**Environment Setup**

**Project Setup**

**Component**

**Module**

**Data Binding: one way, two way**

**Events**

**Model (MVC)**

**Type Script (data type, class, function, constructor, properties, default events)**

**Directive/import**

**ngFor**

**ngIf**

**Pipe**

**Template : html**

**css: custom css**

**Bootstrap:**

**ES6: Loop, Array , arrow sign =>**

**\*Service**

**\*HTTP Service/Web Service**

**\*Form Module**

**\*Routing**

**\*Material**

**\*Animation**

**\*ngModule**

**Component**

==============================================

Major part of the development with Angular 4 is done in the components. Components are basically classes that interact with the .html file of the component, which gets displayed on the browser. We have seen the file structure in one of our previous chapters. The file structure has the app component and it consists of the following files −

app.component.css

app.component.html

app.component.spec.ts

app.component.ts

app.module.ts

**Module**

======================================

Module in Angular refers to a place where you can group the components, directives, pipes, and services, which are related to the application.

In case you are developing a website, the header, footer, left, center and the right section become part of a module.

To define module, we can use the NgModule. When you create a new project using

the Angular –cli command, the ngmodule is created in the app.module.ts file by

default and it looks as follows −

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

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

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

@NgModule({

declarations: [

AppComponent

],

imports: [

BrowserModule

],

providers: [],

bootstrap: [AppComponent]

})

export class AppModule { }

The NgModule needs to be imported as follows −

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

The structure for the ngmodule is as shown below −

@NgModule({

declarations: [

AppComponent

],

imports: [

BrowserModule

],

providers: [],

bootstrap: [AppComponent]

})

It starts with @NgModule and contains an object which has declarations, import

, providers and bootstrap.

Declaration

It is an array of components created. If any new component gets created, it

will be imported first and the reference will be included in declarations as

shown below −

declarations: [

AppComponent,

NewCmpComponent

]

Import

It is an array of modules required to be used in the application. It can also

be used by the components in the Declaration array. For example, right now in

the @NgModule we see the Browser Module imported. In case your application

needs forms, you can include the module as follows −

import { FormsModule } from @angular/forms;

The import in the @NgModule will be like the following −

imports: [

BrowserModule,

FormsModule

]

Providers

This will include the services created.

Bootstrap

This includes the main app component for starting the execution.

=================================================

Event Binding and ngIf

===============================================

<!--The content below is only a placeholder and can be replaced.-->

<div style = "text-align:center">

<h1>

Welcome to {{title}}.

</h1>

</div>

<div> Months :

<select>

<option \*ngFor = "let i of months">{{i}}</option>

</select>

</div>

<br/>

<div>

<span \*ngIf = "isavailable; then condition1 else condition2">

Condition is valid.

</span>

<ng-template #condition1>Condition is valid</ng-template>

<ng-template #condition2>Condition is invalid</ng-template>

</div>

<button (click)="myClickFunction($event)">

Click Me

</button>

-------------

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

@Component({

selector: 'app-root',

templateUrl: './app.component.html',

styleUrls: ['./app.component.css']

})

export class AppComponent {

title = 'Angular 4 Project!';

//array of months.

months = ["January", "Feburary", "March", "April",

"May", "June", "July", "August", "September",

"October", "November", "December"];

isavailable = true;

myClickFunction(event) {

//just added console.log which will display the event details in browser

on click of the button.

alert("Button is clicked");

console.log(event);

}

}

Onchange event

<div> Months :

<select (change) = "changemonths($event)" name = "month">

<option \*ngFor = "let i of months">{{i}}</option>

</select>

</div>

<br/>

Directives

**Directives** in Angular is a **js** class, which is declared as **@directive**. We have 3 directives in Angular. The directives are listed below −

### Component Directives

These form the main class having details of how the component should be processed, instantiated and used at runtime.

### Structural Directives

A structure directive basically deals with manipulating the dom elements. Structural directives have a \* sign before the directive. For example, **\*ngIf**and **\*ngFor**.

### Attribute Directives

Attribute directives deal with changing the look and behavior of the dom element. You can create your own directives as shown below.

## How to Create Custom Directives?

In this section, we will discuss about Custom Directives to be used in components. Custom directives are created by us and are not standard.

Let us see how to create the custom directive. We will create the directive using the command line. The command to create the directive using the command line is −

ng g directive nameofthedirective

e.g

ng g directive changeText

This is how it appears in the command line

C:\projectA4\Angular 4-app>ng g directive changeText

installing directive

create src\app\change-text.directive.spec.ts

create src\app\change-text.directive.ts

update src\app\app.module.ts

The above files, i.e., **change-text.directive.spec.ts** and **change-text.directive.ts** get created and the **app.module.ts** file is updated.

### app.module.ts

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

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

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

import { NewCmpComponent } from './new-cmp/new-cmp.component';

import { ChangeTextDirective } from './change-text.directive';

@NgModule({

declarations: [

AppComponent,

NewCmpComponent,

ChangeTextDirective

],

imports: [

BrowserModule

],

providers: [],

bootstrap: [AppComponent]

})

export class AppModule { }

The **ChangeTextDirective** class is included in the declarations in the above file. The class is also imported from the file given below.

### change-text. directive

import { Directive } from '@angular/core';

@Directive({

selector: '[changeText]'

})

export class ChangeTextDirective {

constructor() { }

}

The above file has a directive and it also has a selector property. Whatever we define in the selector, the same has to match in the view, where we assign the custom directive.

