ATTUNE

Getting Started with Angular 2

By Juned Laliwala

About this Angular 2 e-Book.

- Basic Understanding of Angular 2.
- Understanding Modules, Components and Templates.
- Understanding the Concepts of TypeScripts.
- Basic Web Pages using Angular 2.

ATTUNE

Contents

1. Preface

1.1 About this Guide

This guide is for the developers or the users who wants to have the basic understanding of the new JavaScripts that are coming into the market. Angular 2 is one of them. Angular 2 is the advanced concept of AngularJs. In this guide, you will come across many aspects of the Angular Application like making modules, templates and components for the particular application. We will be showing many demonstrations along with the desired code so that the reader will get the benefits for this e-Book.

1.2 Intended Audience

This guide is purely intended for the users or the developers who have a keen interest in the scripting side. The developers who have basic understanding of the AngularJs will be finding quite interesting concepts as they have gone through the scripting languages. For the learners who have just entered into the scripting side will not be a troublesome for them as we have explained in this e-book all the basic understanding of Angular 2

1.3 Revision History

This book is in reference with the AngularJs. The users or the developers who have previewed the basic concepts of AngularJs will find a good knowledge of the Angular 2 Application. By looking at the AngularJs concept we have started making this books for the Angular 2.

2. Introduction to Angular 2

2.1 What is Angular 2?

Angular 2, which is the enhanced version of AngularJs. As we all are aware about that AngularJs is one of the best JavaScript framework for building various Web and Mobile Applications. Similarly, with Angular 2 we are going to do the same work by adding some more good functionality to the project, but this additional functionality is very simple to create and as a result we can very perfect output.

2.2 Advantages of using Angular 2

- High-Performance Applications will be delivered with the help of Angular 2.
- We can create many Desktop and Mobile based Applications.
- Speed for developing application increased as compared to other scripting languages.
- Testing, Coding and Animation can also be accessed very effectively.

3. Installation of Angular 2

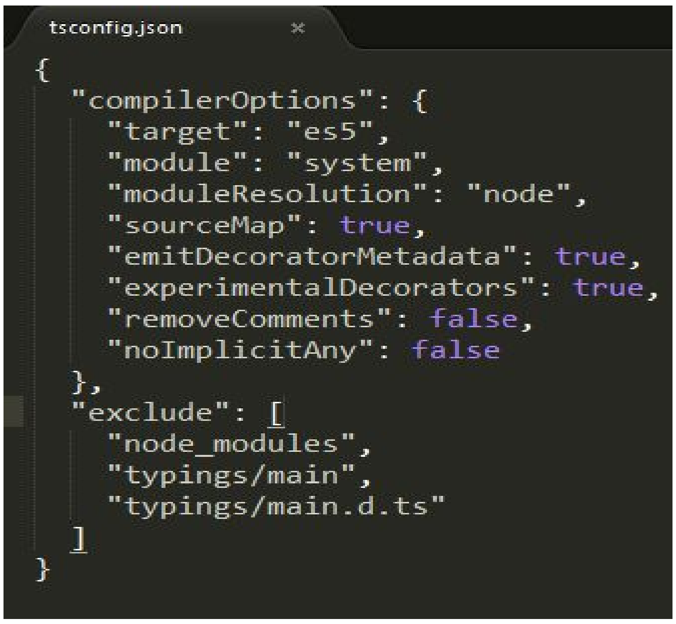
3.1Basic Environment Setup

First of all, we have to understand the basic meaning of the typescript before moving forward to the real-world application. The word "TypeScript" which is the main and the primary language for developing any angularjs Application. Angular 2 is mainly concerned with the word typescript, so we will make the whole application with the help of typescript. Remember one thing that the extension of the typescript files is ".ts".

Create a Project Folder using terminal by giving the following command. "mkdir Attune_angular2 and cd Attune_Angular2" or you can also make a folder by your own choice. This folder should be kept in any of your desired location. Inside the project folder, in order to install Angular 2 in your application some mandatory files need to be included to the project folder.

- **Tsconfig.json**
- **Typings.json**
- **Package.json**

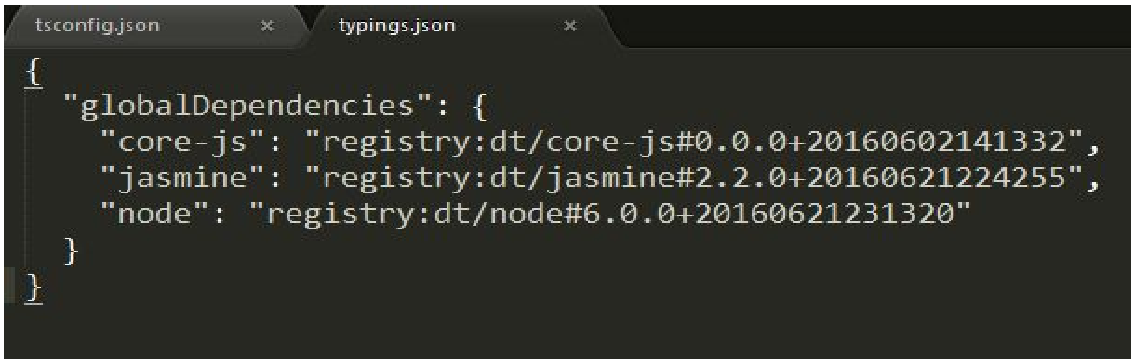
A configuration file is one of the base part of the application which can run on the platform on which the file is being developed. So now as usual we will have to create one configuration file for the typescript for making our application comfortable with typescript.



```
{
  "compilerOptions": {
    "target": "es5",
    "module": "system",
    "moduleResolution": "node",
    "sourceMap": true,
    "emitDecoratorMetadata": true,
    "experimentalDecorators": true,
    "removeComments": false,
    "noImplicitAny": false
  },
  "exclude": [
    "node_modules",
    "typings/main",
    "typings/main.d.ts"
  ]
}
```

Fig: "tsconfig.json"

Similarly, now we will create typings.json file



```json
{
  "globalDependencies": {
    "core-js": "registry:dt/core-js#0.0.0+20160602141332",
    "jasmine": "registry:dt/jasmine#2.2.0+20160621224255",
    "node": "registry:dt/node#6.0.0+20160621231320"
  }
}
```

The definition of the typescripting is usually included into "typings.json" file needed for our angularjs Application.

From the above figure we can see that there are three typings files that have been included so let us discuss about them in more detailed way:

- **"core-js":** Its main use is to bring the ES201516/ES6 Capabilities to our platform dependent browsers.
- **"jasmine":** Its nothing but the typing jasmine framework.
- **"node":** For providing node.js environment to our application.

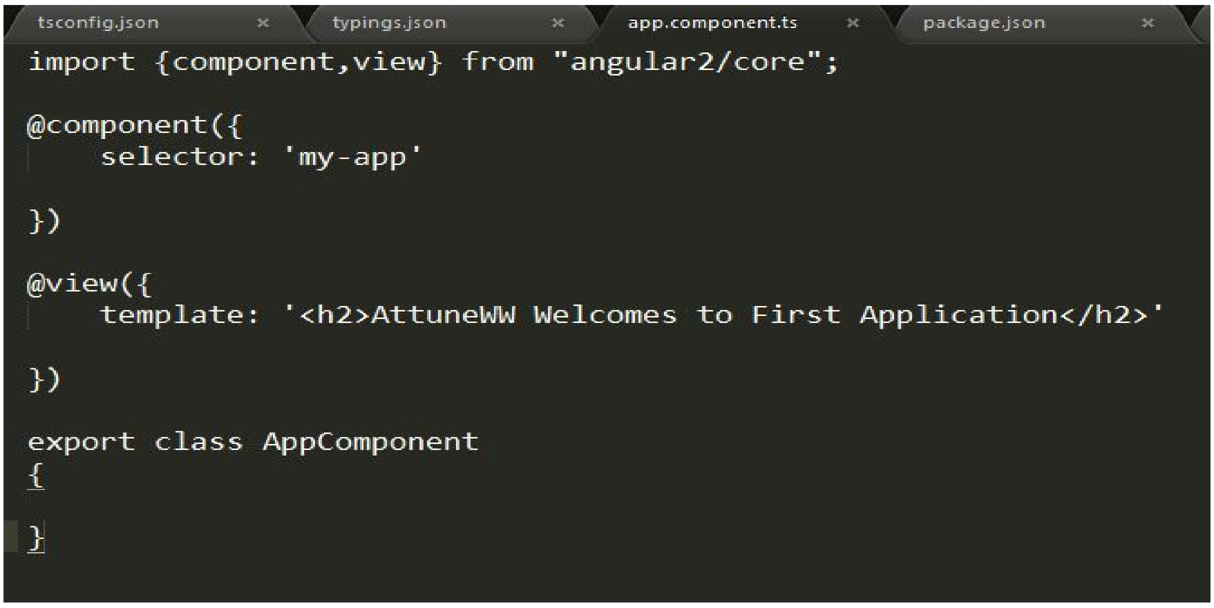
The main Part of the Angular JavaScript total depends upon the installed packages dependencies. So if there are not any single dependency within our application then there are very much large amount of error that can be encountered.

After that we have to give the command "npm install" to install all the dependencies that we have used or encountered.

3.2 Creating First Angular Component

After successfully installing all the dependencies the main part of the Angular 2 starts. Component, which is the core part of Angular is being created by making a new subfolder inside our project root folder. So now all our component related functionalities will be handles inside the newly created subfolder. (In our case "app" subfolder).

The files that have been created for the typescripting will be saved in ".ts" extension. Now let us create a new typescript to make better understanding of the files.



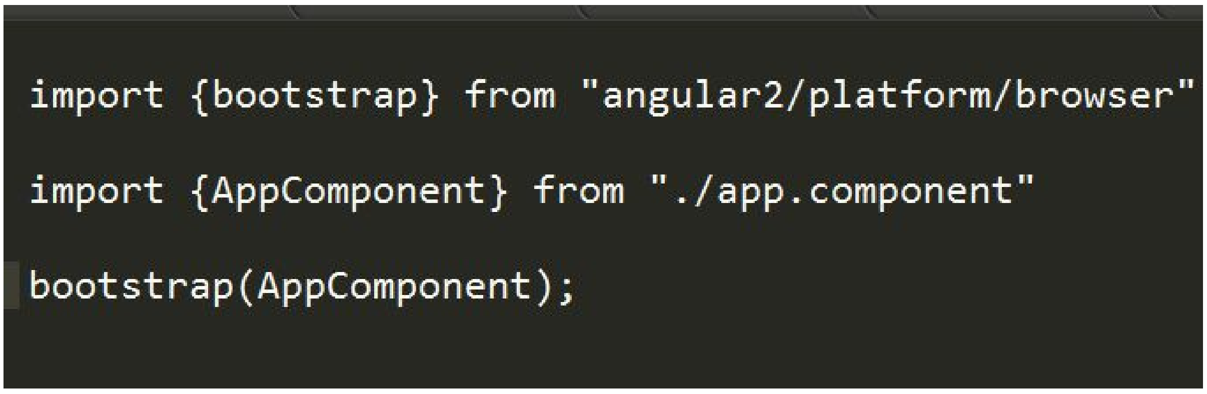
```
import {component,view} from "angular2/core";

@component({
    selector: 'my-app'

})

@view({
    template: '<h2>AttuneWW Welcomes to First Application</h2>'

})

export class AppComponent
{

}
```

Fig: Simple App Component

From the above figure we can see the basic typescripting inside our app folder. The name of the file is "app.component.ts". We will see the better understanding of the below created ts file.

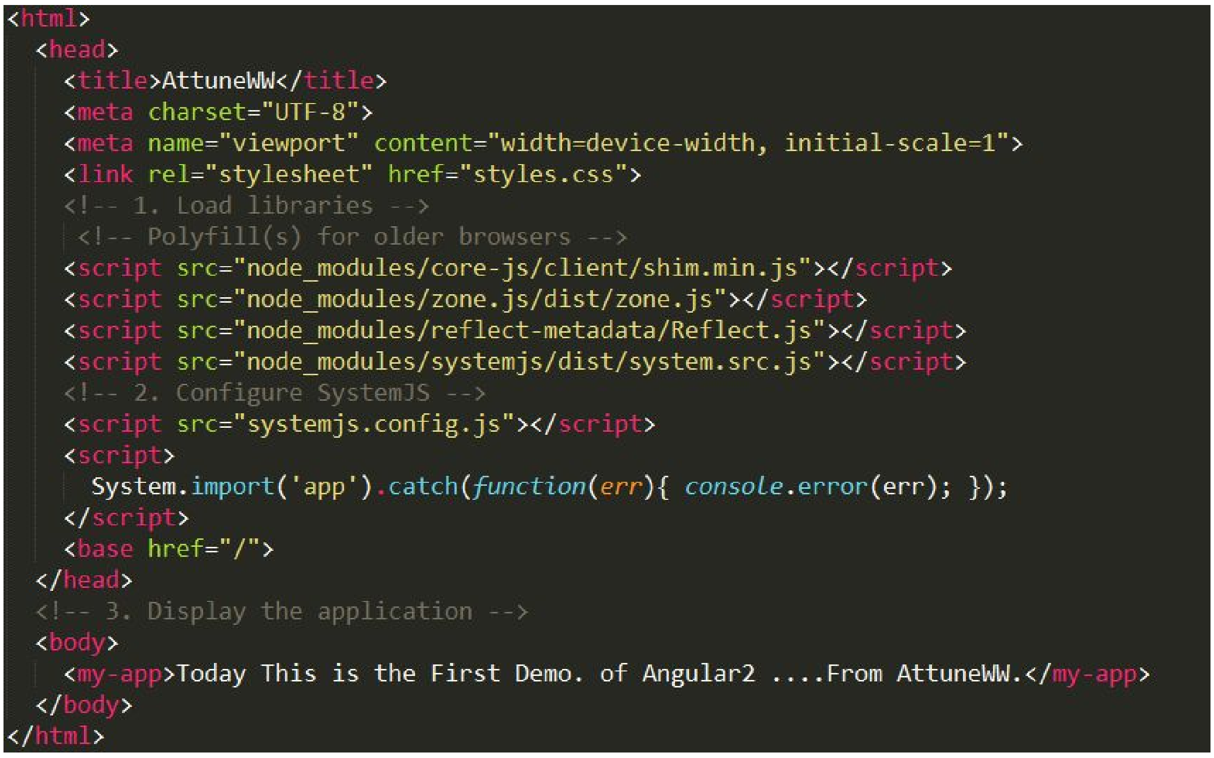
- ➢ Component and View will be imported from the angular2/core
- ➢ @component will allow us to associate metadata with the component class.
- ➢ My-app can be used as an HTML tag.
- ➢ @view will contain the Viewing part of the application.
- ➢ Export is the part which is solely responsible for the components that are outside the files.

Now we will move next step further to create a new file named "main.ts"



```
import {bootstrap} from "angular2/platform/browser"

import {AppComponent} from "./app.component"

bootstrap(AppComponent);
```

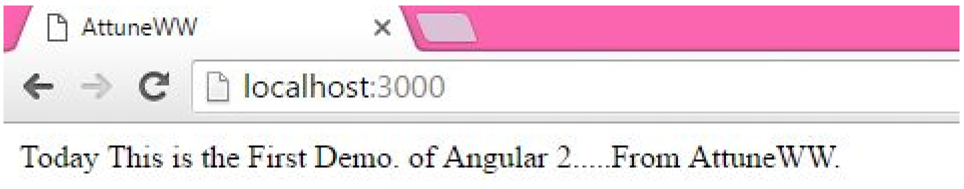
Now the Index.html file will be:



```html
<html>
  <head>
    <title>AttuneWW</title>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <link rel="stylesheet" href="styles.css">
    <!-- 1. Load libraries -->
     <!-- Polyfill(s) for older browsers -->
    <script src="node_modules/core-js/client/shim.min.js"></script>
    <script src="node_modules/zone.js/dist/zone.js"></script>
    <script src="node_modules/reflect-metadata/Reflect.js"></script>
    <script src="node_modules/systemjs/dist/system.src.js"></script>
    <!-- 2. Configure SystemJS -->
    <script src="systemjs.config.js"></script>
    <script>
      System.import('app').catch(function(err){ console.error(err); });
    </script>
    <base href="/">
  </head>
  <!-- 3. Display the application -->
  <body>
    <my-app>Today This is the First Demo. of Angular2 ....From AttuneWW.</my-app>
  </body>
</html>
```

Fig: Index.html

Remember that currently we are displaying the simple web page so this is just the basic html file. Inside this html file we are importing the “app”.



```
AttuneWW
localhost:3000
Today This is the First Demo. of Angular 2.....From AttuneWW.
```

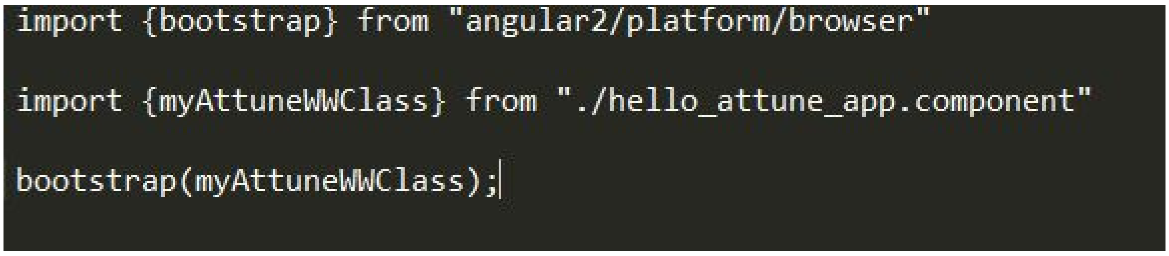
Fig: Output of a Simple Demo

From the above figure we can see that we have imported the above statement inside the <my-app> tag which is nothing but the selector field in the Component part. So whatever content that has been displayed onto my-app tag will be displayed onto the screen.

Advanced concept of Components will be covered in the coming sections. So this was just the overview about the installation and simple demonstration has been included in this section about how to start with the Angular 2 and its corresponding environment. Now we will come to the practical session in which we will implement some task so that we can easily understand the advanced concept.

4. Creating First Simple Example

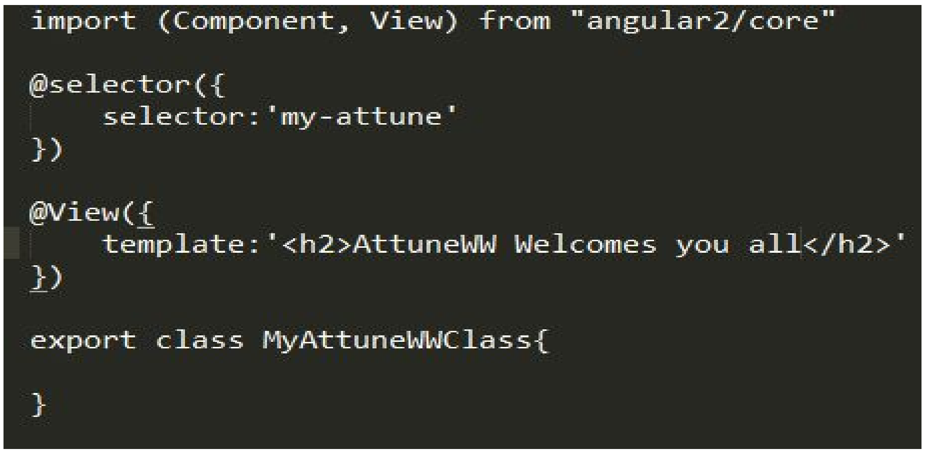
Now we are going to demonstrate you a simple and straight forward example which will clear all your understanding about the Angular 2 Application. Simply we will have to create two important file one file which will handle the view part and another will be the component file for the created part. The former file will be the main file whereas the second will be the component.ts file which is illustrated in the above diagram.



```
import {bootstrap} from "angular2/platform/browser"

import {myAttuneWWClass} from "./hello_attune_app.component"

bootstrap(myAttuneWWClass);
```

Fig: hello_attune_main.ts

This file is the file which will handle the class of the component file. The bootstrap has been used to import the view part of the currently exported file that we will show in the next diagram.



```
import (Component, View) from "angular2/core"

@selector({
    selector:'my-attune'
})

@View({
    template:'<h2>AttuneWW Welcomes you all</h2>'
})

export class MyAttuneWWClass{

}
```

Fig: hello_attune_app. component.ts

The above diagram is the component file of the hello_attune file. In the first line of this file we can see that we are importing Component and View from the core of the angular application. As we know that selector is nothing but the html tag which we are preferring to be used by the component. So in our case the selector “my-attune” will be handling the template which is being included into View part of the file. At last we are exporting the class, this exported class is nothing but it will be used by the main file that we have discussed in the precious article.

So this was just the simple way to start with our Angular 2 Application.

