**Angular**

A javascript framework for building client-side applications.

Angular 2 application is comprised of a **set of components** and **services** that provide functionality across those components

A component is the combination of an HTML template and a component class that controls a portion of the screen.

Language used in Angular application is **TypeScript**

**Setting up Environment**

* Install npm (node package manager) is a command line utility
  + <https://nodejs.org/en/download/current/>
* Setting up angular 2 application
  + Download **git** if not installed
  + Use git clone <https://github.com/DeborahK/Angular2-GettingStarted>
  + In the terminal type **npm install**
    - The above command will install the required modules needed for your angular application based on the dependencies specified in package.json
  + In the terminal type **npm start**
  + It will start a server to launch our application (lite-server) and it will run the **typescript compiler tsc - w (watch mode)**
  + Both typescript compiler and the file server watch for any file changes, so if we make any changes to our file the typescript recompiles them and the changes happens in the browser automatically
* Understanding the basic application
  + Index.html file contains
  + SystemJs :
    - SystemJS is a module loader that can import modules (ES6 modules).
    - loads all the application code files, so we don't need to add <script> for all the files that we use in our application.
  + **System.import('app').catch(function(err){ console.error(err); });**
    - Starts our application by loading all the required modules which is in the **app folder,**
  + **systemjs.config** : file which loads the main entry point to the application which is specified in packages, so it loads and run the **main.js** file.
  + **Main.ts**
  + **platformBrowserDynamic :** to compile the application dynamically
  + **appmodule** : application module
    - Defined using **export class AppModule { }**
    - Use **@NgModule** as decorator and pass meta data defining details of module.

**Creating components**

As said angular application is comprised of a set of components, we create each components and arrange them to form our application.

A component contains the following parts

* **A template** (view layout ) created using html, which defines what is rendered on the page
* **Class** for the code associated with the view, which will be created using **typescript**, it also contains all the data elements available for use in the view, methods were all the logic is written for the view.
* Metadata which provides additional information about the component to angular. It is defined using a **decorator**.
* **Component => Template + Class + Metadata**
* Simple component file would look like
* import {Component} from '@angular/core';  
    
  @Component({  
   selector:'ml-app',  
   template:`  
   <div>  
   <h1>{{pageTitle}}</h1>  
   <p>My first angular2 component</p>  
   </div>  
   `  
  })  
  export class AppComponent {  
   pageTitle : string = 'Marlabs - Angular Traning !!!';  
  }

**Creating component class**

We define the class using the **class** keyword followed by component name. The component name here is App and Component suffix will be appended to it, so it becomes **AppComponent.**

export class AppComponent {

pageTitle : string = “Marlabs”;

}

* By convention the root component of an application is called AppComponent.
* **pageTitle** : is the property with a value Marlabs
* With typescript we specify the datatype for each property that we create, here it is **string**. Methods are also added in the class body.

**Defining metadata**

Angular need metadata data to understand how to instantiate the component, construct the view and interact with the component.

Defined using angular **Component()** and in typeScript we attach that function to the class as a **decorator** and is always prefixed with a **@ symbol, @Component()**

We apply a decorator by placing immediately in front of the feature we are decorating.

Since we are decorating the class we place decorator in immediately in front of the class.

@Component({  
 selector: 'pm-app', // directive name  
 template: `  
 <h1>{{pageTitle}}</h1>

<div>Angular2: Getting Started</div>  
 `  
})  
export class AppComponent {

pageTitle : string = “Marlabs”;

}

**Importing Components**

In the component code we used the **@Component()** from angularto define our class as a component. Now we need to tell the module loader where to find this function.

This is done using the import statments.

**import {Component} from ‘@angular/core’;**

**Note :** Let's create our first component

import {Component} from '@angular/core';  
  
@Component({  
 selector: 'mat-app',  
 template : `  
 <div>  
 <h1>{{pageTitle}}</h1>  
 <div>My first component</div><br />  
 </div>  
 `  
})  
  
export class AppComponent {  
 pageTitle : string = `Marlabs - Angular training`;  
}

**Bootstrapping our App Component**

* Load the root component (bootstrapping)
* Setup index.html file to host our application

index.html contains the main page for the application, and could be the only webpage in the application and all the other html code will be displayed into this web page and hence called **single page application**

We use the directive we created **mat-app** in the index.html and give a **loading…** message.