In the **app.component.html** view, let us add the directive as follows −

<div style="text-align:center">

lt;span changeText >Welcome to {{title}}.</span>

</div>

We will write the changes in **change-text.directive.ts** file as follows −

### change-text.directive.ts

import { Directive, ElementRef} from '@angular/core';

@Directive({

selector: '[changeText]'

})

export class ChangeTextDirective {

constructor(Element: ElementRef) {

console.log(Element);

Element.nativeElement.innerText="Text is changed by changeText Directive. ";

}

}

# Routing

Routing basically means navigating between pages. You have seen many sites with links that direct you to a new page. This can be achieved using routing. Here the pages that we are referring to will be in the form of components. We have already seen how to create a component. Let us now create a component and see how to use routing with it.

In the main parent component **app.module.ts**, we have to now include the router module as shown below −

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

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

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

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

import { NewCmpComponent } from './new-cmp/new-cmp.component';

import { ChangeTextDirective } from './change-text.directive';

import { SqrtPipe } from './app.sqrt';

@NgModule({

declarations: [

SqrtPipe,

AppComponent,

NewCmpComponent,

ChangeTextDirective

],

imports: [

BrowserModule,

RouterModule.forRoot([

{

path: 'new-cmp',

component: NewCmpComponent

}

])

],

providers: [],

bootstrap: [AppComponent]

})

export class AppModule { }

### import { RouterModule} from '@angular/router'

Here, the RouterModule is imported from angular/router. The module is included in the imports as shown below −

RouterModule.forRoot([

{

path: 'new-cmp',

component: NewCmpComponent

}

])

RouterModule refers to the **forRoot** which takes an input as an array, which in turn has the object of the path and the component. Path is the name of the router and component is the name of the class, i.e., the component created.

Let us now see the component created file −

### New-cmp.component.ts

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

@Component({

selector: 'app-new-cmp',

templateUrl: './new-cmp.component.html',

styleUrls: ['./new-cmp.component.css']

})

export class NewCmpComponent implements OnInit {

newcomponent = "Entered in new component created";

constructor() {}

ngOnInit() { }

}

The highlighted class is mentioned in the imports of the main module.

### New-cmp.component.html

<p>

{{newcomponent}}

</p>

<p>

new-cmp works!

</p>

Now, we need the above content from the html file to be displayed whenever required or clicked from the main module. For this, we need to add the router details in the **app.component.html**.

<h1>Custom Pipe</h1>

<b>Square root of 25 is: {{25 | sqrt}}</b><br/>

<b>Square root of 729 is: {{729 | sqrt}}</b>

<br />

<br />

<br />

<a routerLink = "new-cmp">New component</a>

<br />

<br/>

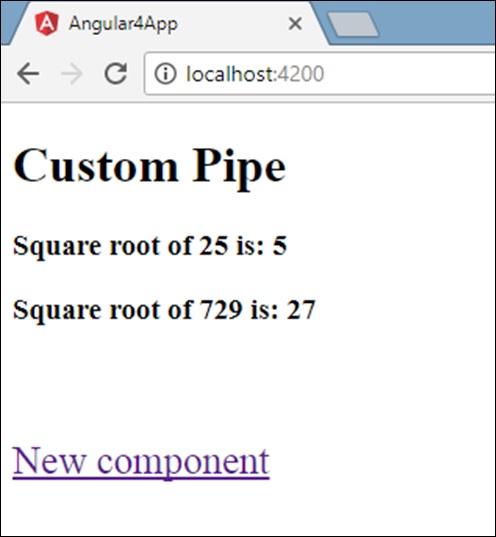
<router-outlet></router-outlet>

In the above code, we have created the anchor link tag and given routerLink as **“new-cmp”**. This is referred in **app.module.ts** as the path.

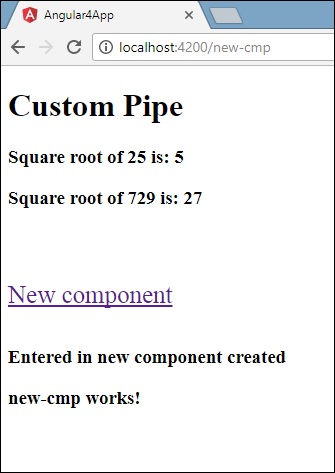
When a user clicks **new component**, the page should display the content. For this, we need the following tag - **<router-outlet> </router-outlet>**.

The above tag ensures that the content in the **new-cmp.component.html**will be displayed on the page when a user clicks **new component**.

Let us now see how the output is displayed on the browser.



When a user clicks New component, you will see the following in the browser.



The url contains **http://localhost:4200/new-cmp**. Here, the new-cmp gets appended to the original url, which is the path given in the **app.module.ts**and the router-link in the **app.component.html**.

When a user clicks New component, the page is not refreshed and the contents are shown to the user without any reloading. Only a particular piece of the site code will be reloaded when clicked. This feature helps when we have heavy content on the page and needs to be loaded based on the user interaction. The feature also gives a good user experience as the page is not reloaded.

# Services

We might come across a situation where we need some code to be used everywhere on the page. It can be for data connection that needs to be shared across components, etc. Services help us achieve that. With services, we can access methods and properties across other components in the entire project.

To create a service, we need to make use of the command line. The command for the same is −

C:\projectA4\Angular 4-app>ng g service myservice

installing service

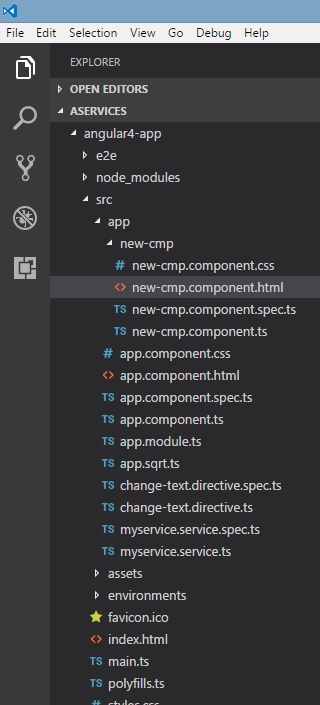
create src\app\myservice.service.spec.ts

create src\app\myservice.service.ts

WARNING Service is generated but not provided, it must be provided to be used

C:\projectA4\Angular 4-app>

The files are created in the app folder as follows −



Following are the files created at the bottom - **myservice.service.specs.ts** and **myservice.service.ts**.