5. Basic Architecture of Angular 2

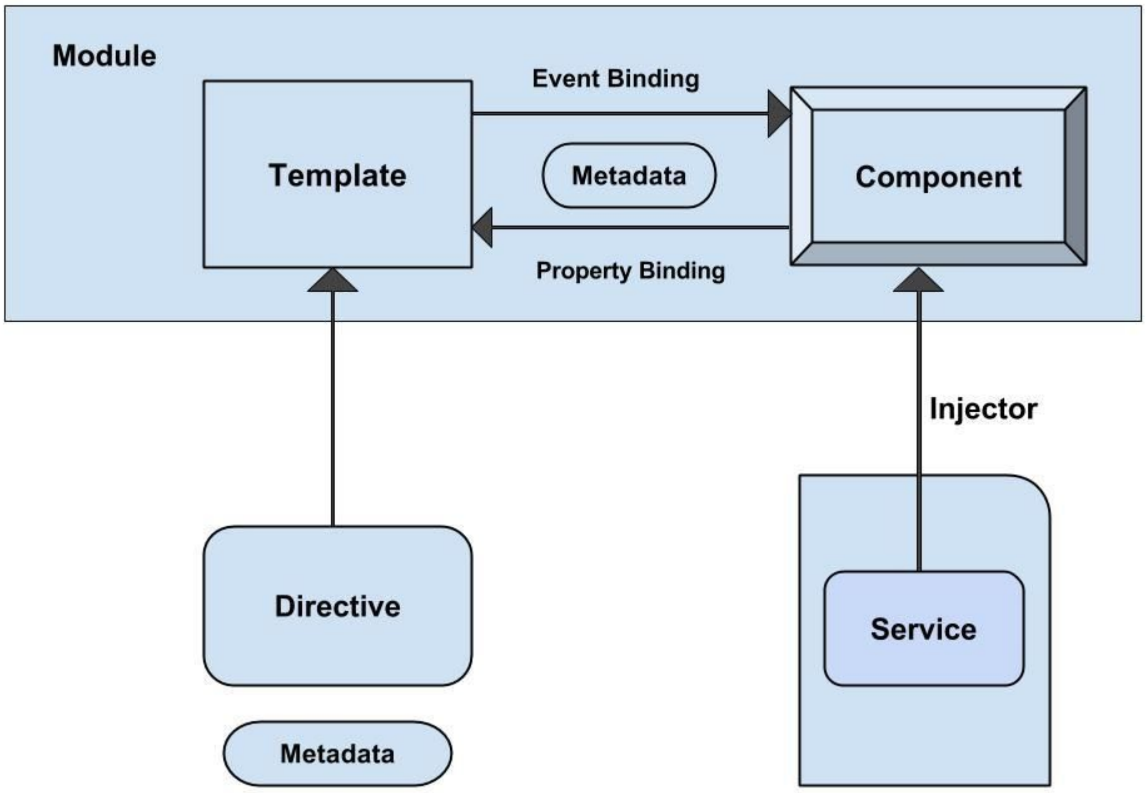


Fig: Angular 2 Architecture

From the above diagram we can see that the overall architecture is being divided into mainly 8 different parts.

1. Module
2. Component
3. Template
4. Metadata
5. Data Binding
6. Service
7. Directives
8. Dependency Injection

Module as also reviewed in Angularjs that the main role of the module to perform the single task that we usually use during the block of code. Apart from the Angularjs we have an extra functionality of exporting the class using export method in Angular2. Applications which are being made in Angular2 are basically Modules and as such by using as much as modules we can create Angular2 App.

In our Angular2 Application we will find "*AppComponent*" in many cases which is nothing but the component class and will be contained inside *app.component.ts* file.

"***export class AppComponent { }"*** is the statement which is used to export the components that we have created in the modules. The meaning of the export in the statement is that it is specifying the module and this AppComponent is public or we can say viewable for all the other modules in the application.

Component is mainly associated with the view of the application and the logics that we are applying to our application. The main role of component class in the overall application is to check the dependency injection and based on that it can render its content itself. @Component is the statement to identify that it is a component and its main use to break the application into separate different parts to identify a particular task. One thing that everyone should remember that we can use only one DOM element per Component Class.

Template part of the Angular 2 is nothing but the view that the user wants to see is displayed in this section. We just have to import the basic HTML syntax that we want to view the templates.

Metadata is in simple words means that data of data. In our Angular Concept we can say that the class that the overall processing of the function is being handled by the metadata. Suppose we have made one simple component and this component will remain the class only until we clearly define this class as a component. We are referring to the term named "Decorator" to attach the typescript to a Metadata and this is denoted by @Component.

```
@Component ({
        selector: 'myemployees',
        template: '<h2>Employee Details</h2>',
        directives: '[myEmployees ]'
})
export class EmployeeComponent{ .. }
```

@Component is a directive which usually contains three basic parameters. From the above example we can see that we have created one selector named "myemployees". Selector will basically create the instance of the component wherever it finds <myemployees> tag in the parent HTML. The template as we know it will simple display the templates or the view of the page. Directives is used to represent the array of the components or the directives that we have specified in the application.

Data Binding is the process by which we can collaborate with the application values by declaring the bindings between source and target html elements. Template part is combined with the component part using Data Binding. There are usually four types of Data Binding: Interpolation, Property Binding, Event Binding and Two-way Binding.

Services are nothing but the functions that we have implemented in our applications can be easily identified using the services. Its main purpose is to identify or perform some specific tasks only. Various services are available for logging, registration, user service, data service and so on. Services are the fundamental core to Angular Applications.

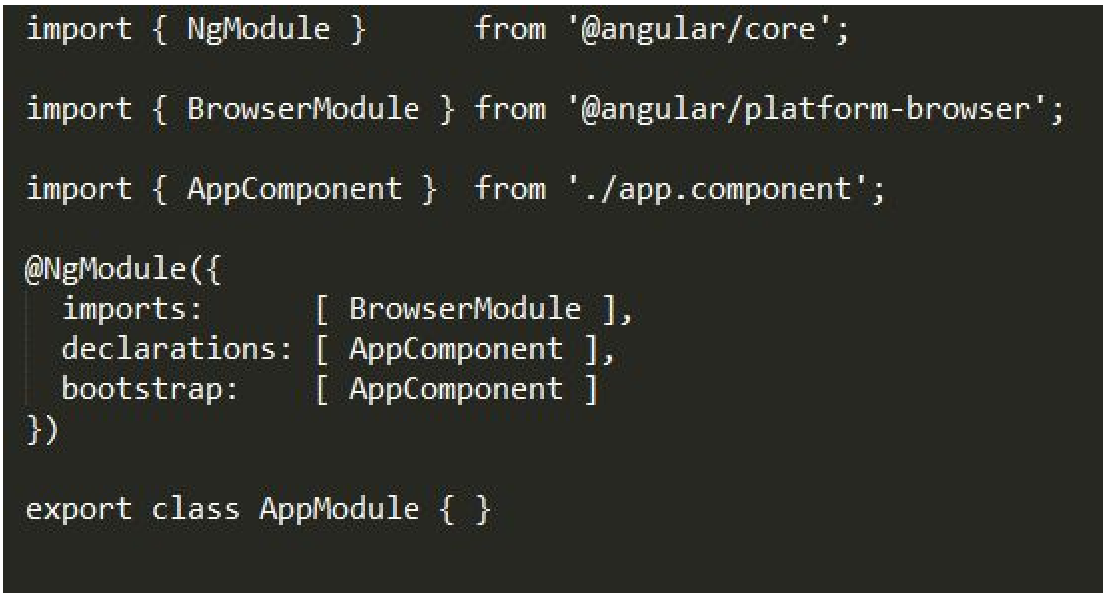
Directive is solely responsible for Metadata representation in our Angular Applications. Component Directives, Decorator Directives and Template Directives are the three types of Directives in Angular.

Dependency Injection is nothing but the Object Passing as dependencies in different components of the application. New creation of the class along with its dependencies will be created using Dependency Injection.

6. Modules, Components and Templates

6.1 Creating A Module

A module can be created as many as module in a single application. In the typical sense we can say that a module is a group of code which is integrated with other modules for running the particular applications.



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

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

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

@NgModule({
  imports:      [ BrowserModule ],
  declarations: [ AppComponent ],
  bootstrap:    [ AppComponent ]
})

export class AppModule { }
```

Fig: Creating a Simple Module

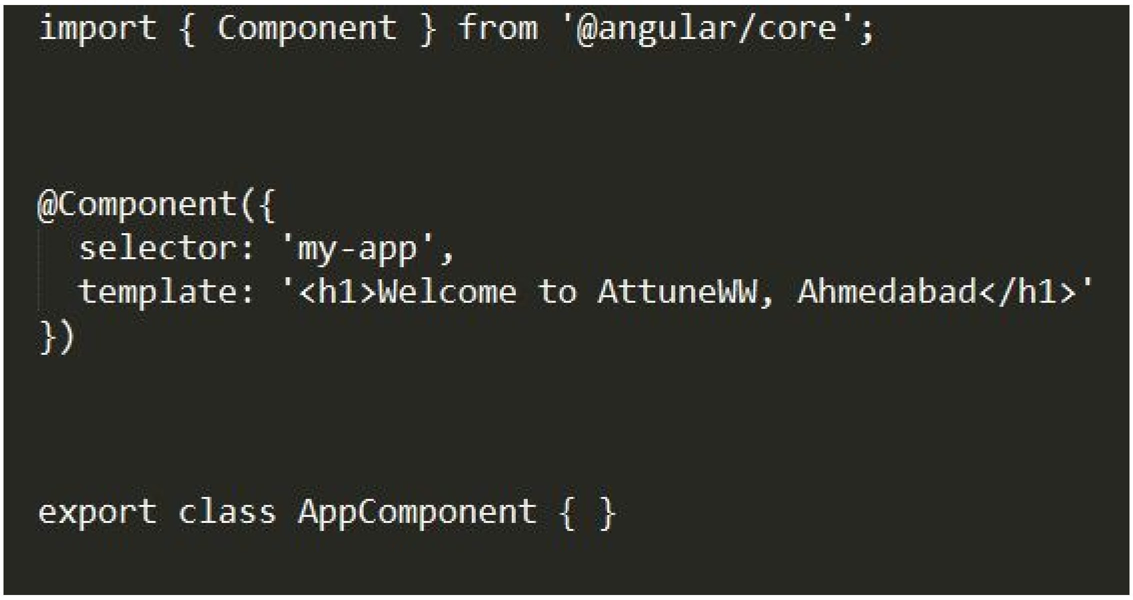
From the above figure we can see that we are importing Angular Module “NgModule” from core elements of Angular. Apart from it we are also importing Browser Module which is solely responsible for handling Browser Specific tasks.

@NgModule is the function through which we can import, declare and identify the bootstrap which are being used for the particular project.

6.2 Creating a Component and Templates

Component is the main part of the Angular 2 applications. From the figure given below, we can see that @Component statement is used to invoke a Component part. Inside this we have included two main parts (Just for this Example) “Selector” and “template”. Selector part as we have also discussed in the previous section that it will handle the HTML syntax of <mya-pp> part. In templates part, we will have to define whatever HTML code we want to render to display the Viewing of the web pages.

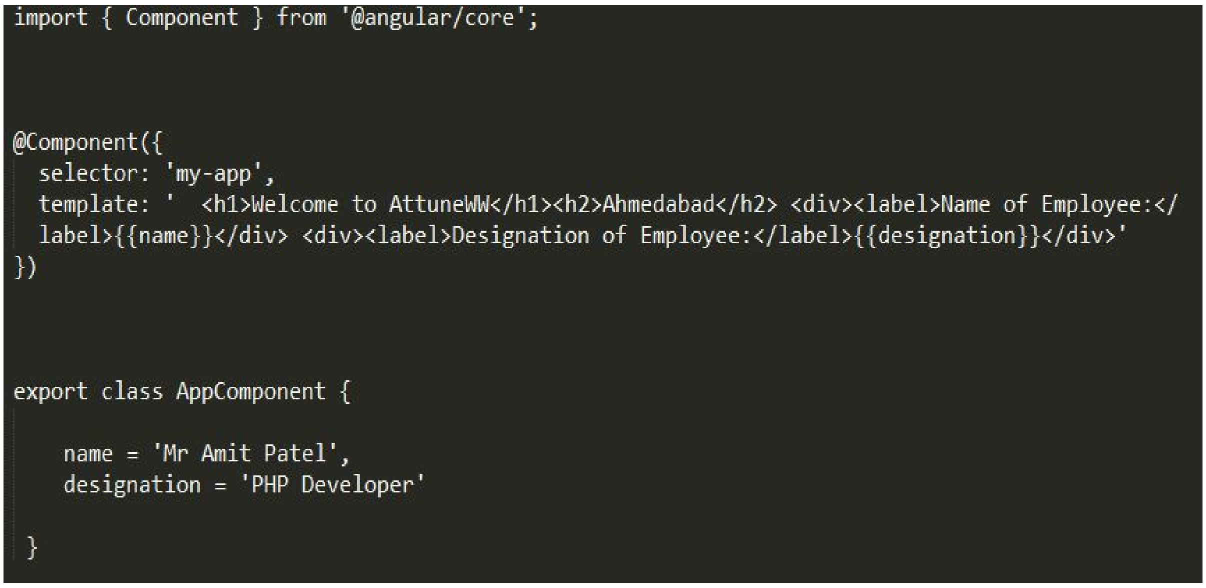
In figure given below, we can see that we are passing simple text in the template section.



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


@Component({
  selector: 'my-app',
  template: '<h1>Welcome to AttuneWW, Ahmedabad</h1>'
})


export class AppComponent { }
```

Fig: Creation of Component and Template

As we can see in the last line of the figure, “export class AppComponent”, the main role is to add some more parameters into it and to call this parameter to use in our angular 2 Application. From the figure shown below, we can see that we are passing two parameters namely “name” and “designation” to display on our output.

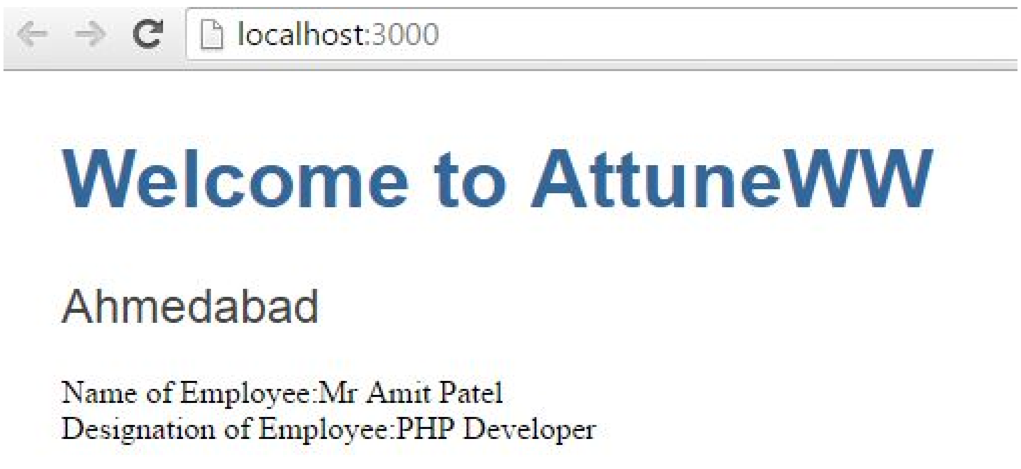


```
import { Component } from '@angular/core';


@Component({
  selector: 'my-app',
  template: '  <h1>Welcome to AttuneWW</h1><h2>Ahmedabad</h2> <div><label>Name of Employee:</
  label>{{name}}</div> <div><label>Designation of Employee:</label>{{designation}}</div>'
})


export class AppComponent {

    name = 'Mr Amit Patel',
    designation = 'PHP Developer'

 }
```

Fig: Component with Parameters

From the above figure we can see that we have passed two values into our name and designation parameters and in @Component function we are defining those values using angular directives. So the desired output is:

ATTUNE


localhost:3000

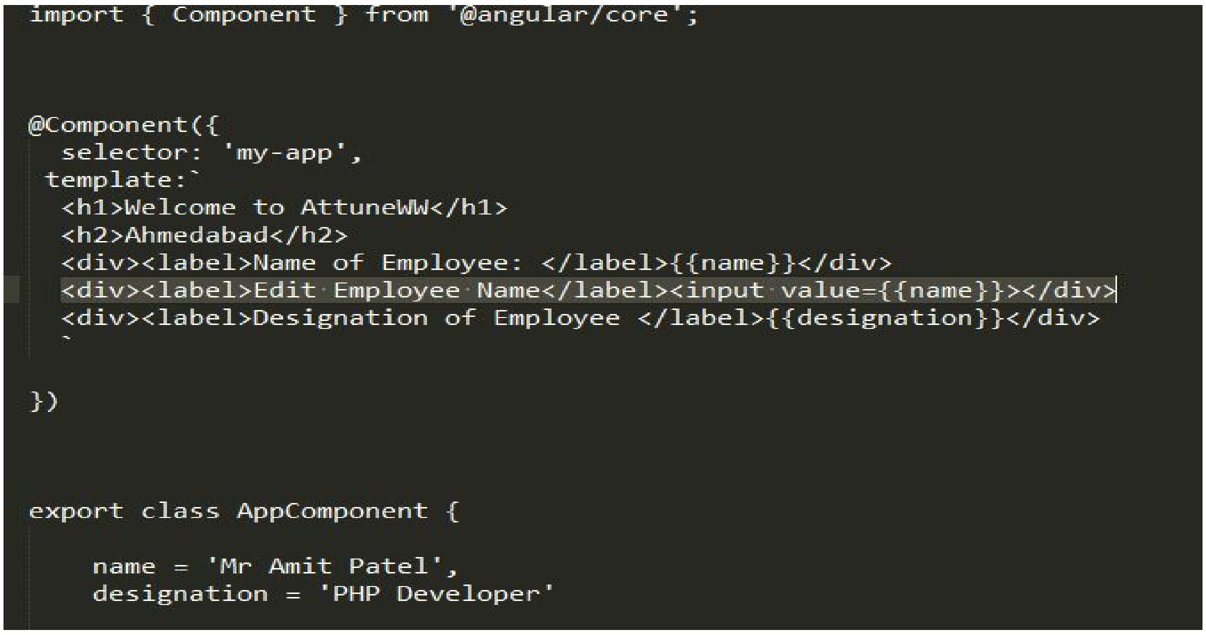
Welcome to AttuneWW

Ahmedabad

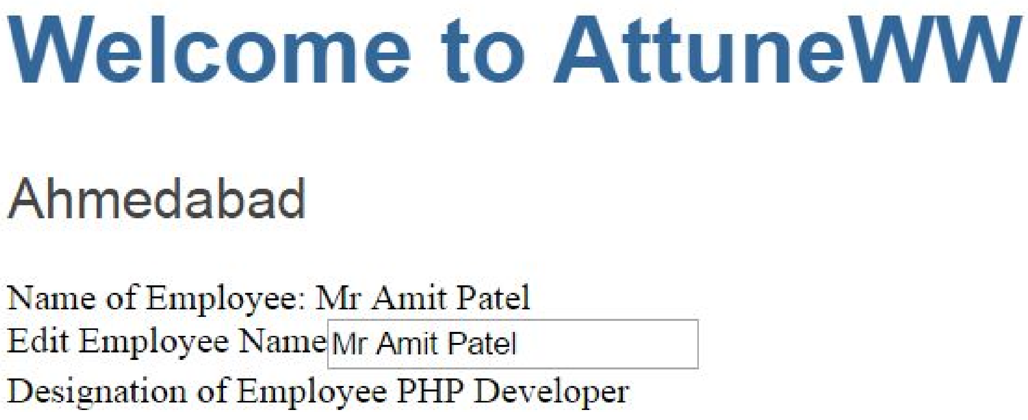
Name of Employee:Mr Amit Patel
Designation of Employee:PHP Developer

Fig: Output of Simple Component & Template.