<mat-app>Loading app….</mat-app>

When the **root element** is loaded the Loading app… message appears and as soon as the loading is complete the html code defined inside the template is inserted in between the selector element tag

**Angularjs Application Startup**

index.html => System.import(‘app) => Systemjs.config.js => main.ts => app.module.ts => app.component.ts

1. index.html file loads and executes the module main.ts
2. main.ts bootstraps the angular module (AppModule)
3. AppModule bootstraps the root application component (AppComponent)
4. Component template appears in the browser

**Creating ProductListComponent**

1. Create a folder inside app folder **products**
2. Create component file **product-list.component.ts**
3. import {Component} from '@angular/core';  
     
   @Component({  
    selector:'ml-products',  
    templateUrl:'app/products/product-list-component.html'  
   })  
   export class ProductListComponent {  
      
   }
4. Use **ml-products** inside AppComponent
   1. <ml-products></ml-products>
5. Need to include the component inside AppModule
   1. Import the component
      1. import { ProductListComponent } from './products/product-list.component';
   2. Include the component inside the declarations array
      1. declarations: [   
          AppComponent,  
          ProductListComponent  
          ]
6. Build the template for the component
   1. Use product-list.component.html file to add all the html content
   2. <div class="panel panel-primary">  
       <div class="panel-heading">  
       Product List  
       </div>  
       <div class="panel-body">  
       <div class="row">  
       <div class="col-md-2">Filter by</div>  
       <div class="col-md-4">  
       <input type="text" />  
       </div>  
       </div>  
       <div class="row">  
       <div class="col-md-6">  
       <h3>Filtered by : </h3>  
       </div>  
       </div>  
       <div class="table-responsive">  
       <table class="table">  
       <thead>  
       <tr>  
       <th>  
       <button class="btn btn-primary">Show image</button>  
       </th>  
       <th>Products</th>  
       <th>Code</th>  
       <th>Available</th>  
       <th>Price</th>  
       <th>5 Star rating</th>  
       </tr>  
       </thead>  
       <tbody>  
         
       </tbody>  
       </table>  
       </div>  
       </div>  
      </div>

**Data binding**

1. Coordinates the communication between the component class and the template and also involves passing data
2. Angular use **interpolation** for passing data.
3. Interpolation is a special syntax that Angular converts into a property binding **{{}},** it is a one way binding from the class property to the template.
4. {{pageTitle}} : evaluates the pageTitle variable and displays the the result.
5. So let's create a new variable inside productlist component.

**Adding logic using angular directives**

1. Let's create a new variable inside the component class to display products
   1. products : any[] = [  
       {  
       "productId": 1,  
       "productName": "Leaf Rake",  
       "productCode": "GDN-0011",  
       "releaseDate": "March 19, 2016",  
       "description": "Leaf rake with 48-inch wooden handle.",  
       "price": 19.95,  
       "starRating": 3.2,  
       "imageUrl": "http://openclipart.org/image/300px/svg\_to\_png/26215/Anonymous\_Leaf\_Rake.png"  
       },  
       {  
       "productId": 2,  
       "productName": "Garden Cart",  
       "productCode": "GDN-0023",  
       "releaseDate": "March 18, 2016",  
       "description": "15 gallon capacity rolling garden cart",  
       "price": 32.99,  
       "starRating": 4.2,  
       "imageUrl": "http://openclipart.org/image/300px/svg\_to\_png/58471/garden\_cart.png"  
       },  
       ];
   2. While defining products array, we used the data type as **any[]**
   3. **any[]** is the data type used when we aren't sure about the datatype in typescript.
2. Let's use ngIf directive to check whether the products array is declared and contains some elements.
   1. <table class="table" \*ngIf="products && products.length"></table>
3. Using angular directive \*ngFor, we can iterate through the array elements.
   1. <tr \*ngFor='let product of products'>  
       <td></td>  
       <td>{{product.productName}}</td>  
       <td>{{product.productCode}}</td>  
       <td>{{product.releaseDate}}</td>  
       <td>{{product.price}}</td>  
       <td>{{product.starRating}}</td>  
       </tr>