### myservice.service.ts

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

@Injectable()

export class MyserviceService {

constructor() { }

}

Here, the Injectable module is imported from the **@angular/core**. It contains the **@Injectable** method and a class called **MyserviceService**. We will create our service function in this class.

Before creating a new service, we need to include the service created in the main parent **app.module.ts**.

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

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

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

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

import { MyserviceService } from './myservice.service';

import { NewCmpComponent } from './new-cmp/new-cmp.component';

import { ChangeTextDirective } from './change-text.directive';

import { SqrtPipe } from './app.sqrt';

@NgModule({

declarations: [

SqrtPipe,

AppComponent,

NewCmpComponent,

ChangeTextDirective

],

imports: [

BrowserModule,

RouterModule.forRoot([

{

path: 'new-cmp',

component: NewCmpComponent

}

])

],

providers: [MyserviceService],

bootstrap: [AppComponent]

})

export class AppModule { }

We have imported the Service with the class name and the same class is used in the providers. Let us now switch back to the service class and create a service function.

In the service class, we will create a function which will display today’s date. We can use the same function in the main parent component **app.component.ts** and also in the new component **new-cmp.component.ts**that we created in the previous chapter.

Let us now see how the function looks in the service and how to use it in components.

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

@Injectable()

export class MyserviceService {

constructor() { }

showTodayDate() {

let ndate = new Date();

return ndate;

}

}

In the above service file, we have created a function **showTodayDate**. Now we will return the new Date () created. Let us see how we can access this function in the component class.

### app.component.ts

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

import { MyserviceService } from './myservice.service';

@Component({

selector: 'app-root',

templateUrl: './app.component.html',

styleUrls: ['./app.component.css']

})

export class AppComponent {

title = 'Angular 4 Project!';

todaydate;

constructor(private myservice: MyserviceService) {}

ngOnInit() {

this.todaydate = this.myservice.showTodayDate();

}

}

The **ngOnInit** function gets called by default in any component created. The date is fetched from the service as shown above. To fetch more details of the service, we need to first include the service in the component **ts** file.

We will display the date in the **.html** file as shown below −

{{todaydate}}

<app-new-cmp></app-new-cmp>

// data to be displayed to user from the new component class.

Let us now see how to use the service in the new component created.

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

import { MyserviceService } from './../myservice.service';

@Component({

selector: 'app-new-cmp',

templateUrl: './new-cmp.component.html',

styleUrls: ['./new-cmp.component.css']

})

export class NewCmpComponent implements OnInit {

todaydate;

newcomponent = "Entered in new component created";

constructor(private myservice: MyserviceService) {}

ngOnInit() {

this.todaydate = this.myservice.showTodayDate();

}

}

In the new component that we have created, we need to first import the service that we want and access the methods and properties of the same. Please see the code highlighted. The todaydate is displayed in the component html as follows −

<p>

{{newcomponent}}

</p>

<p>

Today's Date : {{todaydate}}

</p>

The selector of the new component is used in the **app.component.html** file. The contents from the above html file will be displayed in the browser as shown below −



If you change the property of the service in any component, the same is changed in other components too. Let us now see how this works.

We will define one variable in the service and use it in the parent and the new component. We will again change the property in the parent component and will see if the same is changed in the new component or not.

In **myservice.service.ts**, we have created a property and used the same in other parent and new component.

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

@Injectable()

export class MyserviceService {

serviceproperty = "Service Created";

constructor() { }

showTodayDate() {

let ndate = new Date();

return ndate;

}

}

Let us now use the **serviceproperty** variable in other components. In **app.component.ts**, we are accessing the variable as follows −

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

import { MyserviceService } from './myservice.service';

@Component({

selector: 'app-root',

templateUrl: './app.component.html',

styleUrls: ['./app.component.css']

})

export class AppComponent {

title = 'Angular 4 Project!';

todaydate;

componentproperty;

constructor(private myservice: MyserviceService) {}

ngOnInit() {

this.todaydate = this.myservice.showTodayDate();

console.log(this.myservice.serviceproperty);

this.myservice.serviceproperty = "component created"; // value is changed.

this.componentproperty = this.myservice.serviceproperty;

}

}

We will now fetch the variable and work on the console.log. In the next line, we will change the value of the variable to “**component created**”. We will do the same in **new-cmp.component.ts**.

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

import { MyserviceService } from './../myservice.service';

@Component({

selector: 'app-new-cmp',

templateUrl: './new-cmp.component.html',

styleUrls: ['./new-cmp.component.css']

})

export class NewCmpComponent implements OnInit {

todaydate;

newcomponentproperty;

newcomponent = "Entered in newcomponent";

constructor(private myservice: MyserviceService) {}

ngOnInit() {

this.todaydate = this.myservice.showTodayDate();

this.newcomponentproperty = this.myservice.serviceproperty;

}

}

In the above component, we are not changing anything but directly assigning the property to the component property.

Now when you execute it in the browser, the service property will be changed since the value of it is changed in **app.component.ts** and the same will be displayed for the **new-cmp.component.ts**.

Also check the value in the console before it is changed.



# Http Service

Http Service will help us fetch external data, post to it, etc. We need to import the http module to make use of the http service. Let us consider an example to understand how to make use of the http service.

To start using the http service, we need to import the module in **app.module.ts**as shown below −

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

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

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

import { HttpModule } from '@angular/http';

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

@NgModule({

declarations: [

AppComponent

],

imports: [

BrowserModule,

BrowserAnimationsModule,

HttpModule

],

providers: [],

bootstrap: [AppComponent]

})

export class AppModule { }

If you see the highlighted code, we have imported the HttpModule from @angular/http and the same is also added in the imports array.