As far as we have seen the basic implementation of the Angular 2. Now we will focus on creating textbox for inputting the values



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


@Component({
  selector: 'my-app',
 template:`
  <h1>Welcome to AttuneWW</h1>
  <h2>Ahmedabad</h2>
  <div><label>Name of Employee: </label>{{name}}</div>
  <div><label>Edit Employee Name</label><input value={{name}}></div>
  <div><label>Designation of Employee </label>{{designation}}</div>
  `

})


export class AppComponent {

    name = 'Mr Amit Patel',
    designation = 'PHP Developer'
```


Welcome to AttuneWW

Ahmedabad

Name of Employee: Mr Amit Patel
Edit Employee Name Mr Amit Patel
Designation of Employee PHP Developer

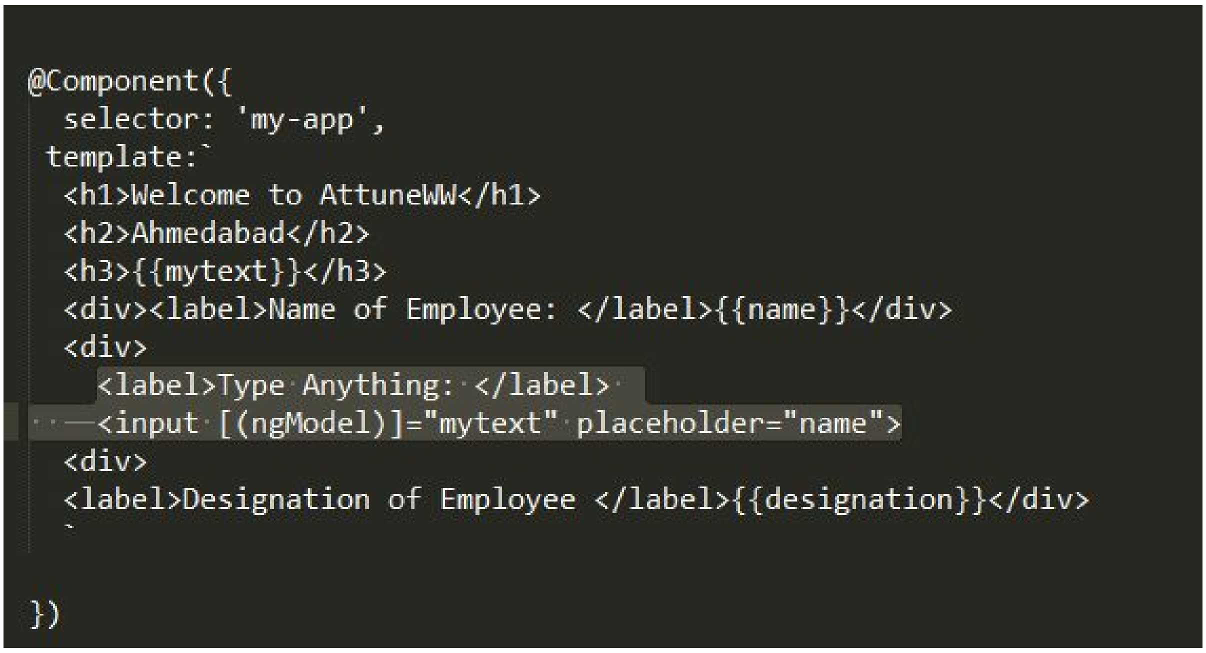
Fig: Textbox in Application

This was the simple example to illustrate the use of textbox.

6.3 Two-way Data Binding

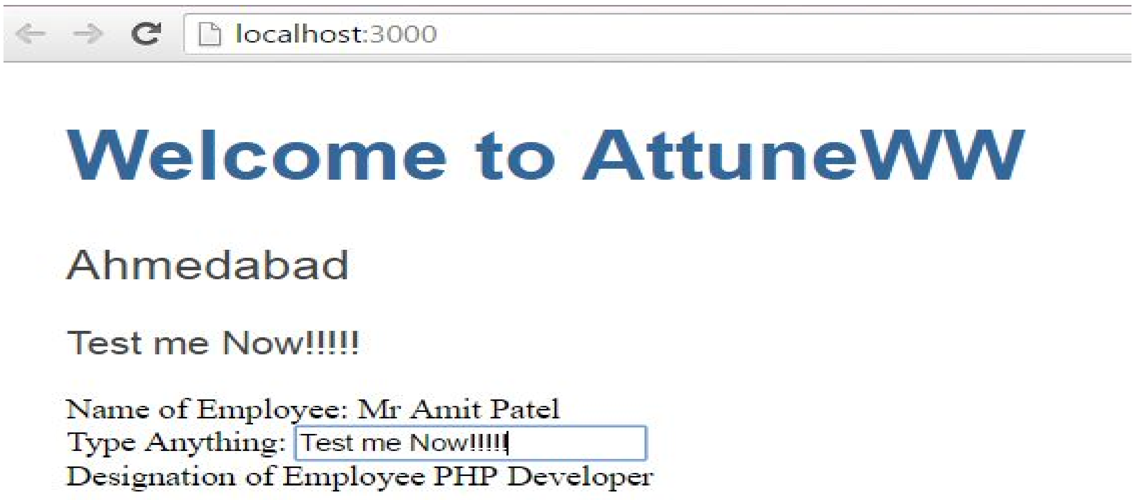
As we all are aware about the data binding that we have being doing in Angularjs by putting ng-model in input tag. The same case is here; we are putting the same ng-model in a different syntax as [(ng-model)]. So let us see the basic functionality of the Two-way Data Binding.

For Two-way Data Binding comes into action, we have to keep “FormsModule” in our module typescript configuration file.



```
@Component({
  selector: 'my-app',
 template:`
  <h1>Welcome to AttuneWW</h1>
  <h2>Ahmedabad</h2>
  <h3>{{mytext}}</h3>
  <div><label>Name of Employee: </label>{{name}}</div>
  <div>
    <label>Type Anything: </label>
    <input [(ngModel)]="mytext" placeholder="name">
  <div>
  <label>Designation of Employee </label>{{designation}}</div>
  `

})
```

Fig: Component with ng-model



```
localhost:3000

Welcome to AttuneWW

Ahmedabad

Test me Now!!!!!

Name of Employee: Mr Amit Patel
Type Anything: Test me Now!!!!!
Designation of Employee PHP Developer
```

Fig: Output with ng-model

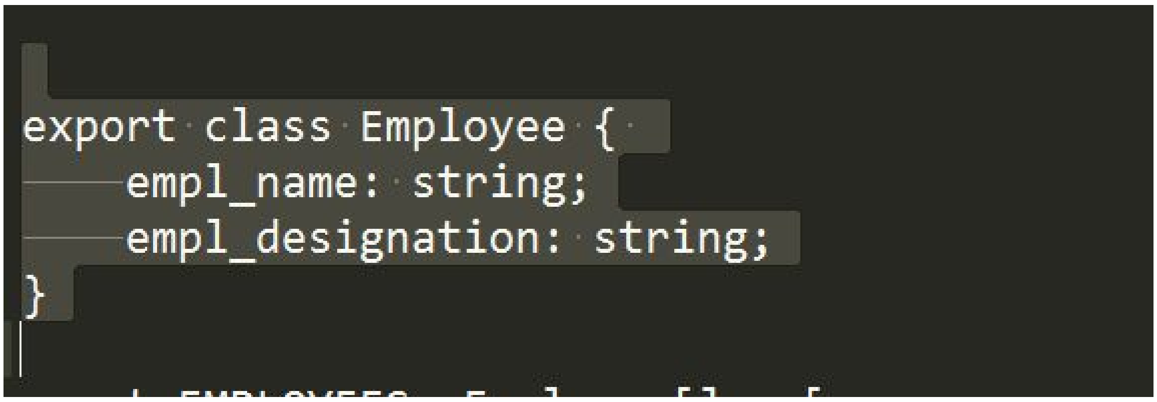
Till now we have seen the entry of only single element. Now we will pass the Employee list as an array. So with the help of array we will statically insert the Name and Designation of Employees.

Steps for Creating Array:

1. First of all, export the class of Employee.

2. Add the static values to the Name and Designation of the Employees.
3. List the Elements.

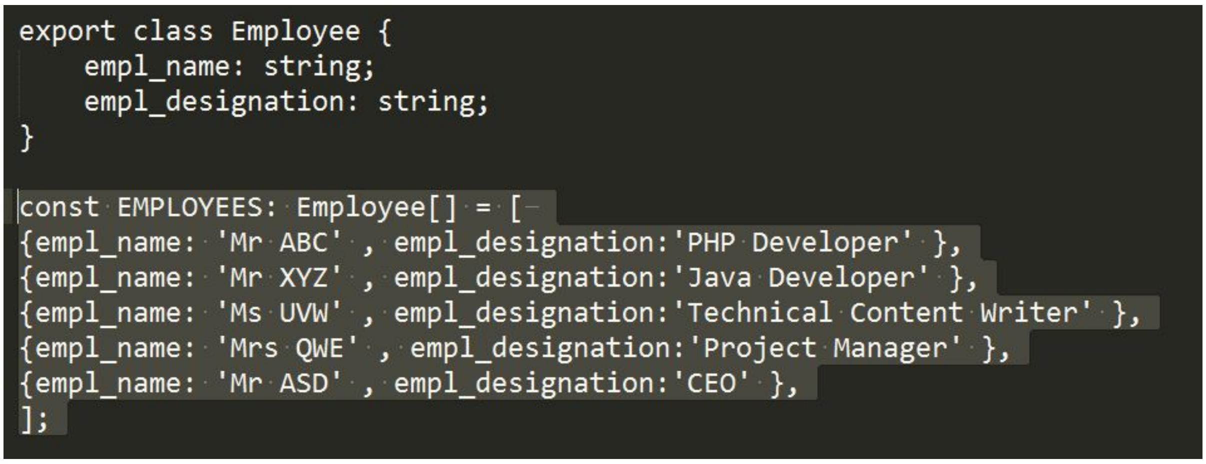
This three steps will be now shown with the demonstration.



```
export class Employee {
    empl_name: string;
    empl_designation: string;
}
```

Fig: Defining Name and Designation by Exporting Class

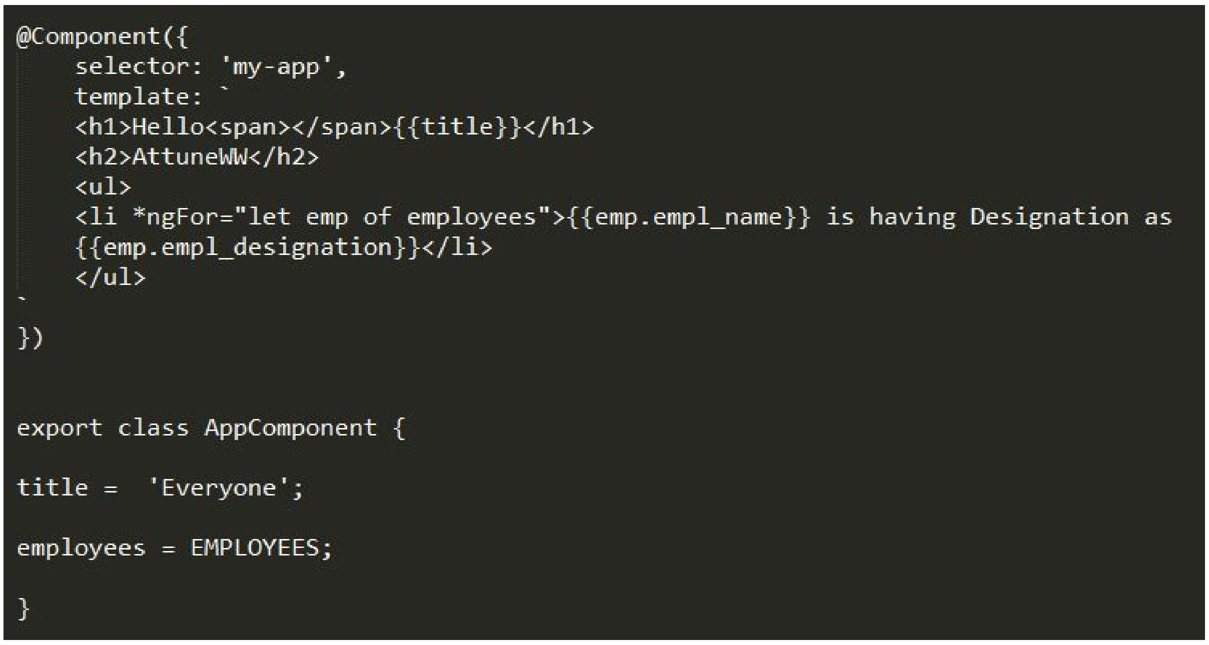
From the above figure we can see that we have included “empl_name” and “empl_designation” to denote the name and designation of the employees. This Class usually means that inside the Employee there are two main parameters namely their name and designation. Now we will insert the static values into the Employee Class.



```
export class Employee {
    empl_name: string;
    empl_designation: string;
}

const EMPLOYEES: Employee[] = [
{empl_name: 'Mr ABC' , empl_designation:'PHP Developer' },
{empl_name: 'Mr XYZ' , empl_designation:'Java Developer' },
{empl_name: 'Ms UVW' , empl_designation:'Technical Content Writer' },
{empl_name: 'Mrs QWE' , empl_designation:'Project Manager' },
{empl_name: 'Mr ASD' , empl_designation:'CEO' },
];
```

Fig: Static Values passed into Array

From the above figure we can say that we have inserted total five Employees list. The most important part in listing in angular 2 is to define a constant value for the Employee List. In the first line of the Array list we have defined one statement “const EMPLOYEES: Employee[ ]” which means that we are using the constant word “EMPLOYEES” for the currently exported class named “Employee”. So all the values of the name and designation part will now be handled by EMPLOYEES.



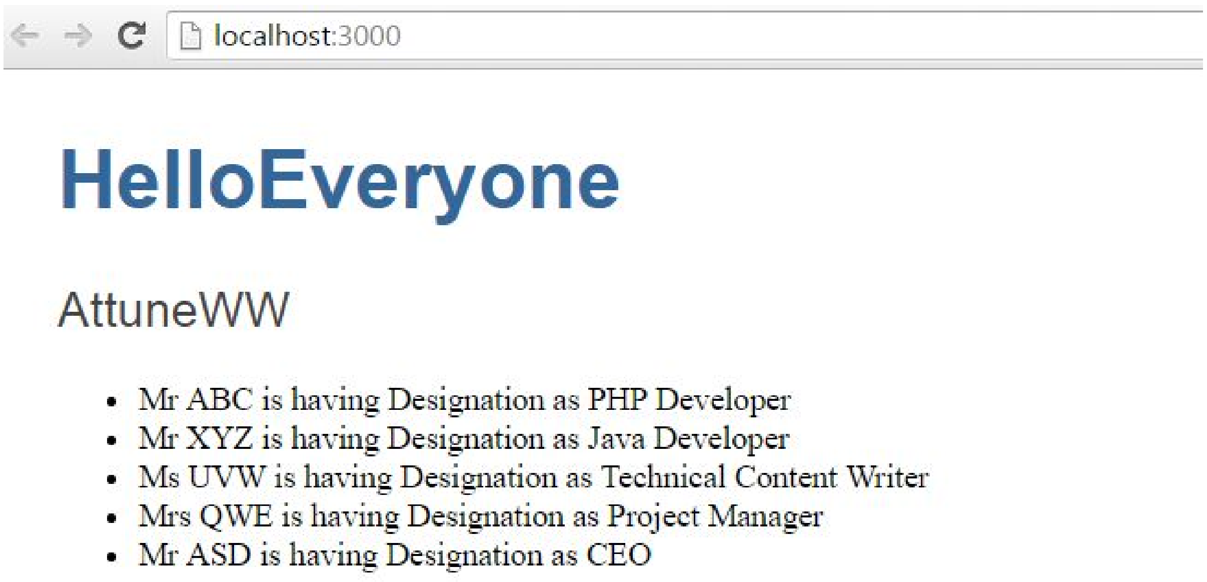
```
@Component({
    selector: 'my-app',
    template: `
    <h1>Hello<span></span>{{title}}</h1>
    <h2>AttuneWW</h2>
    <ul>
    <li *ngFor="let emp of employees">{{emp.empl_name}} is having Designation as
    {{emp.empl_designation}}</li>
    </ul>
`
})


export class AppComponent {

title =  'Everyone';

employees = EMPLOYEES;

}
```

Fig: Component and AppComponent Exported class

Now our main task is to display the Name of all the Employees along with the Designation. So in @Component part we can see that in the list part we are using *ngFor for denoting the acceptable parameter values that are eligible for the list. As we can see in the AppComponent Class, we are referring our Array list as “employees”, so all the static values of the list will be now handled by “employees” parameter.

Employees = Employees is the main statement that is mandatory to be applied in order to pass the values into the list.

So the desired output will be:



localhost:3000

HelloEveryone

AttuneWW

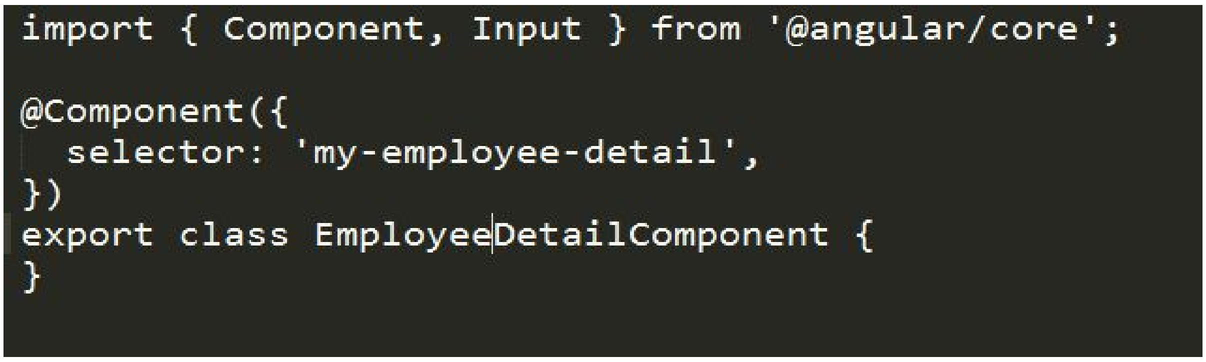
- Mr ABC is having Designation as PHP Developer
- Mr XYZ is having Designation as Java Developer
- Ms UVW is having Designation as Technical Content Writer
- Mrs QWE is having Designation as Project Manager
- Mr ASD is having Designation as CEO

From the above figure, we can see that an automatic generated list has been developed for the static values that has been passed into the list.

7. Concept of Multiple Component

So far we have seen the basic use of only single component in our application. Now we will move one step further and introduce the concept of adding more than one component in a single file. In the previous section we have seen that we have made the list of Employee Name and their respective designation. Now we will make one separate component for the Employee Details which will handle the overall details of the employees.

For this to take into effect, we will create a new file inside our app folder and we will name it as "employee-detail.component.ts".



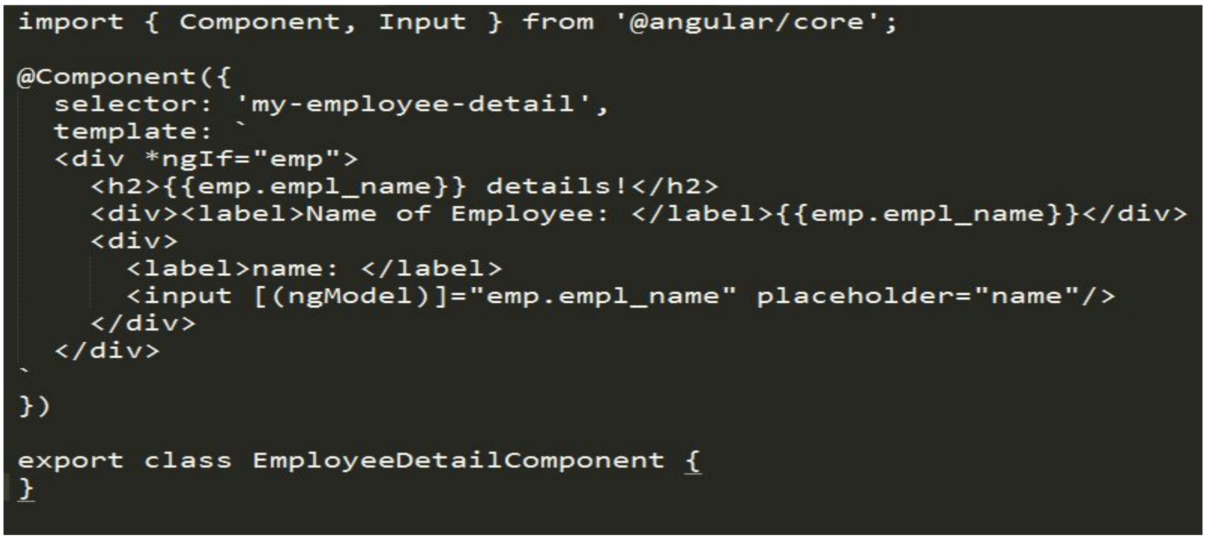
```
import { Component, Input } from '@angular/core';

@Component({
  selector: 'my-employee-detail',
})
export class EmployeeDetailComponent {
}
```

Fig: Employee-Detail Component TypeScript

So now we are ready with a separate typescript file which includes selector as my-employee-detail. The main issue that we should remember is to identify the component and the files which contains that Components. In the above figure we can see that name of the file is "employee-detail.component.ts" and the name of the class is "EmployeeDetailComponent". So this confusion should now be clear in the overall aspects of Angular 2 Applications.

Component names will always end with ***"Component"*** whereas filename will always end with ***".component".***

So, now all the Employee Details will be stored in "employee-detail.component.ts" file.

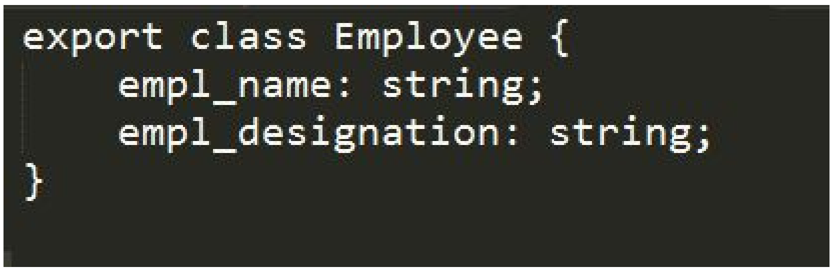


