# ANGULAR BASICS

# What is Angular (Angular )?

- Next version of most successful AngularJS 1.x
- Finally released on **14th Sep, 2016**. It is called Angular.
- It has been optimized for developer productivity, small payload size, and performance.
- Developed using TypeScript, which is Microsoft's extension of JavaScript that allows use of all ES 2015 (ECMAScript 6) features and adds type checking and object-oriented features like interfaces.
- You can write code in either JavaScript or TypeScript or Dart.
- Designed for Web, Mobile and Desktop Apps.
- Not an upgrade of Angular 1. It was completely rewritten from scratch.

# Differences between AngularJS and Angular

- Called as AngularJS 1.x and Angular.
- Components are used instead of Controllers and $scope. A component is a class with its own data and methods.
- Option to write code in different languages.
- Designed for Speed. Supposed to be 5 times faster than Angular 1.
- Designed for Mobile development also.
- More modular. It is broken into many packages.
- Data binding is done with no new directives. We bind to attributes of html elements.
- Event handling is done with DOM events and not directives.
- Simpler API

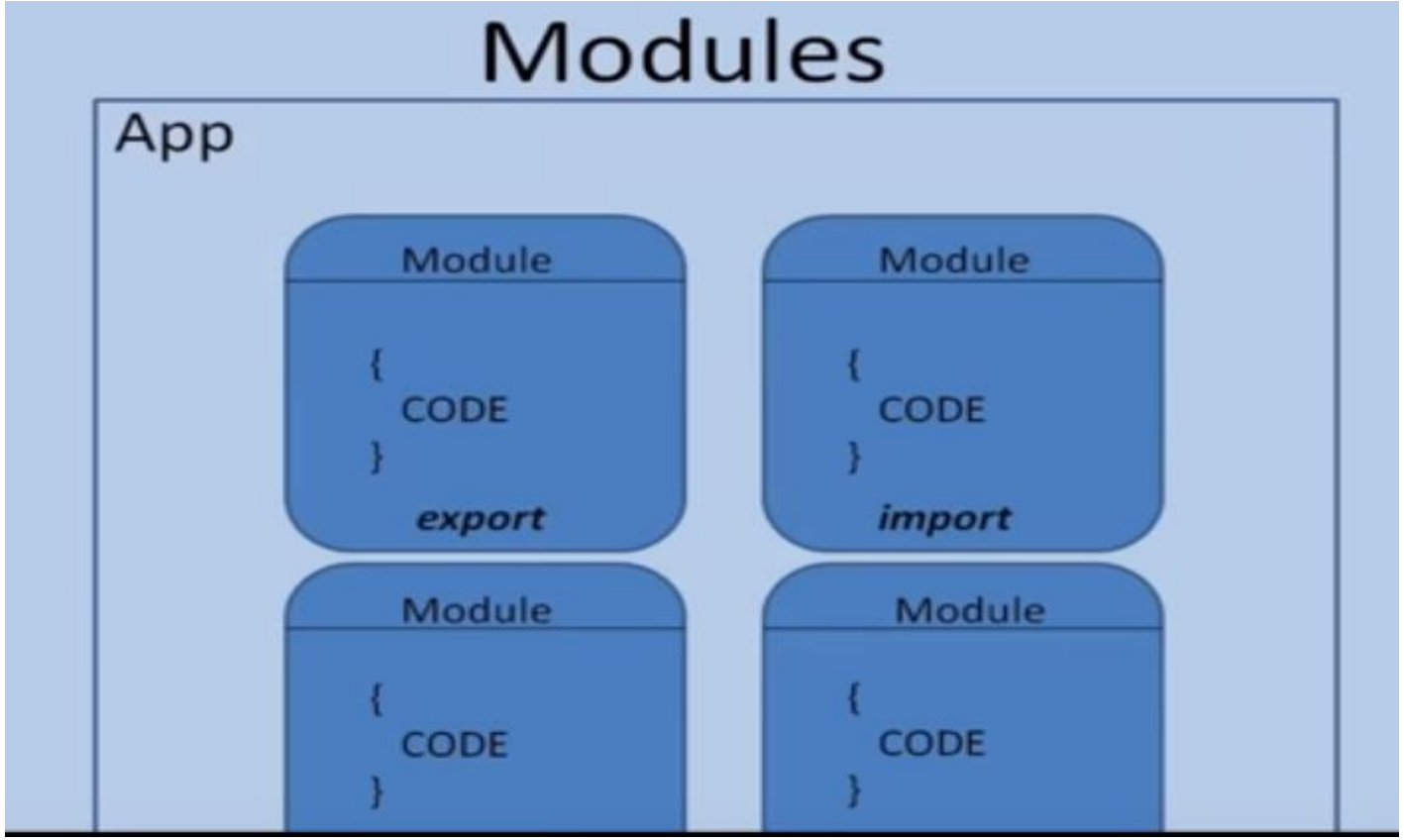
# Building Blocks

The following are important components of an Angular application.

1. Modules
2. Components
3. Templates
4. Metadata
5. Data binding
6. Directives
7. Services
8. Dependency injection

# Module

- Angular application is a collection of many individual modules.
- It contains code that can be export to another module or can be imported by other modules
- Angular framework is a collection of modules



# Module

- A module is a class that is decorated with @NgModule decorator
- Every application contains at least one module called root module, conventionally called as AppModule.
- NgModule decorator provides information about module using properties listed below:
- **Declaration** – classes that belong to this module. They may be components, directives and pipes.
- **Exports** – The subset of declarations that should be visible to other modules.
- **Imports** – Specifies modules whose exported classes are needed in this module.
- **Bootstrap** – Specifies the main application view – root component. It is the base for the rest of the application.
- Providers –species services