Let us now use the http service in the **app.component.ts**.

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

import { Http } from '@angular/http';

import 'rxjs/add/operator/map';

@Component({

selector: 'app-root',

templateUrl: './app.component.html',

styleUrls: ['./app.component.css']

})

export class AppComponent {

constructor(private http: Http) { }

ngOnInit() {

this.http.get("http://jsonplaceholder.typicode.com/users").

map((response) ⇒ response.json()).

subscribe((data) ⇒ console.log(data))

}

}

Let us understand the code highlighted above. We need to import http to make use of the service, which is done as follows −

import { Http } from '@angular/http';

In the class **AppComponent**, a constructor is created and the private variable http of type Http. To fetch the data, we need to use the **get API** available with http as follows

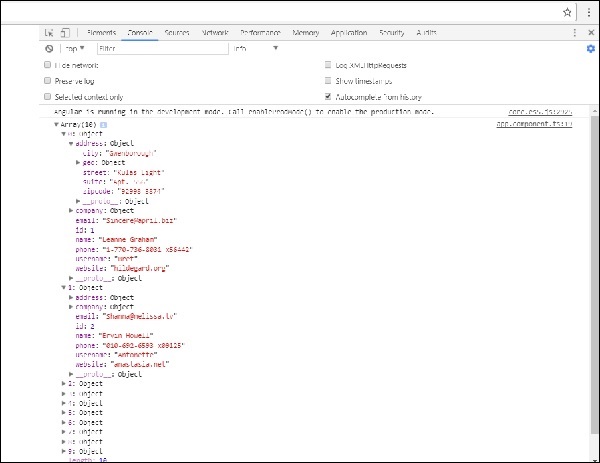
this.http.get();

It takes the url to be fetched as the parameter as shown in the code.

We will use the test url - https://jsonplaceholder.typicode.com/users to fetch the json data. Two operations are performed on the fetched url data map and subscribe. The Map method helps to convert the data to json format. To use the map, we need to import the same as shown below −

import 'rxjs/add/operator/map';

Once the map is done, the subscribe will log the output in the console as shown in the browser −



If you see, the json objects are displayed in the console. The objects can be displayed in the browser too.

For the objects to be displayed in the browser, update the codes in **app.component.html** and **app.component.ts** as follows −

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

import { Http } from '@angular/http';

import 'rxjs/add/operator/map';

@Component({

selector: 'app-root',

templateUrl: './app.component.html',

styleUrls: ['./app.component.css']

})

export class AppComponent {

constructor(private http: Http) { }

httpdata;

ngOnInit() {

this.http.get("http://jsonplaceholder.typicode.com/users").

map(

(response) ⇒ response.json()

).

subscribe(

(data) ⇒ {this.displaydata(data);}

)

}

displaydata(data) {

this.httpdata = data;

}

}

In **app.component.ts**, using the subscribe method we will call the display data method and pass the data fetched as the parameter to it.

In the display data method, we will store the data in a variable httpdata. The data is displayed in the browser using **for** over this httpdata variable, which is done in the **app.component.html** file.

<ul \*ngFor = "let data of httpdata">

<li>Name : {{data.name}} Address: {{data.address.city}}</li>

</ul>

The json object is as follows −

{

"id": 1,

"name": "Leanne Graham",

"username": "Bret",

"email": "Sincere@april.biz",

"address": {

"street": "Kulas Light",

"suite": "Apt. 556",

"city": "Gwenborough",

"zipcode": "92998-3874",

"geo": {

"lat": "-37.3159",

"lng": "81.1496"

}

},

"phone": "1-770-736-8031 x56442",

"website": "hildegard.org",

"company": {

"name": "Romaguera-Crona",

"catchPhrase": "Multi-layered client-server neural-net",

"bs": "harness real-time e-markets"

}

}

The object has properties such as id, name, username, email, and address that internally has street, city, etc. and other details related to phone, website, and company. Using the **for** loop, we will display the name and the city details in the browser as shown in the **app.component.html** file.

This is how the display is shown in the browser −



Let us now add the search parameter, which will filter based on specific data. We need to fetch the data based on the search param passed.

Following are the changes done in **app.component.html** and **app.component.ts** files −

### app.component.ts

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

import { Http } from '@angular/http';

import 'rxjs/add/operator/map';

@Component({

selector: 'app-root',

templateUrl: './app.component.html',

styleUrls: ['./app.component.css']

})

export class AppComponent {

title = 'app';

searchparam = 2;

jsondata;

name;

constructor(private http: Http) { }

ngOnInit() {

this.http.get("http://jsonplaceholder.typicode.com/users?id="+this.searchparam).

map(

(response) ⇒ response.json()

).

subscribe((data) ⇒ this.converttoarray(data))

}

converttoarray(data) {

console.log(data);

this.name = data[0].name;

}

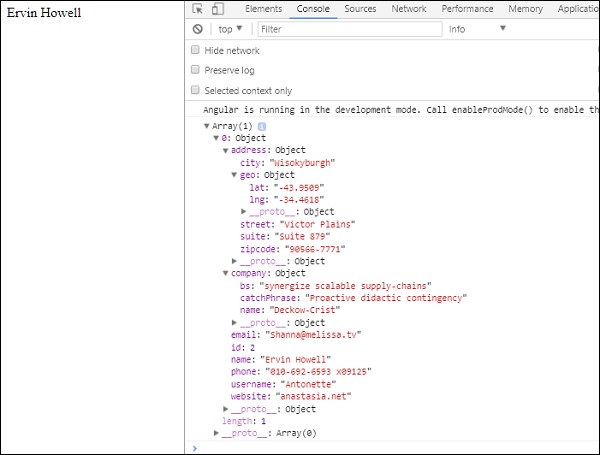
}

For the **get api**, we will add the search param id = this.searchparam. The searchparam is equal to 2. We need the details of **id=2** from the json file.