```
import { Component, Input } from '@angular/core';

@Component({
  selector: 'my-employee-detail',
  template: `
  <div *ngIf="emp">
    <h2>{{emp.empl_name}} details!</h2>
    <div><label>Name of Employee: </label>{{emp.empl_name}}</div>
    <div>
      <label>name: </label>
      <input [(ngModel)]="emp.empl_name" placeholder="name"/>
    </div>
  </div>
`
})

export class EmployeeDetailComponent {
}
```

Fig: employee-detail.component.ts

From the figure shown above we can see that all the values that were defined inside the AppComponent will now be clearly handled by the separate typescript file for Employee Details. Now the conflict occurs as we have declared the Employee in app.component file but we are preferring to this newly created typescript file. No issue, we will now create a new “employee.ts” file

We have exported Employee Class in both the files, so we will have to import the “employee.ts” file in both the component files namely “app.component.ts” and “employee-detail.component.ts” by giving the statement as “**import { Employee } from ‘./employees’**”.

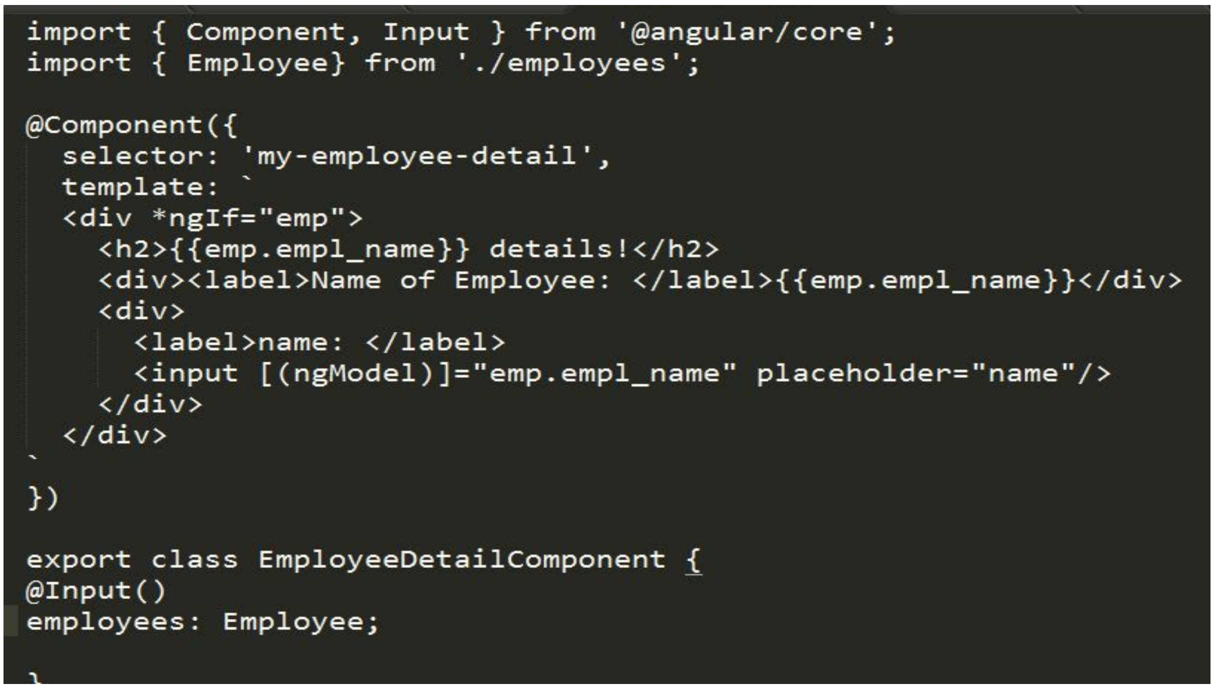


```
export class Employee {
    empl_name: string;
    empl_designation: string;
}
```

Fig: employee.ts

So in the above figure we have simply made a new typescript file for the employees.

Similarly, after modifications that we have discussed, we have imported a newly created file of Employee. Apart from it we have also exported the EmployeeDetailComponent which is handling the user inputs from the Employees.



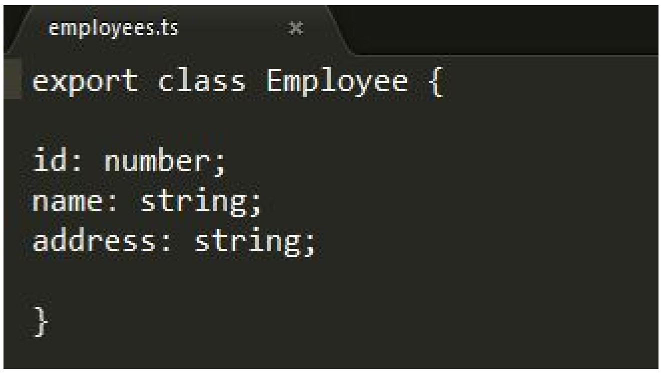
```
import { Component, Input } from '@angular/core';
import { Employee} from './employees';

@Component({
  selector: 'my-employee-detail',
  template: `
  <div *ngIf="emp">
    <h2>{{emp.empl_name}} details!</h2>
    <div><label>Name of Employee: </label>{{emp.empl_name}}</div>
    <div>
      <label>name: </label>
      <input [(ngModel)]="emp.empl_name" placeholder="name"/>
    </div>
  </div>
`
})

export class EmployeeDetailComponent {
@Input()
employees: Employee;

}
```

Fig: Component File of Employee

8. Creating a Service

The main role of the service in the Angular 2 is that a basic functions or the parameter will be called in one simple file and the values that have been defined into this service will be shared by all the files that want to function with that particular file.

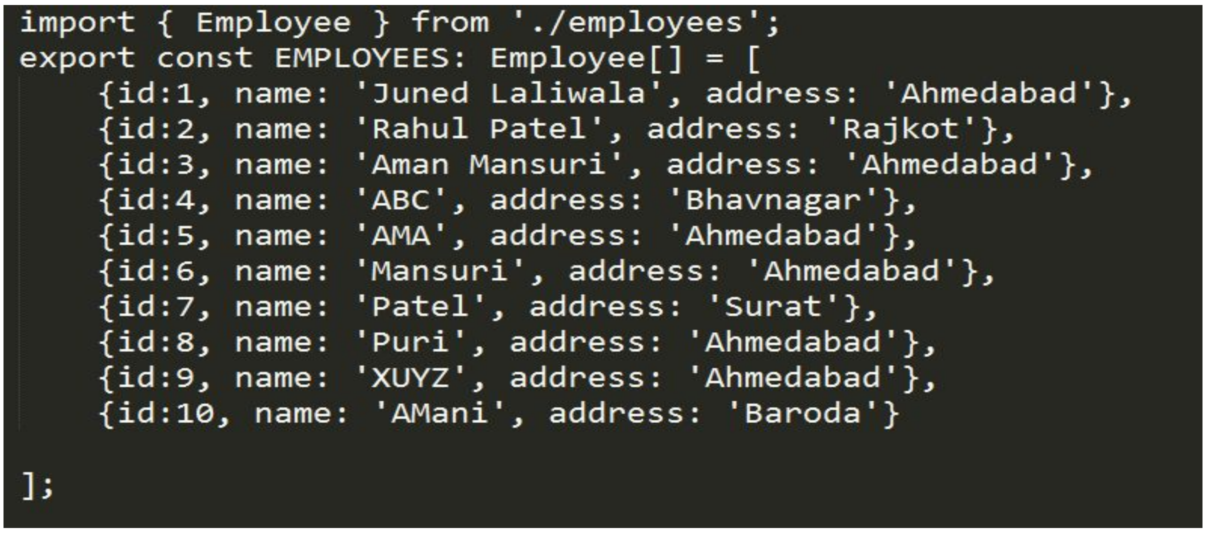
Step 1: Create a typescript file in which we will have to define the parameters that are being used for the data.



```
export class Employee {

id: number;
name: string;
address: string;

}
```

So in the above diagram we can see that we have made one Employee class which is containing ID, Name and Address of the particular employee.

Now we will give some static values to the parameters that we have defined for it.



```
import { Employee } from './employees';
export const EMPLOYEES: Employee[] = [
    {id:1, name: 'Juned Laliwala', address: 'Ahmedabad'},
    {id:2, name: 'Rahul Patel', address: 'Rajkot'},
    {id:3, name: 'Aman Mansuri', address: 'Ahmedabad'},
    {id:4, name: 'ABC', address: 'Bhavnagar'},
    {id:5, name: 'AMA', address: 'Ahmedabad'},
    {id:6, name: 'Mansuri', address: 'Ahmedabad'},
    {id:7, name: 'Patel', address: 'Surat'},
    {id:8, name: 'Puri', address: 'Ahmedabad'},
    {id:9, name: 'XUYZ', address: 'Ahmedabad'},
    {id:10, name: 'AMani', address: 'Baroda'}

];
```

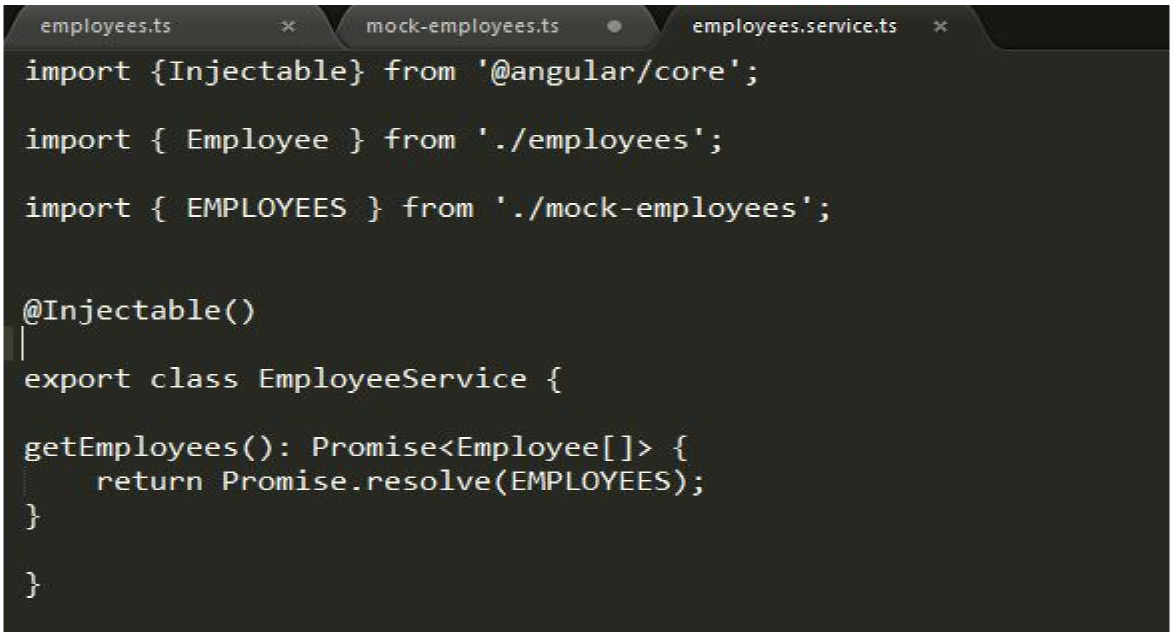
Fig: Adding Static Values to Employees Field

As we can see from the above diagram that we have given some static values to it.

Now we will define a service with the help of the above two files.

In our case, we have created Employee list to show their names and their designation. Now to show proper demonstration of the service we will make one service on Employees and will include more parameters into it. These parameters will be viewed or shared by all other components that wants to use Employee class.

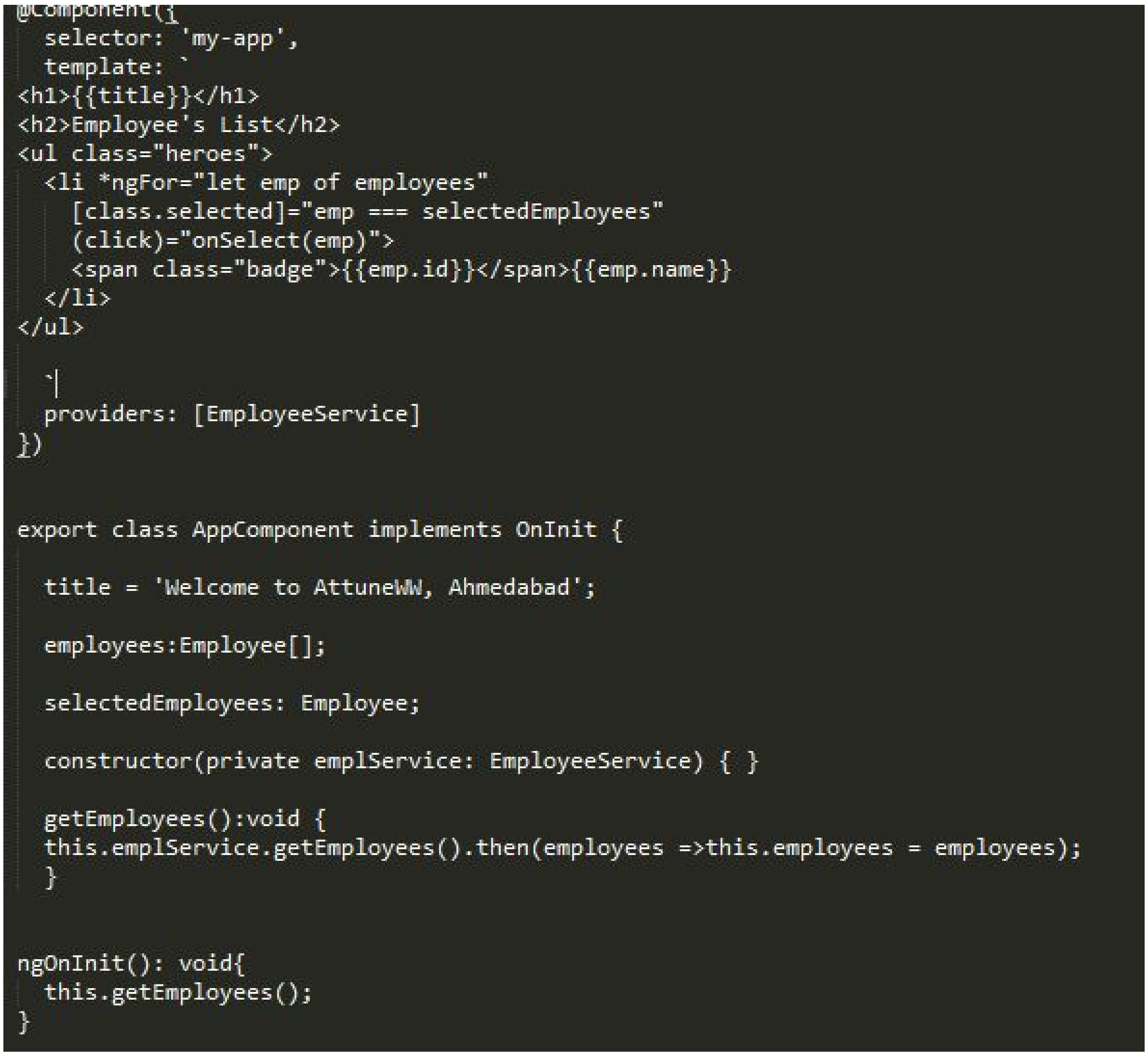
Here we are going to create a service for Employees by making a new file named "employee.service.ts".



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

import { Employee } from './employees';

import { EMPLOYEES } from './mock-employees';


@Injectable()

export class EmployeeService {

getEmployees(): Promise<Employee[]> {
    return Promise.resolve(EMPLOYEES);
}

}
```

Fig: Service for Employee

From the above figure we can see that we have imported "Injectable" function and in the next line we have also applied this as a function using @Injectable().It is recommended to put "@Injectable()" anyhow whether it is in use or not. Now inside the EmployeeService class we have injected one function named "getEmployees" to list the employee related service files. Whatever the service function is will now all be handled by this EmployeeService only. When the results are ready to be declared then with the help of Promise we will get the desired output whenever the Output is available. Promise usually works in Asynchronous way.



```
@Component({
  selector: 'my-app',
  template: `
<h1>{{title}}</h1>
<h2>Employee's List</h2>
<ul class="heroes">
  <li *ngFor="let emp of employees"
    [class.selected]="emp === selectedEmployees"
    (click)="onSelect(emp)">
    <span class="badge">{{emp.id}}</span>{{emp.name}}
  </li>
</ul>

  `
  providers: [EmployeeService]
})


export class AppComponent implements OnInit {

  title = 'Welcome to AttuneWW, Ahmedabad';

  employees:Employee[];

  selectedEmployees: Employee;

  constructor(private emplService: EmployeeService) { }

  getEmployees():void {
  this.emplService.getEmployees().then(employees =>this.employees = employees);
  }


ngOnInit(): void{
  this.getEmployees();
}
```

Fig: app.component.ts

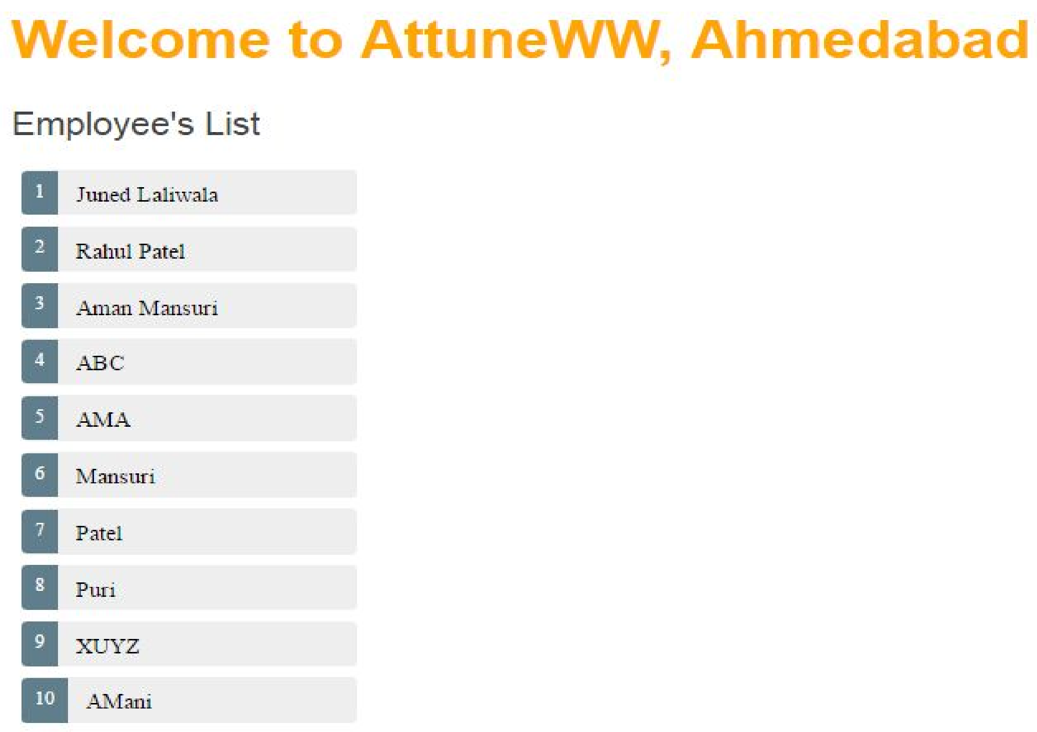
In the first three lines of the above diagram we can easily see that we have imported the methods which we have called for the different purpose.

OnInit is nothing but the initialization of the angular application.

@Component part is already discussed in the previous section.

In the Appcomponent exported class, we can see that we have defined one constructor for calling our services. This service we have declared it as a public. After that we are calling getemployees() function, in which we can see that we are calling all the parameters that have been defined like id, name and address. In the last we have initialize the method with OnInit() to recall our methods.

So the desired output will be:


Welcome to AttuneWW, Ahmedabad

Employee's List

```
1  Juned Laliwala
2  Rahul Patel
3  Aman Mansuri
4  ABC
5  AMA
6  Mansuri
7  Patel
8  Puri
9  XUYZ
10 AMani
```

Fig: Output of Employee List

From the above figure we can see that the list that we have included into our employee list is now available as output. So the complete list of all the employees are being displayed onto the screen.