# Module Example

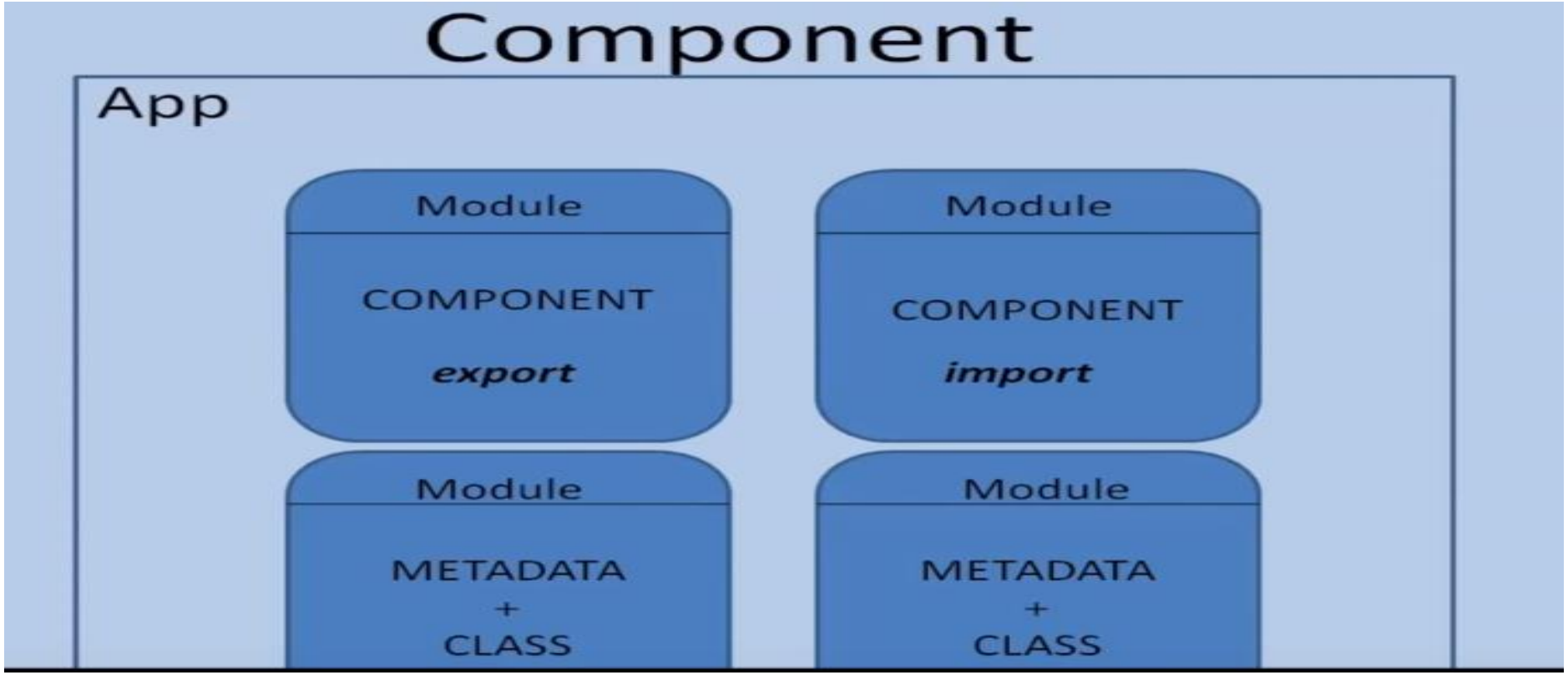
- The following code shows how to create a simple module:

  **AppModule.ts**

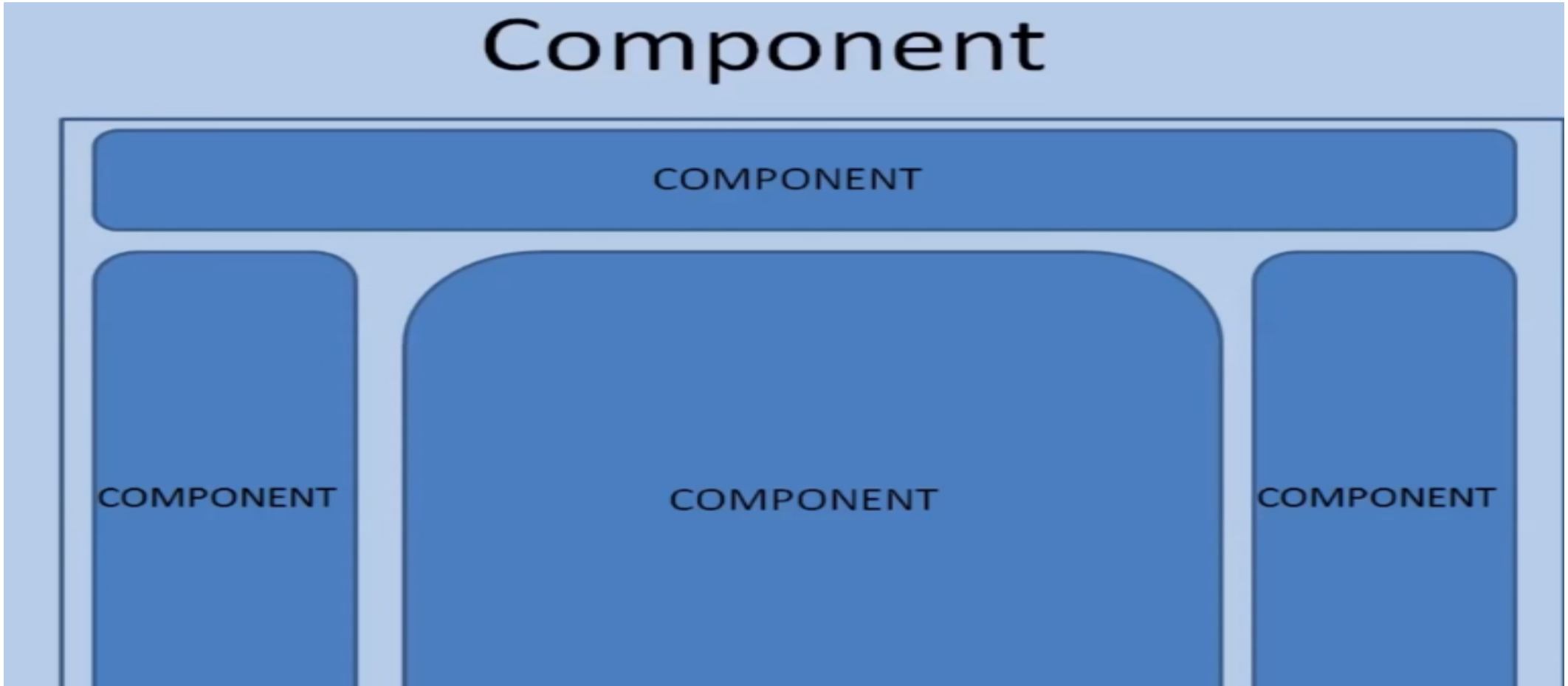
```
import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-
browser';
import { FirstComponent } from './first.component';
@NgModule({
imports: [ BrowserModule],
declarations: [ FirstComponent],
bootstrap: [ FirstComponent ]
Providers:[]
})
export class AppModule { }
```