### app.component.html

{{name}}

This is how the browser is displayed −



We have consoled the data in the browser, which is received from the http. The same is displayed in the browser console. The name from the json with **id=2** is displayed in the browser.

# Forms

## Template Driven Form

With a template driven form, most of the work is done in the template;

and with the model driven form, most of the work is done in the component class.

Let us now consider working on the Template driven form. We will create a simple login form and add the email id, password and submit the button in the form. To start with, we need to import to FormsModule from **@angular/core**which is done in **app.module.ts** as follows −

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

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

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

import { HttpModule } from '@angular/http';

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

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

import { MyserviceService } from './myservice.service';

import { NewCmpComponent } from './new-cmp/new-cmp.component';

import { ChangeTextDirective } from './change-text.directive';

import { SqrtPipe } from './app.sqrt';

@NgModule({

declarations: [

SqrtPipe,

AppComponent,

NewCmpComponent,

ChangeTextDirective

],

imports: [

BrowserModule,

HttpModule,

FormsModule,

RouterModule.forRoot([

{path: 'new-cmp',component: NewCmpComponent}

])

],

providers: [MyserviceService],

bootstrap: [AppComponent]

})

export class AppModule { }

So in **app.module.ts**, we have imported the FormsModule and the same is added in the imports array as shown in the highlighted code.

Let us now create our form in the **app.component.html** file.

<form #userlogin="ngForm" (ngSubmit)="onClickSubmit(userlogin.value)" >

<input type="text" name="emailid" placeholder= "emailid" ngModel>

<br/>

<input type = "password" name="passwd" placeholder="passwd" ngModel>

<br/>

<input type = "submit" value = "submit">

</form>

We have created a simple form with input tags having email id, password and the submit button. We have assigned type, name, and placeholder to it.

In template driven forms, we need to create the model form controls by adding the **ngModel** directive and the **name** attribute. Thus, wherever we want Angular to access our data from forms, add ngModel to that tag as shown above. Now, if we have to read the emailid and passwd, we need to add the ngModel across it.

If you see, we have also added the ngForm to the **#userlogin**. The **ngForm**directive needs to be added to the form template that we have created. We have also added function **onClickSubmit** and assigned **userlogin.value** to it.

Let us now create the function in the **app.component.ts** and fetch the values entered in the form.

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

import { MyserviceService } from './myservice.service';

@Component({

selector: 'app-root',

templateUrl: './app.component.html',

styleUrls: ['./app.component.css']

})

export class AppComponent {

title = 'Angular 4 Project!';

todaydate;

componentproperty;

constructor(private myservice: MyserviceService) { }

ngOnInit() {

this.todaydate = this.myservice.showTodayDate();

}

onClickSubmit(data) {

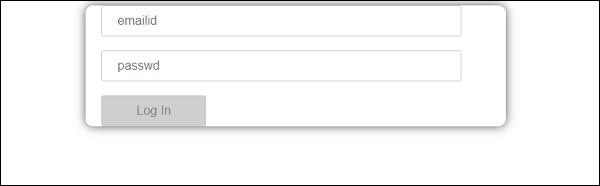
alert("Entered Email id : " + data.emailid);

}

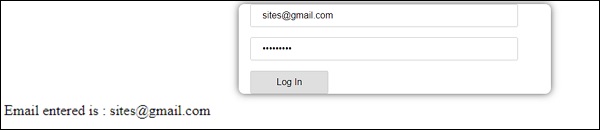
}

In the above **app.component.ts** file, we have defined the function onClickSubmit. When you click on the form submit button, the control will come to the above function.

This is how the browser is displayed −



The form looks like as shown below. Let us enter the data in it and in the submit function, the email id is already entered.



The email id is displayed at the bottom as shown in the above screenshot.

## Model Driven Form

In the model driven form, we need to import the ReactiveFormsModule from @angular/forms and use the same in the imports array.

There is a change which goes in **app.module.ts.**

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

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

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

import { HttpModule } from '@angular/http';

import { ReactiveFormsModule } from '@angular/forms';

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

import { MyserviceService } from './myservice.service';

import { NewCmpComponent } from './new-cmp/new-cmp.component';

import { ChangeTextDirective } from './change-text.directive';

import { SqrtPipe } from './app.sqrt';

@NgModule({

declarations: [

SqrtPipe,

AppComponent,

NewCmpComponent,

ChangeTextDirective

],

imports: [

BrowserModule,

HttpModule,

ReactiveFormsModule,

RouterModule.forRoot([

{

path: 'new-cmp',

component: NewCmpComponent

}

])

],

providers: [MyserviceService],

bootstrap: [AppComponent]

})

export class AppModule { }

In **app.component.ts**, we need to import a few modules for the model driven form. For example, **import { FormGroup, FormControl } from '@angular/forms'**.

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

import { MyserviceService } from './myservice.service';

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

@Component({

selector: 'app-root',

templateUrl: './app.component.html',

styleUrls: ['./app.component.css']

})

export class AppComponent {

title = 'Angular 4 Project!';

todaydate;

componentproperty;

emailid;

formdata;

constructor(private myservice: MyserviceService) { }

ngOnInit() {

this.todaydate = this.myservice.showTodayDate();

this.formdata = new FormGroup({

emailid: new FormControl("angular@gmail.com"),

passwd: new FormControl("abcd1234")

});

}

onClickSubmit(data) {this.emailid = data.emailid;}

}

The variable formdata is initialized at the start of the class and the same is initialized with FormGroup as shown above. The variables emailid and passwd are initialized with default values to be displayed in the form. You can keep it blank in case you want to.