9. Routing Concept in Angular 2

Till now we have seen that the simple list of the employees was displayed onto the output of the application with the help of the service. In this content, we will make some of the pages and we will redirect from one page to another. In our case, we have made list of employees so will make one page which will have the employees name and in the second page we will redirect the additional details of the employees.

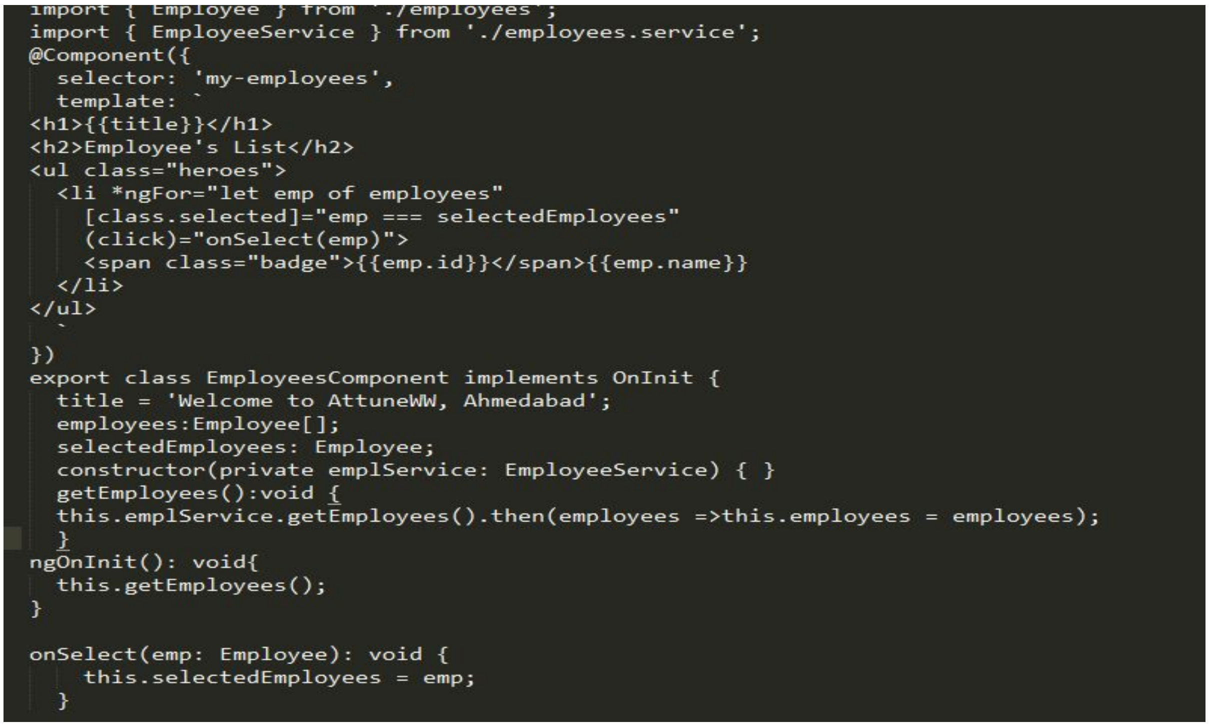
Step 1: Splitting the AppComponent

Till now in the previous section, we have seen that the App was simply showing the list of all the employees that we were putting statically. Now we will make different pages and display different content to the different pages.

In this step, we have to simple rename the three important files namely “app.component file”, “AppComponent” parameter and “my-app” selector to any of our desired name.

So as for example we are changing the value of app.component file to employees.component.ts, AppComponent to EmployeesComponent and my-app selector to my-employees.

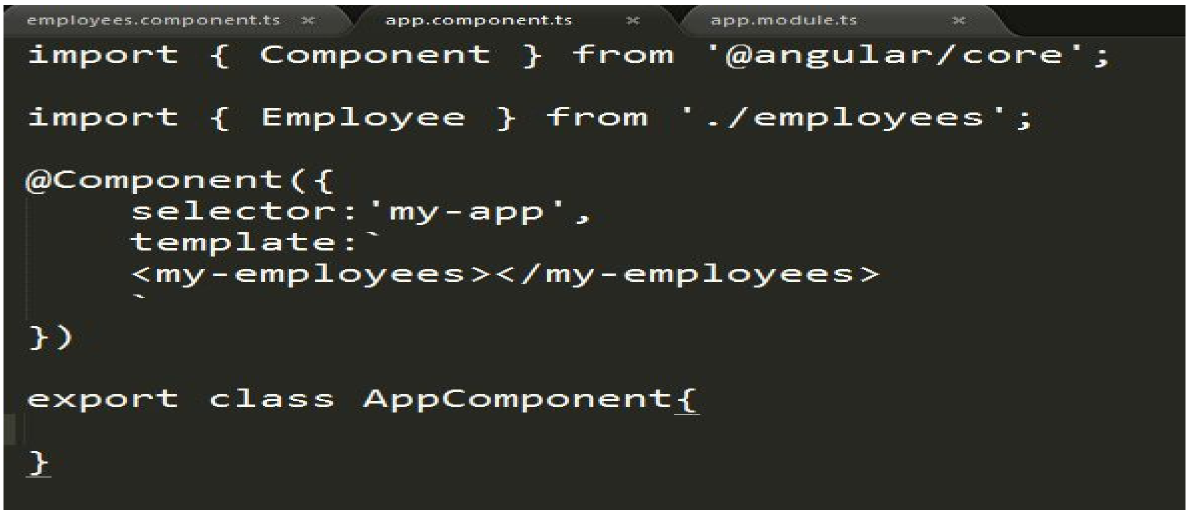
The main reason for changing the app.component file is that this file be now working as a shell. This file will now be containing links for different pages that needs to be navigated from one page to another.



```
import { Employee } from './employees';
import { EmployeeService } from './employees.service';
@Component({
  selector: 'my-employees',
  template: `
<h1>{{title}}</h1>
<h2>Employee's List</h2>
<ul class="heroes">
  <li *ngFor="let emp of employees"
    [class.selected]="emp === selectedEmployees"
    (click)="onSelect(emp)">
    <span class="badge">{{emp.id}}</span>{{emp.name}}
  </li>
</ul>
  `
})
export class EmployeesComponent implements OnInit {
  title = 'Welcome to AttuneWW, Ahmedabad';
  employees:Employee[];
  selectedEmployees: Employee;
  constructor(private emplService: EmployeeService) { }
  getEmployees():void {
  this.emplService.getEmployees().then(employees =>this.employees = employees);
  }
ngOnInit(): void{
  this.getEmployees();
}

onSelect(emp: Employee): void {
    this.selectedEmployees = emp;
  }
```

Fig: employees.component.ts

As we have discussed that we have to change the file name to better understand the concept of the routing. We have simply made the changes that were being encountered in the precious section.



employees.component.ts | app.component.ts | app.module.ts

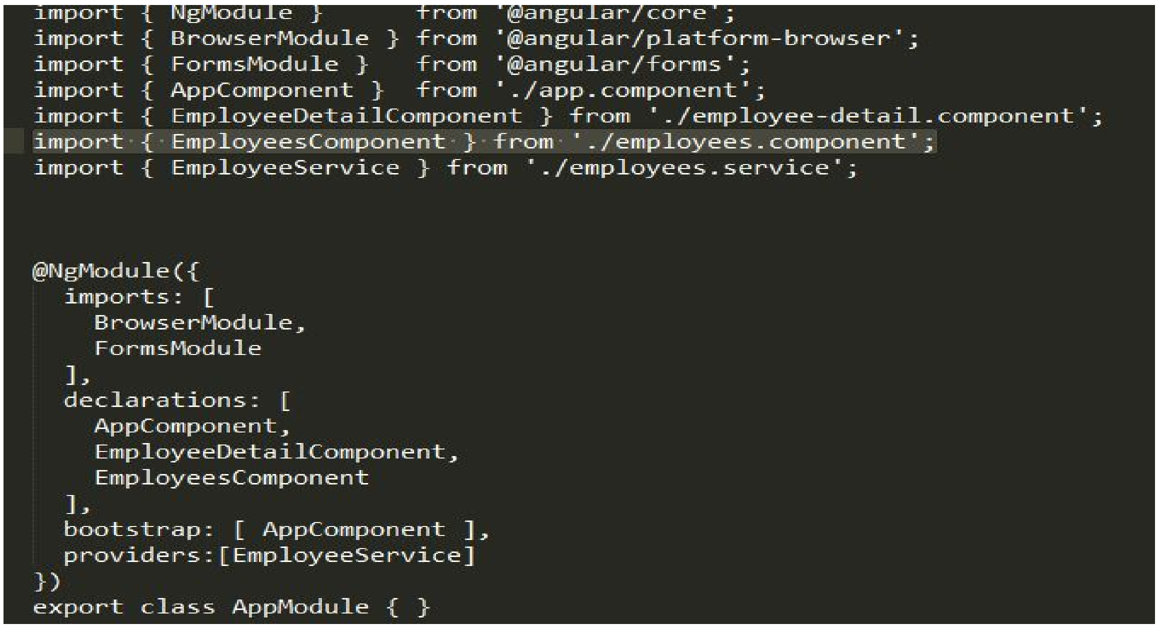
```
import { Component } from '@angular/core';

import { Employee } from './employees';

@Component({
      selector:'my-app',
      template:`
      <my-employees></my-employees>
      `
})

export class AppComponent{

}
```

Fig: Newly Created app.component File

From the above figure we can see that a new app component file has been created with a selector as "my-app". But as we can see that there is some change in the template section, we have included <my-employees> tag. This tag is nothing but the employees list will be called in the output that we have done in the previous sections. The reason for declaring

<my-employees> as a tag is that in the previous figure we can see that the selector has been “my-employees” so we are preferring this tag.

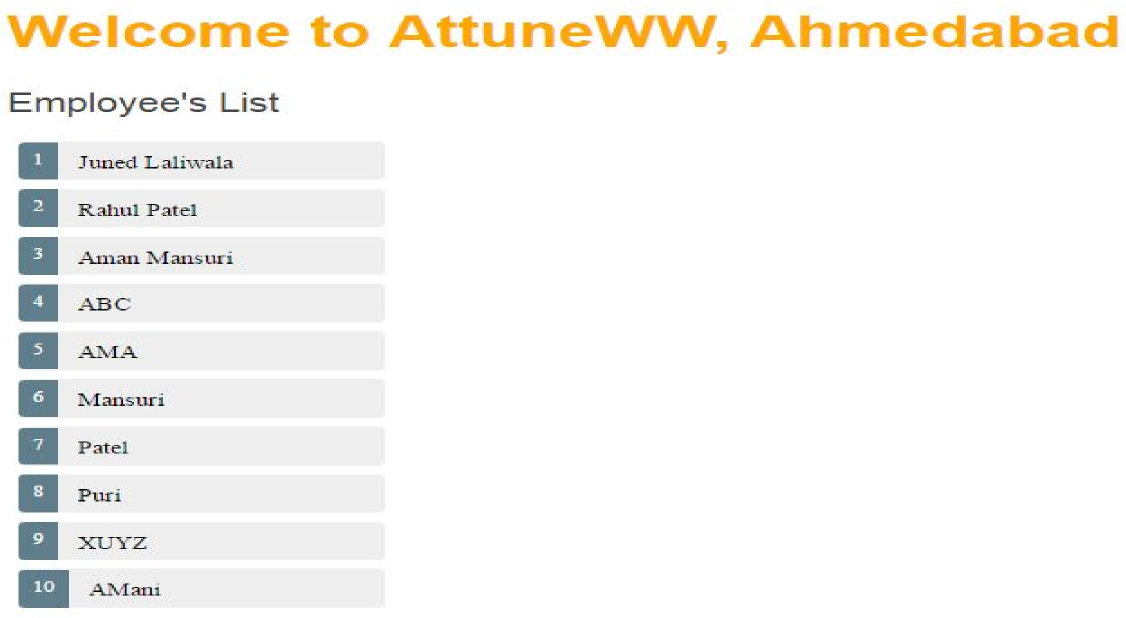
After configuring this changes, we will now have to add some parameters to our Module file for our application. We first simply have to include the Component file of Employee List by importing them from the location where it is available.



```
import { NgModule }       from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
import { FormsModule }    from '@angular/forms';
import { AppComponent }   from './app.component';
import { EmployeeDetailComponent } from './employee-detail.component';
import { EmployeesComponent } from './employees.component';
import { EmployeeService } from './employees.service';



@NgModule({
  imports: [
    BrowserModule,
    FormsModule
  ],
  declarations: [
    AppComponent,
    EmployeeDetailComponent,
    EmployeesComponent
  ],
  bootstrap: [ AppComponent ],
  providers:[EmployeeService]
})
export class AppModule { }
```

Fig: app.module.ts (After changes)

From the above figure we can see that we have imported the files and the corresponding output will be:



```
Welcome to AttuneWW, Ahmedabad

Employee's List

1   Juned Laliwala
2   Rahul Patel
3   Aman Mansuri
4   ABC
5   AMA
6   Mansuri
7   Patel
8   Puri
9   XUYZ
10  AMani
```

Fig: Desired Output

So this result is same as the previous one. So everything till now it's perfect. Now we will add some routing functionalities in such a way that when the user clicks on any of the employees list, then the complete details of the Employees will be displayed on the next page.

Adding Routing

Now we are moving further by clicking on a single employee, we will be able to see the complete details of them on the next page. This process is also called the navigation of the pages.

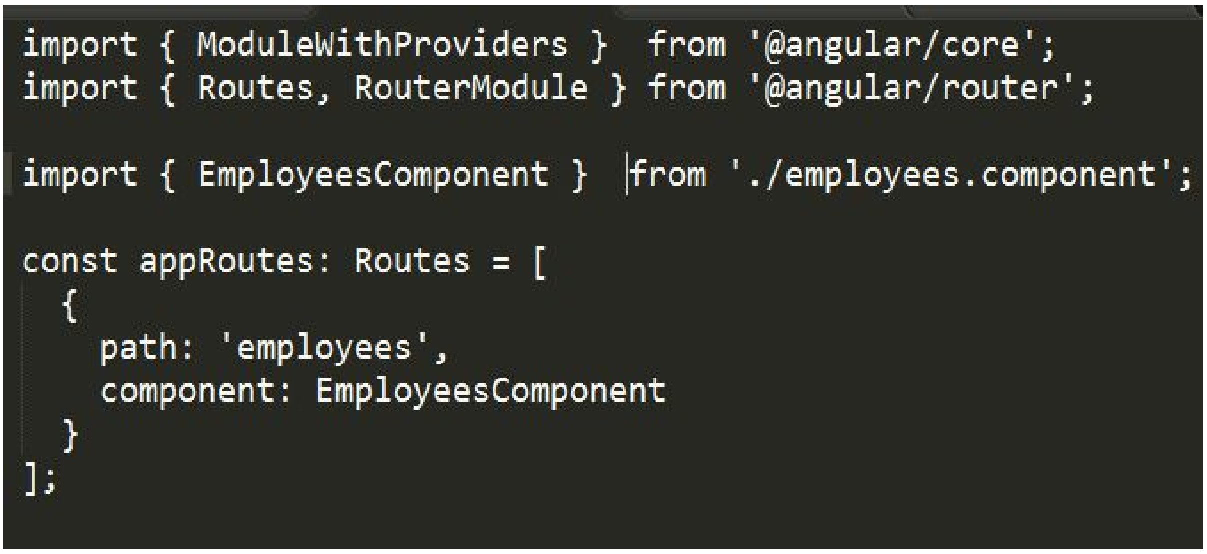
The main functionalities of the Routing will be handled by RouterModule. This router is particularly a combination of multiple provided services (RouterModule), multiple directives (RouterOutlet, RouterLink, RouterLinkActivate) and Configuration(Routes). This three files are the most important part for understanding the routing functionalities in Angular.

Step 1: Base tag

In index.html file, below the head section and at the top it is mandatory top top define one base tag as: *<base href* = *"/">*

Step 2: Configure Routes

Currently we are not having a simple route in our application. So first we will create one configuration file for routing and let's say we name it as "**app.routing.ts**" inside the app folder. The main aim of routes is that it tells the router to link to which page when the user clicks on the button.



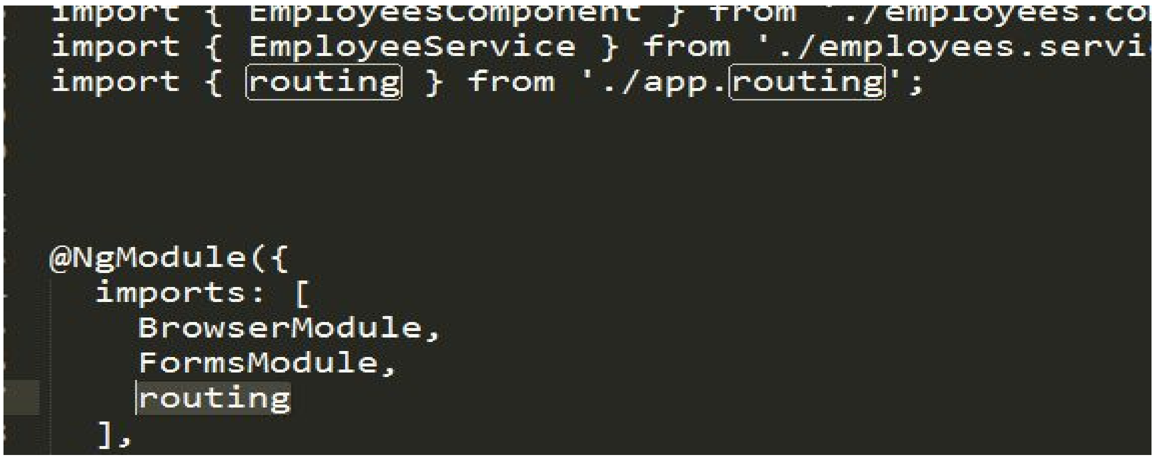
```
import { ModuleWithProviders }  from '@angular/core';
import { Routes, RouterModule } from '@angular/router';

import { EmployeesComponent }  from './employees.component';

const appRoutes: Routes = [
  {
    path: 'employees',
    component: EmployeesComponent
  }
];
```

Fig: app.routing.ts

The Routes that you are seeing in the above definition basically contains an array of the route definition. Currently there is only single route definition, but in the real time we can use as many as required for the application.

Now this routes contains basic two parts: **Path** is nothing but the link or the path that we have provided will be checked by the router in the browser. **Component** is the component name that the router wants to configure the path.

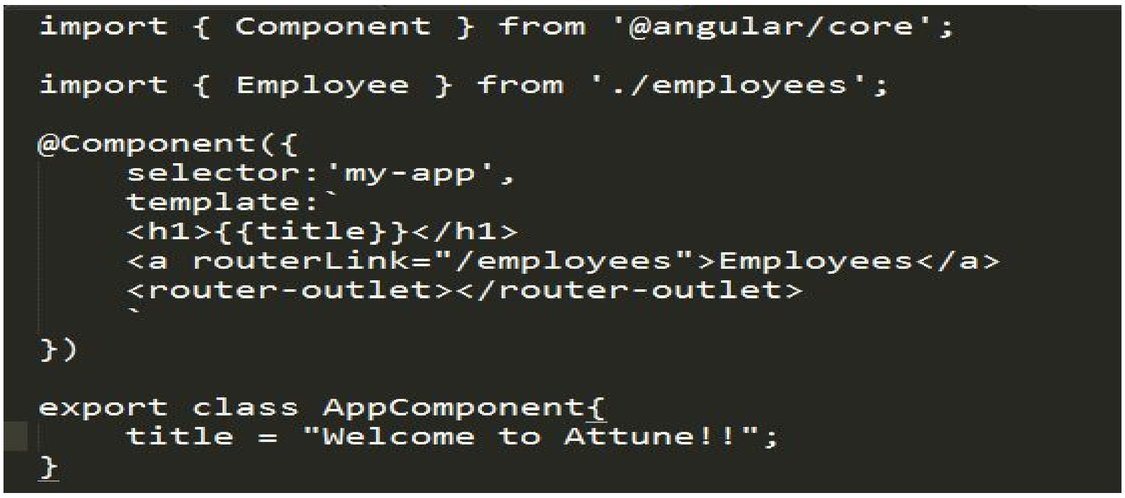
This routes is to be added to the Module file inside the NgModule.



```
import { EmployeesComponent } from './employees.co
import { EmployeeService } from './employees.servi
import { routing } from './app.routing';



@NgModule({
  imports: [
    BrowserModule,
    FormsModule,
    routing
  ],
```

Fig: app.module.ts

<Router-outlet>: is nothing but the directive which is provided by the RouterModule. The main role of router-outlet is that the router will display each and every component immediately below router-outlet



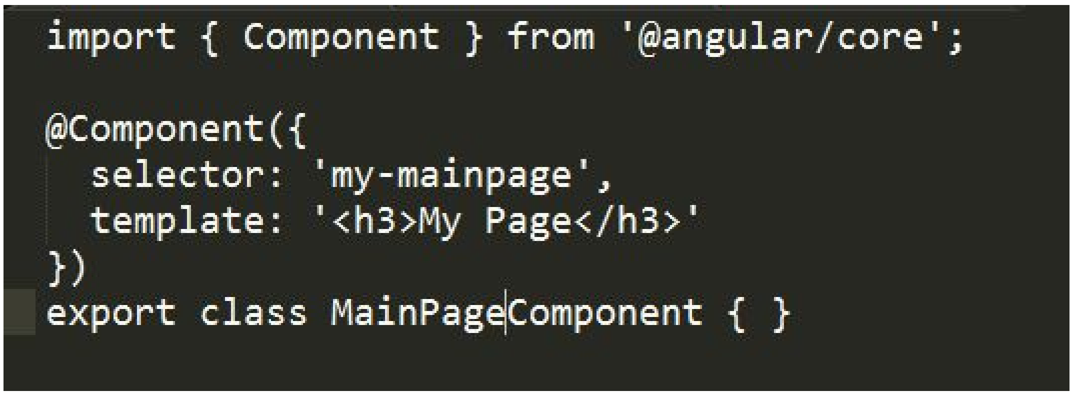
```
import { Component } from '@angular/core';

import { Employee } from './employees';

@Component({
    selector:'my-app',
    template:`
    <h1>{{title}}</h1>
    <a routerLink="/employees">Employees</a>
    <router-outlet></router-outlet>
    `
})

export class AppComponent{
    title = "Welcome to Attune!!";
}
```

Fig:Router Link Example

The main role of routerLink is to provide the location where to navigate.