# Component

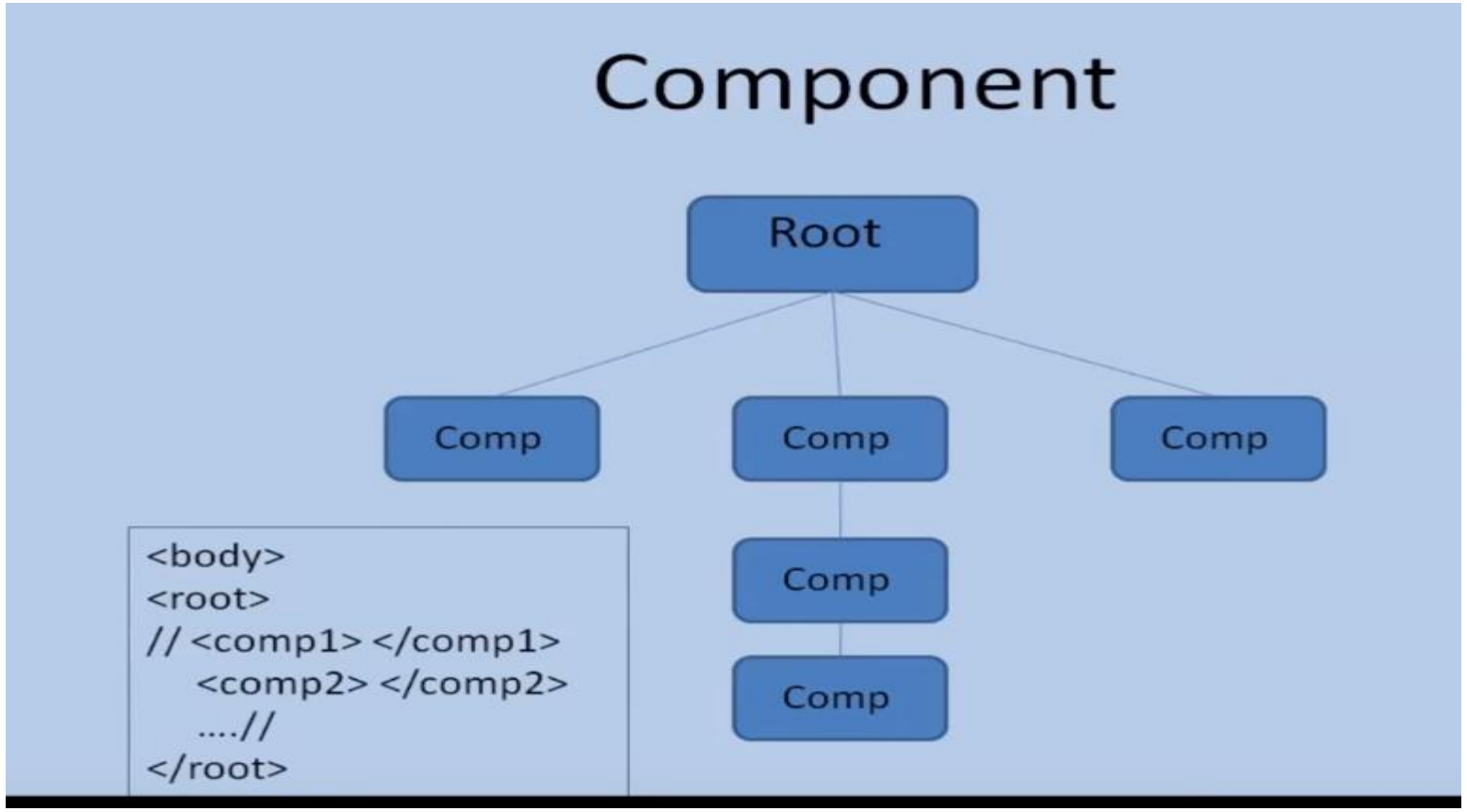
- A component controls a part of the screen called view.
- Every component is a class with its own data and code.
- A component may depend on services that are injected using dependency injection.
- The template, metadata, and component together describe a view.
- Components are decorated with @Component decorator through which we specify **template/templateUrl** and **selector** (tag) related to component,.
- Properties like styles/styleUrls and **providers** can also be used.


Component
App
Module
COMPONENT
export
Module
COMPONENT
import
Module
METADATA
+
CLASS
Module
METADATA
+
CLASS

- E.g if we have a page that contains navigation bar, leftside bar, right side bar, main contents
- Each portion is represented using component



- There is at least one component which is root component and other components are child components of it



- E.g

- **FirstComponent.ts**
- import { Component } from '@angular/core';
- @Component({
- selector: 'my-first’,  // tag to be used in view
- templateUrl : './first.component.html'
- })
- export class FirstComponent {
- title : string = “KLFS Solutions";  //model
- }

- **template/templateUrl**
- You define a component's view with its companion template.

- A template is a form of HTML that tells Angular how to render the component.

- **Metadata**
- Metadata provided using decorators inform Angular how to process a class.