This is how the values will be seen in the form UI.



We have used formdata to initialize the form values; we need to use the same in the form UI **app.component.html**.

<div>

<form [formGroup]="formdata" (ngSubmit) = "onClickSubmit(formdata.value)" >

<input type="text" class="fortextbox" name="emailid" placeholder="emailid"

formControlName="emailid">

<br/>

<input type="password" class="fortextbox" name="passwd"

placeholder="passwd" formControlName="passwd">

<br/>

<input type="submit" class="forsubmit" value="Log In">

</form>

</div>

<p>

Email entered is : {{emailid}}

</p>

In the .html file, we have used formGroup in square bracket for the form; for example, [formGroup]=”formdata”. On submit, the function is called **onClickSubmit** for which **formdata.value** is passed.

The input tag **formControlName** is used. It is given a value that we have used in the **app.component.ts** file.

On clicking submit, the control will pass to the function **onClickSubmit**, which is defined in the **app.component.ts** file.



On clicking Login, the value will be displayed as shown in the above screenshot.

## Form Validation

Let us now discuss form validation using model driven form. You can use the built-in form validation or also use the custom validation approach. We will use both the approaches in the form. We will continue with the same example that we created in one of our previous sections. With Angular 4, we need to import Validators from **@angular/forms** as shown below −

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

Angular has built-in validators such as **mandatory field, minlength, maxlength**, and **pattern**. These are to be accessed using the Validators module.

You can just add validators or an array of validators required to tell Angular if a particular field is mandatory.

Let us now try the same on one of the input textboxes, i.e., email id. For the email id, we have added the following validation parameters −

* Required
* Pattern matching

This is how a code undergoes validation in **app.component.ts**.

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

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

@Component({

selector: 'app-root',

templateUrl: './app.component.html',

styleUrls: ['./app.component.css']

})

export class AppComponent {

title = 'Angular 4 Project!';

todaydate;

componentproperty;

emailid;

formdata;

ngOnInit() {

this.formdata = new FormGroup({

emailid: new FormControl("", Validators.compose([

Validators.required,

Validators.pattern("[^ @]\*@[^ @]\*")

])),

passwd: new FormControl("")

});

}

onClickSubmit(data) {this.emailid = data.emailid;}

}

In **Validators.compose**, you can add the list of things you want to validate on the input field. Right now, we have added the **required** and the **pattern matching** parameters to take only valid email.

In the **app.component.html**, the submit button is disabled if any of the form inputs are not valid. This is done as follows −

<div>

<form [formGroup] = "formdata" (ngSubmit) = "onClickSubmit(formdata.value)" >

<input type = "text" class = "fortextbox" name = "emailid" placeholder = "emailid"

formControlName = "emailid">

<br/>

<input type = "password" class = "fortextbox" name = "passwd"

placeholder = "passwd" formControlName = "passwd">

<br/>

<input type = "submit" [disabled] = "!formdata.valid" class = "forsubmit"

value = "Log In">

</form>

</div>

<p>

Email entered is : {{emailid}}

</p>

For the submit button, we have added disabled in the square bracket, which is given value - **!formdata.valid**. Thus, if the formdata.valid is not valid, the button will remain disabled and the user will not be able to submit it.

Let us see the how this works in the browser −



In the above case, the email id entered is invalid, hence the login button is disabled. Let us now try entering the valid email id and see the difference.



Now, the email id entered is valid. Thus, we can see the login button is enabled and the user will be able to submit it. With this, the email id entered is displayed at the bottom.

Let us now try custom validation with the same form. For custom validation, we can define our own custom function and add the required details in it. We will now see an example for the same.

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

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

@Component({

selector: 'app-root',

templateUrl: './app.component.html',

styleUrls: ['./app.component.css']

})

export class AppComponent {

title = 'Angular 4 Project!';

todaydate;

componentproperty;

emailid;

formdata;

ngOnInit() {

this.formdata = new FormGroup({

emailid: new FormControl("", Validators.compose([

Validators.required,

Validators.pattern("[^ @]\*@[^ @]\*")

])),

passwd: new FormControl("", this.passwordvalidation)

});

}

passwordvalidation(formcontrol) {

if (formcontrol.value.length <'; 5) {

return {"passwd" : true};

}

}

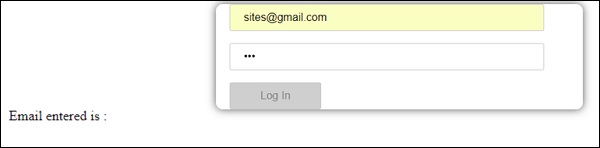
onClickSubmit(data) {this.emailid = data.emailid;}

}

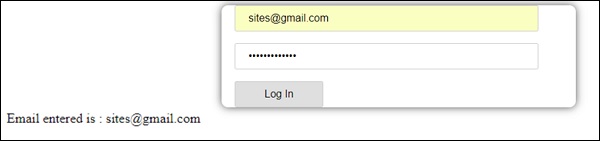
In the above example, we have created a function **password validation** and the same is used in a previous section in the formcontrol - **passwd: new FormControl("", this.passwordvalidation)**.

In the function that we have created, we will check if the length of the characters entered is appropriate. If the characters are less than five, it will return with the passwd true as shown above - **return {"passwd" : true};**. If the characters are more than five, it will consider it as valid and the login will be enabled.

Let us now see how this is displayed in the browser −



We have entered only three characters in the password and the login is disabled. To enable login, we need more than five characters. Let us now enter a valid length of characters and check.



The login is enabled as both the email id and the password are valid. The email is displayed at the bottom as we log in.

# Animations

Animations add a lot of interaction between the html elements. Animation was also available with Angular2. The difference with Angular 4 is that animation is no more a part of the **@angular/core** library, but is a separate package that needs to be imported in **app.module.ts**.

