Angular Document(v3)

Created by Vineet Semwal
(vineetsemwal82@gmail.com)

Angular Overview

Angular is component-oriented framework. You will write tiny components and, together, they will form a complete application. In programming Component is considered a reusable piece of code, Every component does a particular task

Angular Cli

Official toolchain for building angular applications

Download recent nodejs version and install cli

npm install -g @angular/cli

g is the argument that tells npm to install angular cli globally so that it can be used from different projects

1)Helps in creating new project from scratch
2)Helps in creating new angular components
3)Builds the project, that includes compiling typescript
4)The CLI handles the build flow and then starts
5)Gives embedded application server, a server listening on localhost so you can see the results with a live reload feature.

Create new project

```
ng new project-name
ng new project-name --skip-install
```

Let’s see various interesting things in the created project

1)**node_modules :** this directory contains different libraries downloaded for your project

2) **contains one component** :

Each angular component contains

A) markup file(html)
B) Component code class in ts file
C) style file(css)
D)spec.ts files contains code to test our component

Continued in next page

Let’s look at component class in the below code, the name of the class is AppComponent,
i)The name is irrelevant(can be any name), what makes it a component is the @Component decorator(decorator is same as annotation in java)
ii)@Component decorator provides metadata to the angular ie. it provides information about Component to the angular

Code below with explanation

```
@Component({
// selector ie. the name(here app-root) which when angular finds in html, it replaces with the component
selector: 'app-root',
// location of markup file relative to this class
templateUrl: './app.component.html',
  // location of css file relative to this class
styleUrls: ['./app.component.css']
})
export class AppComponent{}
```

3) **Contains one Module :**
Your application will always have at least one module, the root module
Module in angular is used to group components, directives, pipes and services
(there is a module concept in typescript, Dont confuse that with angular module, Both are different things!)
Let’s look at module class, the name of the class is AppModule, The name is irrelevant, what makes this class a module is the @NgModule decorator
Code below with explanation

```
  @NgModule({
//components of application are declared here
   declarations: [
      AppComponent
   ],
//libraries which our project needs are mentioned in imports
   imports: [
      BrowserModule
   ],
 // services(discussed later) which we need to be injected in components etc are mentioned in providers
   providers: [UserService],
 // root component
   bootstrap: [AppComponent]
})
```

```
export class AppModule { }
```

Creating our Component

```
ng generate component component_name
```

Or

```
ng   g   c   component_name
```

This will create a component ts file with its associated template, stylesheet and test file

```
import { Component } from '@angular/core';
@Component({
// selector ie. the name(here user-details) which when angular finds in html, it replaces with the component
selector: 'user-details',
// location of markup file relative to this class
templateUrl: 'userdetails.html' ,
  // location of css file relative to this class
styleUrls: ['./app.component.css']
})
export class UserDetailsComponent {
user: User=new User(1,’scooby’,4);
}
export class User{
id:number;
name:string;
age:number;
constructor(id:number,name:string, age:number){
this.id=id;
this.name=name;
this.age=age;
}
}
```

1) Every time the selector ‘user-details’ (<user-details></user-details>) is found in our HTML, Angular is going to replace the element selected by our component
2) Dash in selector name user-details is a convention
3) path of html representing component is provided in template-Url
4) Path of stylesheets for component is provided in styleUrls

Interpolation

In markup file

{{user.username}}

{{user.age}}

This will display/render value of username in browser
The magic is that, whenever the value of username changes in our object, the template will be automatically updated! That’s called 'change detection'

The expression is used in interpolation which means
1) it can’t have assignments {{user.username="fluffy”}} is wrong
2)it can’t contain let, var, for if etc

if we try to display a variable is undefined, then instead of displaying undefined, Angular is going to display an empty string. The same will happen for a null variable

However if you are mentioning a property in markup file which doesnt exist in component class then you will get property doesn’t exist error

Property Binding

the interpolation is just an easy way to use property binding which is the core of angular templating system
In Angular, every DOM property and not just html attributes can be written to via special attributes on HTML elements surrounded with square brackets []