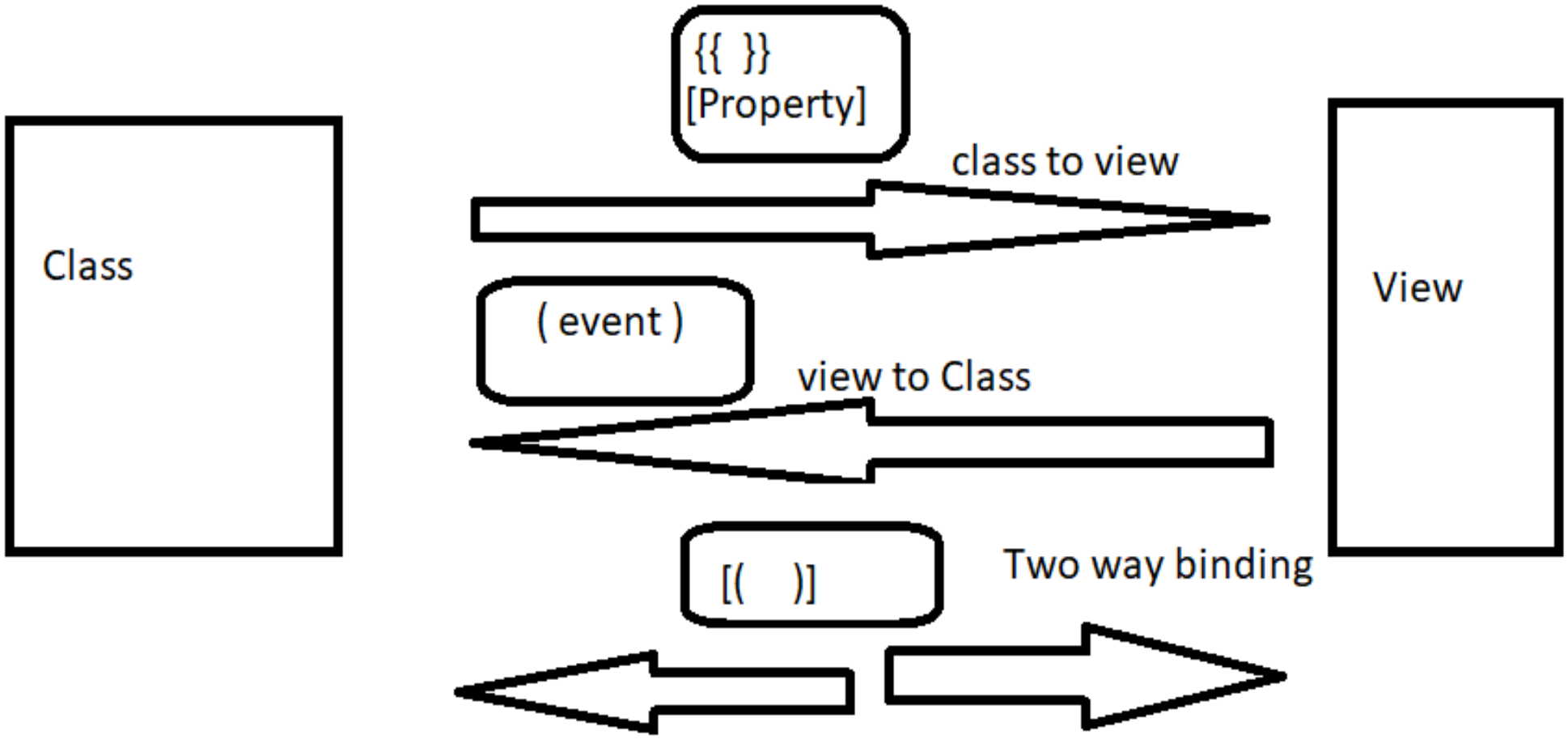
- For example, @Component decorator tells Angular

  - to treat a class as a component and

  - also provides additional information through attributes of decorator (like selector, template etc.)

- **Data Binding**
- Data from objects should be bound to HTML elements and vice-versa, known as data binding.
- Angular takes care of data binding.
- Enclosing property (attribute of HTML element) in [] copies value to property.
- Enclosing event in parentheses () will assign event handler to event.
- Interpolation using {{ }} allows value of an expression to be used in HTML
- The ngModel is used to for two way data binding.

# Data binding



# Directives

- A directive transforms DOM according to instructions given.
- Components are also directives.
- Directives are two types - structural directives and attribute directives.
- Structural directives alter layout by adding, removing, and replacing elements in DOM.
- Attribute directives alter the appearance or behavior of an existing element.
- In templates they look like regular HTML attributes.
- *ngFor and *ngIf are structural directives.
- ngModel and ngClass are attribute directives.

- Components also defines html elements but it is not inside other elements
- But attribute directives are inside other html element
- Directive is also metadata+ class

```
<ul>
     <li *ngFor="let p of perarr">{{p.perid}}-------{{p.pname}}</li>
</ul>
```

# Structural directive

- 
- <p *ngIf=”flagvar”>This is para</p>
- 
- *ngIf ---- if the assigned variable is true then the element will  be displayed otherwise element will be hidden.
- 
- *ngIf = “flagvar; else noproduct”
- <ng-template #noproduct>
- This is else section of ngif
- </ng-template>
- 
- <ul>
- <li *ngFor=”let prod of Productarray”></li>
- </ul>
-

# Attribute directive

- <h1 [ngStyle]="{backgroundColor:getcolor()}">{{ 'Product name: ' }}testing.....{{name}}  {{price}} function ...{{getProductName()}}  {{ productstatus }} </h1>
- color should be red

- <h1 [ngClass]={classname:<condition>}></h1>

- If condition is true classname will be assigned to class attribute otherwise class attribute will not be added.

# Pipes

- {{ name |uppercase }} ---
- {{ name |lowercase }}
- {{ name|slice:’2’:’4’ }} ------ excludes index 4
- {{ name | replace:’the’:’hello’  }}

- {{ 8.567: number:1.2-3}  -----before decimal minimum one number
- ------after decimal min 2 numbers or maximum 3 numbers
- {{ 8.567|number:2.2-2}  ---o/p is 08.57 it will round the number because max
- --------- digits after decimal is 2
- {{8.567| currency : ‘Euro’ }}  -------o/p will be in Euro
-
- {{8.567| currency : ‘USD’ }}  USD8.567
- {{8.567| currency : ‘USD’:true }}  ------$8.567  :true indicates show the symbol
- {{8.567| currency : ‘GBP’:true }}  -----great Brittan pounds
- {{ server_date|date:”fullDate”}