To start with, we need to import the library as follows −

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

The **BrowserAnimationsModule** needs to be added to the import array in **app.module.ts** as shown below −

### app.module.ts

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

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

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

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

@NgModule({

declarations: [

AppComponent

],

imports: [

BrowserModule,

BrowserAnimationsModule

],

providers: [],

bootstrap: [AppComponent]

})

export class AppModule { }

In **app.component.html**, we have added the html elements, which are to be animated.

<div>

<button (click)="animate()">Click Me</button>

<div [@myanimation] = "state" class="rotate">

<img src="assets/images/img.png" width="100" height="100">

</div>

</div>

For the main div, we have added a button and a div with an image. There is a click event for which the animate function is called. And for the div, the **@myanimation** directive is added and given the value as state.

Let us now see the **app.component.ts** where the animation is defined.

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

import { trigger, state, style, transition, animate } from '@angular/animations';

@Component({

selector: 'app-root',

templateUrl: './app.component.html',

styleUrls: ['./app.component.css'],

styles:[`

div{

margin: 0 auto;

text-align: center;

width:200px;

}

.rotate{

width:100px;

height:100px;

border:solid 1px red;

}

`],

animations: [

trigger('myanimation',[

state('smaller',style({

transform : 'translateY(100px)'

})),

state('larger',style({

transform : 'translateY(0px)'

})),

transition('smaller <=> larger',animate('300ms ease-in'))

])

]

})

export class AppComponent {

state: string = "smaller";

animate() {

this.state= this.state == 'larger' ? 'smaller' : 'larger';

}

}

We have to import the animation function that is to be used in the .ts file as shown above.

import { trigger, state, style, transition, animate } from '@angular/animations';

Here we have imported trigger, state, style, transition, and animate from @angular/animations.

Now, we will add the animations property to the @Component () decorator −

animations: [

trigger('myanimation',[

state('smaller',style({

transform : 'translateY(100px)'

})),

state('larger',style({

transform : 'translateY(0px)'

})),

transition('smaller <=> larger',animate('300ms ease-in'))

])

]

Trigger defines the start of the animation. The first param to it is the name of the animation to be given to the html tag to which the animation needs to be applied. The second param are the functions we have imported - state, transition, etc.

The **state** function involves the animation steps, which the element will transition between. Right now we have defined two states, smaller and larger. For smaller state, we have given the style **transform:translateY(100px)**and **transform:translateY(100px)**.

Transition function adds animation to the html element. The first argument takes the states, i.e., start and end; the second argument accepts the animate function. The animate function allows you to define the length, delay, and easing of a transition.

Let us now see the .html file to see how the transition function works

<div>

<button (click)="animate()">Click Me</button>

<div [@myanimation] = "state" class="rotate">

<img src="assets/images/img.png" width="100" height="100">

</div>

</div>

There is a style property added in the **@component** directive, which centrally aligns the div. Let us consider the following example to understand the same −

styles:[`

div{

margin: 0 auto;

text-align: center;

width:200px;

}

.rotate{

width:100px;

height:100px;

border:solid 1px red;

}

`],

Here, a special character [``] is used to add styles to the html element, if any. For the div, we have given the animation name defined in the app.component.ts file.

On the click of a button it calls the animate function, which is defined in the app.component.ts file as follows −

export class AppComponent {

state: string = "smaller";

animate() {

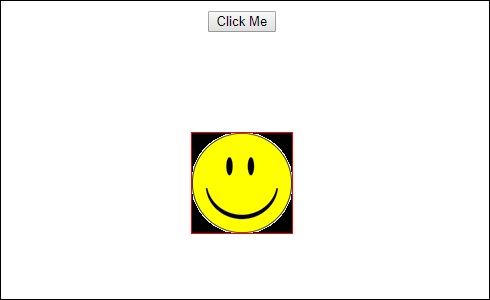
this.state= this.state == ‘larger’? 'smaller' : 'larger';

}

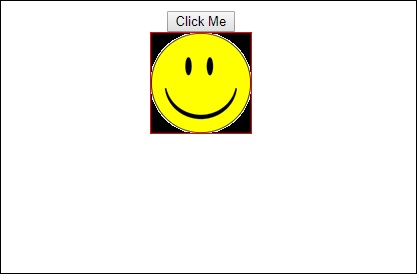
}

The state variable is defined and is given the default value as smaller. The animate function changes the state on click. If the state is larger, it will convert to smaller; and if smaller, it will convert to larger.

This is how the output in the browser (**http://localhost:4200/**) will look like −



Upon clicking the **Click Me** button, the position of the image is changed as shown in the following screenshot −



The transform function is applied in the **y** direction, which is changed from 0 to 100px when the Click Me button is clicked. The image is stored in the **assets/images** folder.

# Materials

**Materials** offer a lot of built-in modules for your project. Features such as autocomplete, datepicker, slider, menus, grids, and toolbar are available for use with materials in Angular 4.

To use materials, we need to import the package. Angular 2 also has all the above features but they are available as part of the @angular/core module. Angular 4 has come up with a separate module **@angular/materials.**. This helps the user to import the required materials.

To start using materials, you need to install two packages - materials and cdk. Material components depend on the animation module for advanced features, hence you need the animation package for the same, i.e., @angular/animations. The package has already been updated in the previous chapter.

npm install --save @angular/material @angular/cdk

Let us now see the package.json. **@angular/material** and **@angular/cdk**are installed.

