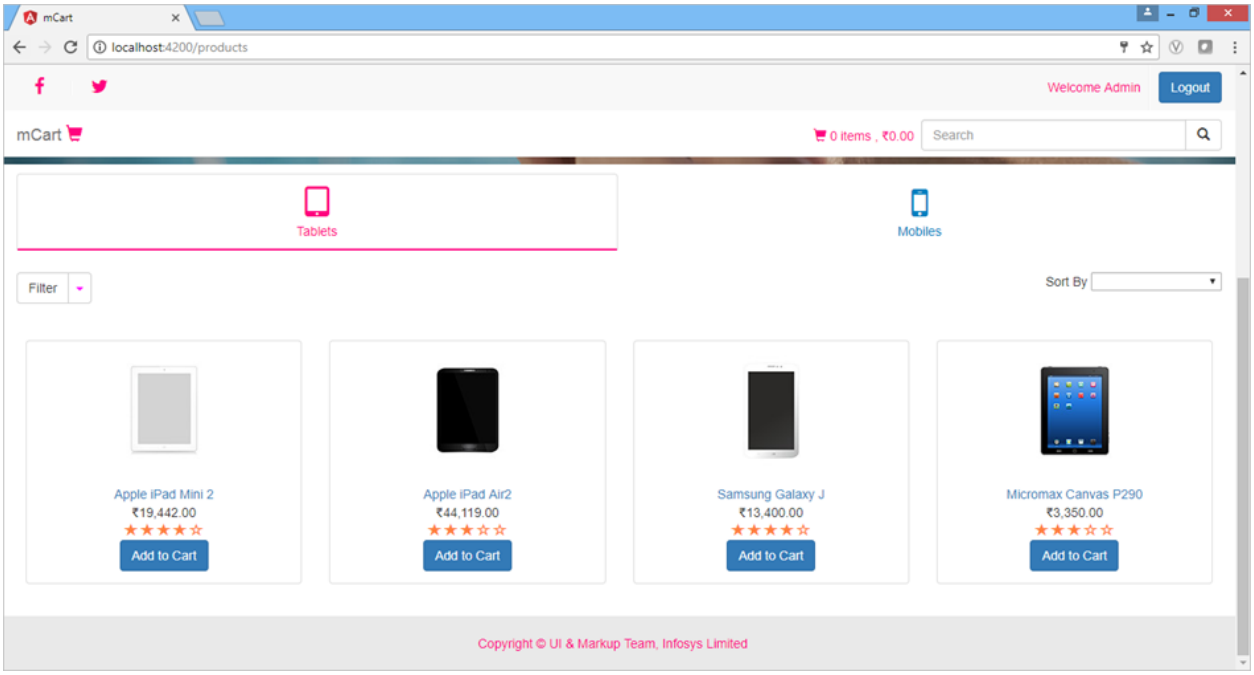
<u>Demo 18 : Data Binding in mCart Application</u>

Highlights:

- Exploring data binding in mCart application
- Understanding two way data binding in mCart application

Demo Steps:

Let us explore ProductListComponent again. Below is the screen for ProductListComponent



- Expand product-list folder under products folder and open product-list.component.html file. Observe the code given below.

```
1. <nav class='navbar navbar-default navbar-fixed-top navbarpos'>
2. <div class='container-fluid'>
3. <a class='navbar-brand txtcolor'>{{pageTitle}} <span
5. class="glyphicon glyphicon-shopping-cart txtcolor"></span></a>
6. <div class="input-group pull-right col-md-3 searchboxpos">
7. <input type="text" class="form-control" placeholder="Search" name="q"
   [(ngModel)]="listFilter" (change)="searchtext()">
8. <div class="input-group-btn">
9. <button class="btn btn-default">
10. <i class="glyphicon glyphicon-search"></i>
11. </button>
12. </div>
13. </div>


14. <div class="pull-right txtcolor cartpos">
15. <span class="glyphicon glyphicon-shopping-cart"> <a
16. [routerLink]="['cart']" class="txtcolor">{{selectedItems}} items</a></span>
17. <span>, {{total | currency:'INR':true:'1.2-2'}} </span>
18. </div>
19. </div>
20. </nav>
21. <br />
```

```
22. <br />
23. <div class="container" class="carouselpos">
24. <div id="carousel-example-generic" class="carousel slide carouselheight" data-ride="carousel"
    data-interval="3000">
25. <ol class="carousel-indicators">
26. <li data-target="#carousel-example-generic" data-slide-to="0" class="active"></li>
27. <li data-target="#carousel-example-generic" data-slide-to="1"></li>
28. <li data-target="#carousel-example-generic" data-slide-to="2"></li>
29. </ol>
30. <div class="carousel-inner">
31. <div class="item active">
32. <img src="assets/imgs/carousel_smart_phone.jpg" alt="First slide" style="min-
    width:100%;height:350px;">


33. </div>
34. <div class="item carouselimgpos">
35. <img src="assets/imgs/carousel1.jpg" alt="Second slide" style="min-
    width:100%;height:350px;">


36. </div>
37. <div class="item">
38. <img src="assets/imgs/tablet_blue_stylus.jpg" alt="Third slide" style="min-
    width:100%;height:350px;">
39. </div>
40. </div>
41. <a class="left carousel-control" href="#carousel-example-generic" role="button" data-
    slide="prev">
42. <span class="glyphicon glyphicon-chevron-left"></span>
43. </a>
44. <a class="right carousel-control" href="#carousel-example-generic" role="button" data-
    slide="next">
45. <span class="glyphicon glyphicon-chevron-right"></span>
46. </a>
47. </div>


48. <div class='panel with-nav-tabs panel-primary noborder'>
49. <div class='panel-heading noborder bgcolor'>
50. <ul class="nav nav-tabs noborder">
51. <li class="active tabpos"><a href="#tabprimary" (click)="tabselect('tablet')" data-
    toggle="tab"><i
52. class="fa fa-tablet fa-3x" aria-hidden="true"></i>
```

```
53. <div>Tablets</div></a></li>
54. <li class="tabpos"><a (click)="tabselect('mobile')" href="#tabprimary" data-toggle="tab"><i
55. class="fa fa-mobile fa-3x" aria-hidden="true"></i>
56. <div>Mobiles</div></a></li>
57. </ul>
58. </div>
59. <div class='panel-body'>
60. <div class="tab-content">
61. <div class="tab-pane fade in active" id="tabprimary">
62. <div class="btn-group">
63. <button type="button" class="btn btn-default">Filter</button>
64. <button type="button" class="btn btn-default dropdown-toggle" data-toggle="dropdown">
65. <span class="caret"></span> <span class="sr-only">Toggle
66. Dropdown</span>
67. </button>
68. <ul class="dropdown-menu multi-column columns-3 noclose">
69. <div class="row vdivide">
70. <div class="col-md-4">
71. <ul class="multi-column-dropdown noclose">
72. <h4>Manufacturer</h4>
73. <li *ngFor="let manufac of manufacturers">
74. <input type="checkbox" [ngModel]="manufac.checked" (change)="filter(manufac)"> <label>
75. {{manufac.id}} </label></li>
76. </ul>
77. </div>
78. <div class="col-md-4">
79. <ul class="multi-column-dropdown noclose">
80. <h4>OS</h4>
81. <li *ngFor="let ostypes of os">
82. <input type="checkbox" [ngModel]="ostypes.checked" (change)="filter(ostypes)">
83. <label> {{ostypes.id}}</label></li>
84. </ul>
85. </div>
86. <div class="col-md-4">
87. <ul class="multi-column-dropdown noclose">
88. <h4>Price Range</h4>
89. <li *ngFor="let price of price_range">
90. <input type="checkbox" [ngModel]="price.checked" (change)=filter(price)> <label>{{ price.id}}
    </label></li>
91. </ul>
92. </div>
93. </div>
94. </ul>
95. </div>
```

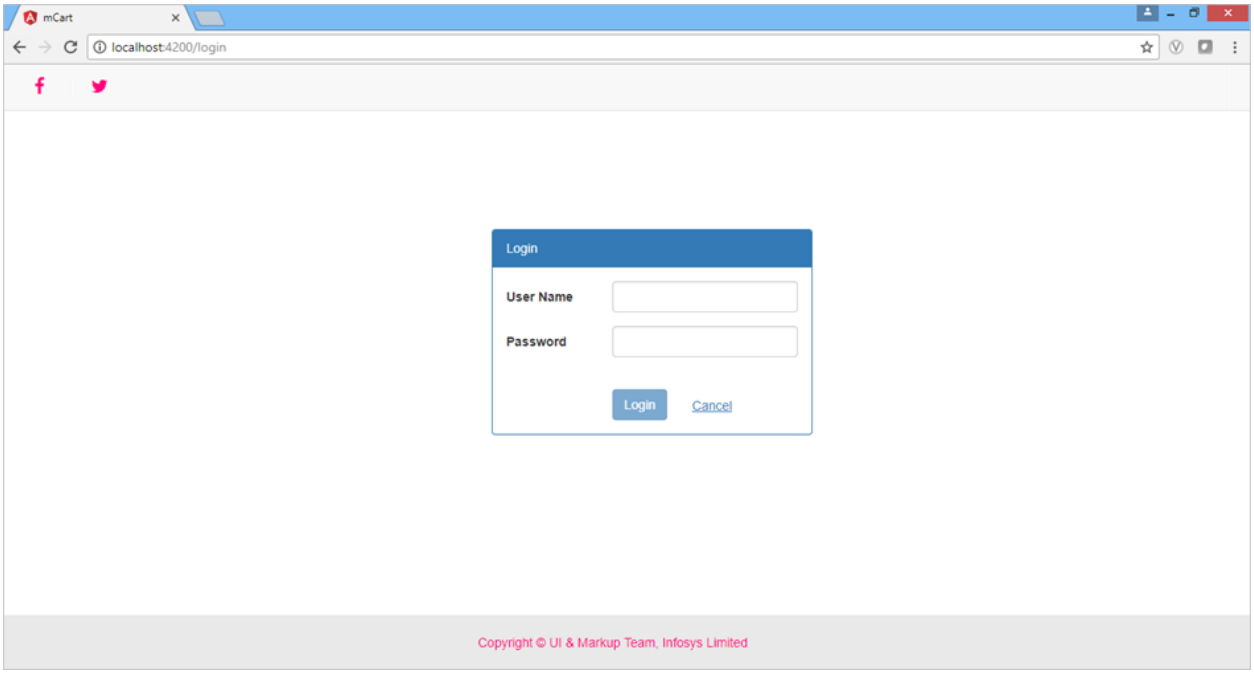
```
96. <span *ngIf="chkmanosprice.length> 0"> {{products.length}}
97. results</span>


98. <div class="pull-right">
99. <span>Sort By </span>
100. <select [ngModel]="sortoption" (change)="onChange($event.target.value)">
101. <option value="popularity">Popularity</option>
102. <option value="pricelh">Price - Low to High</option>
103. <option value="pricehl">Price - High to Low</option>
104. </select>
105. </div>
106. <div *ngIf='products && products.length'>
107. <div class="row" *ngFor='let product of products | orderBy:sortoption ; let i = index'
     [hidden]="(i%4)>0">
108. <div class="col-xs-3">
109. <span class="thumbnail text-center">
110. <div>
111. <img [src] ='product.imageUrl' [title]='product.productName'
112. [style.width.px]='imageWidth' [style.height.px]='imageHeight'
     [style.margin.px]='imageMargin'>
113. </div>
114. <div class="caption">
115. <div>
116. <a [routerLink]="[product.productId]" >
117. {{product.productName}} </a>
118. </div>
119. <div>{{ product.price | currency:'INR':true:'1.2-2'}}</div>
120. <div></div>
121. <ratings  class="ratingcolor" [rate]='product.rating'></ratings>
122. <div>
123. <button (click)="addCart(product.productId)"
124. class="btn btn-primary">Add to Cart</button>
125. </div>
126. </div>
```

Line 107: We have binded hidden property with the expression (i%4)>0. This is property binding. This binding will hide the products display if the given condition is true. In this loop, we are rendering four products per row per iteration. So for the next iteration, it should hide the remaining displayed products as they are already rendered.

Line 111: We have also used property binding here where image src property is binded with the imageUrl property of the component. Similarly title property is also binded with productName

Line 123: We have binded click event with addCart() method which will be invoked when this button is clicked. This is event binding

- Now let us explore LoginComponent again. Below is the output screen for LoginComponent.



- Now open login.component.html file from login folder and observe the code below

```
<div class="container" style="position: relative; top: 180px;width:50%">
 <div class="col-xs-7 col-xs-offset-3">
<div class="panel panel-primary">
 <div class="panel-heading">Login</div>
 <div class="panel-body padding">
<form class="form-horizontal" #loginForm="ngForm">
   <div class="form-group">
  <label for="name" class="col-xs-4 control-label">User Name</label>
 <div class="col-xs-7">
  <input type="text" class="form-control" name="username" [(ngModel)]="login.userName" #username="ngModel" required>
        <div class="alert alert-danger" *ngIf="username.touched && !username.valid">UserName is required</div>
    </div>
   </div>
  <div class="form-group">
    <label for="password" class="col-xs-4 control-label">Password</label>
     <div class="col-xs-7">
        <input type="password" class="form-control" name="password" #password="ngModel" [(ngModel)]="login.password" required>
            <div *ngIf="password.touched && !password.valid" class="alert alert-danger">Password is required</div>
     </div>
  </div>
```

```
21.         <div *ngIf="!valid" class="error">Invalid Credentials...Please try again...</div>
22.     <br />
23.         <div class="form-group">
24.         <span class="col-xs-4"></span>
25.     <div class="col-xs-3">
26.         <button (click)="onSubmit()" class="btn btn-primary"
    [disabled]="!loginForm.form.valid">Login</button>
27.     </div>
28.     <span class="col-xs-5" style="top:8px">
29.         <a [routerLink]="['/welcome']" style="color:#337ab7;text-decoration:
    underline;">Cancel</a>
30.             </span>
31.       </div>
32.   </form>
33.  </div>
34. </div>
35. </div>
36. </div>
```

Line 10: Username textbox is binded with username property of login object using two-way data binding where data flows in both the directions

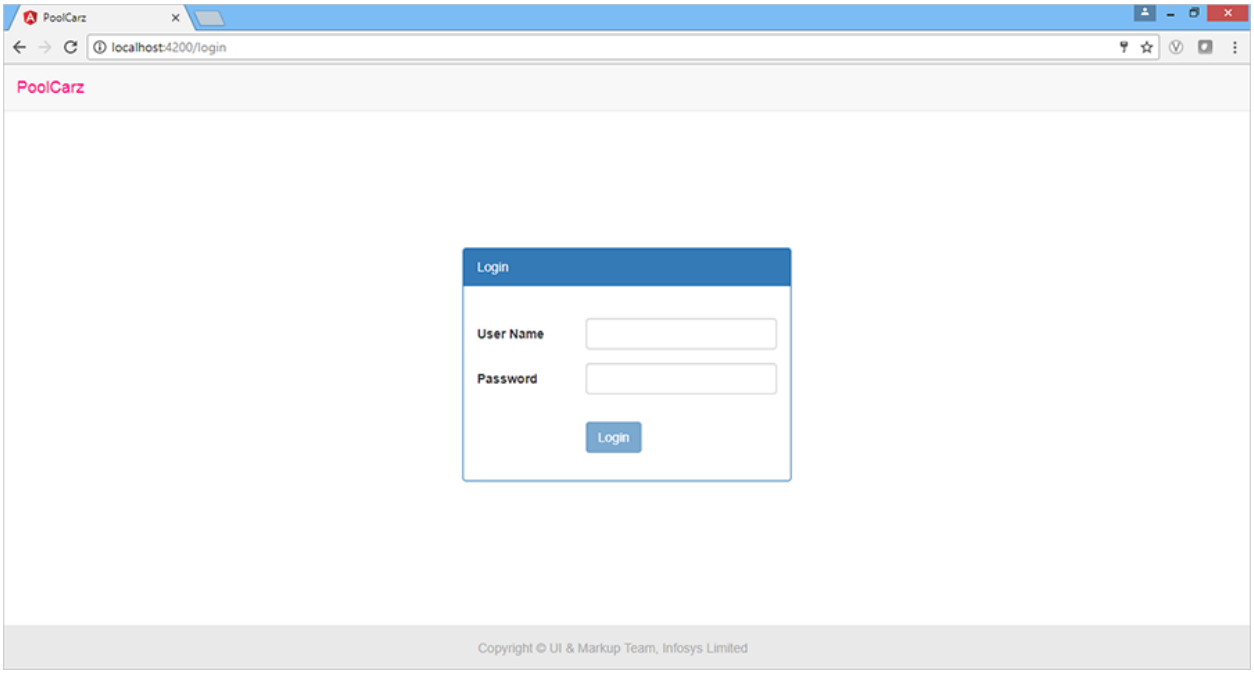
Line 17: Similarly password textbox is binded with password property of login object using two-way binding

<u>Exercise 4 : Implementing Data Binding in Pool Carz Application</u>

Time Limit: 20 Minutes

Problem Statement

Adding data binding to LoginComponent



- Create a model class called Login under login folder with username and password properties
- Create an instance of Login class in login component
- Bind username and password properties with the two text boxes using two way data binding
- Create users array which should contain user details where each object should have username and password values
- When user clicks on Login button after entering the values in text boxes, it should check whether the entered values
- are correct or not by checking with the data in the users array. Display an alert for successful or failure message accordingly.

Demo 18: Built in Pipes

Highlights:

- Exploring built in pipes
- Applying the pipes

Problem Statement: Displaying the product code in lowercase and product name in uppercase using built-in pipes. Output is as shown below

Product Details

Product Code prod_p001
Product Name LAPTOP

1. Write the below given code in **app.component.ts**

```
import { Component } from '@angular/core';
@Component({
        selector: 'app-root',
        templateUrl: './app.component.html',
        styleUrls: ['./app.component.css']
})
export class AppComponent {
        title: string = "product details";
        productCode: string = "PROD_P001";
        productName: string = "Laptop";
}
```

2. Write the below given code in **app.component.html**

```
<h3> {{ title | titlecase}} </h3>
<table style="text-align:left">
  <tr>
    <th> Product Code </th>
    <td> {{ productCode | lowercase }} </td>
  </tr>
```

```
  <tr>
    <th> Product Name </th>
    <td> {{ productName | uppercase }} </td>
  </tr>
</table>
```

3. Save the files and check the output in the browser

<u>Demo19: Passing Parameters to Pipes</u>

Highlights:

- Understanding Built-in Pipes
- Passing parameters to built-in pipes

Problem Statement: Applying built-in pipes with parameters to display product details. Output is as shown below

Product Details

Product Code	P001
Product Name	APPLE MPTT2 MACBOOK PRO
Product Price	217 021,00 ₹
Purchase Date	wednesday, january 17, 2018
Product Tax	10.00%
Product Rating	4.920

We have applied currency pipe to product price with locale setting as 'fr' i.e., French. According to French locale, currency symbol will be displayed at the end of the price as shown in the above output.