Ng4CompleteGuide | API Reference - ts - API

Secure https://angular.io/docs/ts/latest/api/#!?query=pipe

ANGULAR

FEATURES DOCS EVENTS NEWS

API Reference (v2.4.7)

pipe

DOCS HOME

CORE DOCUMENTATION

QUICKSTART

CLI QUICKSTART

GUIDE

API REFERENCE

ADDITIONAL DOCUMENTATION

TUTORIAL

ADVANCED

TYPE: ALL

STATUS: ALL

pipe

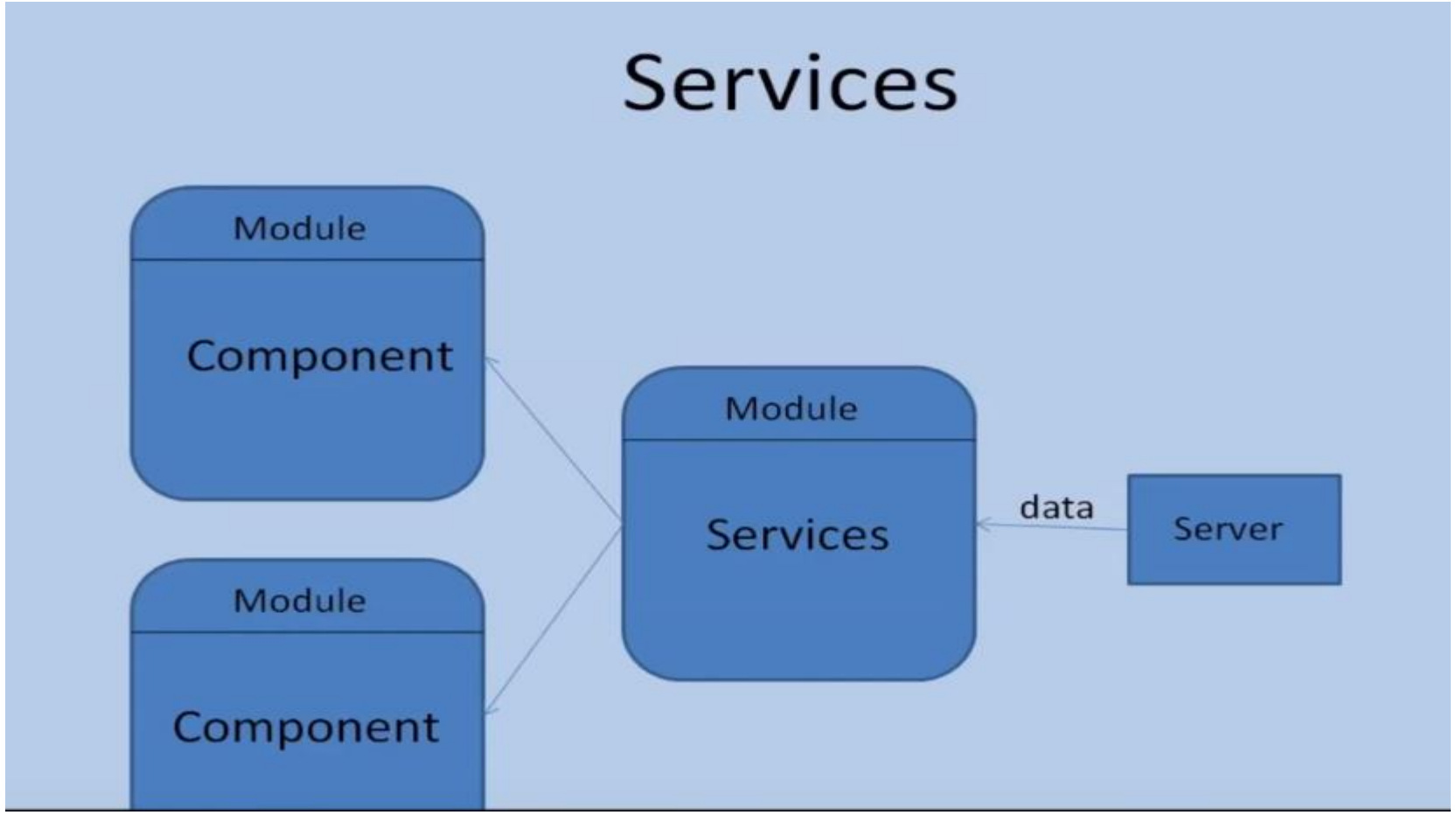
## @angular/common

- AsyncPipe
- CurrencyPipe
- DatePipe
- DecimalPipe
- I18nPluralPipe
- I18nSelectPipe
- JsonPipe
- LowerCasePipe
- PercentPipe
- SlicePipe
- UpperCasePipe