**Property binding**

1. Allows to set property of an element to value of a template expression
2. Binding targets as always enclosed in square brackets []
3. Binding source are always enclosed in quotes
4. [binding target] = ‘binding source’;
5. Example for a property binding
   1. <img [src]='product.imageUrl' [title]='product.productName' [style.width.px] = 'imageWidth' />
6. In the above example we are binding 3 properties
   1. [src]='product.imageUrl'
   2. [title]='product.productName'
   3. [style.width.px] = 'imageWidth'
      1. For this we need to create a new property inside the component class

**Event binding**

1. Allows to bind an event to an element
2. Target event is always enclosed in parentheses
3. Component class method will always be enclosed in quotes
4. (target event) = ‘method()’;
5. Example for event binding
   1. Create a new function toggleImage()
      1. toggleImage() : void {  
          }
   2. Create a new property to toggle image
      1. showImage : boolean = false;  
          toggleImage() : void {  
          this.showImage = !this.showImage;  
          }
   3. Bind the event to button
      1. <button (click)='toggleImage()' class="btn btn-primary">Show image</button>
   4. Apply the condition in <img /> using \*ngIf
      1. <img \*ngIf='showImage' />
6. Toggle the button text using javascript conditional operator
   1. <button (click)='toggleImage()' class="btn btn-primary">  
       {{showImage ? 'Hide Image' : 'Show Image'}}  
       </button>

**Two way binding**

1. Use ngModel directive to specify two way binding
2. Syntax **[(ngModel)] = ‘property name’;**
3. ngModel is enclosed in square bracket [] to indicate property binding
   1. So the value property created inside the class will be bind to the ngModel
4. ngModel is again enclosed in parentheses () to indicate event binding
   1. So the user entered data will be send back to the property in the component class
5. So let's create a new in the component class
   1. listFilter : string = 'cart';
6. Let's use the same inside the filter input element and filtered by
   1. <input type="text" [(ngModel)] = 'listFilter' />
   2. <h3>Filtered by : {{listFilter}}</h3>
7. It throws an error once we try to run this
8. For using ngModel, its needs to included in the AppModule
   1. import { FormsModule } from '@angular/forms';
   2. imports: [   
       BrowserModule,  
       FormsModule  
       ]

**Pipes**

1. Allows to transform data
2. {{product.productName | uppercase}}
3. {{product.productCode | lowercase}}
4. {{product.price | currency:'INR':true | lowercase}}

**Interfaces**

1. We can create interfaces to strongly type a property, when there is no predefined type for a property
2. In the above example the products array we created does not have a predefined property
3. We used products : any[] = []; in this case we can make use of interface
4. An interface is a specification identifying a related set of properties and methods
5. Creating an interface in a separate file in products folder
6. Use interface keyword to create an interface
7. export interface IProduct {  
    productId: number,  
    productName: string,  
    productCode: string,  
    releaseDate: string,  
    description: string,  
    price: number,  
    starRating: number,  
    imageUrl: string  
   }
8. To use the interface as a data type, import the interface inside the product-list.component.ts file
   1. import {IProduct} from './product';
9. Now in the component class replace any[] with IProduct[]

**Encapsulating component styles**

1. styles: array of inline styles
2. styleUrl : array of multiple style urls to be applied for the component
3. You use both the properties inside the component decorator
   1. @Component({  
       selector:'ml-products',  
       templateUrl:'app/products/product-list.component.html',  
       styles : ['thead { color:#FF0000;}']  
      })

**Component life cycle hooks**

1. Angular provides a set of life cycle hooks we can use to tap into component life cycle and perform operations.
2. The most commonly used life cycle hooks  
   1. OnInit : perform component initialization, retrieve data etc
   2. OnChanges : called when a change happens
   3. OnDestroy : called when component gets destroyed
3. To use the we need to
   1. import the interface for the life cycle hooks  
      import {Component, OnInit} from '@angular/core';
   2. Make the component class to implement the interface  
      export class ProductListComponent implements OnInit {}
   3. Each life cycle hook has an associated function which is the **“ng”** followed by life cycle hook name.  
      1. OnInit = ngOnInit()
      2. OnChanges = ngOnChanges()
   4. Define the life cycle hook function  
      ngOnInit() : void {  
       console.log('OnInit called');  
       };