1. Write the below given code in **app.component.ts**

```
import { Component } from '@angular/core';


@Component({
        selector: 'app-root',
        templateUrl: './app.component.html',
        styleUrls: ['./app.component.css']
})
```

```
export class AppComponent {
        title: string = "product details"
         productCode: string = "PROD_P001";
        productName: string = "Apple MPTT2 MacBook Pro"
        productPrice: number = 217021;
        purchaseDate: string = "1/17/2018"
        productTax: string = "0.1";
         productRating: number = 4.92;
}
```

2. Write the below given code in **app.component.html**

```
<h3> {{ title | titlecase}} </h3>
<table style="text-align:left">
  <tr>
    <th> Product Code </th>
    <td> {{ productCode | slice:5:9 }} </td>
  </tr>
  <tr>
    <th> Product Name </th>
    <td> {{ productName | uppercase }} </td>
  </tr>
  <tr>
    <th> Product Price </th>
    <td> {{ productPrice | currency: 'INR':'symbol':'':'fr' }} </td>
  </tr>
  <tr>
    <th> Purchase Date </th>
    <td> {{ purchaseDate | date:'fullDate' | lowercase}} </td>
  </tr>
  <tr>
    <th> Product Tax </th>
```

```
        <td> {{ productTax | percent : '.2' }} </td>
    </tr>
    <tr>
        <th> Product Rating </th>
        <td>{{ productRating | number:'1.3-5'}} </td>
    </tr>
</table>
```

3. Write the below given code in **app.module.ts**

```
import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import { registerLocaleData } from '@angular/common';
import localeFrench from '@angular/common/locales/fr';

registerLocaleData(localeFrench);

@NgModule({
  imports: [BrowserModule],
  declarations: [AppComponent],
  providers:[],
  bootstrap: [AppComponent]
})
export class AppModule {
}
```

4. Save the files and check the output in the browser

Demo 20: JSON Pipe

Highlights:

- Understanding JSON pipe
- Applying JSON pipe

Problem Statement: Applying json pipe to product object to display the following output

Product Details in JSON Format

{ "productCode": "PROD_P001", "productName": "Laptop", "productPrice": 25000, "purchaseDate": "5/12/2017", "productTax": "0.1", "productRating": 4.53 }

1. Write the below given code in **app.component.ts**

```
import { Component } from '@angular/core';

@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent {
  title: string = "product details";

  product: Object = {
    "productCode": "PROD_P001", "productName": "Laptop", "productPrice": 25000,
    "purchaseDate": "5/12/2017", "productTax": "0.1", "productRating": 4.53
  };
}
```

2. Write the below given code in **app.component.html**

```
<h3> Product Details in JSON Format </h3>
{{ product | json }}
```

3. Save the files and check the output in the browser

Demo 21: i18nPlural Pipe

Highlights:

- Understanding i18nPlural pipe
- Applying i18nPlural pipe

Problem Statement: Applying i18nPlural pipe to display product ratings and feedback as shown below

Products Ratings

Product Id	Product Rating & Feedback
1	4 - Excellent
2	3 - Good
3	2 - Average
4	4 - Excellent
5	1 - Very Bad

1. Write the below given code in **app.component.ts**

```
1. import { Component } from '@angular/core';
2. @Component({
3.   selector: 'app-root',
4.   templateUrl: './app.component.html',
5.   styleUrls: ['./app.component.css']
6. })
7. export class AppComponent {
8.   productRatings: number[] = [4, 3, 2, 4, 1]
9.   productMapping = { '=4': ' # - Excellent', '=3': ' # - Good', '=2': '# - Average', 'other': ' # - Very
     Bad
10.  };
11. }
```

Line 9: productRatings property of type number array which is initialized to values

Line 10: productMapping property is an object which has ratings as key and a string as the value

2. Write the below given code in **app.component.html**

```
1. <h3> Products Ratings </h3>
2. <table border=1>
3. <thead>
```

```
4. <tr>
5. <th> Product Id</th>
6. <th> Product Rating & Feedback</th>
7. </tr>
8. </thead>
9. <tbody>
10. <tr *ngFor="let rating of productRatings; index as i">
11. <td align="center"> {{ i+1 }}</td>
12. <td align="center"> {{ rating | i18nPlural:productMapping }} </td>
13. </tr>
14. </tbody>
15. </table>
```

Line 12: i18nPlural pipe displays value from productMapping property based on each iterated value from productRatings.

3. Save the files and check the output in the browser

<u>Demo 22: i18nSelect Pipe</u>

Highlights:

- Understanding i18nSelect pipe
- Applying i18nSelect pipe

Problem Statement: Displaying greeting message based on the language selected using i18nSelect pipe. Output is as shown below

1. Write the below given code in **app.component.ts**

```
1.  import { Component } from '@angular/core';
2.  import { I18nSelectPipe } from '@angular/common';
3.
4.  @Component({
5.  selector: 'app-root',
6.  styleUrls: ['./app.component.css'],
7.  templateUrl: './app.component.html'
8.  })
9.  export class AppComponent {
10.
11. message: string;
12. messageMap: any = { 'en': 'Good Morning', 'fr': 'Bonjour', 'es': 'Buenos dÃƒÂ-as', 'de': 'Guten
    Morgen' };
13. }
```

Line 1: import I18nSelectPipe from @angular/common module

Line 11: message property holds the language code of the selected language from dropdown in the template

Line 12: messageMap is an object having language code as keys and corresponding message as values

2. Write the below given code in **app.component.html**

```
1.  <h2> Wish in Different Languages </h2>
2.
3.  Enter your Name <input type=text #name><br/><br/> Select language to display
4.
5.  <select #language (change)="message = language.value">
6.  <option value="en">English</option>
7.  <option value="fr">French</option>
8.  <option value="es">Spanish</option>
```

```
9.  <option value="de">German</option>
10. </select><br/><br/>
11.
12. <h3 *ngIf="message"> {{ message | i18nSelect:messageMap }}, {{ name.value }} </h3>
```

Line 3: Textbox value will be stored in name template variable

Line 5: When user selects a language from the dropdown, change event will be triggered where the selected language code will be assigned to message property

Line 1: i18nSelect will return the message corresponding to the code passed. This line renders the greeting message in the respective language selected.

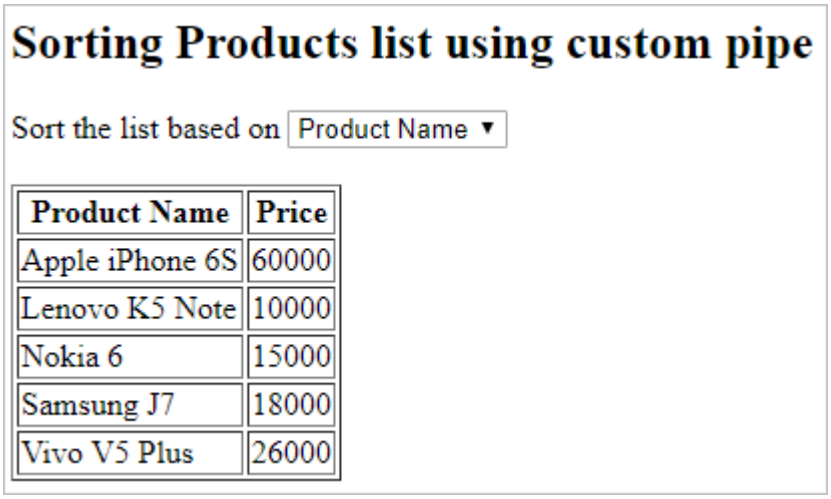
3. Save the files and check the output in the browser

<u>Demo 23: Custom Pipes</u>

Highlights:

- Creating a custom pipe
- Applying custom pipe to the template expression

Problem Statement: Sorting product list based on product name and price using a custom pipe. Output is as shown below



1. Write the below given code in **sort.pipe.ts**

```
import { Pipe, PipeTransform } from '@angular/core';


@Pipe({
```

```
  name: 'sort'
})
export class SortPipe implements PipeTransform {


  transform(value: string[], args?: string): string[] {
    if (args === "prodName") {
      return value.sort((a: any, b: any) => {
        if (a.productName < b.productName) {
          return -1;
        } else if (a.productName > b.productName) {
          return 1;
        } else {
          return 0;
        }
      });
    }
    else if (args === "price") {
      return value.sort((a: any, b: any) => {
        if (a.price < b.price) {
          return -1;
        } else if (a.price > b.price) {
          return 1;
        } else {
          return 0;
        }
      });
    }
    return value;
  }
```

```
}
```

2. Write the below given code in **app.component.ts**

```
import { Component } from '@angular/core';

@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent {
  sortoption: string = "";
  productsList = [
    { productName: "Samsung J7", price: 18000 },
    { productName: "Apple iPhone 6S", price: 60000 },
    { productName: "Lenovo K5 Note", price: 10000 },
    { productName: "Nokia 6", price: 15000 },
    { productName: "Vivo V5 Plus", price: 26000 }
  ];
}
```

3. Write the below given code in **app.component.html**

```
<h2> Sorting Products list using custom pipe </h2>

Sort the list based on
<select [(ngModel)]="sortoption">
      <option value="prodName">Product Name</option>
      <option value="price">Price</option>
     </select><br/><br/>
```

```
<table border="1">
  <thead>
    <tr>
      <th>Product Name</th>
      <th>Price</th>
    </tr>
  </thead>
  <tbody>
    <tr *ngFor="let products of productsList | sort:sortoption">
      <td>{{products.productName}}</td>
      <td>{{products.price}}</td>
    </tr>
  </tbody>
</table>
```

4. Write the below given code in **app.module.ts**

```
import { BrowserModule } from '@angular/platform-browser';
import { NgModule } from '@angular/core';
import { FormsModule } from '@angular/forms';
import { AppComponent } from './app.component';
import { SortPipe } from './sort.pipe';

@NgModule({
 declarations: [
  AppComponent,
  SortPipe
 ],
 imports: [
```

```
    BrowserModule,
    FormsModule
  ],
  providers: [],
  bootstrap: [AppComponent]
})
export class AppModule { }
```

5. Save the files and check the output in the browser

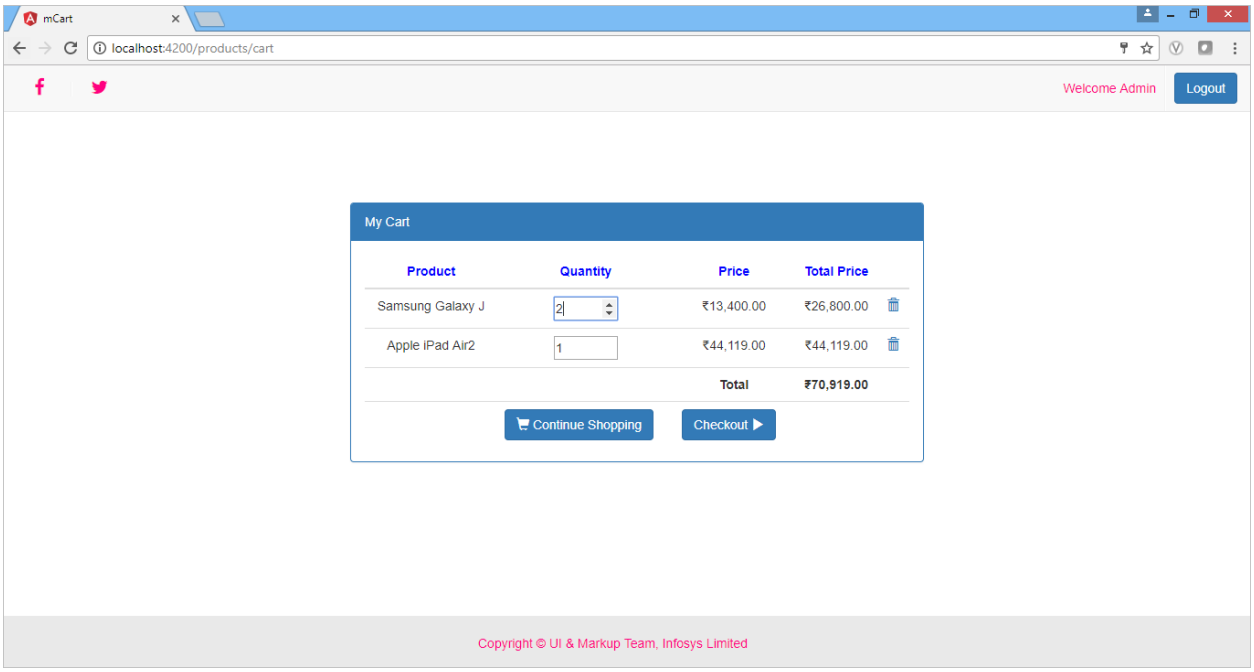
Demo 24: pipes in mCart Application

Highlights:

- Exploring various pipes used in mCart application
- Understanding implementation of pipes

Let us now explore another component called CartComponent which will be rendered when user clicks on cart link after selecting the products for purchase

Observe below the output of **CartComponent**



Our CartComponent acts as a basket and we can add or remove products as per our need.

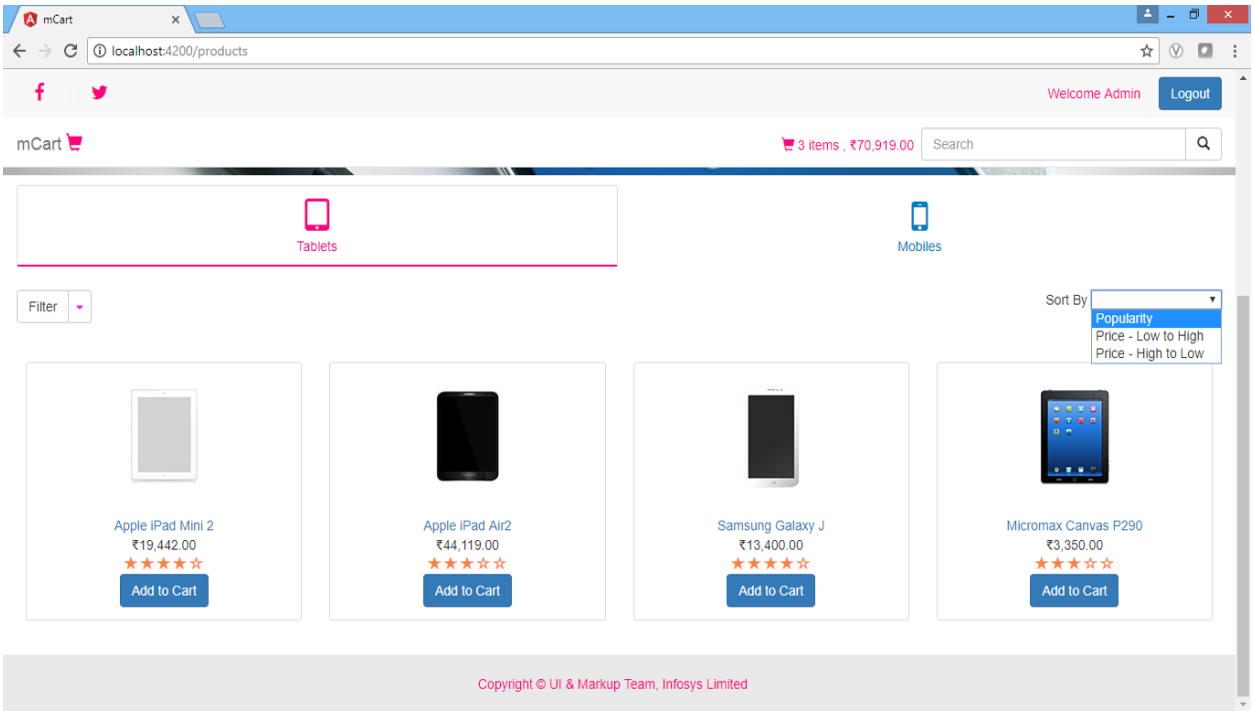
Observe the code inside **cart.component.html** file present under cart folder. Let us understand the application of pipes in the below given code

```
1. <div *ngIf="submit" class='panel panel-primary panelpos'>
2. <div class='panel-heading'>{{ pageTitle }}</div>
3. <div class='panel-body'>
4.
5. <table class='table' *ngIf='selectedProducts && selectedProducts.length' style="margin-
   bottom: 0px; margin-right: 0px">
6. <thead>
7. <tr class="tableheadercolor">
8. <th class="center">Product</th>
9. <th class="center">Quantity</th>
10. <th class="center">Price</th>
11. <th class="center">Total Price</th>
12. <th></th>
13. </tr>
14. </thead>
15. <tbody>
16. <tr *ngFor='let product of selectedProducts; let i=index'>
17. <td class="center">{{product.productName}}</td>
18. <td class="center">
19. <input type="number" id="quant" class="" [(ngModel)]=product.quantity min="1"
   max="100" (change)="updateCart(product)">
20. </td>
21. <td class="center">{{ product.price | currency:'INR':'symbol':'1.2-2' }}</td>
22. <td class="center">{{ product.totalPrice | currency:'INR':'symbol':'1.2-2'}}</td>
23. <td><a (click)="remove(i)"><span title="Delete"
24. class="glyphicon glyphicon-trash"></span></a></td>
25. </tr>
26. <tr>
27. <td></td>
28. <td></td>
29. <th class="center">Total</th>
30. <td class="center"><strong>{{ grandTotal | currency:'INR':'symbol':'1.2-2'
   }}</strong></td>
31. <td></td>
32. </tr>
33. <tr>
34. <td></td>
```

```
35. <td>
36. <button type="button" class="btn btn-primary" (click)='onBack()'>
37. <span class="glyphicon glyphicon-shopping-cart"></span> Continue Shopping
38. </button>
39. </td>
40. <td>
41. <button type="button" class="btn btn-primary" (click)="checkout()">
42. Checkout <span class="glyphicon glyphicon-play"></span>
43. </button>
44. ...
```

Line 21-22: A built-in pipe called currency is applied on price and totalPrice properties which displays the values with currency symbol

Our **ProductListComponent** uses some custom pipes too. Below is the output screen for the ProductListComponent. Let us observe the drop down placed at the right side for sorting



Observe the code inside **product-list.component.html** file. Let us explore the custom pipes in this code

```
1. <nav class='navbar navbar-default navbar-fixed-top navbarpos'>
2. <div class='container-fluid'>
3. <a class='navbar-brand txtcolor'>{{pageTitle}} <span
4. class="glyphicon glyphicon-shopping-cart txtcolor"></span></a>
5. <div class="input-group pull-right col-md-3 searchboxpos">
6. <input type="text" class="form-control" placeholder="Search" name="q"
   [(ngModel)]="listFilter" (change)="searchtext()">
7. <div class="input-group-btn">
8. <button class="btn btn-default">
9. <i class="glyphicon glyphicon-search"></i>
10. </button>
11. </div>
12. </div>
13.
14. <div class="pull-right txtcolor cartpos">
15. <span class="glyphicon glyphicon-shopping-cart"> <a
16. [routerLink]="['cart']" class="txtcolor">{{selectedItems}} items</a></span>
17. <span>, {{total | currency:'INR':'symbol':'1.2-2'}} </span>
18. </div>
19. </div>
20. </nav>
21. <br />
22. <br />
23. <div class="container" class="carouselpos">
24. <div id="carousel-example-generic" class="carousel slide carouselheight" data-ride="carousel"
    data-interval="3000">
25. <ol class="carousel-indicators">
26. <li data-target="#carousel-example-generic" data-slide-to="0" class="active"></li>
27. <li data-target="#carousel-example-generic" data-slide-to="1"></li>
28. <li data-target="#carousel-example-generic" data-slide-to="2"></li>
29. </ol>
30. <div class="carousel-inner">
31. <div class="item active">
32. <img src="assets/imgs/carousel_smart_phone.jpg" alt="First slide" style="min-
    width:100%;height:350px;">
33.
34. </div>
35. <div class="item carouselimgpos">
36. <img src="assets/imgs/carousel1.jpg" alt="Second slide" style="min-
    width:100%;height:350px;">
37.
38. </div>
39. <div class="item">
```

```
40. <img src="assets/imgs/tablet_blue_stylus.jpg" alt="Third slide" style="min-
    width:100%;height:350px;">
41. </div>
42. </div>
43. <a class="left carousel-control" href="#carousel-example-generic" role="button" data-
    slide="prev">
44. <span class="glyphicon glyphicon-chevron-left"></span>
45. </a>
46. <a class="right carousel-control" href="#carousel-example-generic" role="button" data-
    slide="next">
47. <span class="glyphicon glyphicon-chevron-right"></span>
48. </a>
49. </div>
50.
51. <div class='panel with-nav-tabs panel-primary noborder'>
52. <div class='panel-heading noborder bgcolor'>
53. <ul class="nav nav-tabs noborder">
54. <li class="active tabpos"><a href="#tabprimary" (click)="tabselect('tablet')" data-
    toggle="tab"><i
55. class="fa fa-tablet fa-3x" aria-hidden="true"></i>
56. <div>Tablets</div></a></li>
57. <li class="tabpos"><a (click)="tabselect('mobile')" href="#tabprimary" data-toggle="tab"><i
58. class="fa fa-mobile fa-3x" aria-hidden="true"></i>
59. <div>Mobiles</div></a></li>
60. </ul>
61. </div>
62. <div class='panel-body'>
63. <div class="tab-content">
64. <div class="tab-pane fade in active" id="tabprimary">
65. <div class="btn-group">
66. <button type="button" class="btn btn-default">Filter</button>
67. <button type="button" class="btn btn-default dropdown-toggle" data-toggle="dropdown">
68. <span class="caret"></span> <span class="sr-only">Toggle
69. Dropdown</span>
70. </button>
71. <ul class="dropdown-menu multi-column columns-3 noclose">
72. <div class="row vdivide">
73. <div class="col-md-4">
74. <ul class="multi-column-dropdown noclose">
75. <h4>Manufacturer</h4>
76. <li *ngFor="let manufac of manufacturers">
77. <input type="checkbox" [ngModel]="manufac.checked" (change)="filter(manufac)"> <label>
78. {{manufac.id}} </label></li>
79. </ul>
```

```
80. </div>
81. <div class="col-md-4">
82. <ul class="multi-column-dropdown noclose">
83. <h4>OS</h4>
84. <li *ngFor="let ostypes of os">
85. <input type="checkbox" [ngModel]="ostypes.checked" (change)="filter(ostypes)">
86. <label> {{ostypes.id}}</label></li>
87. </ul>
88. </div>
89. <div class="col-md-4">
90. <ul class="multi-column-dropdown noclose">
91. <h4>Price Range</h4>
92. <li *ngFor="let price of price_range">
93. <input type="checkbox" [ngModel]="price.checked" (change)=filter(price)> <label>{{ price.id}}
    </label></li>
94. </ul>
95. </div>
96. </div>
97. </ul>
98. </div>
99. <span *ngIf="chkmanosprice.length> 0"> {{products.length}}
100.        results</span>
101.
102.        <div class="pull-right">
103.        <span>Sort By </span>
104.        <select [ngModel]="sortoption" (change)="onChange($event.target.value)">
105.        <option value="popularity">Popularity</option>
106.        <option value="pricelh">Price - Low to High</option>
107.        <option value="pricehl">Price - High to Low</option>
108.        </select>
109.        </div>
110.        <div *ngIf='products && products.length'>
111.        <div class="row" *ngFor='let product of products | orderBy:sortoption ; let i = index'
    [hidden]="(i%4)>0">
112.        <div class="col-xs-3">
```

...