Making a Home-Page

Now we will make one home page such that whenever the user starts the application then the user will be first redirected to this page.

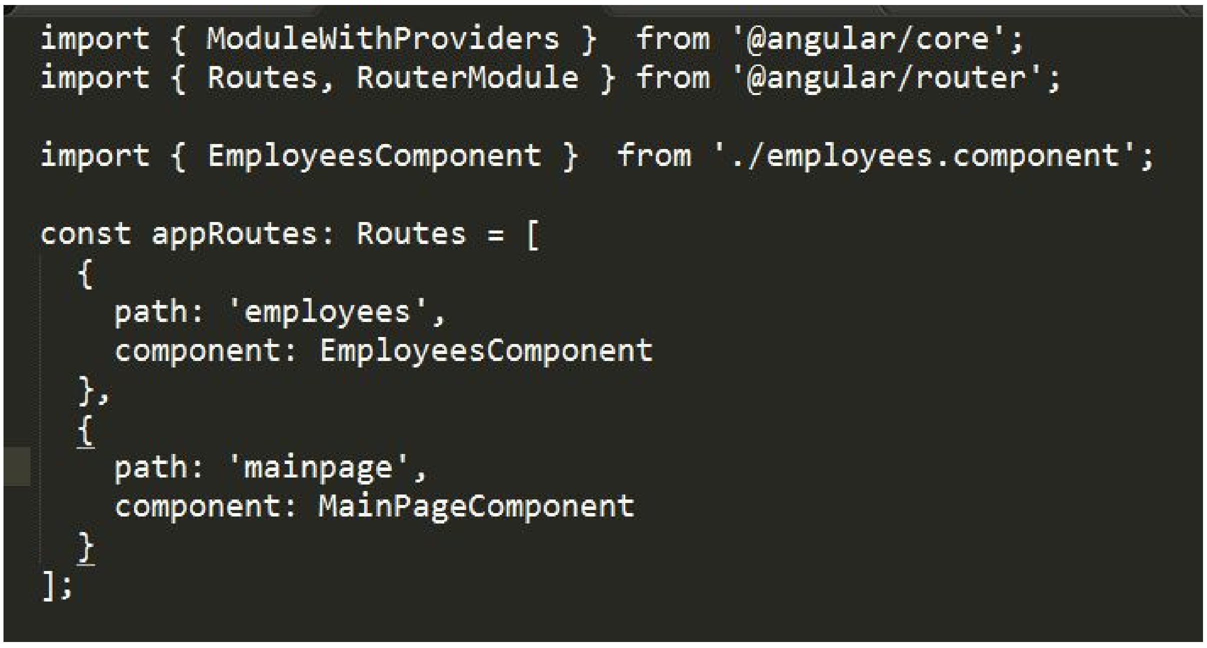


```
import { Component } from '@angular/core';

@Component({
  selector: 'my-mainpage',
  template: '<h3>My Page</h3>'
})
export class MainPageComponent { }
```

Fig: MainPage Component File

1. We have made a file named "mainpage.component.ts" which is nothing but the component file for the main page.
2. Selector for this page is my-mainpage. Whereas Exported class name is MainPageComponent.

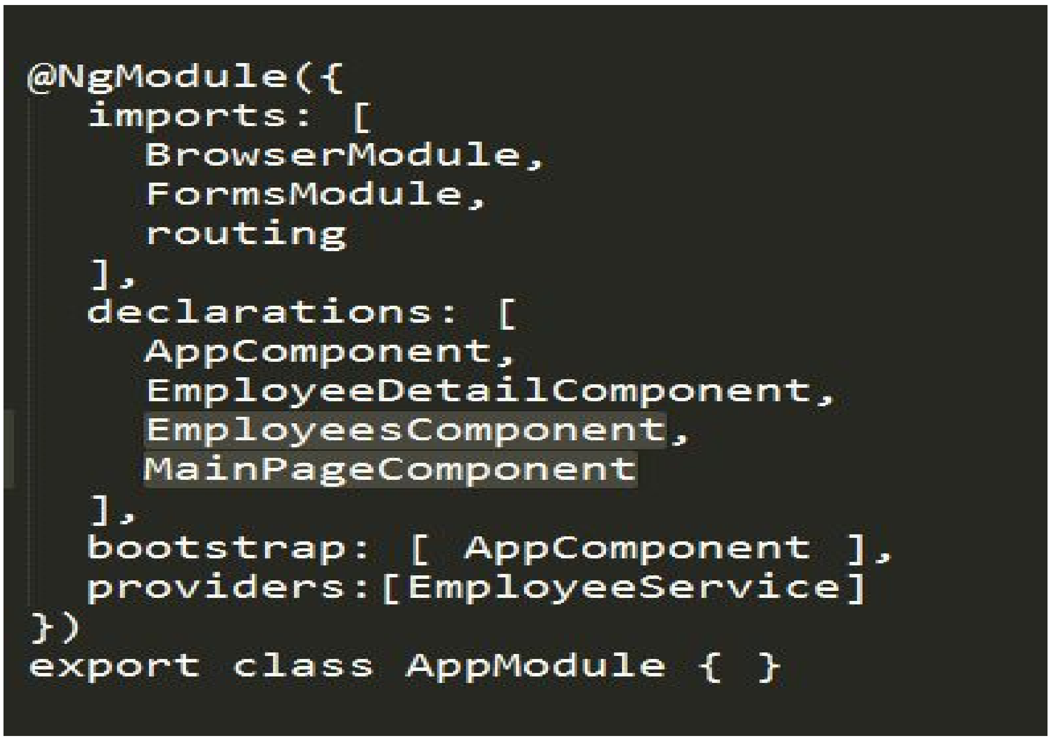
Now to add routing functionality, we will have to go back to **"app.routing.ts"** and add the following route path as shown in the below diagram.



```
import { ModuleWithProviders }  from '@angular/core';
import { Routes, RouterModule } from '@angular/router';

import { EmployeesComponent }  from './employees.component';

const appRoutes: Routes = [
  {
    path: 'employees',
    component: EmployeesComponent
  },
  {
    path: 'mainpage',
    component: MainPageComponent
  }
];
```

Fig: Applying Routes using appRoutes

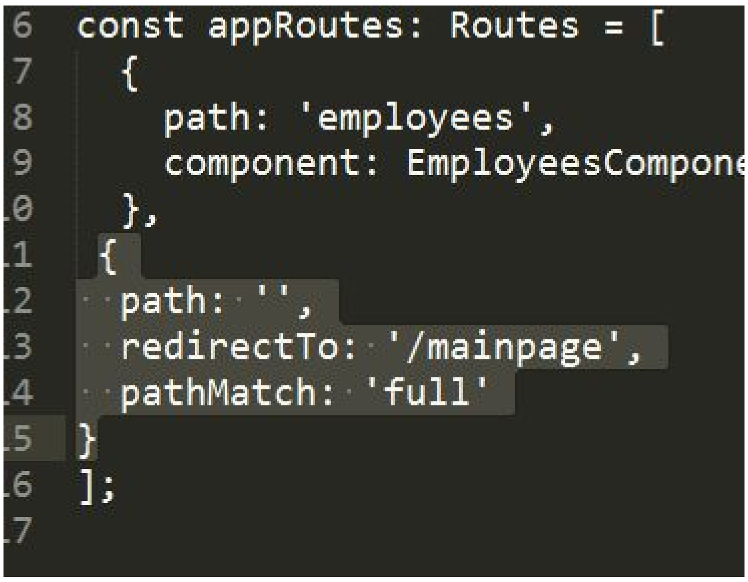
As we can see that there are now two Components, so we have to also include that Components into module file as:



```
@NgModule({
  imports: [
    BrowserModule,
    FormsModule,
    routing
  ],
  declarations: [
    AppComponent,
    EmployeeDetailComponent,
    EmployeesComponent,
    MainPageComponent
  ],
  bootstrap: [ AppComponent ],
  providers:[EmployeeService]
})
export class AppModule { }
```

Fig: Including Employees and MainPage in Declaration

redirectTo is as the name suggests is used to move to the location that has been applied to move. In the figure shown below it is being redirected to the main page. This main page is not considering path as blank which means it is the default page.

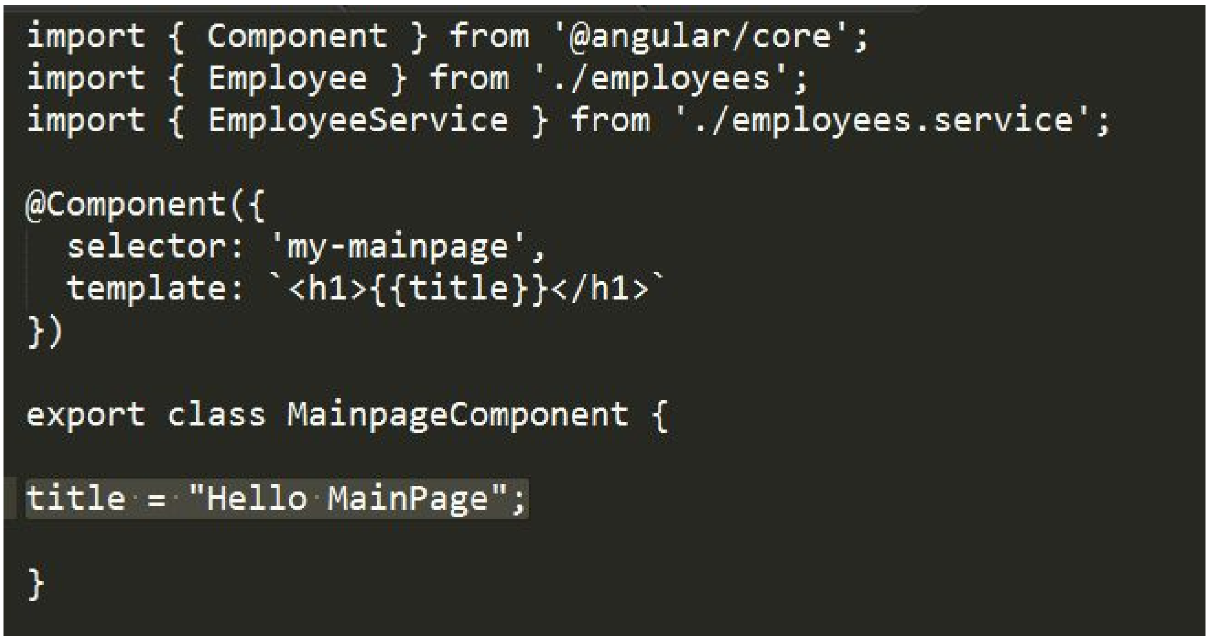


```
const appRoutes: Routes = [
  {
    path: 'employees',
    component: EmployeesCompone
  },
 {
  path: '',
  redirectTo: '/mainpage',
  pathMatch: 'full'
}
];
```

Fig: RedirectTo in routing.ts

Adding Navigation to the templates

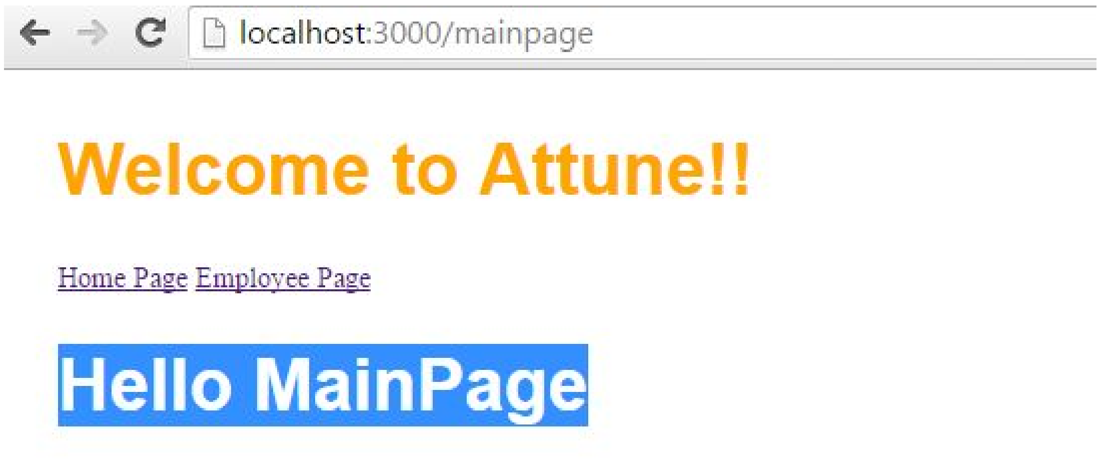
Here we are making a new page named “mainpage” which is nothing but the home page that will be open whenever we start our application.



```
import { Component } from '@angular/core';
import { Employee } from './employees';
import { EmployeeService } from './employees.service';

@Component({
  selector: 'my-mainpage',
  template: `<h1>{{title}}</h1>`
})

export class MainpageComponent {

title = "Hello MainPage";

}
```

Fig: Main Page Component File

So as we can see that we have included a simple title into our mainpage component so that we can just check whether the application shows the main page or not.


localhost:3000/mainpage

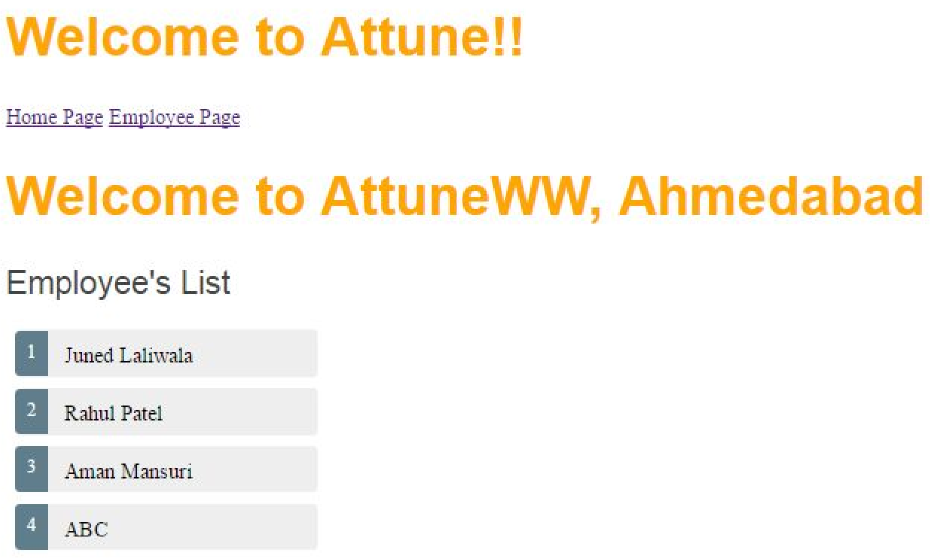
Welcome to Attune!!

Home Page Employee Page

Hello MainPage

Fig: Outcome of the Main Page

The below figure shows that the link that can be provided for the application. From the figure we can see that we have included two links into our project.


Welcome to Attune!!

Home Page Employee Page

Welcome to AttuneWW, Ahmedabad

Employee's List

1 Juned Laliwala

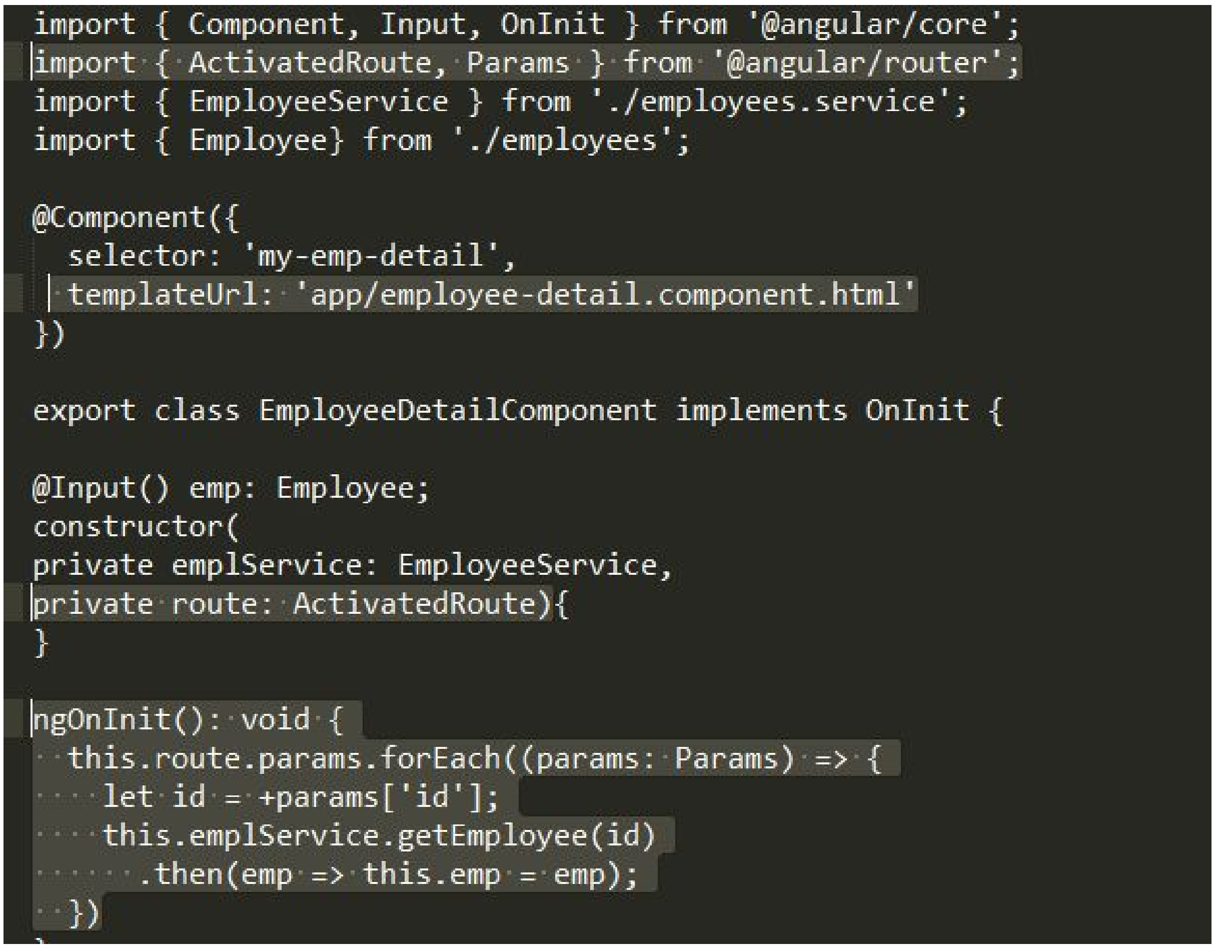
2 Rahul Patel

3 Aman Mansuri

4 ABC

Fig: Employee's List Page with Menu Links

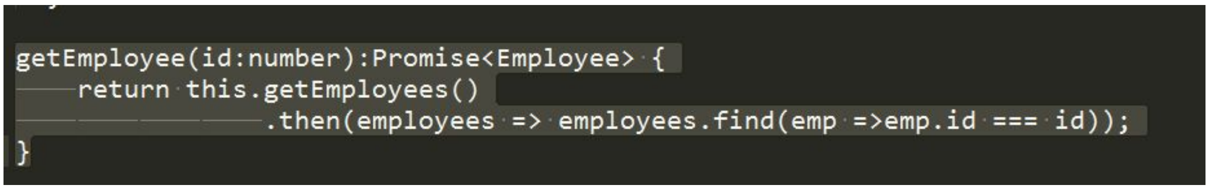
Now with the help of routing we will click on any of our Employees and we will see the complete details of that particular Employees in another page. For performing such a task, we will have to invoke some methods that we will see in the following section.



```
import { Component, Input, OnInit } from '@angular/core';
import { ActivatedRoute, Params } from '@angular/router';
import { EmployeeService } from './employees.service';
import { Employee} from './employees';

@Component({
  selector: 'my-emp-detail',
  templateUrl: 'app/employee-detail.component.html'
})

export class EmployeeDetailComponent implements OnInit {

@Input() emp: Employee;
constructor(
private emplService: EmployeeService,
private route: ActivatedRoute){
}

ngOnInit(): void {
  this.route.params.forEach((params: Params) => {
    let id = +params['id'];
    this.emplService.getEmployee(id)
      .then(emp => this.emp = emp);
  })
```

Fig: Employee-details.Component.ts

This is the Employee Details page. We will discuss the complete details of this page. First we can see that we have imported Activated Router and Params which are mandatory for routing which usually comes the core part of the Angular Applications.In component part we are now defining the path of html file by passing the path of the file. The route file that we have imported in the header part needs to be defined inside the exported Component class. We have as usual imported the Initialization of the angular by passing the ID of the employees in the Employees service and with the help of such parameters we can see the complete details of the Employee.