# Services

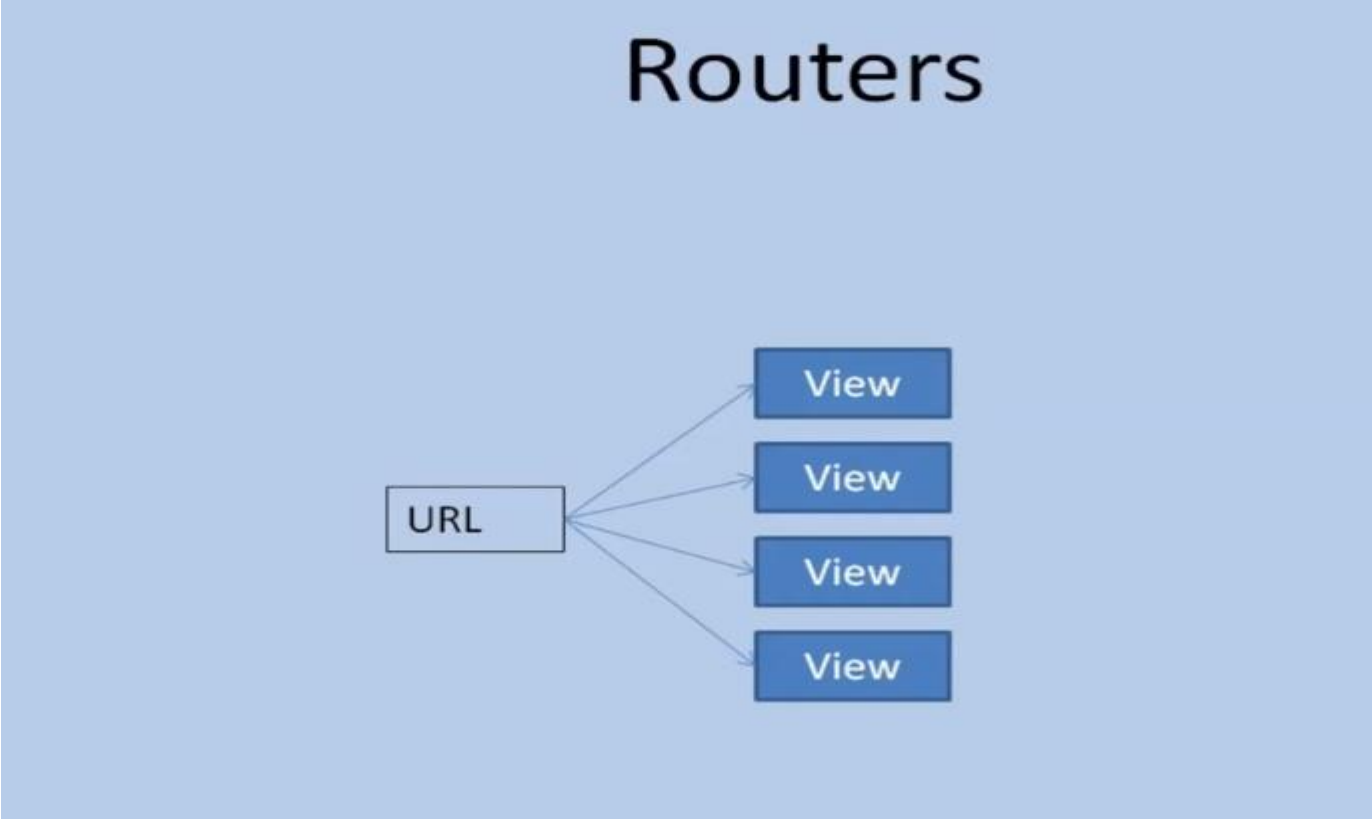
- A service encompasses any functionality.
- A service is a class with a specific purpose(functionality) can be used by multiple components.
- Components consume services.
- Services are injected into Component that use them.
- **Dependency Injection**
- Dependency injection is a way to supply a new instance of a class with the fully-formed dependencies it requires.
- Components get access services they need through dependency injection.
- An injector contains instances of services. It injects them into other services and components as and where they are needed.

# Services


Services
Module
Component
Module
Services
data
Server
Module
Component

# Router

- It decides which view should appear based on URL



# Angular setup

- It is important to setup development environment for Angular in your system. Here are the steps to take up:
- **Install Node.js and NPM**
- Install Node.js and NPM. Node.js is used run JavaScript on server and provide environment for build tools.
- NPM is used to manager packages related to JavaScript libraries. NPM itself is a Node application.
- To install, go to https://nodejs.org/en/download and installer (.msi) for 32-bit or 64-bit. Do the same for other platforms like Mac etc.
- Run .MSI file to install Node.js into your system. It is typically installed into c:\Program Files\nodejs folder.

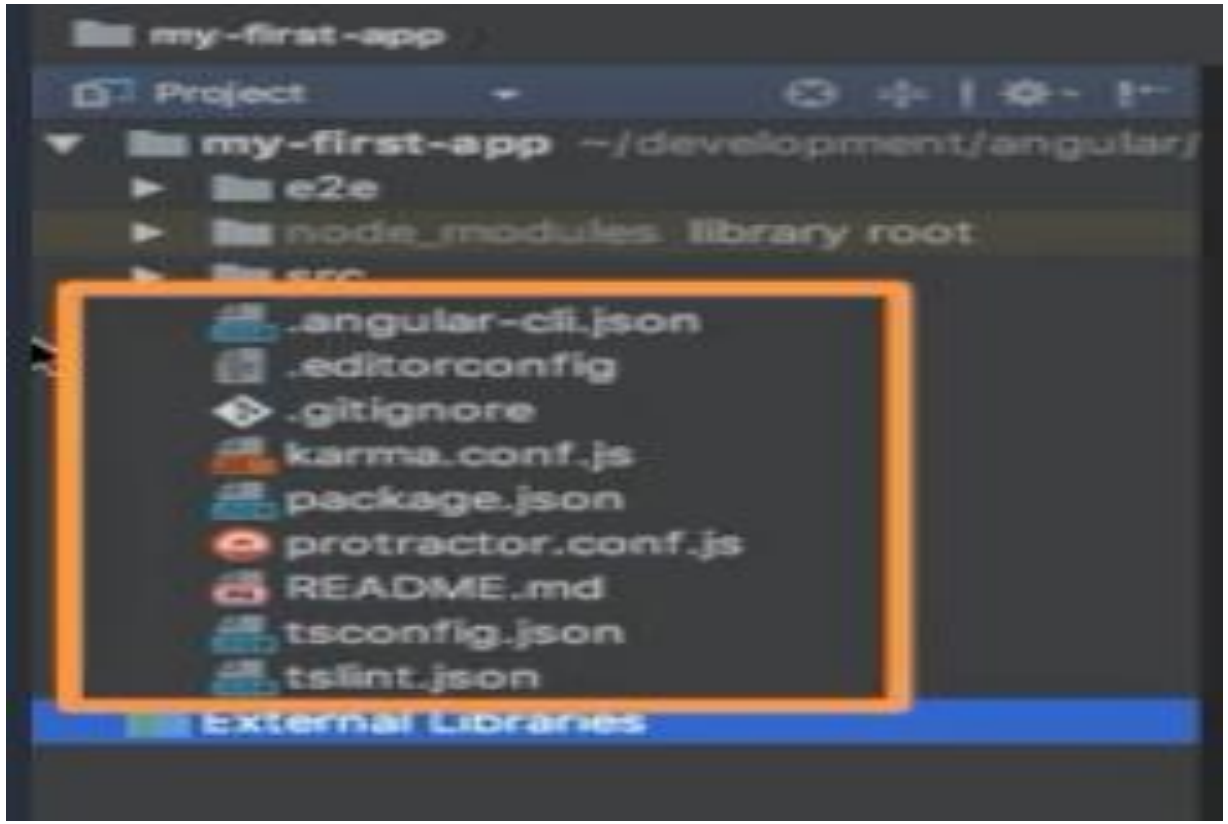
# Steps to create new application

- Change the folder in which you want to store the project and then use following command