Line 104-108: Renders a drop down for sorting with three options called popularity, price – low to high and price – high to low. This control is binded with a property called sortoption which will hold the value selected by the user

Line 111: We have applied a custom pipe called orderBy to the products expression by passing sortoption property as parameter. OrderBy pipe will sort the list of products based on popularity or price in ascending or descending order as per the choice selected by the user

Let us explore the given below code for **orderby.pipe.ts** file under product-list folder to understand the sorting functionality

```
1.  import { PipeTransform, Pipe } from '@angular/core';
2.  import { Product } from '../product';
3.
4.  @Pipe({
5.  name: 'orderBy'
6.  })
7.  export class OrderByPipe implements PipeTransform {
8.
9.  transform(value: Product[], args: string): Product[] {
10. if (args === "popularity") {
11. return value.sort((a: any, b: any) => {
12. if (a.rating > b.rating) {
13. return -1;
14. } else if (a.rating < b.rating) {
15. return 1;
16. } else {
17. return 0;
18. }
19. });
20. }
21. else if (args === "pricelh") {
22. return value.sort((a: any, b: any) => {
        a.  if (a.price < b.price) {
23. return -1;
24. } else if (a.price > b.price) {
25. return 1;
26. } else {
27. return 0;
28. }
29. });
30.
31. }
32. else if (args === 'pricehl') {
33. return value.sort((a: any, b: any) => {
34. if (a.price > b.price) {
35. return -1;
36. } else if (a.price < b.price) {
37. return 1;
38. } else {
39. return 0;
```

```
40. }
41. });
42. }
43. return value;
44. }
45. }
```

Line 4 -6: @Pipe marks the class as a pipe with name as orderBy

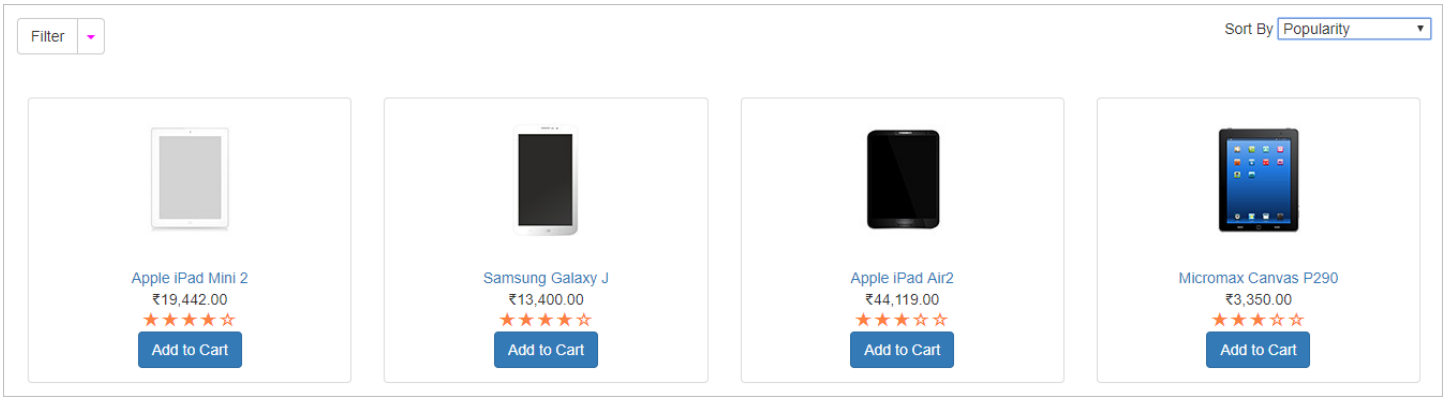
Line 7: Inherits PipeTransform interface

Line 9: Overrides transform method which takes products array as first parameter and the selected value form dropdown into the second parameter called args

Line 10-20: if args value is popularity, then it sorts the products list based on rating property of products

Line 21-30 : Similary if args value is pricelh, it sorts the products list based on price property

Observe the outcome screen below



Demo 25: Nested Components

Highlights:

- Creating nested component
- Loading nested component in container component

Problem Statement: Loading CourseslistComponent in the root component when user clicks on View courses list button as shown below

1. Create a component called coursesList using the following CLI command

   D:\MyApp>ng generate component coursesList

2. Above command will create a folder with name courses-list with the following files

   - courses-list.component.ts
   - courses-list.component.html
   - courses-list.component.css
   - courses-list.component.spec.ts

2. CoursesListComponent class will be added in **app.module.ts** file

```
import { BrowserModule } from '@angular/platform-browser';
import { NgModule } from '@angular/core';
import { AppComponent } from './app.component';
import { CoursesListComponent } from './courses-list/courses-list.component';

@NgModule({
  declarations: [
```

```
    AppComponent,
    CoursesListComponent
  ],
  imports: [
    BrowserModule
  ],
  providers: [],
  bootstrap: [AppComponent]
})
export class AppModule { }
```

3. Write the below given code in **courses-list.component.ts**

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

@Component({
  selector: 'app-courses-list',
  templateUrl: './courses-list.component.html',
  styleUrls: ['./courses-list.component.css']
})
export class CoursesListComponent {

  courses = [
    { courseId: 1, courseName: "Node JS" },
    { courseId: 2, courseName: "Typescript" },
    { courseId: 3, courseName: "Angular" },
    { courseId: 4, courseName: "React JS" }
  ];

}
```

4. Write the below given code in **courses-list.component.html**

```
<table border="1">
  <thead>
    <tr>
      <th>Course ID</th>
      <th>Course Name</th>
    </tr>
  </thead>
  <tbody>
    <tr *ngFor="let course of courses">
      <td>{{course.courseId}}</td>
      <td>{{course.courseName}}</td>
    </tr>
  </tbody>
</table>
```

5. Add the following code in **courses-list.component.css**

```
tr{
   text-align:center;
}
```

6. Write the below given code in app.component.html

```
<h2> Popular Courses </h2>
<button (click)="show=true">View Courses list</button><br/><br/>
<div *ngIf="show">
   <app-courses-list></app-courses-list>
</div>
```

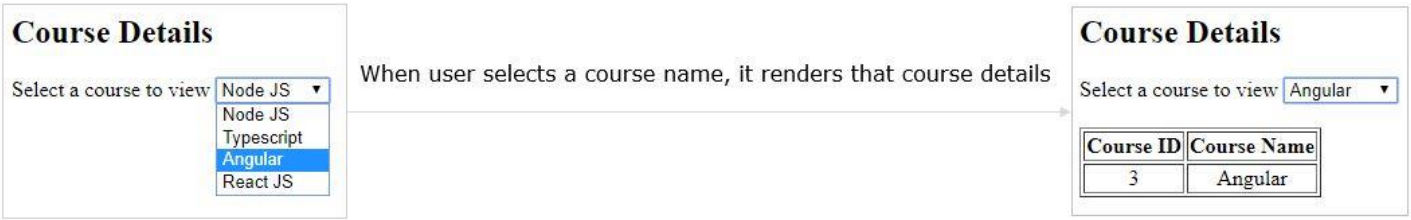
6. Save the files and check the output in the browser

Demo 26: Passing data from Container to child

Highlights:

- Loading nested component
- Passing data from container to child component

Problem Statement: Creating an AppComponent which displays a dropdown with list of courses as values in it. Create another component called CoursesList component and load it in AppComponent which should display the course details. When user selects a course from dropdown, corresponding course details should be loaded. Output is as shown below



1. Open **courses-list.component.ts** file created in nested components example and add the following code

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

@Component({
  selector: 'app-courses-list',
  templateUrl: './courses-list.component.html',
  styleUrls: ['./courses-list.component.css']
})
export class CoursesListComponent {
  courses = [
    { courseId: 1, courseName: "Node JS" },
    { courseId: 2, courseName: "Typescript" },
    { courseId: 3, courseName: "Angular" },
    { courseId: 4, courseName: "React JS" }
  ];

  course: any[];

  @Input() set cName(name: string) {
    this.course = [];
    for (var i = 0; i < this.courses.length; i++) {
      if (this.courses[i].courseName === name) {
        this.course.push(this.courses[i]);
      }
    }
  }
}
```

2. Open **courses-list.component.html** and add the following code

```
<table border="1" *ngIf="course.length>0">
```

```
  <thead>
    <tr>
      <th>Course ID</th>
      <th>Course Name</th>
    </tr>
  </thead>
  <tbody>
    <tr *ngFor="let c of course">
      <td>{{c.courseId}}</td>
      <td>{{c.courseName}}</td>
    </tr>
  </tbody>
</table>
```

3. Add the following in **app.component.html**

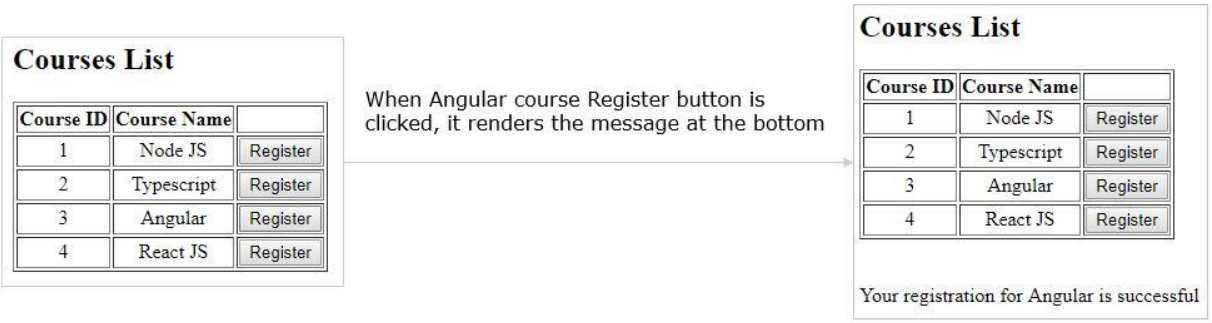
```
<h2> Course Details </h2>

Select a course to view <select #course (change)="name = course.value">
   <option value="Node JS">Node JS</option>
   <option value="Typescript">Typescript</option>
   <option value="Angular">Angular</option>
   <option value="React JS">React JS</option>
   </select><br/><br/>

<app-courses-list [cName]="name"></app-courses-list>
```

<u>Demo 27: Passing data from child to parent</u>

Highlights:

- Loading nested component
- Passing data from child to container component

Problem Statement: Let us create an AppComponent which loads another component called CoursesList component. Create another component called CoursesListComponent which should display the courses list in a table along with register button in each row. When user clicks on register button, it should send that courseName value back to AppComponent where it should display registration successful message along with courseName



1. Open **courses-list.component.ts** file created in the previous example and add the following code

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


@Component({
  selector: 'app-courses-list',
  templateUrl: './courses-list.component.html',
  styleUrls: ['./courses-list.component.css']
})
export class CoursesListComponent {
  @Output() onRegister = new EventEmitter<string>();
  courses = [
    { courseId: 1, courseName: "Node JS" },
    { courseId: 2, courseName: "Typescript" },
    { courseId: 3, courseName: "Angular" },
    { courseId: 4, courseName: "React JS" }
```

```
  ];

  register(courseName: string) {
    this.onRegister.emit(courseName);
  }
}
```

2. Open **courses-list.component.html** and add the following code

```
<table border="1">
  <thead>
    <tr>
      <th>Course ID</th>
      <th>Course Name</th>
      <th></th>
    </tr>
  </thead>
  <tbody>
    <tr *ngFor="let course of courses">
      <td>{{course.courseId}}</td>
      <td>{{course.courseName}}</td>
      <td><button (click)="register(course.courseName)">Register</button></td>
    </tr>
  </tbody>
</table>
```

3. Add the following in **app.component.html**

```
<h2> Courses List </h2>

<app-courses-list (onRegister)="courseReg($event)"></app-courses-list>
<br/><br/>
```

```
<div *ngIf="message">{{message}}</div>
```

4. Add the following code in app.component.ts

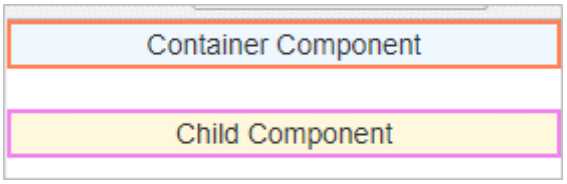
```
import { Component } from '@angular/core';

@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent {

  message: string;

  courseReg(courseName: string) {
    this.message = `Your registration for ${courseName} is successful`;
  }
}
```

Demo 28: Component styling using styleUrls

Highlights:

- Adding CSS styles to components
- Understanding the usage of styleUrls in applying a style

Problem Statement: Applying CSS styles to components using styleUrls property. Output is as shown below



1. Write the below given code in **app.component.ts**

```
import { Component } from '@angular/core';

@Component({
  selector: 'app-root',
  styleUrls: ['./app.component.css'],
  templateUrl: './app.component.html'
})
export class AppComponent {
}
```

2. Write the below given code in **app.component.css**

```
.highlight {
        border: 2px solid coral;
        background-color: aliceblue;
```

```
        text-align: center;
        margin-bottom: 20px;
}
```

3. Write the below given code in app.component.html

```
<div class="highlight">
  Container Component
</div>
<app-child></app-child>
```

4. Create another component called **ChildComponent** using the following CLI command

   D:\MyApp>ng generate component Child

5. Write the below given code in **child.component.ts**

```
import { Component } from '@angular/core';

@Component({
  selector: 'app-child',
  styleUrls: ['./child.component.css'],
  templateUrl: './child.component.html'
})
export class ChildComponent {
}
```

6. Write the below given code in **child.component.css**

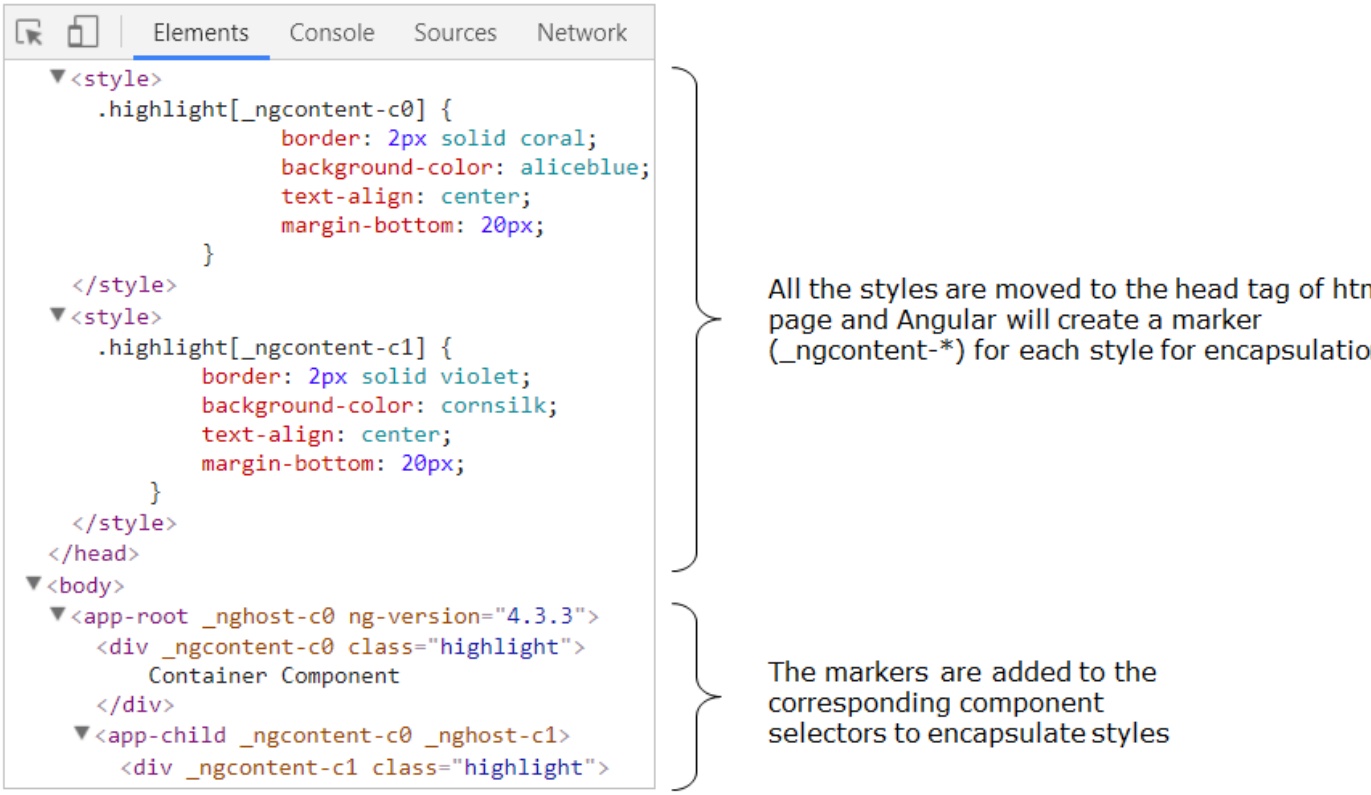
```
.highlight {
        border: 2px solid violet;
        background-color: cornsilk;
        text-align: center;
        margin-bottom: 20px;
```

}

7. Write the below given code in **child.component.html**

```
<div class="highlight">
  Child Component
</div>
```

8. Save the files and check the output in the browser
9. Open developer tools in your chrome browser and go to **Elements** tab

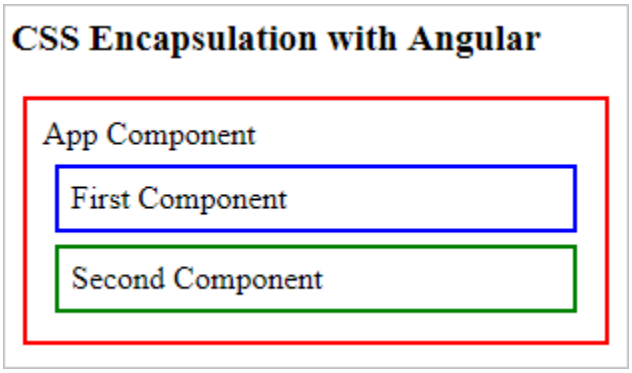


Demo 29: View encapsulation

Highlights:

- Understanding shadow DOM using view encapsulation
- Using ViewEncapsulation.Native and ViewEncapsulation.None

Problem Statement: Applying native and none encapsulation modes to components. Outputs for native mode and none mode are shown below



ViewEncapsulation.Native

1. Create a component called **First** using the following CLI command

   D:\MyApp>ng generate component first

2. Write the below given code in **first.component.css**

   .cmp {

```
  padding: 6px;
  margin: 6px;
  border: blue 2px solid;
}
```

3. Write the below given code in **first.component.html**

```
<div class="cmp">First Component</div>
```

4. Create a component called **Second** using the following CLI command

