Angular 4 - Forms

In this chapter, we will see how forms are used in Angular 4. We will discuss two ways of working with forms - template driven form and model driven 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]

})

exportclassAppModule{}

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']

})

exportclassAppComponent{

title ='Angular 4 Project!';

todaydate;

componentproperty;

constructor(private myservice:MyserviceService){}

ngOnInit(){

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

}

onClickSubmit(data){

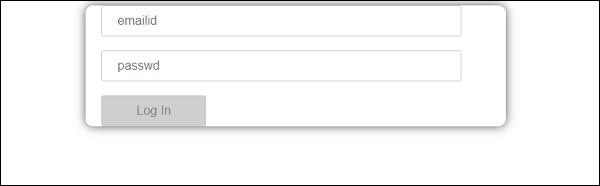
alert("Entered Email id : "+ data.emailid);

}

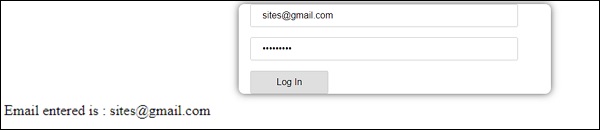
}

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]

})

exportclassAppModule{}

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']

})

exportclassAppComponent{

title ='Angular 4 Project!';

todaydate;

componentproperty;

emailid;

formdata;

constructor(private myservice:MyserviceService){}

ngOnInit(){

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

this.formdata =newFormGroup({

emailid:newFormControl("angular@gmail.com"),

passwd:newFormControl("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)">

<inputtype="text"class="fortextbox"name="emailid"placeholder="emailid"

formControlName="emailid">

<br/>

<inputtype="password"class="fortextbox"name="passwd"

placeholder="passwd"formControlName="passwd">

<br/>

<inputtype="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)">

<inputtype="text"class="fortextbox"name="emailid"placeholder="emailid"

formControlName="emailid">

<br/>

<inputtype="password"class="fortextbox"name="passwd"

placeholder="passwd"formControlName="passwd">

<br/>

<inputtype="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']

})

exportclassAppComponent{

title ='Angular 4 Project!';

todaydate;

componentproperty;

emailid;

formdata;

ngOnInit(){

this.formdata =newFormGroup({

emailid:newFormControl("",Validators.compose([

Validators.required,

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

])),

passwd:newFormControl("",this.passwordvalidation)

});

}

passwordvalidation(formcontrol){

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

return {"passwd" : true};

}

}

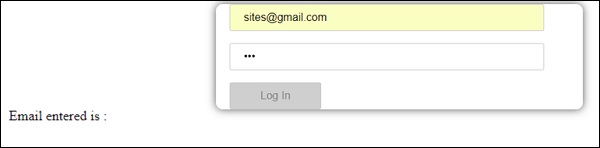
onClickSubmit(data) {this.emailid = data.emailid;}

}

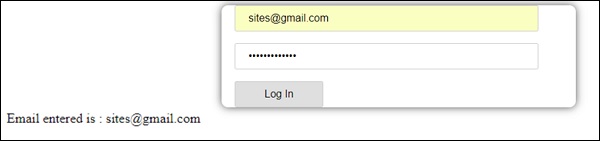
In the above example, we have created a function **password validation** and the same is used in a previous section in the formcontrol - **passwd: new FormControl("", this.passwordvalidation)**.

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.