**Angular 7|8 Short Notes**

1. For Angular Project you Required to install node js first.
2. **Command to install angular 7 0r 8 latest:** npm install -g @angular/cli
3. **Create new angular project** : ng new projectname
4. **Compile project and start web server**: ng serve
5. **Change Port** : 4200 is the default port used when a new project is created. You can change the port with the following command:

**ng serve --host 0.0.0.0 --port 4201**

1. **package.json :** There is list of installed packages and versions.
2. **app.module.ts:** It imports all components, modules and services in project.

In that @NgModule

(

Contains-

**declaration** contains only **components**

**imports** contains only **modules**

**providers** contains only **services**

**Bootstrap This includes the main app component for starting the execution.**

1. **app.routing.module.ts:** For route url and path. if you have selected angular routing during your project setup then this file is generated. This file is generated default when we created new project using the angular-cli command.
2. **app.component.css:**  You can write your css here.
3. **app.component.html:** The html code will be available in this file.

In  **<router-outlet></router-outlet>** tag loads the dynamic components.html files in another components as per routing.

1. **app.component.spec.ts:** These are automatically generated files which contain unit tests for source component.
2. **app.component.ts :** The class for the component is defined over here. The processing of html like connecting to the database, interacting with other components, routing, services, etc.
3. **Index.html :** It contain body has **<app-root></app-root>.** This is the selector which is used in **app.component.ts** file and will display the details from **app.component.html** file.
4. **main.ts** : we start our project development from this file. It import basic module which we need.
5. angular/core
6. angular/platform- browser-dynamic
7. app.module
8. environment is imported by default during angular-cli installation and project setup.
9. **angular-cli has a command to create your own component :**

ng g component componentname

Following changes automatically done in app.module.ts file when we create component :

import {ComponentnameComponent}

from './componentname/componentname.component';

1. **Projet runnig flow:**
2. Command to compile and run: ng serve
3. hit the url in the browser : <http://localhost:4200/>
4. It first executes the index.html file
5. In index.html file contains <app-root></app-root> by default. This tag has the reference in the **main.ts** file.
6. In **main.ts** file **AppModule** is imported from the app of the main parent module( app.module.ts).
7. In **app.module.ts** all components are imported but in that AppComponent are Parent and other are child Components. AppComponent is given to the bootstrap.
8. In **app.component.ts** selector **app-root** which is as a tag placed in the index.html file.
9. So always (by default) run the **app.component.html** in that <app-root></app-root> tag.
10. To run child component you have to add tag <component\_selector\_name></component\_selector\_name> in **app.component.html.**

Then automatically replaced this child component.html to

< component\_selector\_name ></ component\_selector\_name >.

1. To change child component dynamically in app.component.html as per routing, add **<router-outlet></router-outlet>.** when change route then automatically change the child component.html in router-outlet tag.
2. Use **<ng-template>** along with the **if else condition.**

**<ng-template>**  instead of <template> from Angular 4 onwards is because there is a name conflict between the <template> tag and the html <template> standard tag. This was one of the major changes made in Angular 4 version.

1. **Directives:**
2. **Component Directives** These form the main class having details of how the component should be processed, instantiated and used at runtime.
3. **Structural Directives** It basically deals with manipulating the dom elements. Structural directives have a \* sign before the directive. For example, \*ngIf and \*ngFor.
4. **Attribute Directives** It deals with changing the look and behavior of the dom element.
5. We can create **Custom Directives** using command**:**

ng g directive nameofthedirective

then nameofthedirective.directive file created automatically. There is selector property, Whatever we define in the selector, **the same has to match in the view**, where we assign the custom directive.

1. **Pipes:** Syntax : **{{ Welcome to Shree Angular 7 | lowercase}}**

**Command to cearte Pipes:**  ng g pipe new-pipe

We can create a custom pipe, for that create a new ts file.

For ex. Sqrtpipe for square root

Here are some **built-in pipes** available with angular:

1. Lowercasepipe
2. Uppercasepipe
3. Datepipe
4. Currencypipe
5. Jsonpipe
6. Percentpipe
7. Decimalpipe
8. Slicepipe
9. **Services :** Some code to be used everywhere on the page. For example, it can be for data connection that needs to be shared across components. This is achieved with the help of Services.

If you change the property of the service in any component, the same is changed in other components too.

**Comamd to Create Service :**  ng g service myservicename

1. **HttpClient:** It will help us fetch external data, post to it, etc. We need to import the http module to make use of the http service. We will fetch the data from the server using httpclient module.