```
D:\MyApp>ng generate component second
```

5. Write the below given code in **second.component.css**

```
.cmp {
   border: green 2px solid;
   padding: 6px;
   margin: 6px;
}
```

6. Write the below given code in **second.component.html**

```
<div class="cmp">Second Component</div>
```

7. Write the below given code in **app.component.css**

```
.cmp {
  padding: 8px;
  margin: 6px;
  border: 2px solid red;
}
```

8. Write the below given code in **app.component.html**

```
<h3>CSS Encapsulation with Angular</h3>
<div class="cmp">
   App Component
   <app-first></app-first>
   <app-second></app-second>
</div>
```

9. Save the files and check the output in the browser

Conti…

ViewEncapsulation.None

1. Set ViewEncapsulation to none mode in **app.component.ts** file

```
import { Component, ViewEncapsulation } from '@angular/core';

@Component({
  selector: 'app-root',
  styleUrls: ['./app.component.css'],
  templateUrl: './app.component.html',
  encapsulation: ViewEncapsulation.None
})
export class AppComponent {
}
```

2. Set ViewEncapsulation to none mode in **second.component.ts** file

```
import { Component, ViewEncapsulation } from '@angular/core';

@Component({
  selector: 'app-second',
```

```
  templateUrl: './second.component.html',
  styleUrls: ['./second.component.css'],
  encapsulation: ViewEncapsulation.None
})
export class SecondComponent {

}
```

3. Save the files and check the output in the browser

Demo 30: Component Life Cycle

Highlights:

- Understanding component lifecycle
- Exploring and overriding various lifecycle hooks

Problem Statement: Overriding component life cycle hooks and logging the corresponding messages to understand the flow. Output is as shown below

1. Write the below given code in **app.component.ts**

```
import { Component, OnInit,  DoCheck, AfterContentInit, AfterContentChecked,
   AfterViewInit, AfterViewChecked,
   OnDestroy } from '@angular/core';
```

```
@Component({
  selector: 'app-root',
  styleUrls: ['./app.component.css'],
  templateUrl: './app.component.html'
})
export class AppComponent implements OnInit,  DoCheck,
    AfterContentInit, AfterContentChecked,
    AfterViewInit, AfterViewChecked,
    OnDestroy {

    data: string = "Angular 2";

    ngOnInit() {
        console.log("Init");
    }

    ngDoCheck() {
        console.log("Change detected");
    }

    ngAfterContentInit() {
        console.log("After content init");
    }

    ngAfterContentChecked() {
        console.log("After content checked");
    }

    ngAfterViewInit() {
```

```
    console.log("After view init");
  }

  ngAfterViewChecked() {
    console.log("After view checked");
  }

  ngOnDestroy() {
    console.log("Destroy");
  }
}
```

2. Write the below given code in **app.component.html**

```
<div>
  <h1>I'm a container component</h1>
  <input type="text" [(ngModel)]='data'>
  <app-child [title]='data'></app-child>
</div>
```

3. Write the below given code in **child.component.ts**

```
import { Component, OnChanges, Input } from '@angular/core';

@Component({
  selector: 'app-child',
  templateUrl: './child.component.html',
  styleUrls: ['./child.component.css']
})
export class ChildComponent implements OnChanges {

  @Input() title = 'I\'m a nested component';
```

```
  ngOnChanges(changes) {
  console.log("changes in child:"+JSON.stringify(changes));
 }

}
```

4. Write the below given code in **child.component.html**

   ```
   <h2>Child Component</h2>
   <h2>{{title}}</h2>
   ```

5. Save the files and check the output in the browser

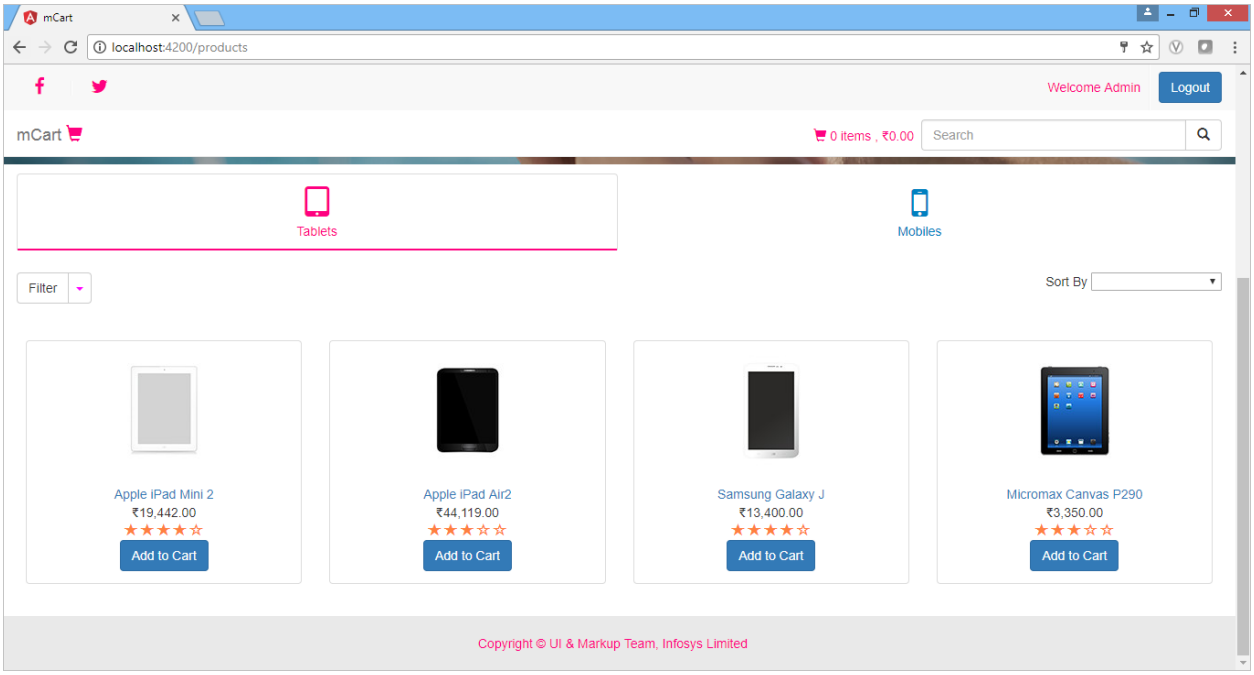
<u>Demo 31: Nested Components and data sharing in mCart Application</u>

Highlights:

- Exploring nested components in mCart application
- Understanding ProductListComponent and RatingComponent

Now let us explore nested components feature used in **ProductListComponent**

Below is the output of **ProductListComponent**. Observe the rating displayed for each product. We have rendered it by loading another component called **RatingComponent**



Code for **product-list.component.html** is given below. This file is under product-list folder. Let us understand the code present

```
1. <nav class='navbar navbar-default navbar-fixed-top navbarpos'>
2. <div class='container-fluid'>
3. <a class='navbar-brand txtcolor'>{{pageTitle}} <span
4. class="glyphicon glyphicon-shopping-cart txtcolor"></span></a>
5. <div class="input-group pull-right col-md-3 searchboxpos">
6. <input type="text" class="form-control" placeholder="Search" name="q"
   [(ngModel)]="listFilter" (change)="searchtext()">
7. <div class="input-group-btn">
8. <button class="btn btn-default">
9. <i class="glyphicon glyphicon-search"></i>
10. </button>
11. </div>
12. </div>
13.
14. <div class="pull-right txtcolor cartpos">
15. <span class="glyphicon glyphicon-shopping-cart"> <a
16. [routerLink]="['cart']" class="txtcolor">{{selectedItems}} items</a></span>
17. <span>, {{total | currency:'INR':'symbol':'1.2-2'}} </span>
18. </div>
19. </div>
20. </nav>
```

```
21. <br />
22. <br />
23. <div class="container" class="carouselpos">
24. <div id="carousel-example-generic" class="carousel slide carouselheight" data-ride="carousel"
    data-interval="3000">
25. <ol class="carousel-indicators">
26. <li data-target="#carousel-example-generic" data-slide-to="0" class="active"></li>
27. <li data-target="#carousel-example-generic" data-slide-to="1"></li>
28. <li data-target="#carousel-example-generic" data-slide-to="2"></li>
29. </ol>
30. <div class="carousel-inner">
31. <div class="item active">
32. <img src="assets/imgs/carousel_smart_phone.jpg" alt="First slide" style="min-
    width:100%;height:350px;">
33.
34. </div>
35. <div class="item carouselimgpos">
36. <img src="assets/imgs/carousel1.jpg" alt="Second slide" style="min-
    width:100%;height:350px;">


37. </div>
38. <div class="item">
39. <img src="assets/imgs/tablet_blue_stylus.jpg" alt="Third slide" style="min-
    width:100%;height:350px;">
40. </div>
41. </div>
42. <a class="left carousel-control" href="#carousel-example-generic" role="button" data-
    slide="prev">
43. <span class="glyphicon glyphicon-chevron-left"></span>
44. </a>
45. <a class="right carousel-control" href="#carousel-example-generic" role="button" data-
    slide="next">
46. <span class="glyphicon glyphicon-chevron-right"></span>
47. </a>
48. </div>
49.
50. <div class='panel with-nav-tabs panel-primary noborder'>
51. <div class='panel-heading noborder bgcolor'>
52. <ul class="nav nav-tabs noborder">
53. <li class="active tabpos"><a href="#tabprimary" (click)="tabselect('tablet')" data-
    toggle="tab"><i
54. class="fa fa-tablet fa-3x" aria-hidden="true"></i>
55. <div>Tablets</div></a></li>
```

```
56. <li class="tabpos"><a (click)="tabselect('mobile')" href="#tabprimary" data-toggle="tab"><i
57. class="fa fa-mobile fa-3x" aria-hidden="true"></i>
58. <div>Mobiles</div></a></li>
59. </ul>
60. </div>
61. <div class='panel-body'>
62. <div class="tab-content">
63. <div class="tab-pane fade in active" id="tabprimary">
64. <div class="btn-group">
65. <button type="button" class="btn btn-default">Filter</button>
66. <button type="button" class="btn btn-default dropdown-toggle" data-toggle="dropdown">
67. <span class="caret"></span> <span class="sr-only">Toggle
68. Dropdown</span>
69. </button>
70. <ul class="dropdown-menu multi-column columns-3 noclose">
71. <div class="row vdivide">
72. <div class="col-md-4">
73. <ul class="multi-column-dropdown noclose">
74. <h4>Manufacturer</h4>
75. <li *ngFor="let manufac of manufacturers">
76. <input type="checkbox" [ngModel]="manufac.checked" (change)="filter(manufac)"> <label>
77. {{manufac.id}} </label></li>
78. </ul>
79. </div>
80. <div class="col-md-4">
81. <ul class="multi-column-dropdown noclose">
82. <h4>OS</h4>
83. <li *ngFor="let ostypes of os">
84. <input type="checkbox" [ngModel]="ostypes.checked" (change)="filter(ostypes)">
85. <label> {{ostypes.id}}</label></li>
86. </ul>
87. </div>
88. <div class="col-md-4">
89. <ul class="multi-column-dropdown noclose">
90. <h4>Price Range</h4>
91. <li *ngFor="let price of price_range">
92. <input type="checkbox" [ngModel]="price.checked" (change)=filter(price)> <label>{{ price.id}}
    </label></li>
93. </ul>
94. </div>
95. </div>
96. </ul>
97. </div>
98. <span *ngIf="chkmanosprice.length> 0"> {{products.length}}
```

```
results</span>

        <div class="pull-right">
        <span>Sort By </span>
        <select [ngModel]="sortoption" (change)="onChange($event.target.value)">
        <option value="popularity">Popularity</option>
        <option value="pricelh">Price - Low to High</option>
        <option value="pricehl">Price - High to Low</option>
        </select>
        </div>
        <div *ngIf='products && products.length'>
        <div class="row" *ngFor='let product of products | orderBy:sortoption ; let i = index' [hidden]="(i%4)>0">
        <div class="col-xs-3">
        <span class="thumbnail text-center">
        <div>
        <img [src] ='product.imageUrl' [title]='product.productName'
        [style.width.px]='imageWidth' [style.height.px]='imageHeight' [style.margin.px]='imageMargin'>
        </div>
        <div class="caption">
        <div>
        <a [routerLink]="[product.productId]" >
        {{product.productName}} </a>
        </div>
        <div>{{ product.price | currency:'INR':'symbol':'1.2-2'}}</div>
        <div></div>
        <ratings  class="ratingcolor" [rate]='product.rating'></ratings>

        <div>
        <button (click)="addCart(product.productId)"
        class="btn btn-primary">Add to Cart</button>
        </div>
        </div>
        </span>
        </div>
        <div class="col-xs-3">
        <div *ngIf="products[i+1]" class="thumbnail text-center">
        <div>
        <img [src]='products[i+1].imageUrl' [title]='products[i+1].productName' [style.width.px]='imageWidth' [style.height.px]='imageHeight'
        [style.margin.px]='imageMargin'>
        </div>
        <div class="caption">
```

```
140.        <div>
141.        <a [routerLink]="[products[i+1].productId]">
142.        {{products[i+1].productName}} </a>
143.        </div>
144.        <div>{{ products[i+1].price | currency:'INR':'symbol':'1.2-2'}}
145.        </div>
146.        <div></div>
147.        <ratings class="ratingcolor" [rate]='products[i+1].rating'></ratings>
148.        <div></div>
149.        <div>
150.        <button (click)="addCart(products[i+1].productId)" class="btn btn-primary">Add to
    Cart</button>
151.        </div>
152.        </div>
153.        </div>
154.        </div>
155.        <div class="col-xs-3">
156.        <div *ngIf="products[i+2]" class="thumbnail text-center">
157.        <div>
158.        <img [src]='products[i+2].imageUrl' [title]='products[i+2].productName'
    [style.width.px]='imageWidth' [style.height.px]='imageHeight'
159.        [style.margin.px]='imageMargin'>
160.        </div>
161.        <div class="caption">
162.        <div>
163.        <a [routerLink]="[products[i+2].productId]">
164.        {{products[i+2].productName}} </a>
165.        </div>
166.        <div>{{ products[i+2].price | currency:'INR':'symbol':'1.2-2'}}
167.        </div>
168.        <div></div>
169.        <ratings class="ratingcolor" [rate]='products[i+2].rating'></ratings>
170.        <div></div>
171.        <div>
172.        <button (click)="addCart(products[i+2].productId)" class="btn btn-primary">Add to
    Cart</button>
173.        </div>
174.        </div>
175.        </div>
176.        </div>
177.        <div class="col-xs-3">
178.        <div *ngIf="products[i+3]" class="thumbnail text-center">
179.        <div>
```

```
180.        <img [src]='products[i+3].imageUrl' [title]='products[i+3].productName'
    [style.width.px]='imageWidth' [style.height.px]='imageHeight'
181.        [style.margin.px]='imageMargin'>
182.        </div>
183.        <div class="caption">
184.        <div>
185.        <a [routerLink]="[products[i+3].productId]">
186.        {{products[i+3].productName}} </a>
187.        </div>
188.
189.        <div>{{ products[i+3].price | currency:'INR':'symbol':'1.2-2'}}
190.        </div>
191.        <div></div>
192.        <ratings class="ratingcolor" [rate]='products[i+3].rating'></ratings>
193.        <div></div>
194.        <div>
195.        <button (click)="addCart(products[i+3].productId)" class="btn btn-primary">Add to
    Cart</button>
196.        </div>
197.        </div>
198.        </div>
199.        </div>
200.        </div>
201.        </div>
202.        </div>
203.        <br/><br/>
204.        </div>
205.        </div>
206.        </div>
207.        </div>
```

Line 125: ratings is another component we are loading here in ProductList Component to display rating of a product. We are passing rating of each product to rate property of Ratingcomponent

Below is the code for **rating.component.ts** file, under products folder. Let us explore and understand the code for RatingComponent

```
1.  import { Component, Input, Output, EventEmitter } from '@angular/core';
2.
3.  @Component({
4.  selector: 'ratings',
5.  template: `
6.  <span *ngFor="let r of range; let i = index">
7.  <i class="glyphicon" [ngClass]="i < rate ? 'glyphicon-star' : 'glyphicon-star-empty'"></i>
```

```
8.  </span>
9.  `
10. })
11. export class RatingComponent {
12. private range: Array<number> = [1, 2, 3, 4, 5];
13. @Input() private rate: number;
14. }
```

Line 12: range property is an array which holds 1 to 5 values which represents rating

Line 13: rate is an input property which receives value from its parent component. It receives rating of each product

Line 6: ngFor loop will iterate over range of values given

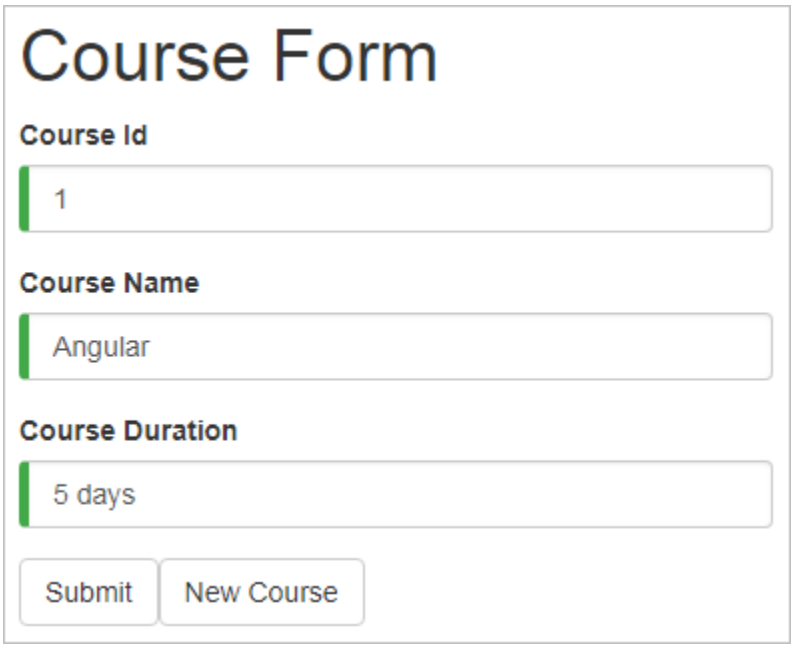
Line 7: It displays filled star symbol or empty star symbol based on the rating. We have used bootstrap css classes to display star symbols.

<u>Demo 32: Template Driven Forms</u>

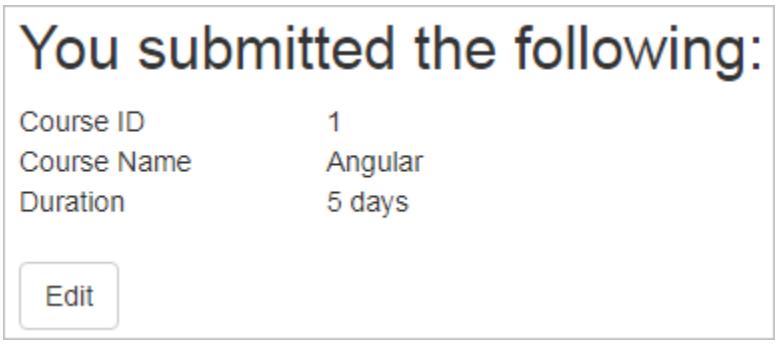
Highlights:

- Creating a template driven form
- Understanding the usage of ngForm

Problem Statement: Creating a course registration form as a template driven form as shown below



When submit button is clicked, it will show the course details



1. Create **course.ts** file under course-form folder and add the following code

```
export class Course {
  constructor(
    public courseId: number,
    public courseName: string,
    public duration: string
  ) { }
}
```

2. Add the following code in **course-form.component.ts** file

```
import { Component } from '@angular/core';
import { Course } from './course';

@Component({
  selector: 'app-course-form',
  templateUrl: './course-form.component.html',
  styleUrls: ['./course-form.component.css']
})
export class CourseFormComponent {

  course = new Course(1, 'Angular', '5 days');
  submitted = false;

  onSubmit() { this.submitted = true; }

}
```

3. Install **bootstrap**

   D:\MyApp>npm install bootstrap@3.3.7 –save

4. Include boostrap.min.css file in **angular.json** file as shown below

```
...
"styles": [
      "styles.css",
      "./node_modules/bootstrap/dist/css/bootstrap.min.css"
     ],
...
```

5. Write the below given code in **course-form.component.html**