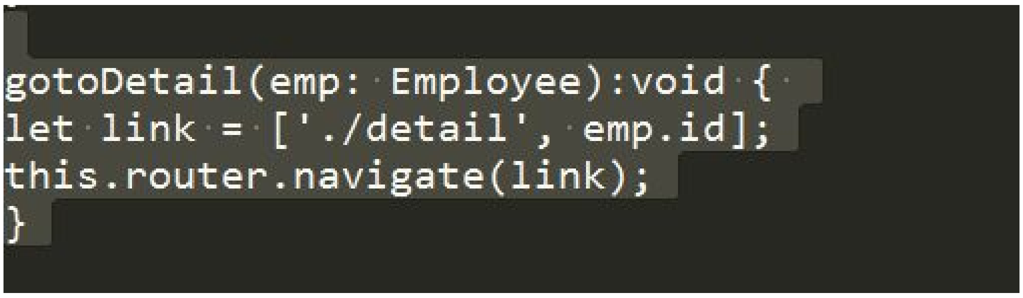


```
getEmployee(id:number):Promise<Employee> {
    return this.getEmployees()
                .then(employees => employees.find(emp =>emp.id === id));
}
```

Fig: getEmployee( ) Method

Now, inside the service file of the employee we are making a new function namely "getEmployee" which is getting the parameter as ID and based on this ID the complete details of the Employees will be called by the getEmployees( ) function as we can see the functionalities that we have defined inside this service.

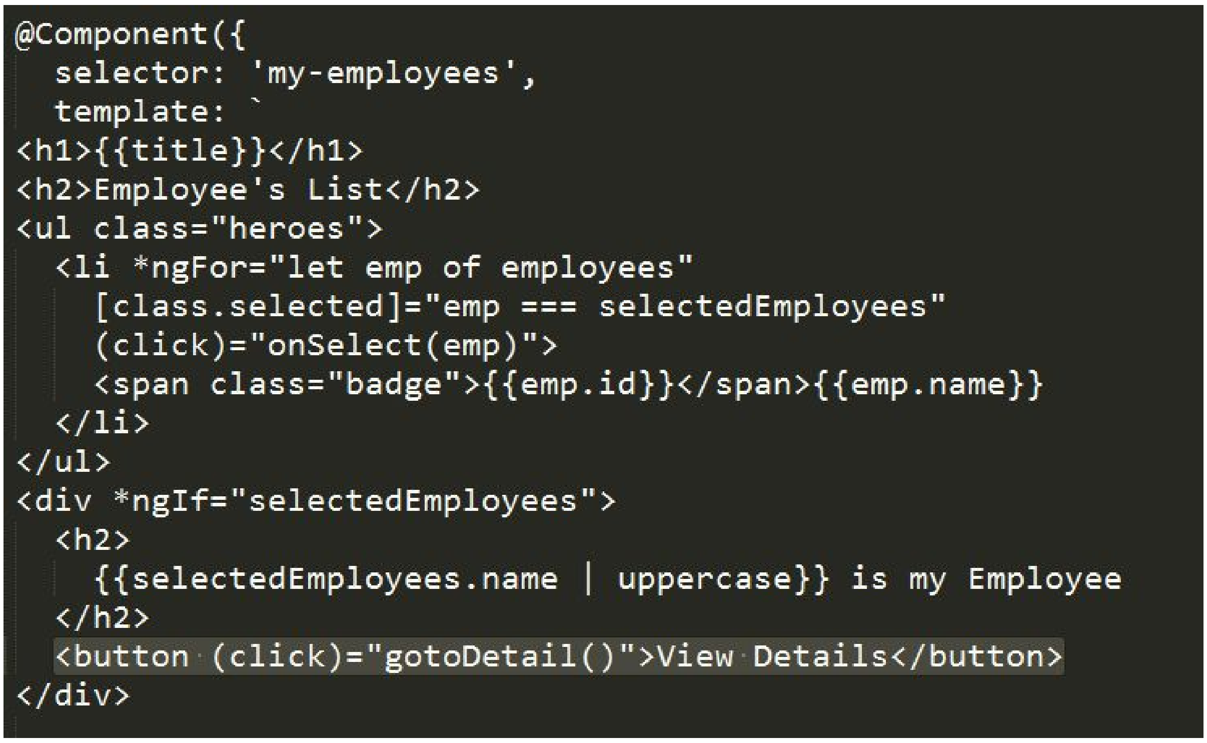
So far we have seen that we have to get the complete details of the employees so that we are defining one of the function inside a button so that when the user clicks on the button, the complete details will be displayed onto the next screen.



```
gotoDetail(emp: Employee):void {
let link = ['./detail', emp.id];
this.router.navigate(link);
}
```

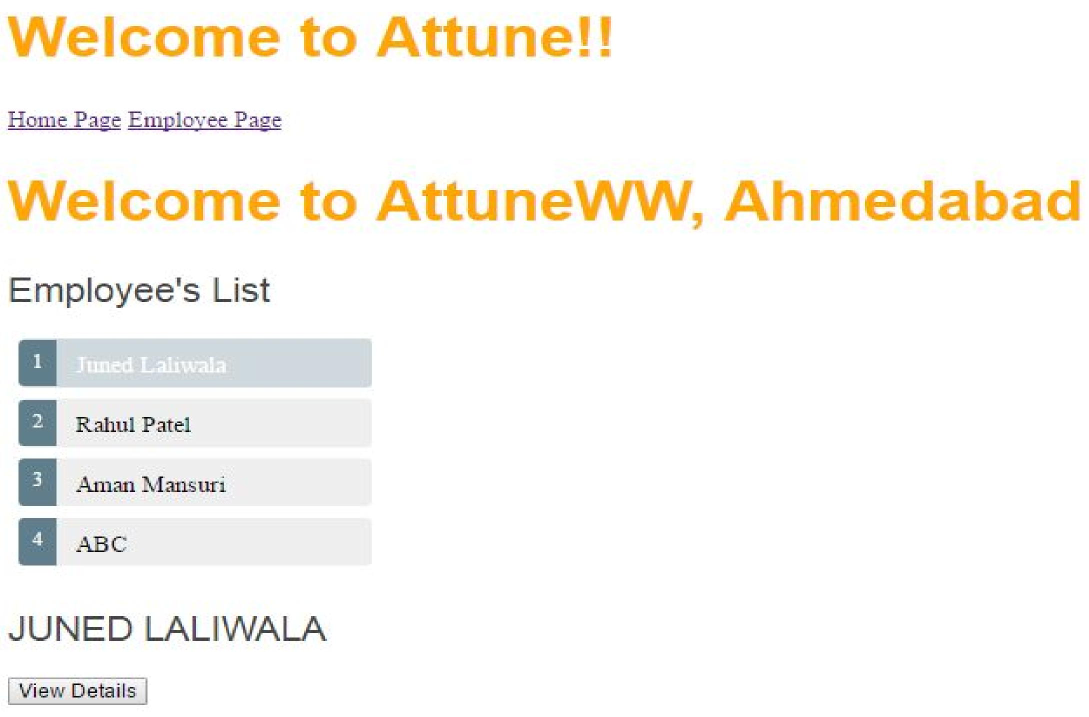
Fig: gotoDetail Function

GotoDetail is the function to get the complete details based on Employee’s ID.



```
@Component({
  selector: 'my-employees',
  template: `
<h1>{{title}}</h1>
<h2>Employee's List</h2>
<ul class="heroes">
  <li *ngFor="let emp of employees"
    [class.selected]="emp === selectedEmployees"
    (click)="onSelect(emp)">
    <span class="badge">{{emp.id}}</span>{{emp.name}}
  </li>
</ul>
<div *ngIf="selectedEmployees">
  <h2>
    {{selectedEmployees.name | uppercase}} is my Employee
  </h2>
  <button (click)="gotoDetail()">View Details</button>
</div>
```

As we can see from the above figure that we have applied a button click and inside that click we have called one function named gotoDetail which is nothing but the details of the employees will be displayed onto the screen.



Welcome to Attune!!

Home Page Employee Page

Welcome to AttuneWW, Ahmedabad

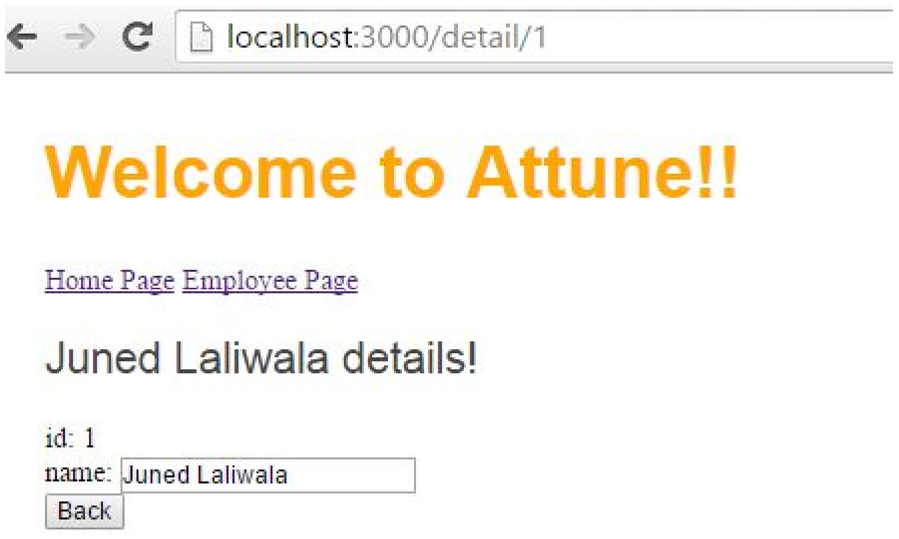
Employee's List

1	Juned Laliwala
2	Rahul Patel
3	Aman Mansuri
4	ABC

JUNED LALIWALA

Fig: Outcome of the Employee Page

Now as we can see that when the user clicks on one of the Employee then name is being displayed on to same page. Now when the user clicks on the view details button then the details of the page will be as:



localhost:3000/detail/1

Welcome to Attune!!

Home Page Employee Page

Juned Laliwala details!

id: 1
name: Juned Laliwala

Fig: Employee Details on the next Page

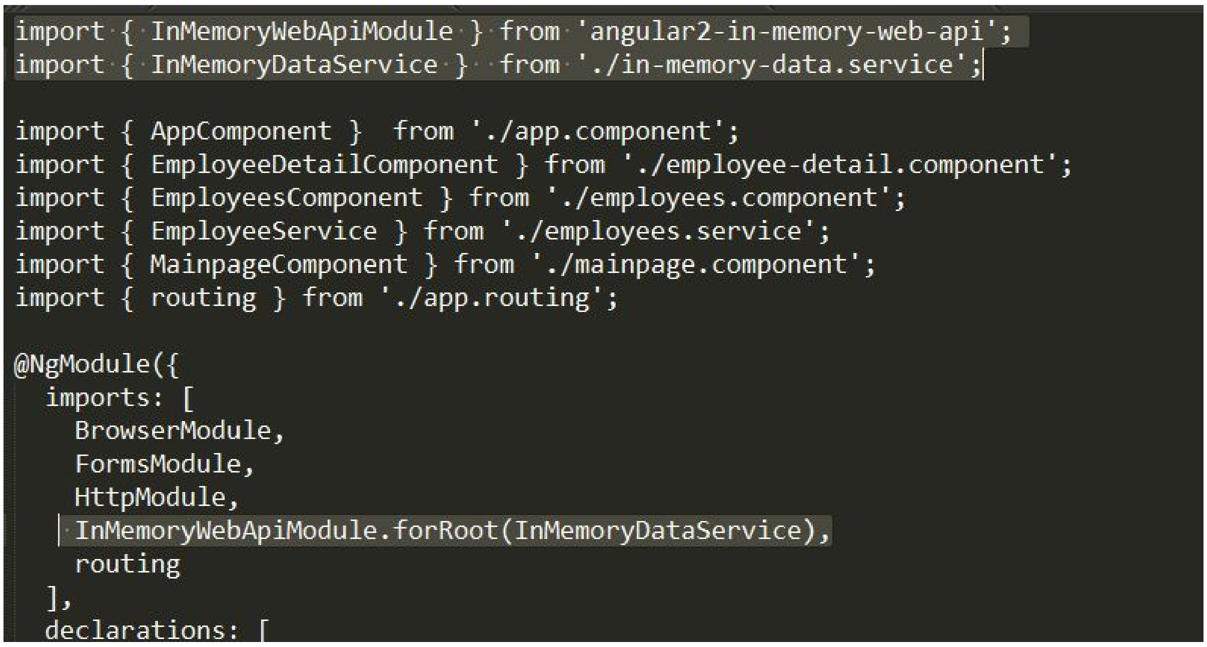
So cool, we have the details of the Employee and in the browser we can see the url part, it is displaying the ID of the displayed Employee.

10. Working with HTTP Concept

So far as we have seen in the AngularJs that the modules that we were referring for using HTTO was HttpModule. The same is the case with the Angular 2, here also we are using HttpModule. The main intense is to access the web pages and it exists on another core part of angular 2 that is '@angular/http'.
Now as we are going to use HTTP Modules into our application, so we have to mandatorily import http modules into our app's modules file.

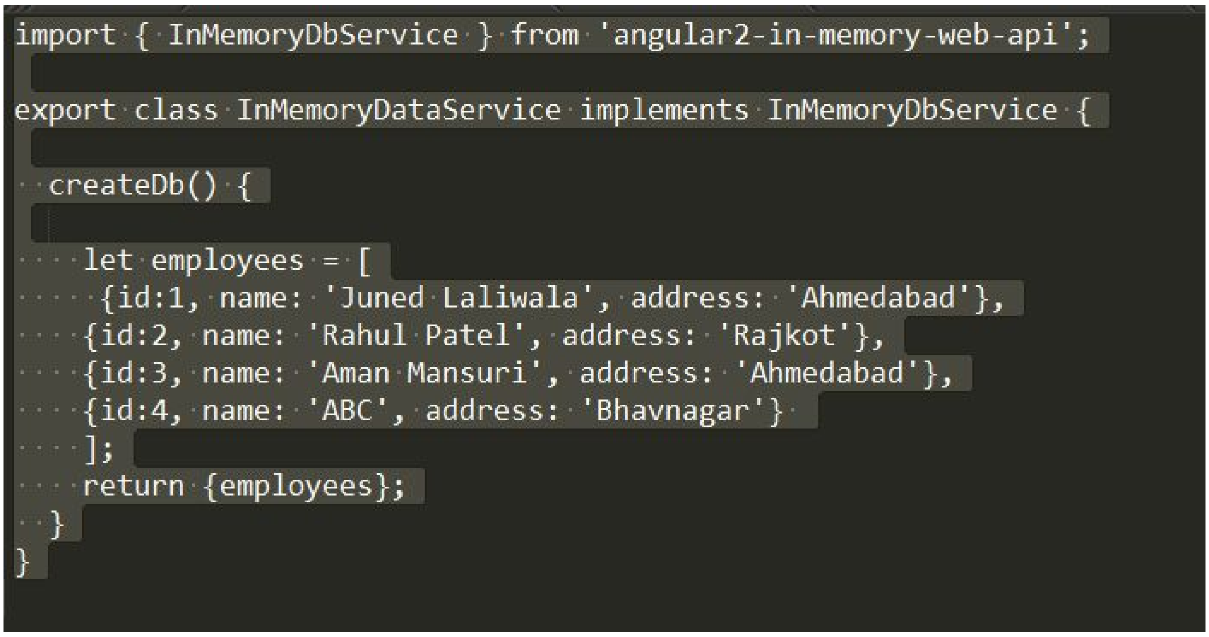
Currently we are not having any web service for handling the data that we are going to access using http modules. So we will make one temporary web service for showing the demonstration of the web services using http requests.

So again we will have to import two important things for Http Modules namely "InMemoryWebApiModule" and "InMemoryDataService" in module file. Apart from it we will have to import in NgModule the important field as shown in the figure below:



```
import { InMemoryWebApiModule } from 'angular2-in-memory-web-api';
import { InMemoryDataService }  from './in-memory-data.service';

import { AppComponent }  from './app.component';
import { EmployeeDetailComponent } from './employee-detail.component';
import { EmployeesComponent } from './employees.component';
import { EmployeeService } from './employees.service';
import { MainpageComponent } from './mainpage.component';
import { routing } from './app.routing';

@NgModule({
  imports: [
    BrowserModule,
    FormsModule,
    HttpModule,
    InMemoryWebApiModule.forRoot(InMemoryDataService),
    routing
  ],
  declarations: [
```

Fig:
From the above figure we can see that we have imported InMemoryWebApiModule, this module replaces the Http Client Backend with the InMemoryDataService. So all the http web client will be now handled by In Memory Data Service.
The main use of "*forroot*" in web Api Module is that will handle the database service as shown in the diagram below:

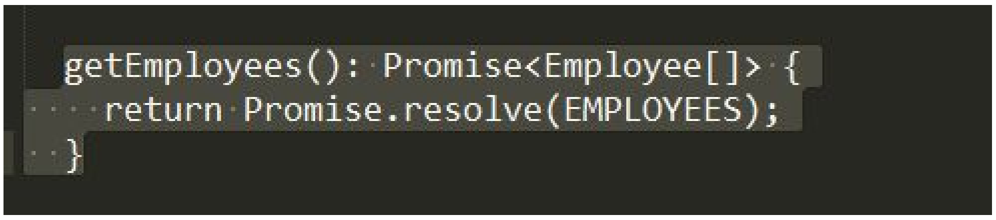


```
import { InMemoryDbService } from 'angular2-in-memory-web-api';

export class InMemoryDataService implements InMemoryDbService {

  createDb() {

    let employees = [
     {id:1, name: 'Juned Laliwala', address: 'Ahmedabad'},
    {id:2, name: 'Rahul Patel', address: 'Rajkot'},
    {id:3, name: 'Aman Mansuri', address: 'Ahmedabad'},
    {id:4, name: 'ABC', address: 'Bhavnagar'}
    ];
    return {employees};
  }
}
```

Fig: Making Database file

From the above figure we can see that we have imported Database Service and after that we are exporting the class of the Data Service. Inside this Data Service class, we are making one Database named employees and adding some static values into it.