- C:\mydata\myangular>ng new myfirstangulardemo
- It will download and install all the required frameworks e.g angular framework and give you a template for the project. It will take some time
- 4. Change the folder to myfirstangulardemo
- 5. To run the project use command
- ng serve ---------- this will compile typescript code and build javascript code and will run index .html file on default port 4200 if you want to change port then use command

- ng serve –port <new port number>

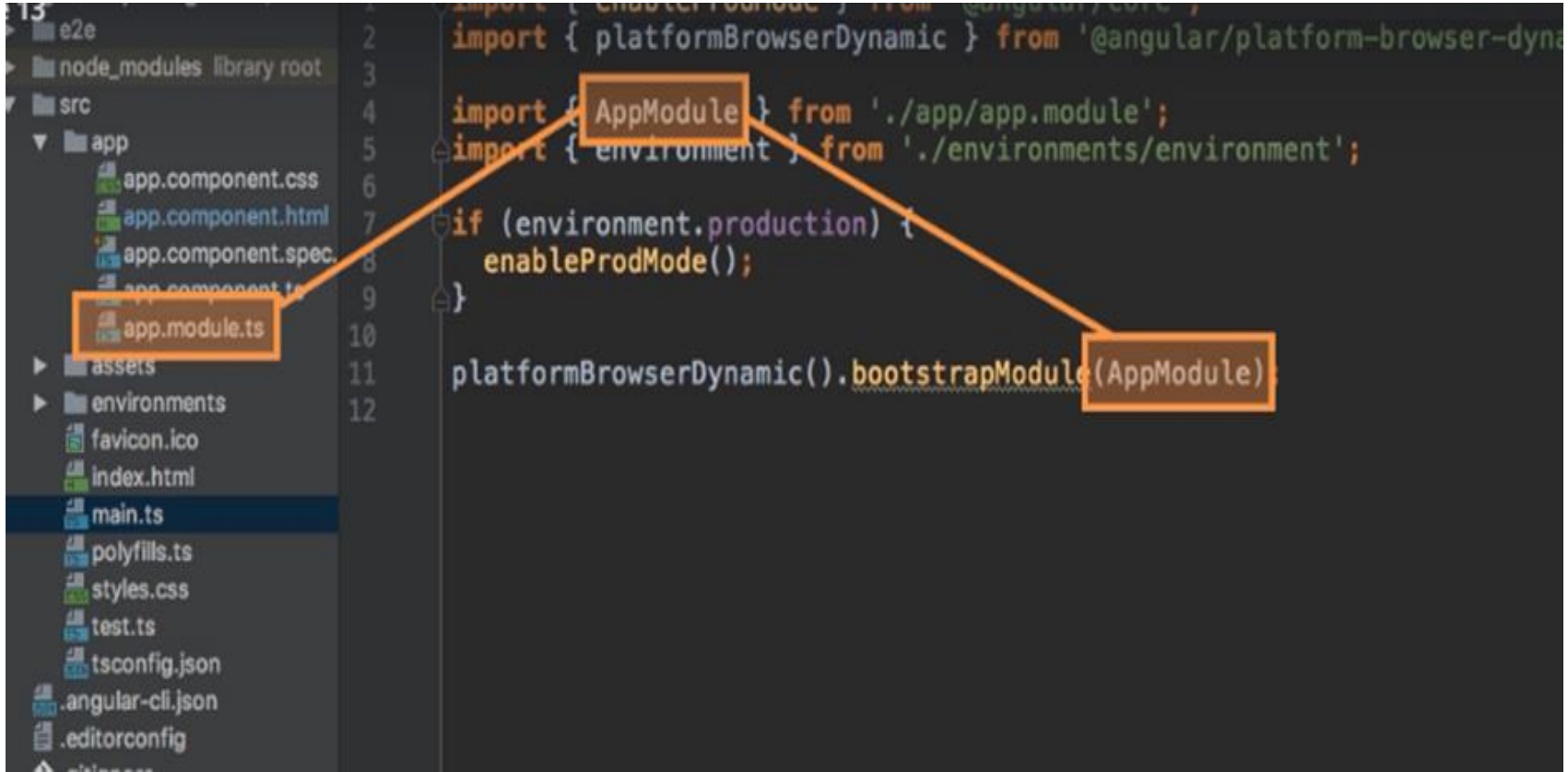
- 6. To see the output open browser and use link localhost:4200
- 7. Lets open the code using some editor
- 8. You may use any editor like sublime, Atom, visual studio code, bracket etc
- 9. In folder structure the files which are outside src folder are used for configuring the project


my-first-app
Project
my-first-app ~/development/angular/
e2e
node_modules library root
.angular-cli.json
.editorconfig
.gitignore
karma.conf.js
package.json
protractor.conf.js
README.md
tsconfig.json
tslint.json
External Libraries