```
<div class="container">


  <div [hidden]="submitted">

    <h1>Course Form</h1>

    <form (ngSubmit)="onSubmit()" #courseForm="ngForm">


      <div class="form-group">

        <label for="id">Course Id</label>

        <input type="text" class="form-control" required [(ngModel)]="course.courseId" name="id"
#id="ngModel">

        <div [hidden]="id.valid || id.pristine" class="alert alert-danger">

          Course Id is required

        </div>

      </div>


      <div class="form-group">

        <label for="name">Course Name</label>

        <input type="text" class="form-control" required [(ngModel)]="course.courseName"
name="name" #name="ngModel">

        <div [hidden]="name.valid || name.pristine" class="alert alert-danger">

          Course Name is required

        </div>

      </div>


      <div class="form-group">

        <label for="duration">Course Duration</label>

        <input type="text" class="form-control" required [(ngModel)]="course.duration"
name="duration" #duration="ngModel">

        <div [hidden]="duration.valid || duration.pristine" class="alert alert-danger">
```

```
        Duration is required
      </div>
    </div>


    <button type="submit" class="btn btn-default"
[disabled]="!courseForm.form.valid">Submit</button>
    <button type="button" class="btn btn-default" (click)="courseForm.reset()">New
Course</button>
  </form>
 </div>


 <div [hidden]="!submitted">
  <h2>You submitted the following:</h2>
  <div class="row">
    <div class="col-xs-3">Course ID</div>
    <div class="col-xs-9  pull-left">{{ course.courseId }}</div>
  </div>
  <div class="row">
    <div class="col-xs-3">Course Name</div>
    <div class="col-xs-9 pull-left">{{ course.courseName }}</div>
  </div>
  <div class="row">
    <div class="col-xs-3">Duration</div>
    <div class="col-xs-9 pull-left">{{ course.duration }}</div>
  </div>
  <br>
  <button class="btn btn-default" (click)="submitted=false">Edit</button>
 </div>
</div>
```

6. Write the below given code in **course-form.component.css**

```
input.ng-valid[required]  {
  border-left: 5px solid #42A948; /* green */
}

input.ng-dirty.ng-invalid:not(form)  {
  border-left: 5px solid #a94442; /* red */
}
```

7. Write the below given code in **app.component.html**

```
<app-course-form></app-course-form>
```

8. Save the files and check the output in the browser

<u>Demo: Reactive Forms</u>

Highlights:

- Creating a reactive form
- Understanding FormBuilder in creating a reactive form

Problem Statement: Creating an employee registration form as a reactive form as shown below

Registration Form

First Name

This field is required!

Last Name

This field is required!

Street

Zip

City

Submit

It renders the values at the bottom after clicking submit button

Registration Form

First Name

James

Last Name

Gosling

Street

ABC Street

Zip

457899

City

New York

Submit

Employee Details

First Name: James

Last Name: Gosling

Street: ABC Street

Zip: 457899

City: New York

1. Write the below given code in **app.module.ts**

```
import { BrowserModule } from '@angular/platform-browser';
import { NgModule } from '@angular/core';
import { ReactiveFormsModule } from '@angular/forms';

import { AppComponent } from './app.component';
```

```
import { RegistrationFormComponent } from './registration-form/registration-form.component';

@NgModule({
  declarations: [
    AppComponent,
    RegistrationFormComponent
  ],
  imports: [
    BrowserModule,
    ReactiveFormsModule
  ],
  providers: [],
  bootstrap: [AppComponent]
})
export class AppModule { }
```

2. Create a component called **RegistrationForm** using the following CLI command

   D:\MyApp>ng generate component RegistrationForm

3. Add the following code in **registration-form.component.ts** file

```
import { Component, OnInit } from '@angular/core';
import { FormBuilder, FormGroup, Validators } from '@angular/forms';

@Component({
  selector: 'app-registration-form',
  templateUrl: './registration-form.component.html',
  styleUrls: ['./registration-form.component.css']
})
export class RegistrationFormComponent implements OnInit {
```

```
  registerForm: FormGroup;

  constructor(private formBuilder: FormBuilder) { }

  ngOnInit() {
    this.registerForm = this.formBuilder.group({
      firstName: ['', Validators.required],
      lastName: ['', Validators.required],
      address: this.formBuilder.group({
        street: [],
        zip: [],
        city: []
      })
    });
  }

}
```

4. Write the below given code in **registration-form.component.html**

```
<div class="container">
  <h1>Registration Form</h1>
  <form [formGroup]="registerForm">
    <div class="form-group">
      <label>First Name</label>
      <input type="text" class="form-control" formControlName="firstName">
      <p *ngIf="registerForm.controls.firstName.errors" class="alert alert-danger">This field is required!</p>
    </div>
```

```
    <div class="form-group">
      <label>Last Name</label>
      <input type="text" class="form-control" formControlName="lastName">
      <p *ngIf="registerForm.controls.lastName.errors" class="alert alert-danger">This field is
required!</p>
    </div>
    <div class="form-group">
      <fieldset formGroupName="address">
        <label>Street</label>
        <input type="text" class="form-control" formControlName="street">
        <label>Zip</label>
        <input type="text" class="form-control" formControlName="zip">
        <label>City</label>
        <input type="text" class="form-control" formControlName="city">
      </fieldset>
    </div>
    <button type="submit" (click)="submitted=true">Submit</button>
  </form>
<br/>
  <div [hidden]="!submitted">
    <h3> Employee Details </h3>
    <p>First Name: {{ registerForm.get('firstName').value }} </p>
    <p> Last Name: {{ registerForm.get('lastName').value }} </p>
    <p> Street: {{ registerForm.get('address.street').value }}</p>
    <p> Zip: {{ registerForm.get('address.zip').value }} </p>
    <p> City: {{ registerForm.get('address.city').value }}</p>
  </div>
</div>
```

5. Write the below given code in **registration-form.component.css**

```
.ng-valid[required]  {
  border-left: 5px solid #42A948; /* green */
}

.ng-invalid:not(form)  {
  border-left: 5px solid #a94442; /* red */
}
```

6. Write the below given code in **app.component.html**

```
<app-registration-form></app-registration-form>
```

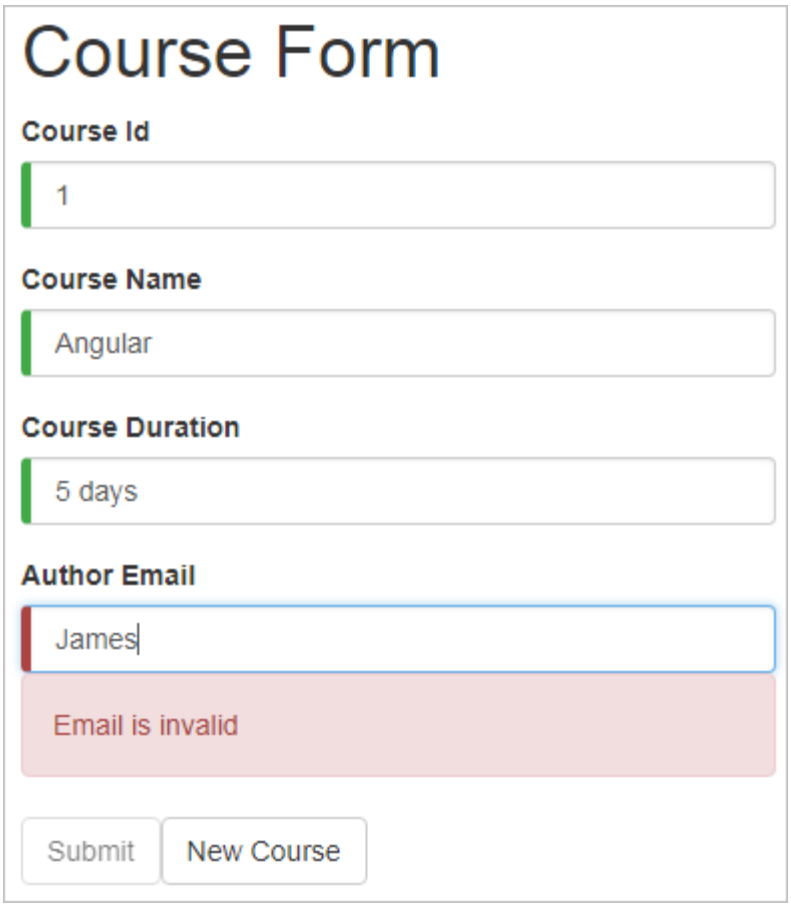
7. Save the files and check the output in the browser

<u>Demo: Custom Validators in Template driven forms</u>

Highlights:

- Creating a custom validator for a template driven form
- Applying a custom validator to an HTML element

Problem Statement: Creating a custom validator for email field in the course registration form as shown below



1. Write the code given below in **course.ts**

```
export class Course {
  constructor(
    public courseId: number,
    public courseName: string,
    public duration: string,
    public email: string
  ) { }
```

```
}
```

2. In **course-form.component.ts** file, pass default value to email field as shown below

```
import { Component } from '@angular/core';
import { Course } from './course';

@Component({
  selector: 'app-course-form',
  templateUrl: './course-form.component.html',
  styleUrls: ['./course-form.component.css']
})
export class CourseFormComponent {

  course = new Course(1, 'Angular', '5 days','james@gmail.com');
  submitted = false;

  onSubmit() { this.submitted = true; }

}
```

3. Create a file with name **email.validator.ts** under course-form folder to implement custom validation functionality for email field

```
import { Directive } from '@angular/core';
import { NG_VALIDATORS, FormControl, Validator } from '@angular/forms';

@Directive({
   selector: '[validateEmail]',
   providers: [
      { provide: NG_VALIDATORS, useExisting: EmailValidator, multi: true }
   ]
```

```
})
export class EmailValidator implements Validator {

  validate(control: FormControl): { [key: string]: any } {
    const emailRegexp = /^([a-zA-Z0-9_\-\.]+)@([a-zA-Z0-9_\-\.]+)\.([a-zA-Z]{2,5})$/;
    if (!emailRegexp.test(control.value)) {
      return { "emailValid": true };
    }
    return null;
  }
}
```

4. Add EmailValidator class in the root module i.e., **app.module.ts** as shown below

```
import { BrowserModule } from '@angular/platform-browser';
import { NgModule } from '@angular/core';
import { FormsModule } from '@angular/forms';

import { AppComponent } from './app.component';
import { CourseFormComponent } from './course-form/course-form.component';
import { EmailValidator } from './course-form/email.validator';

@NgModule({
 declarations: [
  AppComponent,
  CourseFormComponent,
  EmailValidator
 ],
 imports: [
  BrowserModule,
```

```
    FormsModule
  ],
  providers: [],
  bootstrap: [AppComponent]
})
export class AppModule { }
```

5. Add the following code in **course-form.component.html** file for email field as shown below

```
<div class="container">

  <div [hidden]="submitted">
    <h1>Course Form</h1>

    <form (ngSubmit)="onSubmit()" #courseForm="ngForm">

      <div class="form-group">
        <label for="id">Course Id</label>
        <input type="text" class="form-control" required [(ngModel)]="course.courseId" name="id"
#id="ngModel">
        <div [hidden]="id.valid || id.pristine" class="alert alert-danger">
          Course Id is required</div>
      </div>

      <div class="form-group">
        <label for="name">Course Name</label>
        <input type="text" class="form-control" required [(ngModel)]="course.courseName"
          minlength="4" name="name" #name="ngModel">
        <div *ngIf="name.errors && (name.dirty || name.touched)" class="alert alert-danger">
          <div [hidden]="!name.errors.required">Name is required</div>
```

```
        <div [hidden]="!name.errors.minlength">Name must be at least 4 characters long.</div>
      </div>
    </div>

    <div class="form-group">
      <label for="duration">Course Duration</label>
      <input type="text" class="form-control" required [(ngModel)]="course.duration"
        name="duration" #duration="ngModel">
      <div [hidden]="duration.valid || duration.pristine" class="alert alert-danger">Duration is
required</div>
    </div>

    <div class="form-group">
      <label for="email">Author Email</label>
      <input type="text" class="form-control" required [(ngModel)]="course.email"
        name="email" #email="ngModel" validateEmail>

      <div *ngIf="email.errors && (email.dirty || email.touched)" class="alert alert-danger">
        <div [hidden]="!email.errors.required">Email is required</div>
        <div [hidden]="!email.errors.emailValid">Email is invalid</div>
      </div>
    </div>

    <button type="submit" class="btn btn-default"
[disabled]="!courseForm.form.valid">Submit</button>
    <button type="button" class="btn btn-default" (click)="courseForm.reset()">New
          Course</button>
  </form>
 </div>
```

```
<div [hidden]="!submitted">
  <h2>You submitted the following:</h2>
  <div class="row">
    <div class="col-xs-3">Course ID</div>
    <div class="col-xs-9  pull-left">{{ course.courseId }}</div>
  </div>
  <div class="row">
    <div class="col-xs-3">Course Name</div>
    <div class="col-xs-9 pull-left">{{ course.courseName }}</div>
  </div>
  <div class="row">
    <div class="col-xs-3">Duration</div>
    <div class="col-xs-9 pull-left">{{ course.duration }}</div>
  </div>
  <br>
  <button class="btn btn-default" (click)="submitted=false">Edit</button>
 </div>
</div>
```

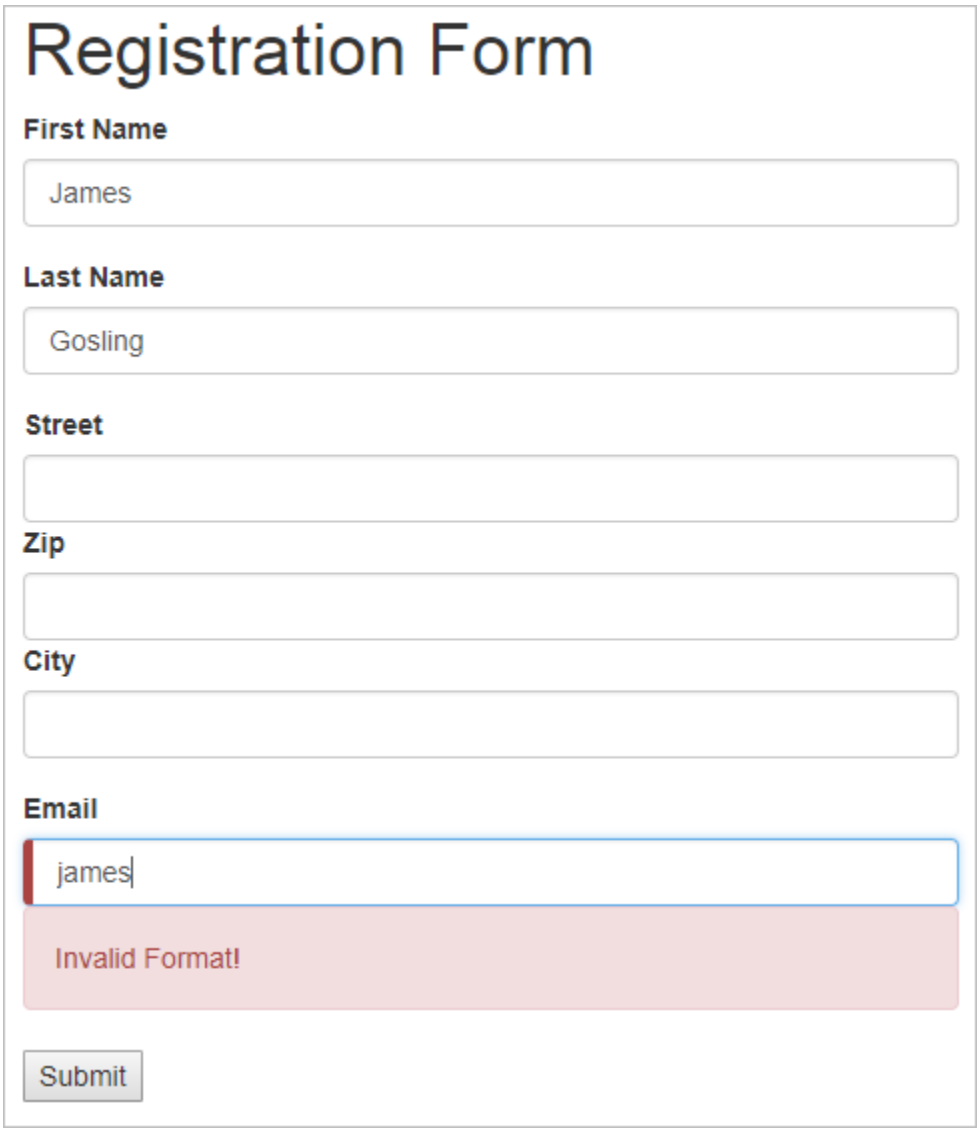
6. Save the files and check the output in the browser

Demo: Custom Validators in reactive forms

Highlights:

- Creating a custom validator for a reactive form
- Applying custom validator to a HTML field

Problem Statement: Creating a custom validator for email field in employee registration form (reactive form) as shown below



1. Write a separate function in **registration-form.component.ts** for custom validation as shown below.

   import { Component, OnInit } from '@angular/core';

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

```
@Component({
  selector: 'app-registration-form',
  templateUrl: './registration-form.component.html',
  styleUrls: ['./registration-form.component.css']
})
export class RegistrationFormComponent implements OnInit {

  registerForm: FormGroup;

  constructor(private formBuilder: FormBuilder) { }

  ngOnInit() {
    this.registerForm = this.formBuilder.group({
      firstName: ['', Validators.required],
      lastName: ['', Validators.required],
      address: this.formBuilder.group({
        street: [],
        zip: [],
        city: []
      }),
      email:['',validateEmail]
    });
  }
}

function validateEmail(c: FormControl) {
    let EMAIL_REGEXP = /^([a-zA-Z0-9_\-\.]+)@([a-zA-Z0-9_\-\.]+)\.([a-zA-Z]{2,5})$/;

    return EMAIL_REGEXP.test(c.value) ? null : {
```

```
      emailValid: {
        valid: false
      }
    };
}
```

2. Add html controls for email field in **registration-form.component.html** file as shown below

```
<div class="container">
  <h1>Registration Form</h1>
  <form [formGroup]="registerForm">
    <div class="form-group">
      <label>First Name</label>
      <input type="text" class="form-control" formControlName="firstName">
      <p *ngIf="registerForm.controls.firstName.errors" class="alert alert-danger">This field is required!</p>
    </div>
    <div class="form-group">
      <label>Last Name</label>
      <input type="text" class="form-control" formControlName="lastName">
      <p *ngIf="registerForm.controls.lastName.errors" class="alert alert-danger">This field is required!</p>
    </div>
    <div class="form-group">
      <fieldset formGroupName="address">
        <label>Street</label>
        <input type="text" class="form-control" formControlName="street">
        <label>Zip</label>
        <input type="text" class="form-control" formControlName="zip">
        <label>City</label>
```

```
      <input type="text" class="form-control" formControlName="city">
     </fieldset>
   </div>
   <div class="form-group">
                                <label>Email</label> <input type="text" class="form-control"
formControlName="email">
                                <p *ngIf="registerForm.controls.email.errors?.emailValid" class="alert
alert-danger">Invalid Format!</p>
                        </div>
   <button type="submit" (click)="submitted=true">Submit</button>
  </form>
<br/>
  <div [hidden]="!submitted">
   <h3> Employee Details </h3>
   <p>First Name: {{ registerForm.get('firstName').value }} </p>
   <p> Last Name: {{ registerForm.get('lastName').value }} </p>
   <p> Street: {{ registerForm.get('address.street').value }}</p>
   <p> Zip: {{ registerForm.get('address.zip').value }} </p>
   <p> City: {{ registerForm.get('address.city').value }}</p>
   <p> Email: {{ registerForm.get('email').value }} </p>
  </div>
</div>
```

3. Save the files and check the output in the browser

Demo: Services

Highlights:

- Creating a service
- Injecting a service into a component

Problem Statement: Create a Book Component which fetches book details like id, name and displays them on the page in a list format. Store the book details in an array and fetch the data using custom service. Output is as shown below

1. Create **BookComponent** by using the following CLI command

   D:\MyApp>ng generate component book

2. Create a file with name **book.ts** under book folder and add the following code.

```
export class Book {
   id: number;
   name: string;
}
```

3. Create a file with name **books-data.ts** under book folder and add the following code.