**Building a custom pipe**

1. To create a custom pipe let's create a new file product-filter.pipe.ts inside products folder
2. Create the pipe class by implementing **PipeTransform interface**export class ProductFilterPipe implements PipeTransform {  
      
   }
3. Add the pipe decorator  
   @Pipe({  
    name : 'productFilter'  
   })
4. Import both decorator and interface used in the pipe  
     
   import { PipeTransform, Pipe } from '@angular/core';
5. As said before each interface has a method associated with it  
     
   transform(value:IProduct[], filterBy:string) : IProduct[] {  
    return value;  
   }  
   1. First parameter : products array
   2. Second parameter : search string
6. Since we are using IProduct[] as the data type for products array import the same.  
   import {IProduct} from './product';
7. Inside the **transform function,** use javascript array filter function to filter array of items based on a user defined function
8. The custom pipe file will finally look like
   1. import { PipeTransform, Pipe } from '@angular/core';  
        
      import {IProduct} from './product';  
        
      @Pipe({  
       name : 'productFilter'  
      })  
      export class ProductFilterPipe implements PipeTransform {  
         
       transform(value:IProduct[], filterBy:string) : IProduct[] {  
       return filterBy ? value.filter(function(items) {  
       if(items.productName.toLowerCase().indexOf(filterBy.toLowerCase())!= -1) {  
       return value;  
       }  
       }) : value;  
       }  
         
      }
9. To use it in our html file
   1. <tr \*ngFor='let product of products | productFilter:listFilter'></tr>
   2. **customPipe : filter variable**
10. After creating the custom pipe to make it available for the application import and include it inside AppModule declarations array  
      
    import { ProductFilterPipe } from './products/product-filter.pipe';  
      
    declarations: [   
     AppComponent,  
     ProductListComponent,  
     ProductFilterPipe  
    ]

**Services**

1. Create a service class  
     
   export class ProductService {  
   }
2. Define the metadata with a decorator (@Injectable())
3. Import the decorator  
   import { Injectable } from '@angular/core';
4. Define a function inside the service class to retrieve products  
   getProducts() : IProduct[] {  
    return [  
    {  
    "productId": 1,  
    "productName": "Leaf Rake",  
    "productCode": "GDN-0011",  
    "releaseDate": "March 19, 2016",  
    "description": "Leaf rake with 48-inch wooden handle.",  
    "price": 19.95,  
    "starRating": 3.2,  
    "imageUrl": "http://openclipart.org/image/300px/svg\_to\_png/26215/Anonymous\_Leaf\_Rake.png"  
    },  
    {  
    "productId": 2,  
    "productName": "Garden Cart",  
    "productCode": "GDN-0023",  
    "releaseDate": "March 18, 2016",  
    "description": "15 gallon capacity rolling garden cart",  
    "price": 32.99,  
    "starRating": 4.2,  
    "imageUrl": "http://openclipart.org/image/300px/svg\_to\_png/58471/garden\_cart.png"  
    },  
    ];  
    }
5. To use service in a component it needs to imported first  
   import {ProductService} from './product.service';
6. To register a service we need to specify the service as a **provider** in the **component decorator** where we are using it.  
     
   @Component({  
    selector:'ml-products',  
    templateUrl:'app/products/product-list.component.html',  
    styles : ['thead { color:#FF0000;}'],  
    providers : [ProductService]  
   })
7. Injecting the service in product-list.component.ts file
   1. Specify the dependency inside the constructor function of the class  
      constructor(private \_productService : ProductService) {  
       this.products = this.\_productService.getProducts();  
      }
   2. Either you set the products array inside the constructor like above or  
      ngOnInit() : void {  
       this.products = this.\_productService.getProducts();  
      };

**Retrieving data using HTTP Promise**

1. import HttpModule in AppModule, include in imports array  
   import { HttpModule } from '@angular/http';  
     
   imports: [   
    BrowserModule,  
    FormsModule,  
    HttpModule  
    ]
2. Inside product.service.ts file import http service  
   import {Http} from '@angular/http';
3. Specify the dependency inside the constructor of service function  
    constructor(private \_http : Http) {  
      