Property binding is especially useful at places where existence of attribute itself guarantee the certain behavior which can't be changed unless attribute is removed, hidden demo in one of the example below

<span [textContent]="user.username"></span>
Same in interpolation
{{user.username}}

Another example
If your link source was not dynamic
<a href="url to the page”>visit link </a>

If your link source is dynamic
<a   [href]="linksrc">visit link</a>
Where linksrc is the field in the component class

<div [hidden]="isHidden">Hidden or not</div>
Where isHidden is the field in component class
If it was not dynamic, it is hidden
<div hidden>Hidden </div>
<div>not Hidden </div>

Event Binding

Html has a lot of predefined events like click event on any field, keyup/keydown event on form input like <input> etc , we can attach handler functions to these html events ,
see example below

<button onclick=”myfun()”>Click me   </button>
Calls myfun function in the javascript loaded with page

Angular has good support for event binding, event binding is not just possible on just html events but also on custom events
<button (click)="myfun()">Click me</button>
Where myfun() is the function defined in typescript file

```
export class AppComponent{
title = 'experiments-app2';
counter=0;
myfun(){
this.counter++;
console.log("counter="+this.counter);
}
}
```

Similary bindings can be done with other events
<input type="text" (keyup)="myfun()">
Call myfun() on every keyup event

Two way Binding

We have seen two bindings till now
1) Data Binding : Change in data changes the view using interpolation ie. using{} or property binding ie. using []
2) Event binding: we can attach handler function to events like above using which we can also save/change data when there is input in view so we can say change in view can change data

Angular supports both these powerful features at once in one syntax
[(ngModel)] : known as banana in box, [] represents property binding, () represents event binding
The ngModel directive allows you to display a data property and update that property when

the user makes changes.
myInput is field in the component class

```
<input type="text" [(ngModel)]="myInput" name="myTextInput">
my input is {{myInput}}
```

Requirement: Application should mention FormsModule in imports section of module file

Local Variables(In template)

Local variables are variables that you can dynamically declare in your template using the # syntax , precede the local variable with #

```
<input type="text" #description>
<button (click)="description.focus()">
   Focus on textfield
</button>
```

This local variable description can now be used anywhere in the template
Focus on text field when button is clicked

See below example

```
<input type="text" (keyup)="true"   #description>
Description is {{description.value}}
```

Directives

Directive means instructions
Directives are a powerful tool to teach HTML elements new skills that doesn’t exist in the standard html

Three kinds of directives

1) **Attribute directives:** changes the appearance or behavior of an element or component.
2) **Components**—directives with a template(<user-details></user-details> yes this is our directive)
3) **Structural directives**—change the DOM layout by adding and removing DOM elements.

Structural directives are preceded with * (star)
***ngFor, *ngIf, *ngSwitch**

Attribute Directives

1) changes the appearance or behavior of an element or component.
2) used as attributes of elements

Builtin examples **ngStyle , ngClass**
ngModel, ngForm (discussed later in form section)

ngStyle
1)updates style of the containing HTML elements dynamically
2)Style Properties are specified in key value pairs

Example below
Here titleColor and titleBackGround are fields in the component containing colors

```
<div [ngStyle]="{'color':titleColor , 'background-color':titleBackGround}">
{{title}}
</div>
```

ngClass
Helps in Adding and removing CSS classes on an HTML element dynamically
Property value can be boolean variable or expression that evaluates to boolean or a function returning boolean

titleClass is the css class which will be added to the element if the booleanVariable evaluates to true

```
<div [ngClass]="{'titleClass':true}">
{{title}}
</div>
```

Structural directives—change the DOM layout by adding and removing DOM elements.
Structural directives are preceded with * (star)
***ngFor, *ngIf**

***ngIf**
It takes a boolean expression and makes an element appear or disappear.
See example
mynumber is a field in the component

```
<div *ngIf="mynumber%2===0">
<h2>Number is even</h2>
</div>
```

The ngIf will or will not display the div whenever the value of mynumber field in the component class changes.
mynumber is a field in the component