```
import { Book } from './book';
```

```
export var BOOKS: Book[] = [
   { "id": 1, "name": "HTML 5" },
   { "id": 2, "name": "CSS 3" },
   { "id": 3, "name": "Java Script" },
   { "id": 4, "name": "Ajax Programming" },
   { "id": 5, "name": "jQuery" },
   { "id": 6, "name": "Mastering Node.js" },
   { "id": 7, "name": "Angular JS 1.x" },
   { "id": 8, "name": "ng-book 2" },
   { "id": 9, "name": "Backbone JS" },
   { "id": 10, "name": "Yeoman" }
];
```

4. Create a service called **BookService** under book folder using the following CLI command

   D:\MyApp\src\app\book>ng generate service book --module=app

5. Add the following code in **book.service.ts**

```
import { Injectable } from '@angular/core';
import { Book } from './book';
import { BOOKS } from './books-data';

@Injectable()
export class BookService {

  getBooks() {
    return Promise.resolve(BOOKS);
  }
}
```

6. Add the following code in **book.component.ts** file

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

import { Book } from './book';
import { BookService } from './book.service';

@Component({
  selector: 'app-book',
  templateUrl: './book.component.html',
  styleUrls: ['./book.component.css']
})
export class BookComponent implements OnInit {

  books: Book[];

  constructor(private bookService: BookService) { }

  getBooks() {
    this.bookService.getBooks().then(books => this.books = books);
  }

  ngOnInit() {
    this.getBooks();
  }
}
```

7. Write the below given code in **book.component.html**

```
<h2>My Books</h2>
<ul class="books">
  <li *ngFor="let book of books">
    <span class="badge">{{book.id}}</span> {{book.name}}
  </li>
</ul>
```

8. Add the following code in **book.component.css** which has styles for books

```
.books {
    margin: 0 0 2em 0;
    list-style-type: none;
    padding: 0;
    width: 15em;
}
.books li {
    cursor: pointer;
    position: relative;
    left: 0;
    background-color: #EEE;
    margin: .5em;
    padding: .3em 0;
    height: 1.6em;
    border-radius: 4px;
}
.books li:hover {
    color: #607D8B;
    background-color: #DDD;
    left: .1em;
}
```

```
.books .badge {
    display: inline-block;
    font-size: small;
    color: white;
    padding: 0.8em 0.7em 0 0.7em;
    background-color: #607D8B;
    line-height: 1em;
    position: relative;
    left: -1px;
    top: -4px;
    height: 1.8em;
    margin-right: .8em;
    border-radius: 4px 0 0 4px;
}
```

9. Add the following code in app.component.html

   ```
   <app-book></app-book>
   ```

10. Save the files and check the output in the browser

<u>Demo: Server Communication using httpClient</u>

Highlights:

- Understanding communication with server
- Exploring HttpClient class

Problem Statement: Fetching books data using HttpClient class. Output is as shown below

1. In the example used for custom services concept, add HttpModule to the **app.module.ts** to make use of Http class.

```
import { NgModule } from '@angular/core';

import { BrowserModule } from '@angular/platform-browser';

import { HttpClientModule } from '@angular/common/http';


import { AppComponent } from './app.component';

import { BookComponent } from './book/book.component';

import { BookService } from './book/book.service';


@NgModule({

  imports: [BrowserModule, HttpClientModule],
```

```
  declarations: [AppComponent, BookComponent],
  providers: [BookService],
  bootstrap: [AppComponent]
})
export class AppModule { }
```

2. Create **books.json** under assets folder and move books data to it

```
[
   {   "id": 1, "name": "HTML 5"  },
   {   "id": 2, "name": "CSS 3"   },
   {   "id": 3, "name": "Java Script"  },
   {   "id": 4, "name": "Ajax Programming" },
   {   "id": 5, "name": "jQuery"  },
   {   "id": 6, "name": "Mastering Node.js"  },
   {   "id": 7, "name": "Angular JS 1.x"  },
   {   "id": 8, "name": "ng-book 2" },
   {   "id": 9, "name": "Backbone JS" },
   {   "id": 10, "name": "Yeoman" }
]
```

3. Add the following code in **book.service.ts** file

```
import { Injectable } from '@angular/core';
import { HttpClient } from '@angular/common/http';
import { catchError, tap } from 'rxjs/operators';
import { Observable } from 'rxjs';
import { HttpErrorResponse } from '@angular/common/http';

import { Book } from './book';
```

```
@Injectable()
export class BookService {

  private booksUrl = './assets/books.json';

  constructor(private http: HttpClient) { }

  getBooks(): Observable<Book[]> {
    return this.http.get<Book[]>(this.booksUrl).pipe(
      tap(data => console.log('Data fetched:'+JSON.stringify(data))),
      catchError(this.handleError));
  }
  private handleError(err:HttpErrorResponse) {
    let errMsg:string='';
    if (err.error instanceof Error) {
       // A client-side or network error occurred. Handle it accordingly.
       console.log('An error occurred:', err.error.message);
        errMsg=err.error.message;}
       else {
       // The backend returned an unsuccessful response code.
       // The response body may contain clues as to what went wrong,
       console.log(`Backend returned code ${err.status}`);
          errMsg=err.error.status;
     }
        return Observable.throw(errMsg);
  }
}
```

4. Write the code given below in **book.component.ts**

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

import { Book } from './book';
import { BookService } from './book.service';

@Component({
  selector: 'app-book',
  templateUrl: './book.component.html',
  styleUrls: ['./book.component.css']
})
export class BookComponent implements OnInit {

  books: Book[];
  errorMessage: string;

  constructor(private bookService: BookService) { }

  getBooks() {
    this.bookService.getBooks().subscribe(
      books => this.books = books,
      error => this.errorMessage = <any>error);
  }

  ngOnInit() {
    this.getBooks();
  }
}
```

5. Write the code given below in **book.component.html**

```
<h2>My Books</h2>
<ul class="books">
  <li *ngFor="let book of books">
    <span class="badge">{{book.id}}</span> {{book.name}}
  </li>
</ul>

<div class="error" *ngIf="errorMessage">{{errorMessage}}</div>
```

6. Save the files and check the output in the browser

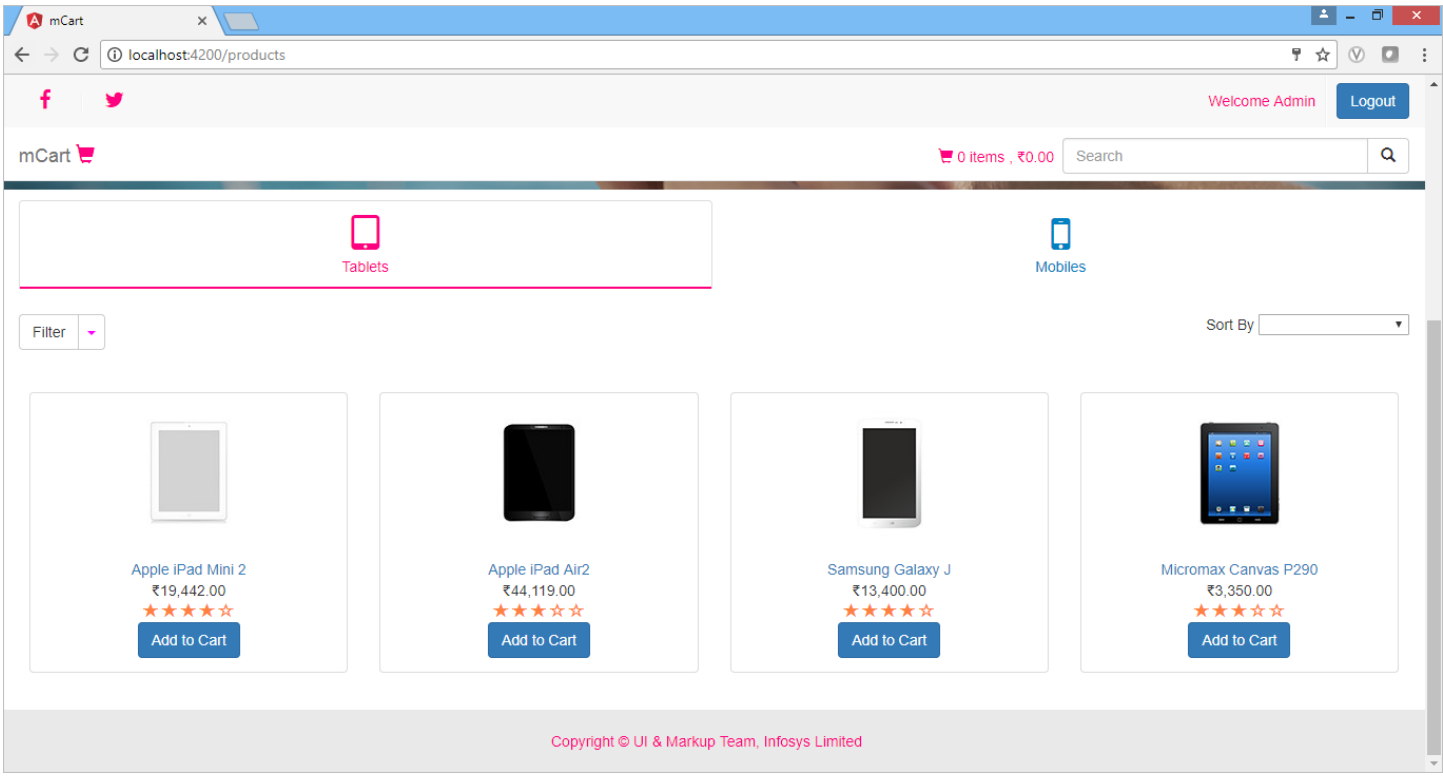
<u>Demo 39: Services in mCart</u>

Highlights:

- Exploring services in mCart application
- Fetching data from json files to mCart application

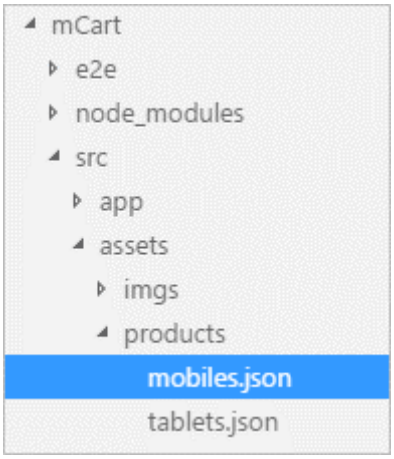
Now let us understand how our mCart application fetches the data. We have created a custom service called **ProductService** in which we have used Http class to fetch data stored in the json files

Observe below the output of ProductListComponent,



Here, the products list is displayed by making http call to the json file.

The json files are stored under the assets folder.



Observe the data present inside the json files.

mobiles.json

```
[
    {
        "productId": 1,
        "productName": "Samsung Galaxy Note 7",
```

"productCode": "MOB-120",
"description": "64GB, Coral Blue",
"price": 60569,
"imageUrl": "assets/imgs/samsung_note7_coralblue.JPG",
"manufacturer": "Samsung",
"ostype": "Android",
"rating": 4
},
{
"productId": 2,
"productName": "Samsung Galaxy Note 7",
"productCode": "MOB-124",
"description": "64GB, Gold",
"price": 60200,
"imageUrl": "assets/imgs/samsung_note7_gold.JPG",
"manufacturer": "Samsung",
"ostype": "Android",
"rating": 4
},
...

tablets.json

[
{
"productId": 1,
"productName": "Apple iPad Mini 2",
"productCode": "TAB-120",
"description": "16GB, White",
"price": 19442,
"imageUrl": "assets/imgs/apple_ipad_mini.jpg",

```
        "manufacturer": "Apple",
        "ostype": "iOS",
        "rating": 4
    },
    {
        "productId": 2,
        "productName": "Apple iPad Air2",
        "productCode": "TAB-124",
        "description": "64GB, Black",
        "price": 44119,
        "imageUrl": "assets/imgs/ipad_air.jpg",
        "manufacturer": "Apple",
        "ostype": "iOS",
        "rating": 3
    },
    ...
```

Let us explore the methods present in **product.service.ts** file from products folder. Observe the code given below

```
import { Injectable } from '@angular/core';
import { HttpClient, HttpErrorResponse } from '@angular/common/http';
import { Observable } from 'rxjs';
import { catchError } from 'rxjs/operators';
import { map } from 'rxjs/operators';

import { Product } from './product';

@Injectable()
export class ProductService {
```

```
11. private _productUrl = 'assets/products/mobiles.json';
12. selectedProducts: any = [];
13. products: any = [];
14. producttype = 'tablet';
15. username: string;
16.
17. constructor(private http: HttpClient) {
18. if (sessionStorage.getItem('selectedProducts')) {
19. this.selectedProducts = JSON.parse(sessionStorage.getItem('selectedProducts'));
20. }
21. }
22.
23. getProducts(): Observable<Product[]> {
24. if (this.producttype === 'tablet') {
25. return this.http.get<Product[]>('assets/products/tablets.json').pipe(
        a.  catchError(this.handleError));
26. } else if (this.producttype === 'mobile') {
27. return this.http.get<Product[]>('assets/products/mobiles.json').pipe(
        a.  catchError(this.handleError));
28. }
29. }
30.
31. getProduct(id: number): Observable<Product> {
32. return this.getProducts()
33. .map(products => products.filter(product => product.productId === id)[0]);
34. }
35.
36. private handleError(err: HttpErrorResponse) {
37. console.log(err);
```

```
38. return Observable.throw(err.error() || 'Server error');
39. }
40. }
```

Line 2-5: We have imported HttpClient, HttpErrorResponse, Observable and its operators

Line 9: @Injectable() decorator makes the class as a service class which can be injected into other classes in the application

Line 17: We are injecting HttpClient class to make asynchronous calls to json files

Line 23: getProducts() method contains functionality to fetch products data from json files

Line 25 : Based on the product type selected, it makes http call to tablets.json or mobiles.json

Line 34-37: getProduct() method is to fetch a particular product details from json file

Line 39-42: handleError() is a error handling method which throws the error back to the component

Now open **products.module.ts** to explore adding service class to the module

```
import { NgModule } from '@angular/core';
import { CommonModule } from '@angular/common';
import { FormsModule } from '@angular/forms';

import { ProductsRoutingModule } from './products-routing.module';
import { ProductListComponent } from './product-list/product-list.component';
import { ProductDetailComponent } from './product-detail/product-detail.component';
import { CartComponent } from './cart/cart.component';
import { OrderByPipe } from './product-list/orderby.pipe';
import { RatingComponent } from './rating.component';
import { ProductService } from './product.service';
import { AuthGuardService } from './auth-guard.service';

@NgModule({
imports: [
```

```
16. CommonModule,
17. FormsModule,
18. ProductsRoutingModule
19. ],
20. declarations:
    [ProductListComponent,ProductDetailComponent,CartComponent,OrderByPipe,RatingC
    omponent],
21. providers:[ProductService,AuthGuardService]
22. })
23. export class ProductsModule { }
```

Line 11: Imports ProductService class

Line 21: Add it to the providers property to make it available in the entire module

Now open **product-list.component.ts** file to explore injecting a product service class

```
1. import { Component, Input } from '@angular/core';
2.
3. import { ProductService } from '../product.service';
4. import { Cart } from '../cart/Cart';
5. import { Product } from '../product';
6.
7. @Component({
8. templateUrl: 'product-list.component.html',
9. styleUrls: ['product-list.component.css']
10. })
11. export class ProductListComponent {
12. rate: number;
13. pageTitle: string = 'mCart';
14. imageWidth: number = 80;
15. imageHeight: number = 120;
```

```
imageMargin: number = 12;
showImage: boolean = false;
listFilter: string;
manufacturers = [{ "id": "Samsung", "checked": false },
{ "id": "Microsoft", "checked": false },
{ "id": "Apple", "checked": false },
{ "id": "Micromax", "checked": false }
];
os = [{ "id": "Android", "checked": false },
{ "id": "Windows", "checked": false },

{ "id": "iOS", "checked": false }];
price_range = [{ "id": "3000-5000", "checked": false },
{ "id": "13000-15000", "checked": false },
{ "id": "19000-35000", "checked": false },
{ "id": "40000-70000", "checked": false }];
errorMessage: string;
products: any = [];
selectedItems: any = 0;
cart: Cart;
total: number = 0;
orderId: number = 0;
selectedManufacturers: string[] = [];
selectedOStypes: string[];
selectedPrice: string[];
checkedManufacturers: any[];
checkedOS: any[];
checkedPrice: any[];
sub: any;
```

```
45. i: number = 0;
46. sortoption: string = "";
47. chkmanosprice: any = [];
48.
49. constructor(private _productService: ProductService) {
50. document.getElementById("login").style.display = "";
51. document.getElementById("login").innerHTML ="Logout";
52. sessionStorage.setItem("loginTitle","Logout");
53. this.orderId++;
54. document.getElementById("welcome").style.display = "";
55. document.getElementById("welcome").innerHTML = "Welcome " +
    sessionStorage.getItem("username");
56. document.getElementById("welcome").style.color = "#ff0080";
57. document.getElementById("welcome").style.position = "relative";
58. document.getElementById("welcome").style.top = "8px";
59. this._productService.getProducts()
60. .subscribe(
61. products => {
62. this._productService.products = products;
63. this.products = this._productService.products;
64. },
65. error => this.errorMessage = <any>error);
66.
67. if (_productService.selectedProducts.length > 0) {
68. this.selectedItems = Number(sessionStorage.getItem("selectedItems"));
69. this.total = Number(sessionStorage.getItem("grandTotal"));
70. }
71. }
```

Line 49: ProductService is injected into the component

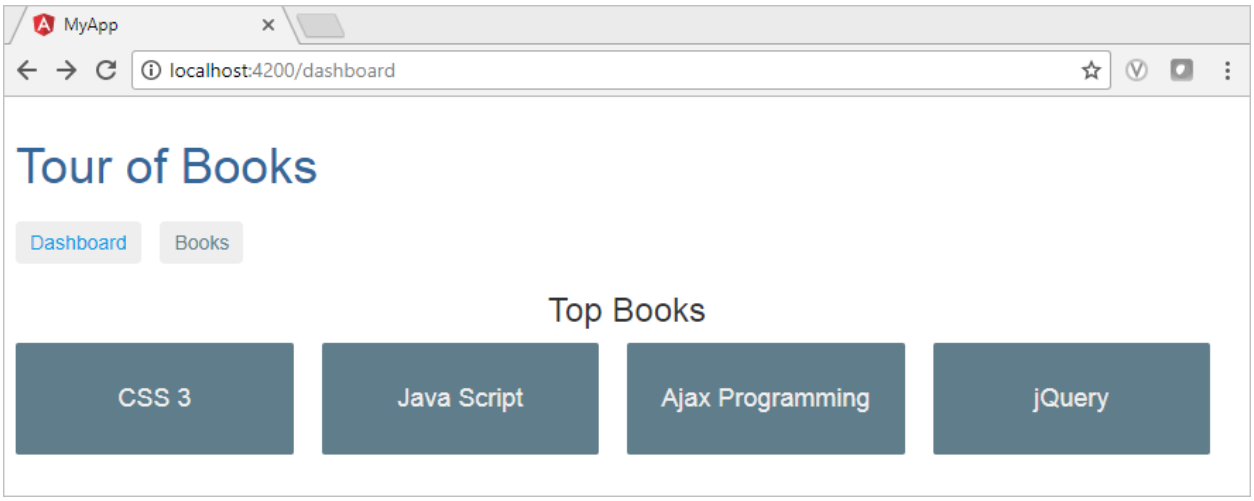
Line 59-65: We are invoking getProducts() method from ProductService calss which returns an observable response. On top of it, we have subscribe() method which will assign products data into local property on successful response or assigns error thrown into local property called errorMessage on error response

Demo: Routing

Highlights:

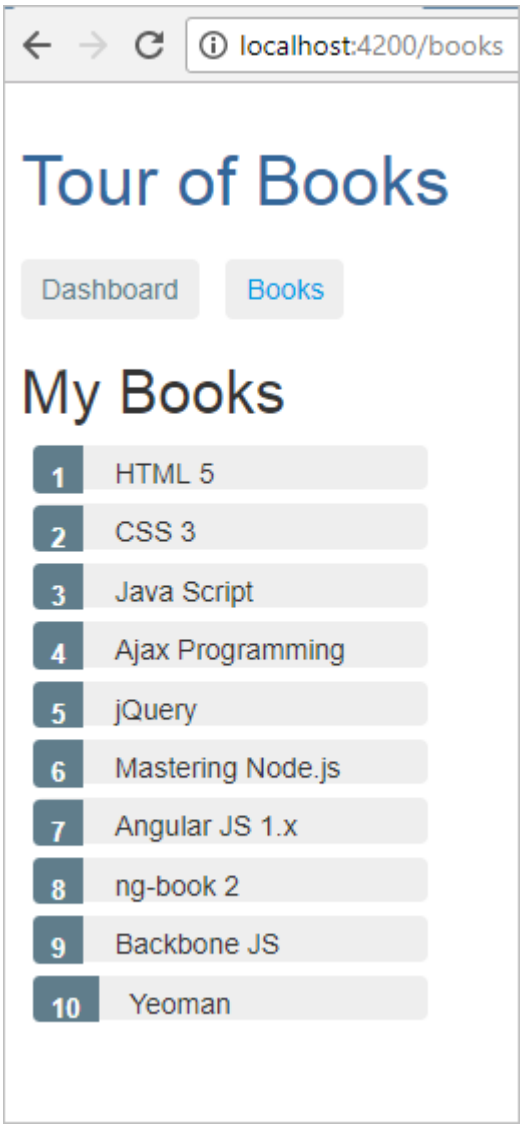
- Navigating between multiple views
- Understanding routing in an angular application

Problem Statement: Creating multiple components and adding routing to provide navigation between them. Output is as shown below



When a specific book is clicked, it renders the BookDetailComponent as shown below

When books link is clicked, it navigates to BooksComponent as shown below



1. Consider the example used for Http concept.
2. Create another component with name dashboard using the following command

   D:\MyApp>ng generate component dashboard

3. Open **dashboard.component.ts** and add the following code

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

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


   import { Book } from '../book/book';

   import { BookService } from '../book/book.service';
   ```

```
@Component({
  selector: 'app-dashboard',
  templateUrl: './dashboard.component.html',
  styleUrls: ['./dashboard.component.css']
})
export class DashboardComponent implements OnInit {

  books: Book[] = [];
  constructor(
    private router: Router,
    private bookService: BookService) { }

  ngOnInit() {
    this.bookService.getBooks()
      .subscribe(books => this.books = books.slice(1, 5));
  }
  gotoDetail(book: Book) {
    this.router.navigate(['/detail', book.id]);
  }
}
```