    }
4. Modify the getProducts() function to return a promise  
   getProducts() : Promise<IProduct[]> {  
    return this.\_http.get('api/products/products.json')  
    .map((response) => <IProduct[]>response.json()).toPromise();  
    }  
     
   **Note : map method takes the raw http response object returned by the http get method and translates into an array of products.**
5. To use **toPromise()** function we need to import the operator  
   +**import 'rxjs/add/operator/toPromise';**
6. To use **map()** function it needs to be imported  
   **import 'rxjs/add/operator/map';**
7. Inside the map function we have the ES6 arrow function, which let's you create a function in different way
   1. var add = function(x,y) {  
       return x + y;  
      }  
        
      var resp = add(4,5);  
        
      var add = (x,y) => x + y;  
      var resp = add(4,5);  
        
      alert(resp);
8. Modify the OnInit() in product-list.component.ts file to incorporate the changes made in the getProducts()  
     
   ngOnInit() : void {  
    this.\_productService.getProducts().then(products\_resp => this.products = products\_resp);  
    };

**Retrieving data using HTTP Observables**

* Make necessary changes to **product-service.ts** file
* import { Injectable } from '@angular/core';  
  import { Http, Response } from '@angular/http';  
  import { Observable } from 'rxjs/Observable';  
  import 'rxjs/add/operator/map';  
    
  import { IProduct } from './product';  
    
  @Injectable()  
  export class ProductService {  
     
   constructor(private \_http : Http) {  
     
   }  
   getProducts() : Observable<IProduct[]> {  
   return this.\_http.get('api/products/products.json')  
   .map((response: Response) => <IProduct[]>response.json());  
   }  
  }
* **product-list.component.ts** file
* ngOnInit(): void {  
   console.log('Product list component initialized !!!');  
   this.\_productService.getProducts().subscribe(data => this.products = data);  
   }

**Nested components**

1. Let's create a nested component to modify the 5 star rating
2. Put the files in shared folder since it should be shared between multiple components
3. Let's create a new component file star.component.ts inside the shared folder
4. Just like any other component follow the same steps of creating a component  
     
   import {Component} from '@angular/core';  
     
   @Component({  
    selector:'ml-star',  
    templateUrl:'app/shared/star.component.html',  
    styleUrls:['app/shared/star.component.css']  
   })  
   export class StarComponent {  
      
   }
5. star.component.html files has two properties **starWidth**, **rating** used, so let's define the same in the component file  
   export class StarComponent {  
    rating : number = 4;  
    starWidth : number;  
   }
6. The value of **starWidth** has to be calculated based on the rating, so let's implement **OnChanges** life cycle hook to do that.  
     
   import {Component,OnChanges} from '@angular/core';  
     
   export class StarComponent implements OnChanges{  
    rating : number = 4;  
    starWidth : number;  
    ngOnChanges() : void {  
    this.starWidth = this.rating \* 86/5;  
    }  
   }
7. Import the star component inside AppModule  
   import { StarComponent } from './shared/star.component';  
   declarations: [   
    AppComponent,  
    ProductListComponent,  
    ProductFilterPipe,  
    StarComponent  
    ],
8. In product-list.component.html file instead of showing the rating value used the nested component  
   <ml-star></ml-star>

**Passing data to nested component**

1. If the nested component want to receive a data from the container it must expose a property to the container
2. To expose the property we use **@Input()** (decorator)
3. So first we need to import Input  
     
   import {Component,OnChanges,Input} from '@angular/core';
4. Expose the rating property using @Input() (decorator)  
   @Input() rating : number;
5. Now using property binding we can pass values to rating property  
   <ml-star [rating]='product.starRating'></ml-star>

**Sending data from nested component back to container**

1. If the nested component wants to send the information back to container, nested component needs to **raise an event**
2. The nested component exposes the event using **@Output()** (decorator)
3. The only way for nested component to send the data back to the container is by raising an event and the event payload becomes the data to be passed.
4. Event is defined using **EventEmitter object**
5. Let's import Output , EventEmiiter in the component  
     
   import {Component,OnChanges,Input,Output,EventEmitter} from '@angular/core';
6. Let's create the property which raises the event  
   @Output() ratingClicked : EventEmitter<string> = new EventEmitter<string>();
7. Create a new function which will emit an event  
    onClick() {  
    this.ratingClicked.emit('The rating '+this.rating+' was clicked');  
    }
8. Now call the onClick() function when a click event happens in the nested component  
   (click)='onClick()'
9. Now do the event binding in nested component used in the product-list.component.html file.  
     