***ngFor**

Helps in looping on the data

Example below

users is User[] , where User class contains id, name

```
<div *ngFor="let user of users; let i=index">
    {{user.id}} ,{{ user.name}}
</div>
```

Pipes

Used for transforming and formatting data, they take input and give output

Pipes can be used for chaining foreg. {{input|pipe1| pipe2}}

Various kind of pipes are

lowercase, uppercase, currency, percent, date

Example below, will transform data in myField to uppercase data

{{ myField | uppercase }}

Formatting to indian currency with currency pipe, value could have come from field in component

{{'10.45'| currency:'INR'}}

Creating custom Pipe

Procedure

1)Create your pipe class and implements PipeTransform

2)Implements transform method with takes input

3)Mention @Pipe decorator on your pipe class and specify name of pipe

4)Mention your created pipe class in declarations section in module file

Example below

{{ “hello” | length}}

```
import { PipeTransform, Pipe } from '@angular/core';
// name of pipe is length
@Pipe({name:'length' })
/**
This pipe takes string and returns length of string
**/
export class LengthPipe implements PipeTransform{
```

```
transform(value: string):number {
return value.length;
}
}
```

In app.module.ts

```
@NgModule({
declarations: [
AppComponent,
LengthPipe // mention name of pipe class here
],
imports: [BrowserModule],
providers: [],
bootstrap: [AppComponent]
})
export class AppModule { }
```

Forms

Requirement: Application should mention FormsModule in imports section of module file

In Angular Forms handling can be done in two ways

1) Template Driven way
2) Code Driven way(Reactive Forms)

Template Driven way

we add the proper directives in the template and the angular framework takes care of the most of the form handling (Form representation creation which is done in code driven form handling)

1) #data is local template variable and we assign it the ngForm object created by Angular for the form
2) ngSubmit event is emitted by the ngForm directive when submit is triggered
3) ngModel creates the formcontrol which tracks the value, user interaction, and validation status of the formcontrol (formcontrols we create on our own in code driven form handling)

Look at below code and read the above points again

```
<form   #data="ngForm"   (ngSubmit)="addUser(data.value)">
    <label>Name</label>
ngModel creates the formcontrol which tracks the value, user interaction, and validation status of the control
        <input type="text" ngModel name="username">
      <label>Id</label>
      <input type="number" ngModel name="id">
      <input type="submit" value="submit">
</form>
```

And in Component class is the addUser() handler function which is mentioned for ngSubmit event in html above

```
addUser(details:any){
    let id=details.id;
    let name=details.username;
    console.log(“received id=”+id+” name=“+name);
}
```

Form Validation

Angular supports form validation for both template driven and Code driven

```
<!--
template variable #myform represents form in code driven side
-->
<form #myform=ngForm
(ngSubmit)="myform.valid && register(myform)">

<!--
template variable name(mentioned by #name)
represents formcontrol for the username field
name.errors represents the validation errs
errors.required represents the validation errs for required
name.errors.minlength represents the validations errs for minlength
cheking names.errors!=null because it will be null when there are no errs
-->
<input name="username"
class="form-control" required minlength="4"
ngModel #name="ngModel">
<div *ngIf="name.errors!=null &&
name.errors.required">
Name is required.
</div>

<div *ngIf="name.errors!=null && name.errors.minlength">
minimum length should be 4
</div>

<button type="submit">Submit</button>
</form>
<div *ngIf="name.errors==null">
registered with {{username}}
</div>
```

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


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


constructor() { }
username="";
ngOnInit(): void {
}
register(myform){
console.log("myform="+myform.value.username);
this.username=myform.value.username;
}
}
```

Reactive Programming

In reactive programming, everything is a stream. A stream is an ordered sequence of events. These events represent 1)values, 2)errors or 3)completion events. All these are pushed from the data producer to the consumer.

Observable

It is used to work with asynchronous data,
Observable represents any set of values over any amount of time.
Observer is used for passing data between publisher and subscriber in the application.
Unlike Promise which was introduced in echma6, Observable can deliver multiple values to subscriber over a period of time.

Three handlers(functions) are specified for next, error, complete while subscribing to Observable

next : Required. A handler for each delivered value. Called zero or more times after execution starts.

error : Optional. A handler for an error notification. An error halts execution of

the observable instance.

complete: Optional. A handler for the execution-complete notification.

Two ways to subscribe to observable object

First Way