4. Open **dashboard.component.html** and add the following code

```
<h3>Top Books</h3>
<div class="grid grid-pad">
  <div *ngFor="let book of books" (click)="gotoDetail(book)" class="col-1-4">
    <div class="module book">
      <h4>{{book.name}}</h4>
    </div>
```

```
  </div>
</div>
```

5. Open **dashboard.component.css** and add the following code

```
[class*='col-'] {
        float: left;
}

*, *:after, *:before {
        -webkit-box-sizing: border-box;
        -moz-box-sizing: border-box;
        box-sizing: border-box;
}

h3 {
        text-align: center;
        margin-bottom: 0;
}

[class*='col-'] {
        padding-right: 20px;
        padding-bottom: 20px;
}

[class*='col-']:last-of-type {
        padding-right: 0;
}

.grid {
```

```
        margin: 0;
}

.col-1-4 {
        width: 25%;
}

.module {
        padding: 20px;
        text-align: center;
        color: #eee;
        max-height: 120px;
        min-width: 120px;
        background-color: #607D8B;
        border-radius: 2px;
}

h4 {
        position: relative;
}

.module:hover {
        background-color: #EEE;
        cursor: pointer;
        color: #607d8b;
}

.grid-pad {
        padding: 10px 0;
```

```
}

.grid-pad>[class*='col-']:last-of-type {
        padding-right: 20px;
}

@media ( max-width : 600px) {
        .module {
                font-size: 10px;
                max-height: 75px;
        }
}

@media ( max-width : 1024px) {
        .grid {
                margin: 0;
        }
        .module {
                min-width: 60px;
        }
}
```

6. Create another component called **book-detail** using the following command

   D:\MyApp>ng generate component book-detail

7. Open **book.service.ts** and add **getbook()** method as shown below to fetch a specific book details

   ```
   import { Injectable } from '@angular/core';
   ```

```
import { HttpClient } from '@angular/common/http';
import { Observable } from 'rxjs';
import { Headers, RequestOptions } from '@angular/http';
import { tap, catchError, map } from 'rxjs/operators';

import { Book } from './book';

@Injectable()
export class BookService {

  private booksUrl = 'assets/books.json';

  constructor(private http: HttpClient) { }

  getBooks(): Observable<Book[]> {
    return this.http.get<Book[]>(this.booksUrl).pipe(
      .tap(data => console.log("All: " + JSON.stringify(data)))
      .catchError(this.handleError);
  }

  private handleError(error: any) {
    let errMsg = (error.message) ? error.message :
      error.status ? `${error.status} - ${error.statusText}` : 'Server error';
    console.error(errMsg);
    return Observable.throw(errMsg);
  }

 getBook(id: number) {
        return this.getBooks()
```

```
            .map(books => books.find(book => book.id === id));
    }
}
```

8. Open **book-detail.component.ts** and add the following code

```
import { Component, OnInit } from '@angular/core';
import { ActivatedRoute, ParamMap } from '@angular/router';
import { switchMap } from 'rxjs/operators';
import { Observable } from 'rxjs';

import { Book } from '../book/book';
import { BookService } from '../book/book.service';

@Component({
  selector: 'app-book-detail',
  templateUrl: './book-detail.component.html',
  styleUrls: ['./book-detail.component.css']
})
export class BookDetailComponent implements OnInit {

  book: Book;
  error: any;
  sub: any;

  constructor(private bookService: BookService, private route: ActivatedRoute) { }

  ngOnInit() {
    this.sub = this.route.paramMap.pipe(switchMap((params: ParamMap) =>
      this.bookService.getBook(+params.get('id'))))
```

```
    .subscribe(book => this.book = book);
  }

  ngOnDestroy() {
    this.sub.unsubscribe();
  }

  goBack() {
    window.history.back();
  }
}
```

9. Open **book-detail.component.html** and add the following code

```
<div *ngIf="book">
        <h2>{{book.name}} details!</h2>
        <div>
                <label>id: </label>{{book.id}}
        </div>
        <div>
                <label>name: </label> <input [(ngModel)]="book.name"
                        placeholder="name" />
        </div>
        <button (click)="goBack()">Back</button>
</div>
```

10. Open **book-detail.component.css** and add the following code

```
label {
  display: inline-block;
  width: 3em;
```

```
  margin: .5em 0;
  color: #607D8B;
  font-weight: bold;
}
input {
  height: 2em;
  font-size: 1em;
  padding-left: .4em;
}
button {
  margin-top: 20px;
  font-family: Arial;
  background-color: #eee;
  border: none;
  padding: 5px 10px;
  border-radius: 4px;
  cursor: pointer; cursor: hand;
}
button:hover {
  background-color: #cfd8dc;
}
button:disabled {
  background-color: #eee;
  color: #ccc;
  cursor: auto;
}
```

11. Write the below given code in **app-routing.module.ts**

```
import { NgModule } from '@angular/core';
```

```
import { RouterModule, Routes } from '@angular/router';

import { BookComponent } from './book/book.component';
import { DashboardComponent } from './dashboard/dashboard.component';
import { BookDetailComponent } from './book-detail/book-detail.component';

const appRoutes: Routes = [
  { path: 'dashboard', component: DashboardComponent },
  { path: '', redirectTo: '/dashboard', pathMatch: 'full' },
  { path: 'books', component: BookComponent },
  { path: 'detail/:id', component: BookDetailComponent }
];

@NgModule({
  imports: [
    RouterModule.forRoot(appRoutes)
  ],
  exports: [
    RouterModule
  ]
})
export class AppRoutingModule { }
```

12. Write the below given code in **app.module.ts**

```
import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
import { HttpClientModule } from '@angular/common/http';
import { FormsModule } from '@angular/forms';
```

```
import { AppComponent } from './app.component';
import { BookComponent } from './book/book.component';
import { BookService } from './book/book.service';
import { DashboardComponent } from './dashboard/dashboard.component';
import { BookDetailComponent } from './book-detail/book-detail.component';
import { AppRoutingModule } from './app-routing.module';

@NgModule({
  imports: [BrowserModule, HttpClientModule, FormsModule, AppRoutingModule],
  declarations: [AppComponent, BookComponent, DashboardComponent,
BookDetailComponent],
  providers: [BookService],
  bootstrap: [AppComponent]
})
export class AppModule { }
```

13. Write the below given code in **app.component.ts**

```
import { Component } from '@angular/core';

@Component({
  selector: 'app-root',
  styleUrls: ['./app.component.css'],
  templateUrl: './app.component.html'
})
export class AppComponent {
  title = 'Tour of Books';
}
```

14. Write the below given code in **app.component.html**

```
<h1>{{title}}</h1>
<nav>
  <a [routerLink]='["/dashboard"]' routerLinkActive="active">Dashboard</a>
  <a [routerLink]='["/books"]' routerLinkActive="active">Books</a>
</nav>
<router-outlet></router-outlet>
```

15. Open **app.component.css** and add the following code

```
/* Master Styles */
h1 {
  color: #369;
  font-family: Arial, Helvetica, sans-serif;
  font-size: 250%;
}
h2, h3 {
  color: #444;
  font-family: Arial, Helvetica, sans-serif;
  font-weight: lighter;
}
body {
  margin: 2em;
}
body, input[text], button {
  color: #888;
  font-family: Cambria, Georgia;
}
a {
  cursor: pointer;
  cursor: hand;
```

```
}
button {
  font-family: Arial;
  background-color: #eee;
  border: none;
  padding: 5px 10px;
  border-radius: 4px;
  cursor: pointer;
  cursor: hand;
}
button:hover {
  background-color: #cfd8dc;
}
button:disabled {
  background-color: #eee;
  color: #aaa;
  cursor: auto;
}

/* Navigation link styles */
nav a {
  padding: 5px 10px;
  text-decoration: none;
  margin-right: 10px;
  margin-top: 10px;
  display: inline-block;
  background-color: #eee;
  border-radius: 4px;
}
```

```
nav a:visited, a:link {
  color: #607D8B;
}
nav a:hover {
  color: #039be5;
  background-color: #CFD8DC;
}
nav a.active {
  color: #039be5;
}

/* everywhere else */
* {
  font-family: Arial, Helvetica, sans-serif;
}
```

16. Open **styles.css** under src folder and add the following code

```
/* You can add global styles to this file, and also import other style files */
body{
    padding:10px;
}
```

17. Open **book.component.ts** file in book folder and add the following code

```
import { Component, OnInit } from '@angular/core';
import { Router } from '@angular/router';

import { Book } from './book';
```

```
import { BookService } from './book.service';

@Component({
  selector: 'app-book',
  templateUrl: './book.component.html',
  styleUrls: ['./book.component.css']
})
export class BookComponent implements OnInit {

  books: Book[];
  errorMessage: string;
  selectedBook: Book;

  constructor(private router: Router, private bookService: BookService) { }

  getBooks() {
    this.bookService.getBooks().subscribe(
      books => this.books = books,
      error => this.errorMessage = <any>error);
  }

  ngOnInit() {
    this.getBooks();
  }
}
```

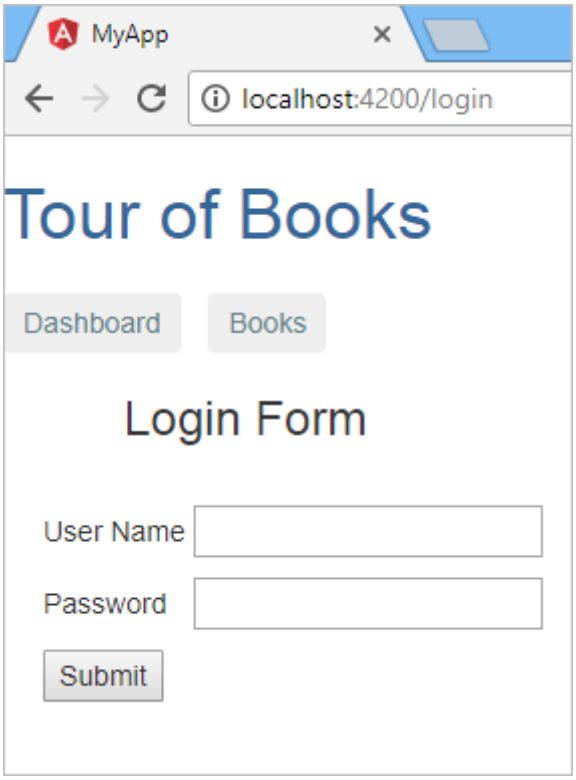
18. Save the files and check the output in the browser

Demo: Route guards

Highlights:

- Understanding routing guards
- Using guards to navigate

Problem Statement: Consider the same example used for routing. Add route guard to BooksComponent. Only after logging in , user should be able to access BooksComponent. If user tries to give url of Bookscomponent in another tab or window, or if user tries to reload the BooksComponent page, it should be redirected to LoginComponent. Output is as shown below



1. Add the following code to **login.component.html** file

```
<h3 style="position:relative;left:60px">Login Form</h3>

<div *ngIf="invalidCredentialMsg" style="color:red">{{invalidCredentialMsg}}</div><br/>

<div style="position:relative;left:20px">

        <form [formGroup]="loginForm" (ngSubmit)="onFormSubmit()">

                <p>User Name <input formControlName="username"></p>

                <p>Password <input type="password" formControlName="password"
style="position:relative;left:10px"></p>
```

```
                <p><button type="submit">Submit</button></p>
        </form>
</div>
```

2. Add the following code to **login.component.ts** file

```
import { Component } from '@angular/core';
import { FormBuilder, FormGroup } from '@angular/forms';
import { Router } from '@angular/router';
import { LoginService } from './login.service';

@Component({
  templateUrl: './login.component.html',
  styleUrls: ['./login.component.css']
})
export class LoginComponent {
  invalidCredentialMsg: string;
  loginForm: FormGroup;

  constructor(private loginService: LoginService, private router: Router, private formbuilder:
FormBuilder) {
    this.loginForm = this.formbuilder.group({
      username: [],
      password: []
    });
  }

  onFormSubmit() {
    let uname = this.loginForm.get('username').value;
    let pwd = this.loginForm.get('password').value;
```

```
    this.loginService.isUserAuthenticated(uname, pwd).subscribe(
      authenticated => {
        if (authenticated) {
          this.router.navigate(['/books']);
        } else {
          this.invalidCredentialMsg = 'Invalid Credentials. Try again.';
        }
      }
    );
  }
}
```

3. Create user.ts file under login folder and add the following code to **user.ts** file

```
export class User {
  constructor(public userId:number, public username:string, public password:string) { }
}
```

4. Add the following code to **login.service.ts** file

```
import { Injectable } from '@angular/core';
import { Observable, of } from 'rxjs';
import { map } from 'rxjs/operators';

import { User } from './user';

const USERS = [
  new User(1, 'user1', 'user1'),
  new User(2, 'user2', 'user2')
];
let usersObservable = Observable.of(USERS);
```

```
@Injectable()
export class LoginService {

    private isloggedIn: boolean = false;

    getAllUsers(): Observable<User[]> {
        return usersObservable;
    }

    isUserAuthenticated(username: string, password: string): Observable<boolean> {
        return this.getAllUsers()
            .map(users => {
                let user = users.find(user => (user.username === username) && (user.password ===
password));
                if (user) {
                    this.isloggedIn = true;

                } else {
                    this.isloggedIn = false;
                }
                return this.isloggedIn;
            });
    }

    isUserLoggedIn(): boolean {
        return this.isloggedIn;
    }
```

```
}
```

5. Create another service class called **login-guard.service** and add the following code

```
import { Injectable } from '@angular/core';
import { CanActivate, Router } from '@angular/router';

import { LoginService } from './login.service';

@Injectable()
export class LoginGuardService implements CanActivate {

  constructor(private loginService: LoginService, private router: Router) { }

  canActivate(): boolean {

    if (this.loginService.isUserLoggedIn()) {
      return true;
    }

    this.router.navigate(['/login']);
    return false;
  }
}
```

6. Add the following code in **login-routing.module.ts**

```
import { NgModule } from '@angular/core';
```

```
import { RouterModule, Routes } from '@angular/router';

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

const loginRoutes: Routes = [
    {
      path: '',
      component: LoginComponent
    }
];

@NgModule({
  imports: [ RouterModule.forChild(loginRoutes) ],
  exports: [ RouterModule ]
})
export class LoginRoutingModule{ }
```

7. Add the following code in **app.module.ts**

```
import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
import { HttpClientModule } from '@angular/common/http';
import { FormsModule, ReactiveFormsModule } from '@angular/forms';

import { AppComponent } from './app.component';
import { BookComponent } from './book/book.component';
import { BookService } from './book/book.service';
import { DashboardComponent } from './dashboard/dashboard.component';
import { BookDetailComponent } from './book-detail/book-detail.component';
import { AppRoutingModule } from './app-routing.module';
```

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

import {LoginService } from './login/login.service';

import {LoginGuardService} from './login/login-guard.service';


@NgModule({

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

  declarations: [AppComponent, BookComponent, DashboardComponent,
BookDetailComponent, LoginComponent],

  providers: [BookService, LoginService, LoginGuardService],

  bootstrap: [AppComponent]

})

export class AppModule { }
```

8. Save the files and check the output in the browser

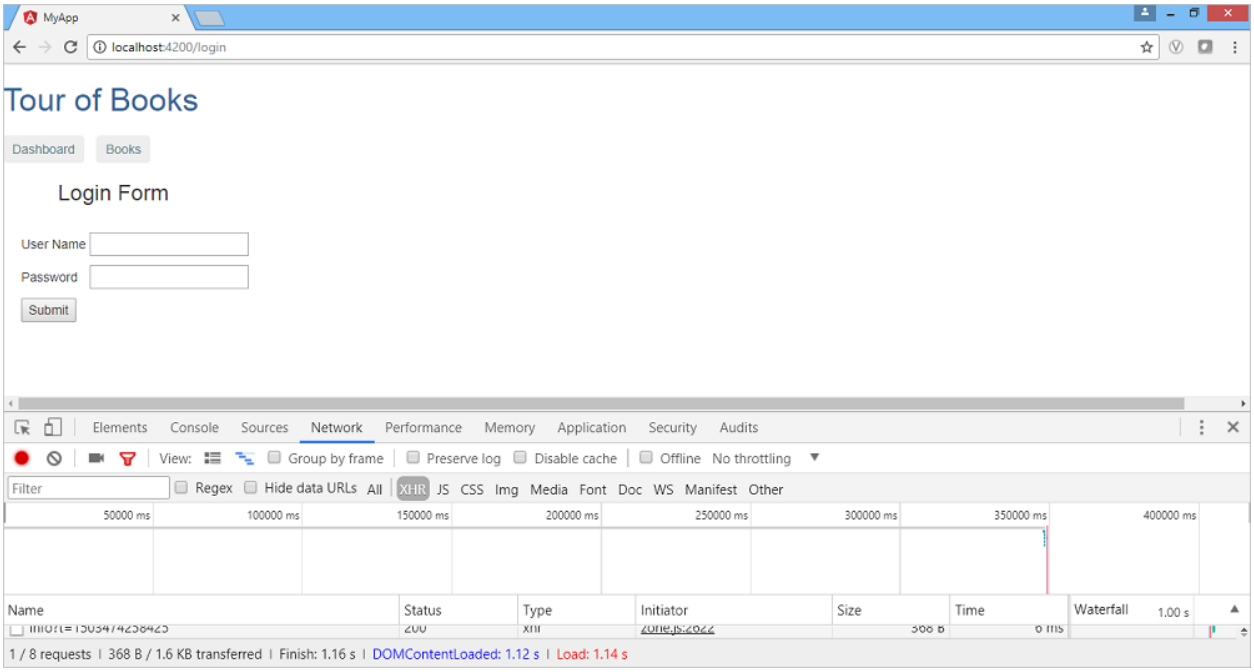
Demo: Asynchronous Routing

Highlights:

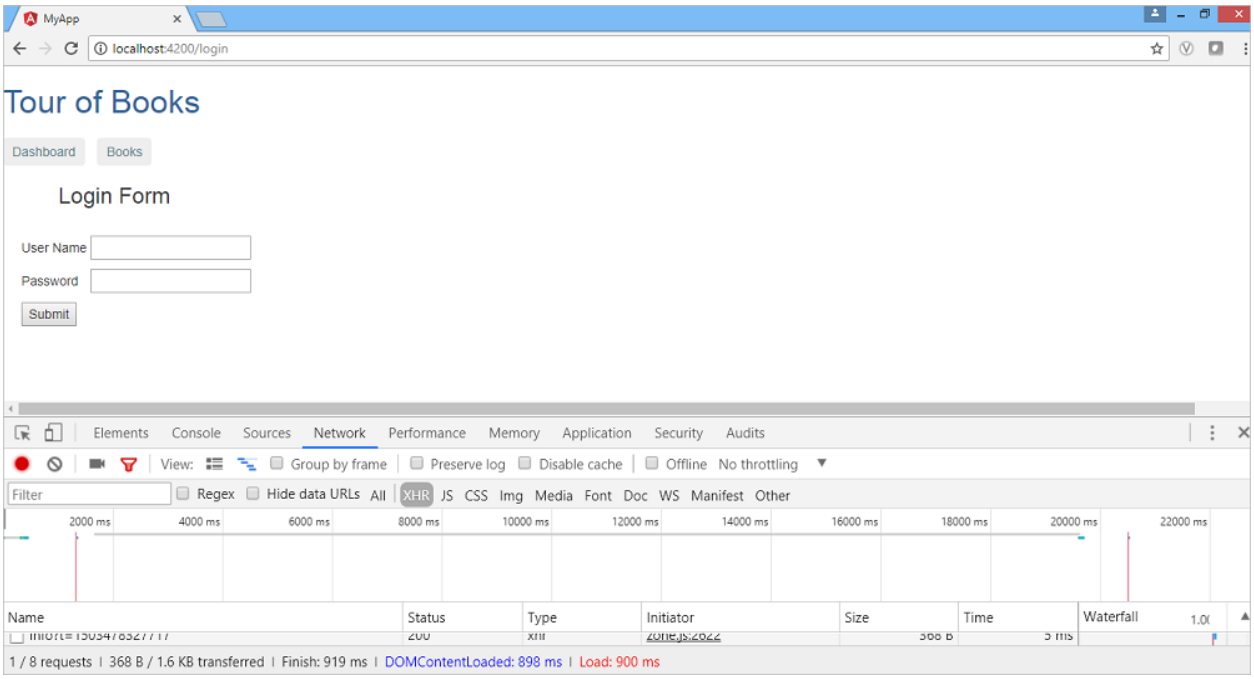
- Understanding asynchronous routing
- Exploring lazy loading route configurations

Problem Statement: Applying lazy loading to Bookcomponent. Output is as shown below

If lazy loading is not added to the demo, it has loaded in 1.14 s. Observe the load time at the bootom of the browser console. Press F12 in the browser and click Network tab and check the Load time



If lazy loading is added to the demo, it has loaded in 900 ms. As BookComponent will be loaded after login, the load time is reduced initially



1. Write the code given below in **book-routing.module.ts** file