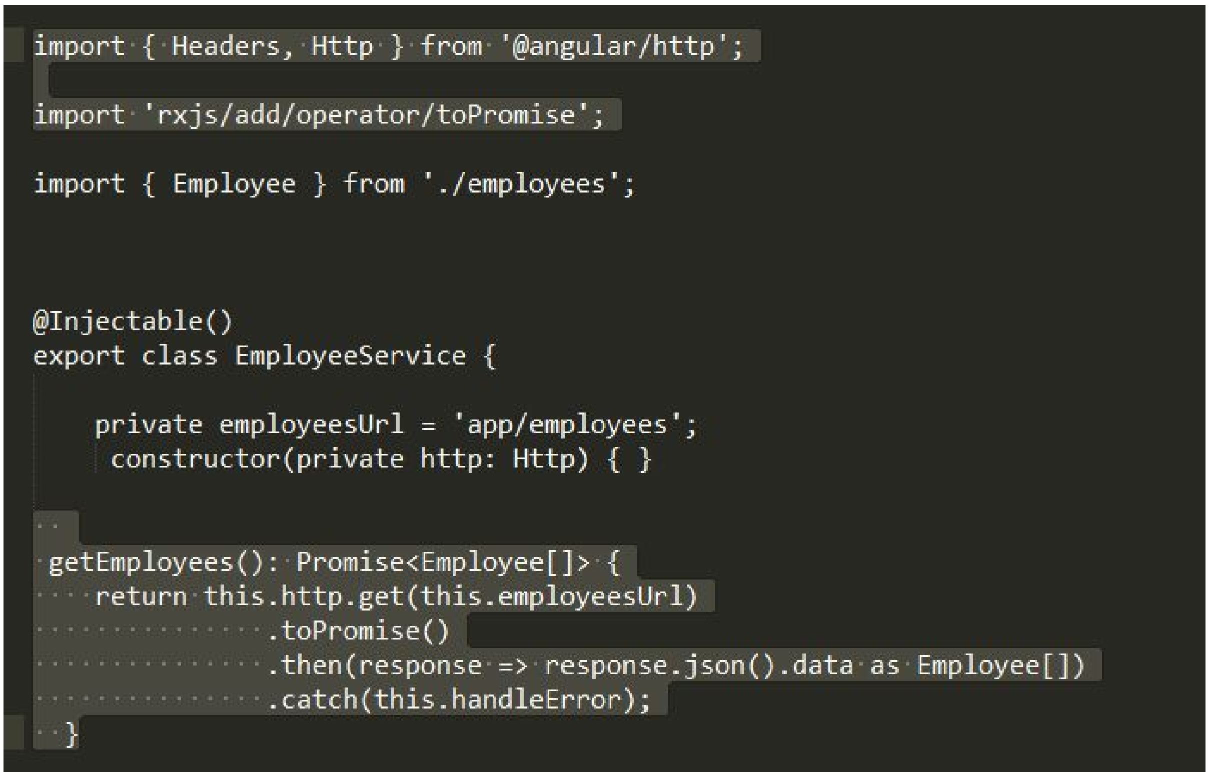
In the file that we have made named “mock-employees.ts” which were handling all the details about the employees so now it’s safe to delete that file because now we have made one database file in which we will handle the employee’s Complete details.



```
  getEmployees(): Promise<Employee[]> {
    return Promise.resolve(EMPLOYEES);
  }
```

Fig: Old GetEmployees Service File

From the above figure we can see that we have made one getEmployees() service file in which we were getting the complete details of the employees by returning the Promise parameter to fetch the details. Now the time has come to replace this promise by putting the data base file in regards to Promise to fetch the details.
The changes that are being made to Service file of the Employees is as follows:



```
import { Headers, Http } from '@angular/http';

import 'rxjs/add/operator/toPromise';

import { Employee } from './employees';



@Injectable()
export class EmployeeService {

    private employeesUrl = 'app/employees';
     constructor(private http: Http) { }


 getEmployees(): Promise<Employee[]> {
    return this.http.get(this.employeesUrl)
               .toPromise()
               .then(response => response.json().data as Employee[])
               .catch(this.handleError);
  }
```

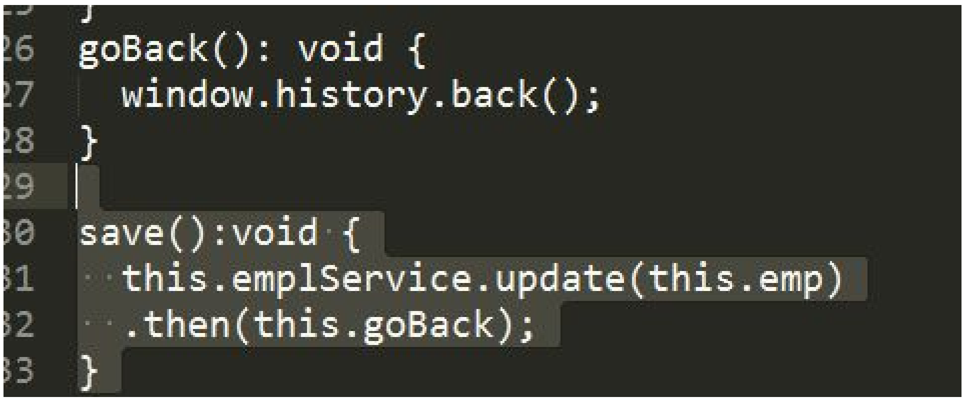
Fig: Changes in Service Files

So after refreshing the browser, we can see that overall functionalities remains the same. So that means we are on the right path.
In the previous service file we were using the promise parameter to fetch the details, but now we are preferring the rxjs Observable parameter to work the same details. Instead of the using Promise method we are now using ".toPromise() function". After that we are calling the parameters by using response.json data, which will return all the details of the employees into our application.

Saving Employees Details

As we have seen so far that we were adding the employees details and were able to view the employees but now when we edit the employees then we can easily save that employees details on to the next page. We will make a button to save the details. When the user clicks on the save button then the details will be updated.

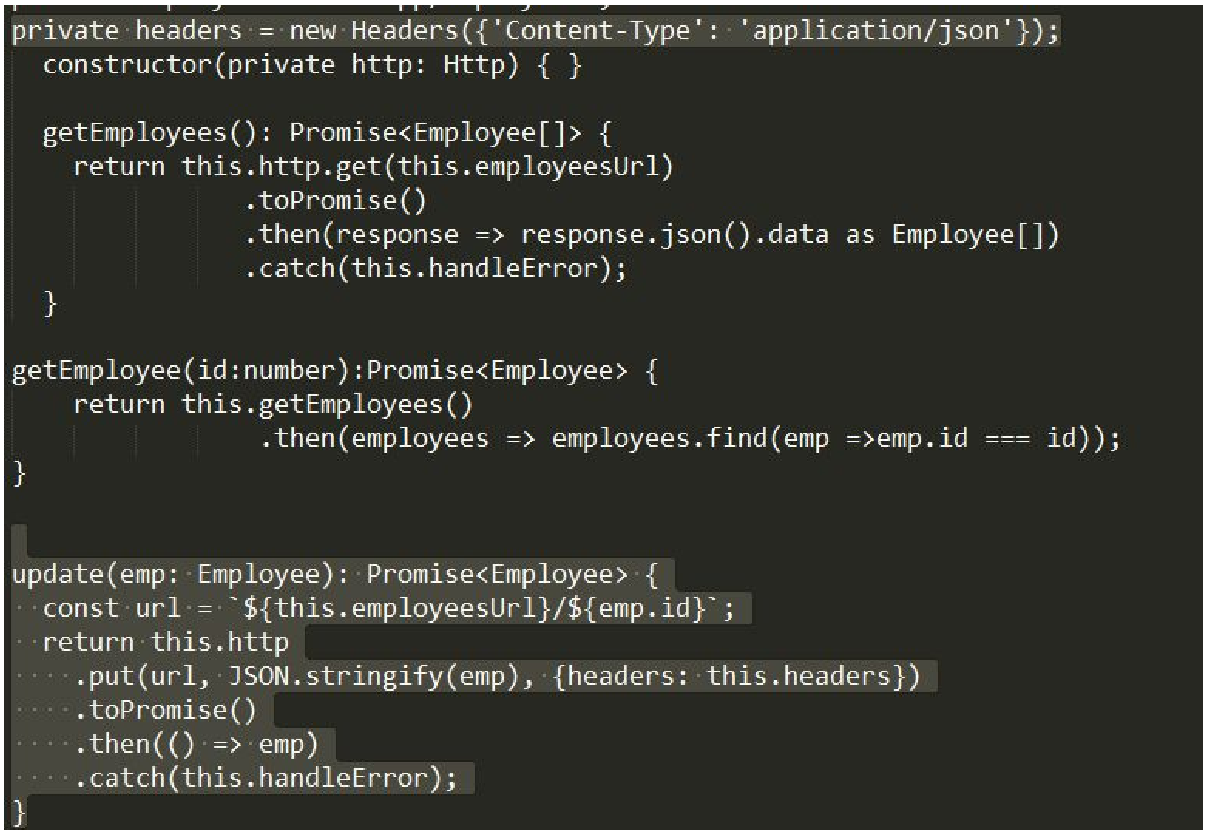
In employee-detail.component html file , we are adding one button and inside that button we are passing the save function to save the details of the employees. Now inside the employee-detail component ts file we will add the save functionalities as shown below:



```
goBack(): void {
  window.history.back();
}

save():void {
  this.emplService.update(this.emp)
  .then(this.goBack);
}
```

Fig: Save the Employee Details

Now we have to apply the Update Method that we have passed in the above figure. In Employee-service.ts file we have to update as follows:

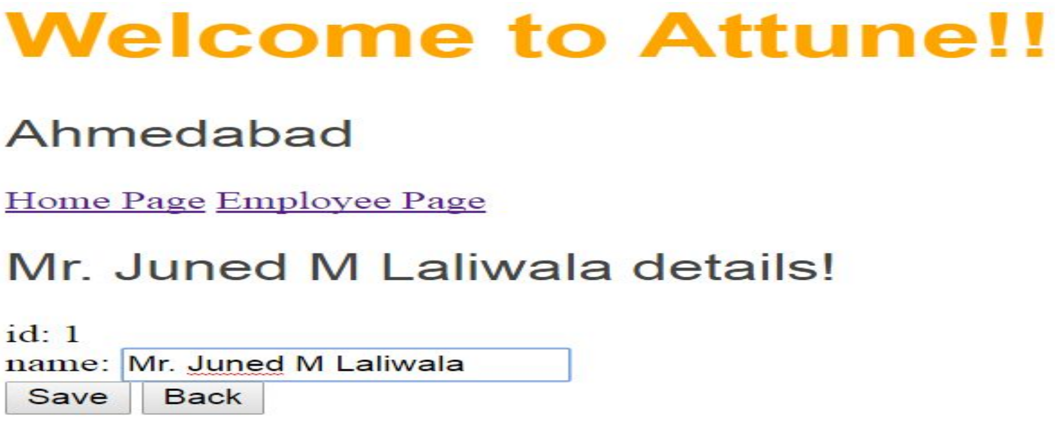


```
private headers = new Headers({'Content-Type': 'application/json'});
  constructor(private http: Http) { }

  getEmployees(): Promise<Employee[]> {
    return this.http.get(this.employeesUrl)
               .toPromise()
               .then(response => response.json().data as Employee[])
               .catch(this.handleError);
  }

getEmployee(id:number):Promise<Employee> {
    return this.getEmployees()
                .then(employees => employees.find(emp =>emp.id === id));
}


update(emp: Employee): Promise<Employee> {
  const url = `${this.employeesUrl}/${emp.id}`;
  return this.http
    .put(url, JSON.stringify(emp), {headers: this.headers})
    .toPromise()
    .then(() => emp)
    .catch(this.handleError);
}
```

Fig: Update Function in Employee Service File

As we can see from the above figure that we have updated the employee details for the particular Employee. So when the user clicks on the save button then updated details will be saved onto the main page as:

ATTUNE


Welcome to Attune!!

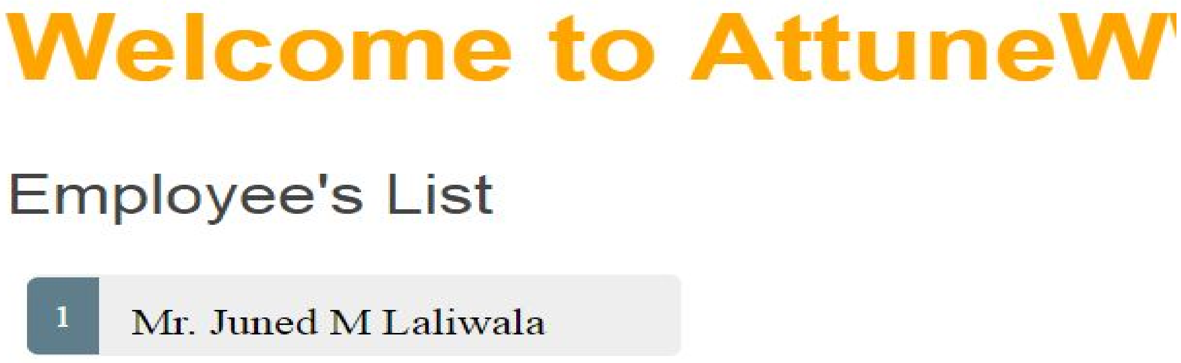
Ahmedabad

Home Page Employee Page

Mr. Juned M Laliwala details!

id: 1
name: Mr. Juned M Laliwala

Fig: Changing the Employee Details


Welcome to AttuneW

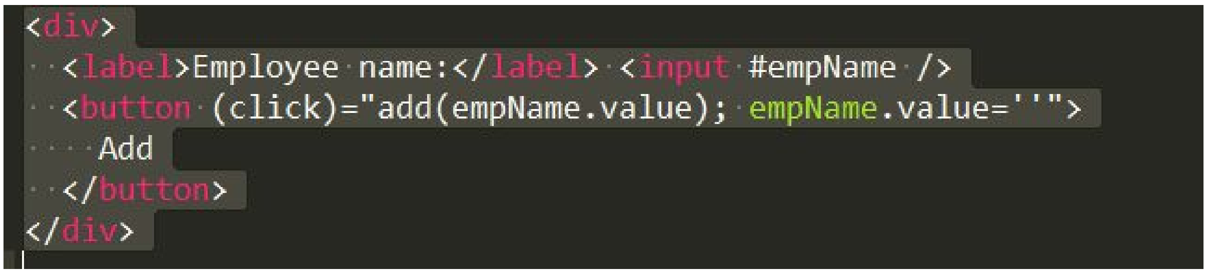
Employee's List

1 Mr. Juned M Laliwala

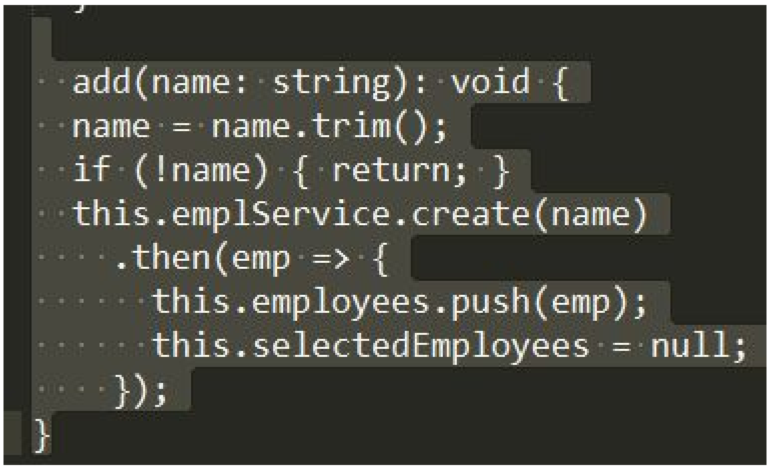
Fig: Corresponding Output of Employee

Adding new Employee

Now in this content, we are going to add the new Employees into our Application. So for that we will make one Add button into our html file. Inside that Add button we will pass one Add Employee Function which will add the Employee details to the Server.

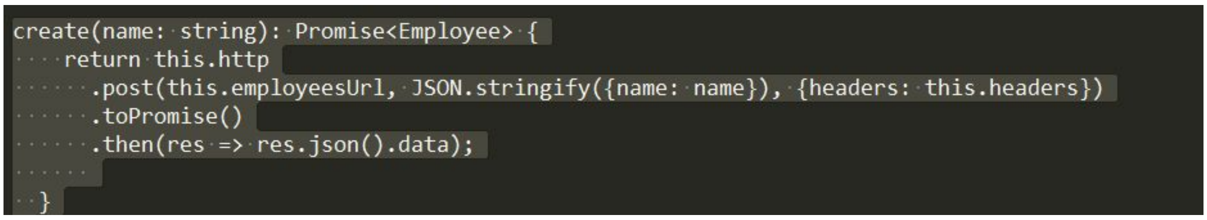


```
<div>
  <label>Employee name:</label> <input #empName />
  <button (click)="add(empName.value); empName.value=''">
    Add
  </button>
</div>
```



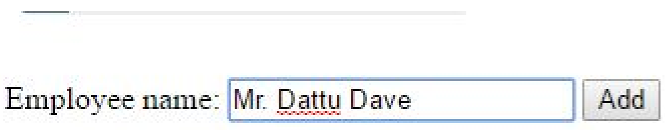
```
  add(name: string): void {
  name = name.trim();
  if (!name) { return; }
  this.emplService.create(name)
      .then(emp => {
        this.employees.push(emp);
        this.selectedEmployees = null;
      });
}
```

Fig: Adding Employee



```
create(name: string): Promise<Employee> {
    return this.http
      .post(this.employeesUrl, JSON.stringify({name: name}), {headers: this.headers})
      .toPromise()
      .then(res => res.json().data);
      
  }
```

Fig: Creating Employee Method



Employee name: Mr. Dattu Dave

Fig: Adding Employee

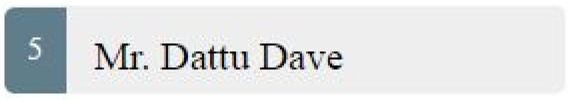
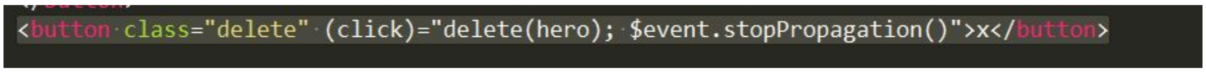


Fig: Added Employee

Deleting Employee

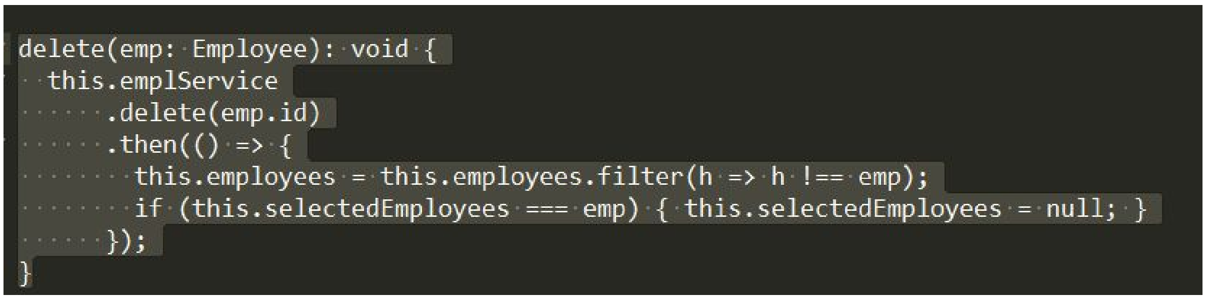
Now we will delete a particular Employee from the list.

Add a Button to delete Employee.



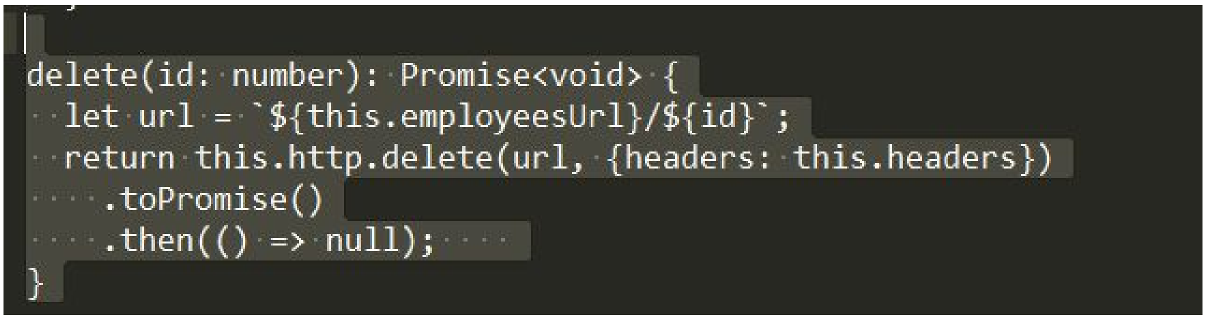
```
<button class="delete" (click)="delete(hero); $event.stopPropagation()">x</button>
```

Apply delete function.



```
delete(emp: Employee): void {
  this.emplService
      .delete(emp.id)
      .then(() => {
        this.employees = this.employees.filter(h => h !== emp);
        if (this.selectedEmployees === emp) { this.selectedEmployees = null; }
      });
}
```

Fig: Delete Employee in Component File



```
delete(id: number): Promise<void> {
  let url = `${this.employeesUrl}/${id}`;
  return this.http.delete(url, {headers: this.headers})
    .toPromise()
    .then(() => null);
}
```

Fig: Service File

So far we have seen the basic use of HTTP in Angular 2 Application. Similarly, we can make any desired application or any of the functionalities that we can work with.

11.

Summary

This was the basic concept of the Angular 2 Application. With the help of this tutorial we can get the basic idea about the usage of the Modules, Templates and Components. In this tutorial we have almost seen many examples with which we can work with. Angular 2 is the very simple and easy to code technology. We also understood that with the help of typescript we can generate javascript files automatically. So this was just the basic concept of the Angular 2. In the next section, we will come back with more extra functionalities of the Angular 2 Applications. Thank You.