```
fetched.subscribe(
      users=>{
    this.users=users;
 },
     err=>{
       console.log("err in fetching="+err);
      },
      ()=>{
     console.log("completed");
   }
);
```

Second way

```
let observer:Observer<User[]>={
            next:users=>{
           this.users=users;
},
            error:err=>{
            console.log("error="+err);
},
      complete:()=>{console.log(“completed”);}
         };
 fetched.subscribe(observer);
```

Observables can be transformed like arrays are transformed by functions like map(*), filter(*), reduce(*). Observables can also be transformed,   See below example

```
 let observable:Observable<number>= of(1,2,3,4);// data 1,2,3,4 will be provided over period of time
   observable.pipe(filter(num=>num%2==0))   //chain by using pipe, use filter function of rxjs
 // subscribing observable for data that will be available over period of time
//subscribed and provided handler functions 1) success 2) error 3) complete
   observable.subscribe(num=>console.log(num),
   err=>console.log("err="+err),
   ()=>console.log("completed")
   );
   }
```

Routing

In angular routing is done to navigate from one component to another

Procedure

1) Mention routes to components in app-routingmodule.ts

Below we have mentioned two components and their routes

```
const routes: Routes = [
     {     path:'item-details',   component:ItemDetailsComponent   },
     {     path:'list-items',      component:ListItemsComponent   }];
```

2) Provide the link where there is a need

the link href will be computed by the router

```
<a href="" routerLink="list-items">List items</a>
```

3) Don’t forget to mention <router-outlet><router-outlet> in the html , It works as a place holder for the component which is referred in routerLink above

4) Don’t forget to mention AppRoutingModule in the app.module.ts as this contains the library that supports routing

Programmatic Navigation

Navigation can also be done by router.navigate() method in the below way

Here when app component is rendered, router.navigate([‘list-items’]) which causes the navigation to component with list-item route , Oninit is a lifecycle hook discussed later, For now assume ngOnit() is called as soon as component is instantiated by angular for rendering

```
@Component({
selector: 'app-root',
templateUrl: './app.component.html',
styleUrls: ['./app.component.css']
})
export class AppComponent implements OnInit{
title = 'routing-ex';
router:Router;
constructor(router:Router ){
this.router=router;
}
ngOnInit(){
this.router.navigate(['list-items']);
}
}
```

Pass data in navigation

Below we are passing routerLink

```
<a href="" routerLink="../item-details/9">Item details</a>
const routes: Routes = [
{      path:'item-details/:id',   component:ItemDetailsComponent   },
{      path:'list-items',      component:ListItemsComponent   }
];
@Component({
selector: 'item-details',
templateUrl: './item-details.component.html',
styleUrls: ['./item-details.component.css']
})
export class ItemDetailsComponent implements OnInit {

route:ActivatedRoute;//Provides information about a route associated with a component that is loaded in router outlet
```

```
constructor(route:ActivatedRoute) {
this.route=route;
}

ngOnInit(): void {
//
//param map is the property which is just an observable so we are subscribing to it and passing the success
//handler and inside getting the parameters value
//
this.route.paramMap.subscribe(
(params:ParamMap)=>{
let idValue= params.get("id");
console.log("param name=id"+" value="+idValue);
});
}
```

Adding bootstrap support

Procedure

1) npm install --save bootstrap   jquery

This will download and save bootstrap and jquery modules in node_modules of your project

2) add the bootstrap core css and bootstrap js and jquery js scripts in angular.json

After adding, styles and scripts section will appear like this in angular.json

```
"styles": [
"src/styles.css",
"node_modules/bootstrap/dist/css/bootstrap.min.css"
],
"scripts": [
"node_modules/jquery/dist/jquery.min.js",
"node_modules/bootstrap/dist/js/bootstrap.min.js"
```

]

3)Feel free to use the bootstrap components now