```
import { NgModule } from '@angular/core';
import { RouterModule, Routes } from '@angular/router';

import { BookComponent }  from './book.component';
import { LoginGuardService } from '../login/login-guard.service';

const bookRoutes: Routes = [
        {
          path: '',
          component: BookComponent,
    canActivate: [ LoginGuardService ]
        }
];
```

```
@NgModule({
  imports: [ RouterModule.forChild(bookRoutes) ],
  exports: [ RouterModule ]
})
export class BookRoutingModule{ }
```

2. Create **book.module.ts** file and add the following code

```
import { NgModule }   from '@angular/core';

import { BookComponent }  from './book.component';
import { BookRoutingModule } from './book-routing.module';

@NgModule({
  imports: [ BookRoutingModule ],
  declarations: [ BookComponent ]
})
export class BookModule { }
```

3. Add the following code to **app-routing.module.ts** file

```
import { NgModule } from '@angular/core';
import { RouterModule, Routes } from '@angular/router';

import { DashboardComponent } from './dashboard/dashboard.component';
import { BookDetailComponent } from './book-detail/book-detail.component';
import {LoginComponent } from './login/login.component';

const appRoutes: Routes = [
    { path: 'dashboard', component: DashboardComponent },
    { path: '', redirectTo: '/login', pathMatch: 'full' },
```

```
    { path: 'books', loadChildren: './book/book.module#BookModule'},
    { path: 'detail/:id', component: BookDetailComponent },
    {path:'login', component: LoginComponent}
];

@NgModule({
    imports: [
        RouterModule.forRoot(appRoutes)
    ],
    exports: [
        RouterModule
    ]
})
export class AppRoutingModule { }
```

4. Save the files and check the output in the browser

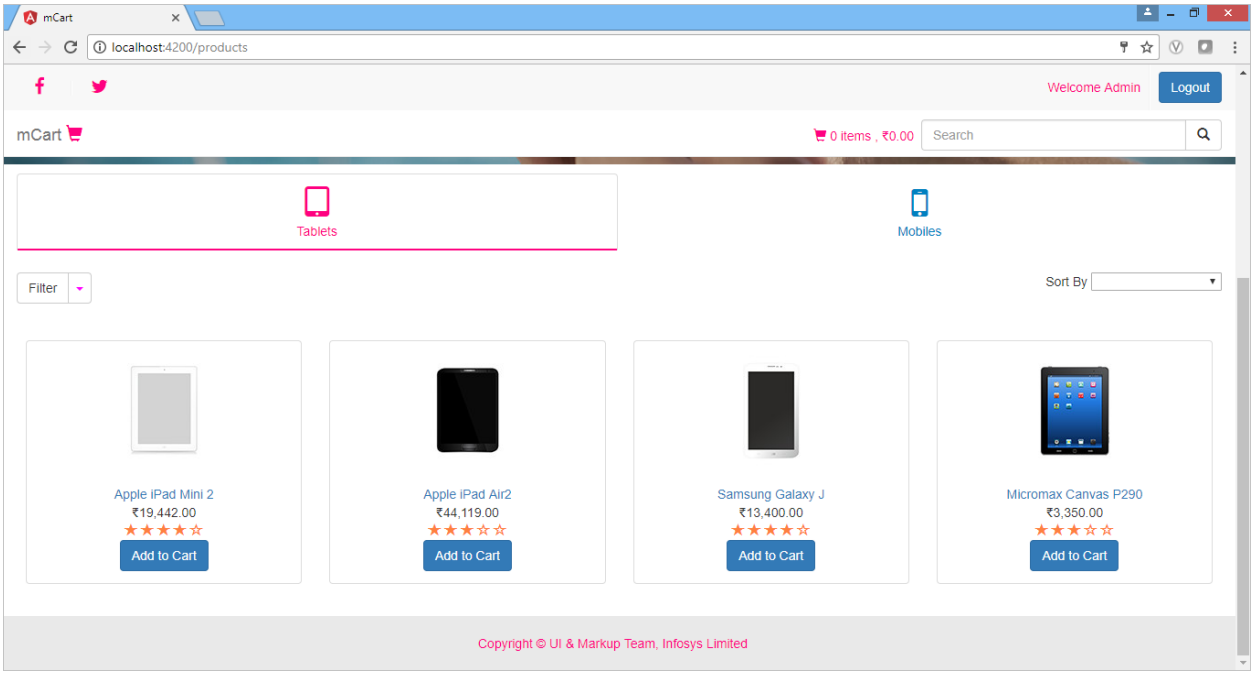
Demo 43: Routing in mCart

Highlights:

- Exploring routing in mCart application
- Navigation between views

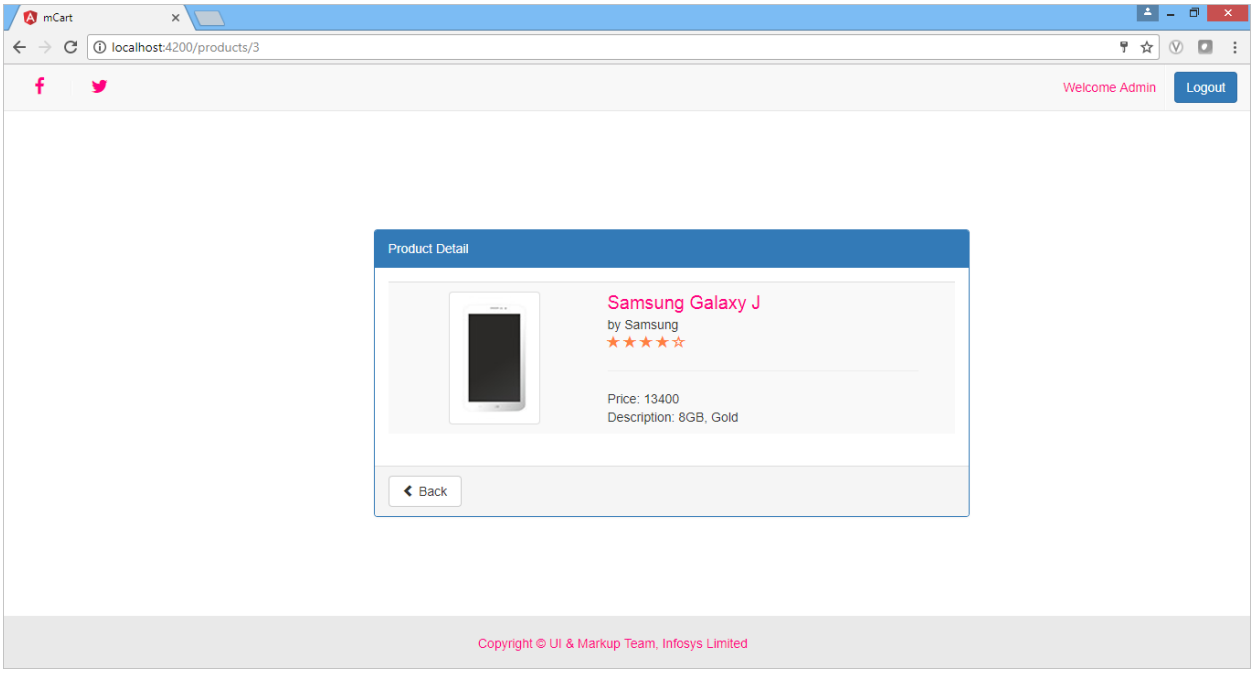
We are navigating between views in our mCart application too. Let us explore the routing in mCart application.

Below is the screen for **ProductList** Component. Observe the url in the address bar. **/products** is the path to render ProductsListComponent



When we click on product name hyperlink, it navigates us to ProductDetailComponent.

Below is the output of **ProductDetailComponent**. Observe the url in the address bar. The url is product/2 where 2 is the product id of the product selected



1. Let us understand the code inside **app-routing.module.ts** file under app folder

```
1. import { NgModule } from '@angular/core';
2. import { Routes, RouterModule } from '@angular/router';
3.
4. import { WelcomeComponent } from './welcome/welcome.component';
5. import { LoginComponent } from './login/login.component';
6.
7. const appRoutes: Routes = [
8. { path: 'welcome', component: WelcomeComponent },
9. { path: 'login', component: LoginComponent },
10. { path: 'products', loadChildren: 'app/products/products.module#ProductsModule' },
11. { path: '', redirectTo: '/welcome', pathMatch: 'full' }
12. ];
13.
```

```
14. @NgModule({
15. imports: [
16. RouterModule.forRoot(appRoutes)
17. ],
18. exports: [
19. RouterModule
20. ]
21. })
22. export class AppRoutingModule { }
```

Line 2: Imports Routes, RouterModule from @angular/router module

Line 7-12: We have mapped paths with respective components and stored in appRoutes variable

Line 16: We have added appRoutes to the RouterModule

Line 19: We have added RouterModule to exports property so that it is available to the entire application

2. Understand the code in **app.module.ts** file to explore the addition routing module to the root module

```
1. import { NgModule } from '@angular/core';
2. import { BrowserModule } from '@angular/platform-browser';
3. import { HttpClientModule } from '@angular/common/http';
4. import { FormsModule } from '@angular/forms';
5.
6. import { AppComponent } from './app.component';
7. import { AppRoutingModule } from './app-routing.module';
8. import { WelcomeComponent } from './welcome/welcome.component';
9. import { LoginComponent } from './login/login.component';
10.
11.
12. @NgModule({
```

13. imports: [BrowserModule, HttpClientModule, FormsModule, AppRoutingModule],
14. declarations: [AppComponent, WelcomeComponent, LoginComponent],
15. providers: [],
16. bootstrap: [AppComponent]
17. })
18. export class AppModule { }

Line 7: imports ApproutingModule class

Line 13: Add it to the imports property so that it is available to the entire module

3 Explore the code present in **product-list.component.html** page to understand implementation of routing

1. <nav class='navbar navbar-default navbar-fixed-top navbarpos'>
2.
3. {{pageTitle}} <span
4. class="glyphicon glyphicon-shopping-cart txtcolor">
5.
6. <input type="text" class="form-control" placeholder="Search" name="q" [(ngModel)]="listFilter" (change)="searchtext()">
7.
8. <button class="btn btn-default">
9.
10. </button>
11.
12.

13.
14. <a
15. [routerLink]="['cart']" class="txtcolor">{{selectedItems}} items
16. , {{total | currency:'INR':true:'1.2-2'}}

```
17. </div>
18. </div>
19. </nav>
20. <br />
21. <br />
22. <div class="container" class="carouselpos">
23. <div id="carousel-example-generic" class="carousel slide carouselheight" data-
    ride="carousel" data-interval="3000">
24. <ol class="carousel-indicators">
25. <li data-target="#carousel-example-generic" data-slide-to="0" class="active"></li>
26. <li data-target="#carousel-example-generic" data-slide-to="1"></li>
27. <li data-target="#carousel-example-generic" data-slide-to="2"></li>
28. </ol>
29. <div class="carousel-inner">
30. <div class="item active">
31. <img src="assets/imgs/carousel_smart_phone.jpg" alt="First slide" style="min-
    width:100%;height:350px;">

32. </div>
33. <div class="item carouselimgpos">
34. <img src="assets/imgs/carousel1.jpg" alt="Second slide" style="min-
    width:100%;height:350px;">

35. </div>
36. <div class="item">
37. <img src="assets/imgs/tablet_blue_stylus.jpg" alt="Third slide" style="min-
    width:100%;height:350px;">
38. </div>
39. </div>
```

```
40. <a class="left carousel-control" href="#carousel-example-generic" role="button" data-
    slide="prev">
41. <span class="glyphicon glyphicon-chevron-left"></span>
42. </a>
43. <a class="right carousel-control" href="#carousel-example-generic" role="button" data-
    slide="next">
44. <span class="glyphicon glyphicon-chevron-right"></span>
45. </a>
46. </div>


47. <div class='panel with-nav-tabs panel-primary noborder'>
48. <div class='panel-heading noborder bgcolor'>
49. <ul class="nav nav-tabs noborder">
50. <li class="active tabpos"><a href="#tabprimary" (click)="tabselect('tablet')" data-
    toggle="tab"><i
51. class="fa fa-tablet fa-3x" aria-hidden="true"></i>
52. <div>Tablets</div></a></li>
53. <li class="tabpos"><a (click)="tabselect('mobile')" href="#tabprimary" data-
    toggle="tab"><i
54. class="fa fa-mobile fa-3x" aria-hidden="true"></i>
55. <div>Mobiles</div></a></li>
56. </ul>
57. </div>
58. <div class='panel-body'>
59. <div class="tab-content">
60. <div class="tab-pane fade in active" id="tabprimary">
61. <div class="btn-group">
62. <button type="button" class="btn btn-default">Filter</button>
63. <button type="button" class="btn btn-default dropdown-toggle" data-
    toggle="dropdown">
```

```
64. <span class="caret"></span> <span class="sr-only">Toggle
65. Dropdown</span>
66. </button>
67. <ul class="dropdown-menu multi-column columns-3 noclose">
68. <div class="row vdivide">
69. <div class="col-md-4">
70. <ul class="multi-column-dropdown noclose">
71. <h4>Manufacturer</h4>
72. <li *ngFor="let manufac of manufacturers">
73. <input type="checkbox" [ngModel]="manufac.checked" (change)="filter(manufac)">
    <label>
74. {{manufac.id}} </label></li>
75. </ul>
76. </div>
77. <div class="col-md-4">
78. <ul class="multi-column-dropdown noclose">
79. <h4>OS</h4>
80. <li *ngFor="let ostypes of os">
81. <input type="checkbox" [ngModel]="ostypes.checked" (change)="filter(ostypes)">
82. <label> {{ostypes.id}}</label></li>
83. </ul>
84. </div>
85. <div class="col-md-4">
86. <ul class="multi-column-dropdown noclose">
87. <h4>Price Range</h4>
88. <li *ngFor="let price of price_range">
89. <input type="checkbox" [ngModel]="price.checked" (change)=filter(price)> <label>{{
    price.id}} </label></li>
90. </ul>
```

```
</div>
</div>
</ul>
</div>
<span *ngIf="chkmanosprice.length> 0"> {{products.length}}
results</span>

<div class="pull-right">
<span>Sort By </span>
<select [ngModel]="sortoption" (change)="onChange($event.target.value)">
        <option value="popularity">Popularity</option>
        <option value="pricelh">Price - Low to High</option>
        <option value="pricehl">Price - High to Low</option>
        </select>
        </div>
        <div *ngIf='products && products.length'>
        <div class="row" *ngFor='let product of products | orderBy:sortoption ; let i = index' [hidden]="(i%4)>0">
        <div class="col-xs-3">
        <span class="thumbnail text-center">
        <div>
        <img [src] ='product.imageUrl' [title]='product.productName'
        [style.width.px]='imageWidth' [style.height.px]='imageHeight' [style.margin.px]='imageMargin'>
        </div>
        <div class="caption">
        <div>
        <a [routerLink]="[product.productId]" >
        {{product.productName}} </a>
```

```
117.        </div>
118.        <div>{{ product.price | currency:'INR':true:'1.2-2'}}</div>
119.        <div></div>
120.        <ratings  class="ratingcolor"
    [rate]='product.rating'></ratings>
121.        <div>
122.        <button (click)="addCart(product.productId)"
123.        class="btn btn-primary">Add to Cart</button>
124.        </div>
125.        </div>
126.        </span>
127.        </div>
128.        <div class="col-xs-3">
129.        <div *ngIf="products[i+1]" class="thumbnail text-center">
130.        <div>
131.        <img [src]='products[i+1].imageUrl' [title]='products[i+1].productName'
    [style.width.px]='imageWidth' [style.height.px]='imageHeight'
132.        [style.margin.px]='imageMargin'>
133.        </div>
134.        <div class="caption">
135.        <div>
136.        <a [routerLink]="[products[i+1].productId]">
137.        {{products[i+1].productName}} </a>
138.        </div>
139.        <div>{{ products[i+1].price | currency:'INR':true:'1.2-2'}}
140.        </div>
141.        <div></div>
142.        <ratings class="ratingcolor" [rate]='products[i+1].rating'></ratings>
143.        <div></div>
```

```
144.        <div>
145.        <button (click)="addCart(products[i+1].productId)" class="btn btn-
    primary">Add to Cart</button>
146.        </div>
```

Line 141-142: Hyperlink is binded with routing link whre we are passing product id as route parameter. When this link is clicked, it will navigate to products/<productid> path i.e., it navigates to ProductDetailComponent

4 Open **product-detail.component.ts** file under product-detail folder and understand the code present below

```
1. import { Component } from '@angular/core';
2. import { ActivatedRoute, Router, ParamMap } from '@angular/router';
3. import { Observable } from 'rxjs';
4. import { switchMap } from 'rxjs/operators';

5. import { Product } from '../product';
6. import { ProductService } from '../product.service';

7. @Component({
8. templateUrl: 'product-detail.component.html',
9. styleUrls: ['product-detail.component.css']
10. })
11. export class ProductDetailComponent {
12. pageTitle: string = 'Product Detail';
13. product: Product;
14. imageWidth: number = 100;
15. imageMargin: number = 2;
16. errorMessage: string;
17. id: number = 0;
```

```
18. constructor(private route: ActivatedRoute,
19. private router: Router, public productService: ProductService) {
20. this.id = +this.route.snapshot.paramMap.get('id');
21. this.product = this.productService.products.filter((product: any) => product.productId ===
    this.id)[0];
22. }


23. onBack(): void {
24. this.router.navigate(['/products']);
25. }
```

Line 21: We have injected ActivatedRoute class to fetch route parameters and Router class for navigation

Line 23: We are fetching value of route parameter called id

Line 27: onBack() method is invoked when back button is clicked on product detail page

Line 28: We are navigating to /products url when back button is clicked