   <ml-star [rating]='product.starRating' **(ratingClicked)='ratingClicked($event)'**></ml-star>
10. In the above statement **ratingClicked on the left side** is the **property exposed** by the nested component and **ratingClicked on the right side** is the function which needs to be created inside the product-list.component to receive the data.  
      
    ratingClicked(message:string) : void {  
     alert(message);  
    }

**Routing**

1. Import the RouterModule inside AppModule  
   import { RouterModule } from '@angular/router';
2. Specify RouterModule inside declarations array  
     
   imports: [   
    BrowserModule,  
    FormsModule,  
    HttpModule,  
    RouterModule.forRoot([  
    {path:'products', component:ProductListComponent},  
    {path:'welcome', component:WelcomeComponent},  
    {path:'', redirectTo:'welcome', pathMatch:'full'}, // default path  
    {path:'\*\*', redirectTo:'welcome', pathMatch:'full'} // invalid path  
    ])  
    ]
3. Add the welcome component in AppModule
4. So the final AppModule would look like  
     
   import { NgModule } from '@angular/core';  
   import { BrowserModule } from '@angular/platform-browser';  
   import { FormsModule } from '@angular/forms';  
   import { HttpModule } from '@angular/http';  
   import { RouterModule } from '@angular/router';  
     
     
   import { AppComponent } from './app.component';  
   import { WelcomeComponent } from './home/welcome.component';  
   import { ProductListComponent } from './products/product-list.component';  
   import { ProductFilterPipe } from './products/product-filter.pipe';  
   import { StarComponent } from './shared/star.component';  
     
   @NgModule({  
    imports: [   
    BrowserModule,  
    FormsModule,  
    HttpModule,  
    RouterModule.forRoot([  
    {path:'products', component:ProductListComponent},  
    {path:'welcome', component:WelcomeComponent},  
    {path:'', redirectTo:'welcome', pathMatch:'full'},  
    {path:'\*\*', redirectTo:'welcome', pathMatch:'full'}  
    ])  
    ],  
    declarations: [   
    AppComponent,  
    ProductListComponent,  
    ProductFilterPipe,  
    StarComponent,  
    WelcomeComponent  
    ],  
    bootstrap: [ AppComponent ]  
   })  
   export class AppModule { }
5. Change the template property of AppComponent  
     
   template:`  
    <div class="navbar navbar-default">  
    <ul class="nav navbar-nav">  
    <li>  
    <a **[routerLink]="['/welcome']"**>Welcome</a>  
    </li>  
    <li>  
    <a **[routerLink]="['/products']"**>Products</a>  
    </li>  
    </ul>  
    </div>  
    <div>  
    <h1>{{pageTitle}}</h1>  
    **<router-outlet></router-outlet>**  
    </div>
6. Use routerLink to specify the URL and **router-outlet** to place the rendered views
7. Add <base href='/' /> in the index.html file to specify the default URL

**Pass parameters through URL**

1. Add a new route with params in RouterModule array  
   {path:'products/:id/:username', component:ProductDetailComponent},
2. Let's link the product name in product details component to product details page  
     
   <a [routerLink]="['/products', product.productId, 'marlabs']">{{product.productName}}</a>
3. To access the routeParams we need inject **ActivatedRoute** as dependency to the component
4. The product detail component will finally look like  
   import {Component, OnInit} from '@angular/core';  
   **import {ActivatedRoute} from '@angular/router';**  
     
     
   @Component({  
    template:`  
    <div>  
    <h1>Product detailed page</h1>  
    <h3>{{pageTitle}} - {{productNumber}}</h3>  
    </div>  
    `;  
   })  
   export class ProductDetailComponent implements OnInit{  
    constructor(**private \_activateRoute : ActivatedRoute**) {  
    }  
    pageTitle : string = "Product No";  
    productNumber : number;  
    ngOnInit() : void {  
    this.productNumber = **this.\_activateRoute.snapshot.params['id']**+' = '+**this.\_activateRoute.snapshot.params['username']**;  
    }  
   }