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

1. **We can Create User input Form Using :**
2. **Template-based Forms** with the ngModel, ngForm and ngSubmit Directives -  by including the **FormsModule**

we want Angular to access our data from forms, add ngModel to that tag.

The ngForm directive needs to be added to the form template (tag) that we have created.

1. **Model Driven Forms** with the FormBuilder, FormControl and FormGrop Directives - by including the **ReactiveFormsModule.**

In app.component.ts, we need to import a few modules for the model driven form. For example-

**Ex. import { FormGroup, FormControl } from '@angular/forms'.**

The variable **formdata** is initialized at the start of the class and the same is initialized with **FormGroup**. In that from group we can initialize the **Formcontrol** variables like name,email…etc.

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.

1. **Form Validation :** import { Validators} from '@angular/forms'

**Angular has built-in validators**

1. mandatory field
2. minlength
3. maxlength
4. pattern

In app.component.ts **Validators.compose**, you can add the list of things you want to validate on the input field.

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

For ex. <input type = "submit" [disabled] = "!formdata.valid” name=”submit” value = “Login">

In above ex. Formdata is variable in app.component.ts.

We can create **Custom Validation**, we can define our own custom function and add the required details in it

**23. Materials/CDK -Virtual Scrolling :**

Virtual Scrolling feature is added to CDK (Component Development Kit). Virtual scrolling shows up the **visible dom elements to the user, as the user scrolls, the next list is displayed.** This gives faster experience as the full list is not loaded at one go and only loaded as per the visibility on the screen.

Install the dependency for **virtual scrolling module** using **Command :**

npm install @angular/cdk –save

In app.module.ts file add:

import { ScrollDispatchModule } from '@angular/cdk/scrolling';

We need to add the tag**, <cdk-virtual-scroll-viewport></cdk-virtual-scroll- viewport>** to work with virtual scroll module in .hml file.

We have mostly used \*ngFor so far, but inside **<cdk-virtual-scroll-viewport>,** we have to use **\*cdkVirtualFor** to loop through the data.

1. **Materials/CDK-Drag and Drop**: It helps to drag and drop the elements from the list.

Install the dependency for **Drag and Drop module** using **Command :**

npm install @angular/cdk –save

In app.module.ts file add:

import { DragDropModule } from '@angular/cdk/drag-drop';

we can import { DragDropModule, moveItemInArray, transferArrayItem } from '@angular/cdk/drag-drop'; for additional features.

1. **Animations:** Animations add a lot of interaction between the html elements.

Import the library with the below line of code:

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

@myanimation directive used in .html file for animation use.

1. **Materials :** It offer a lot of built-in modules for your project. Features:
2. **autocomplete**
3. **datepicker**
4. **Slider**
5. **Menus**
6. **Grids**
7. **toolbar** are available for use with materials in Angular 7.

You need to install two packages: **materials and cdk.**

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

npm install @angular/cdk –save

use material tags with **mat-** like **<mat-menu></mat-menu>**

1. **Testing :** There is a .spec.ts file created for **every new** component, service, directive, etc.

For any changes added to your component, services, directives or any other files created, **you can include your test cases in the respective .spec.ts files**. So most of the **unit testing** can be covered at the beginning itself.

**To run the test cases, the command** :

ng test

It shows the erros in console or browser.

1. **Buiding Angular 7 Project** :

Once you are done with the project in Angular, we need to build it so that it can be used in production or stating.

The configuration for build, i.e., **production, staging, development, testing** needs to be defined in your **src/environments.**

In **environment.ts** change production: false to ture when change to production level.

**In app.components.ts file import:**

import { environment } from './../environments/environment';

The environment replacement from default to production which we are trying to do are defined inside angular.json fileReplacements section as follows:

"production": {

"fileReplacements": [

{

"replace": "src/environments/environment.ts",

"with": "src/environments/environment.prod.ts"

} ],

When the command for build runs, the file gets replaced to src/environments/environment.prod.ts. The additional configuration like staging or testing can be added here as shown in the below example:

"configurations": {

"production": { ... },

"staging": {

"fileReplacements": [

{

"replace": "src/environments/environment.ts",

"with": "src/environments/environment.staging.ts"

}

]

}

}

So the command to run the build is as follows:

ng build --configuration=production // for production environmnet

ng build --configuration=staging // for stating enviroment

Now let us run the build command for production, the command will create a dist folder inside our project which will have the final files after build.