{

"name": "angularstart",

"version": "0.0.0",

"license": "MIT",

"scripts": {

"ng": "ng",

"start": "ng serve",

"build": "ng build",

"test": "ng test",

"lint": "ng lint",

"e2e": "ng e2e"

},

"private": true,

"dependencies": {

"@angular/animations": "^4.0.0",

"@angular/cdk": "^2.0.0-beta.8",

"@angular/common": "^4.0.0",

"@angular/compiler": "^4.0.0",

"@angular/core": "^4.0.0",

"@angular/forms": "^4.0.0",

"@angular/http": "^4.0.0",

"@angular/material": "^2.0.0-beta.8",

"@angular/platform-browser": "^4.0.0",

"@angular/platform-browser-dynamic": "^4.0.0",

"@angular/router": "^4.0.0",

"core-js": "^2.4.1",

"rxjs": "^5.1.0",

"zone.js": "^0.8.4"

},

"devDependencies": {

"@angular/cli": "1.2.0",

"@angular/compiler-cli": "^4.0.0",

"@angular/language-service": "^4.0.0",

"@types/jasmine": "~2.5.53",

"@types/jasminewd2": "~2.0.2",

"@types/node": "~6.0.60",

"codelyzer": "~3.0.1",

"jasmine-core": "~2.6.2",

"jasmine-spec-reporter": "~4.1.0",

"karma": "~1.7.0",

"karma-chrome-launcher": "~2.1.1",

"karma-cli": "~1.0.1",

"karma-coverage-istanbul-reporter": "^1.2.1",

"karma-jasmine": "~1.1.0",

"karma-jasmine-html-reporter": "^0.2.2",

"protractor": "~5.1.2",

"ts-node": "~3.0.4",

"tslint": "~5.3.2",

"typescript": "~2.3.3"

}

}

We have highlighted the packages that are installed to work with materials.

We will now import the modules in the parent module - **app.module.ts** as shown below.

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

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

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

import { MdButtonModule, MdMenuModule, MdSidenavModule } from '@angular/material';

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

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

@NgModule({

declarations: [

AppComponent

],

imports: [

BrowserModule,

BrowserAnimationsModule,

MdButtonModule,

MdMenuModule,

FormsModule,

MdSidenavModule

],

providers: [],

bootstrap: [AppComponent]

})

export class AppModule { }

In the above file, we have imported the following modules from @angular/materials.

import { MdButtonModule, MdMenuModule, MdSidenavModule } from '@angular/material';

And the same is used in the imports array as shown below −

imports: [

BrowserModule,

BrowserAnimationsModule,

MdButtonModule,

MdMenuModule,

FormsModule,

MdSidenavModule

]

The **app.component.ts** is as shown below −

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

@Component({

selector: 'app-root',

templateUrl: './app.component.html',

styleUrls: ['./app.component.css']

})

export class AppComponent {

myData: Array<any>;

constructor() {}

}

Let us now add the material in **app.component.html**.

<button md-button [mdMenuTriggerFor]="menu">Menu</button>

<md-menu #menu="mdMenu">

<button md-menu-item>

File

</button>

<button md-menu-item>

Save As

</button>

</md-menu>

<md-sidenav-container class="example-container">

<md-sidenav #sidenav class="example-sidenav">

Angular 4

</md-sidenav>

<div class="example-sidenav-content">

<button type="button" md-button (click)="sidenav.open()">

Open sidenav

</button>

</div>

</md-sidenav-container>

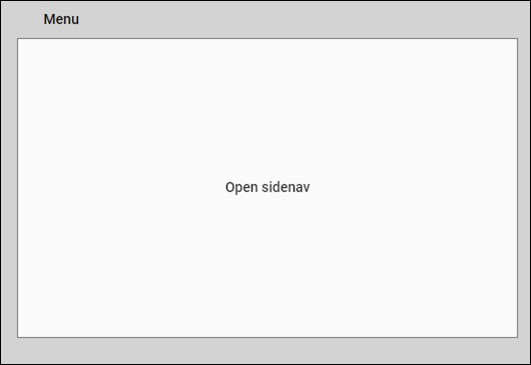
In the above file, we have added Menu and SideNav.

### Menu

To add menu, **<md-menu></md-menu>** is used. The **file** and **Save As**items are added to the button under **md-menu**. There is a main button added **Menu**. The reference of the same is given the <md-menu> by using **[mdMenuTriggerFor]=”menu”** and using the menu with **# in <md-menu>**.

### SideNav

To add sidenav, we need **<md-sidenav-container></md-sidenav-container>**. **<md-sidenav></md-sidenav>** is added as a child to the container. There is another div added, which triggers the sidenav by using **(click)=”sidenav.open()”**. Following is the display of the menu and the sidenav in the browser −



Upon clicking **opensidenav**, it shows the side bar as shown below −



Upon clicking Menu, you will get two items **File** and **Save As** as shown below −



Let us now add a datepicker using materials. To add a datepicker, we need to import the modules required to show the datepicker.

In **app.module.ts**, we have imported the following module as shown below for datepicker.

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

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

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

import { MdDatepickerModule, MdInputModule, MdNativeDateModule } from '@angular/material';

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

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

@NgModule({

declarations: [

AppComponent

],

imports: [

BrowserModule,

BrowserAnimationsModule,

FormsModule,

MdDatepickerModule,

MdInputModule,

MdNativeDateModule

],

providers: [],

bootstrap: [AppComponent]

})

export class AppModule { }

Here, we have imported modules such as **MdDatepickerModule, MdInputModule,** and **MdNativeDateModule**.

Now, the **app.component.ts** is as shown below −

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

@Component({

selector: 'app-root',

templateUrl: './app.component.html',

styleUrls: ['./app.component.css']

})

export class AppComponent {

myData: Array<any>;

constructor() {}

}

The **app.component.html** is as shown below −

<md-input-container>

<input mdInput [mdDatepicker]="picker" placeholder="Choose a date">

<button mdSuffix [mdDatepickerToggle]="picker"></button>

</md-input-container>

<md-datepicker #picker></md-datepicker>

This is how the datepicker is displayed in the browser −