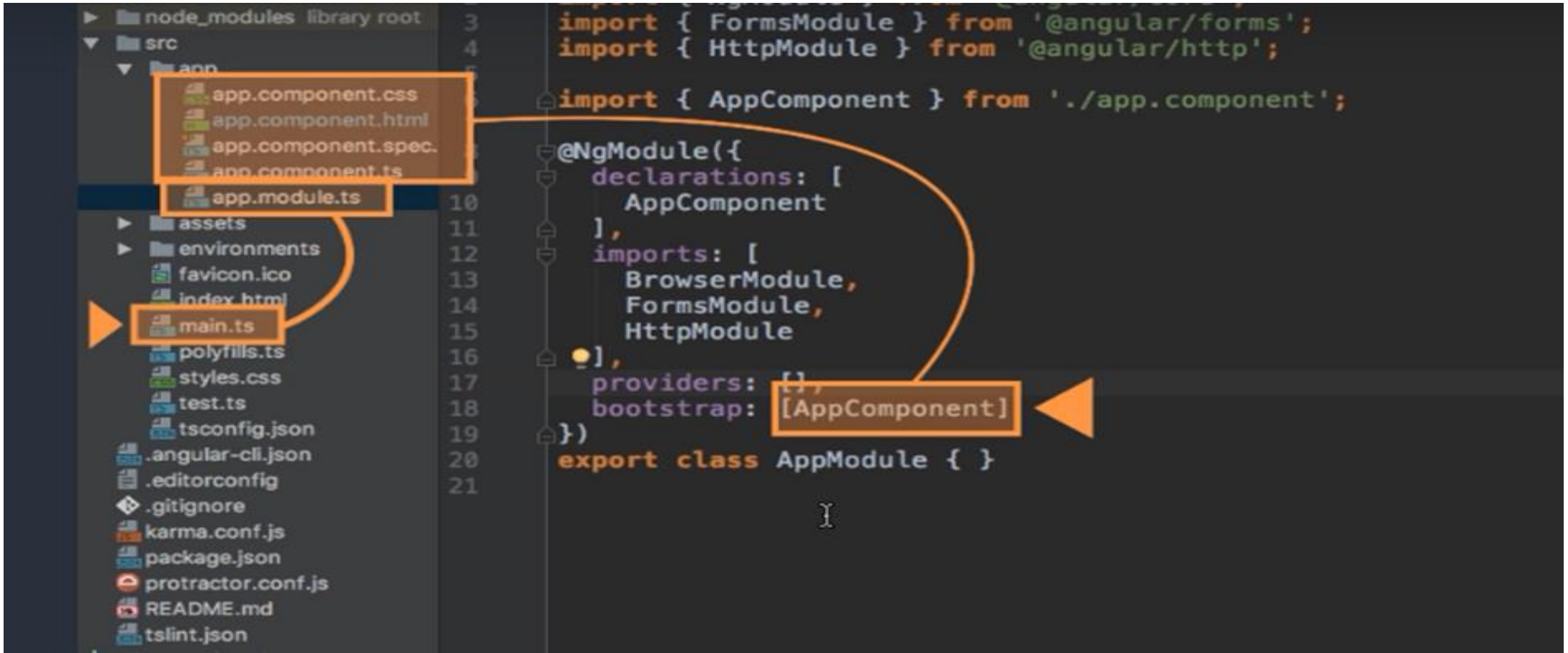
- 10. E2e file is used for testing end to end project
- 11. And the src folder contains actual code.
- 12. In src folder there is app folder in which the code is there
- 13. App.component.html is your display logic and app.component.ts is your business logic
- 14. The properties that you want to show in html page should be bound with properties in the class
- 15. App.module.ts is used to unlock features that you use in your angular application e.g if you use ngmodule then to use it we need to add FormsModule in app.module.ts
- 16. It uses typescript
- 17. For formatting you can use bootstrap we are using bootstrap version 3,to install it and configure it in angular-cli.json file or in angular.json file
- 18. In the file you will find an array named style that contains file name src/style.css which can be used for adding your styles globally for your project
- 19. You can install bootstrap by using following command
- npm install –save bootstrap@3

- 20. And then add following entry in angular-cli.json file
-

Lets open app.module.ts file in which you will find array bootstrap. It contains list of all components those should be known to angular at the time it analyses index.html page



Lets open app.module.ts file in which you will find array bootstrap. It contains list of all components those should be known to angular at the time it analyses index.html page



# OTHER CONFIGURATION FILE

# typings.json

EXPLORER

- WORKING FILES
- TEMPLATE
  - app
    - app.component.js
    - app.component.ts
    - app.component.js.m...
    - main.js
    - main.ts
    - main.js.map
  - node_modules
  - typings
  - index.html
  - package.json
  - style.css
  - systemjs.config.js
  - tsconfig.json
  - typings.json

typings.json

```
{
  "dependencies": {
    "zone.js": "github:gdi2290/typed-zone.js#66ea8a3451542bb7798369306840e46be1d6ec89"
  },
  "devDependencies": {},
  "ambientDependencies": {
    "angular-protractor": "github:DefinitelyTyped/DefinitelyTyped/angular-protractor/angular-protrac
    "core-js": "registry:dt/core-js#0.0.0+20160317120654",
    "hammerjs": "github:DefinitelyTyped/DefinitelyTyped/hammerjs/hammerjs.d.ts#74a4dfc1bc2dfadec47b8
    "jasmine": "github:DefinitelyTyped/DefinitelyTyped/jasmine/jasmine.d.ts#4b36b94d5910aa8a4d20bdcc
    "node": "github:DefinitelyTyped/DefinitelyTyped/node/node.d.ts#8cf8164641be73e8f1e652c2a5b967c72
    "selenium-webdriver": "github:DefinitelyTyped/DefinitelyTyped/selenium-webdriver/selenium-webdri
    "webpack": "github:DefinitelyTyped/DefinitelyTyped/webpack/webpack.d.ts#95c02169ba8fa58ac1092422
  }
}
```

- We need one more file to tell javascript compiler about typescript code
- So the tsconfig.json is there

# Systemjs.config.ts

- This file helps in loding of the modules and stores details about default extension

EXPLORER
- WORKING FILES
- TEMPLATE
  - app
    - app.component.js
    - app.component.ts
    - app.component.js.m...
    - main.js
    - main.ts
    - main.js.map
  - node_modules
  - typings
  - index.html
  - package.json
  - style.css
  - systemjs.config.js
  - tsconfig.json
  - typings.json

systemjs.config.js

```
(function(global) {

  // map tells the System loader where to look for things
  var map = {
    'app':                        'app', // 'dist',
    'rxjs':                       'node_modules/rxjs',
    'angular2-in-memory-web-api': 'node_modules/angular2-in-memory-web-api',
    '@angular':                   'node_modules/@angular'
  };

  // packages tells the System loader how to load when no filename and/or no extension
  var packages = {
    'app':                        { main: 'main.js',  defaultExtension: 'js' },
    'rxjs':                       { defaultExtension: 'js' },
    'angular2-in-memory-web-api': { defaultExtension: 'js' },
  };

  var packageNames = [
    '@angular/common',
    '@angular/compiler',
    '@angular/core',
    '@angular/http',
    '@angular/platform-browser',
    '@angular/platform-browser-dynamic',
    '@angular/router',
    '@angular/router-deprecated',
    '@angular/testing',
    '@angular/upgrade'
```

# Index.html

- This file can be divided into  3 parts

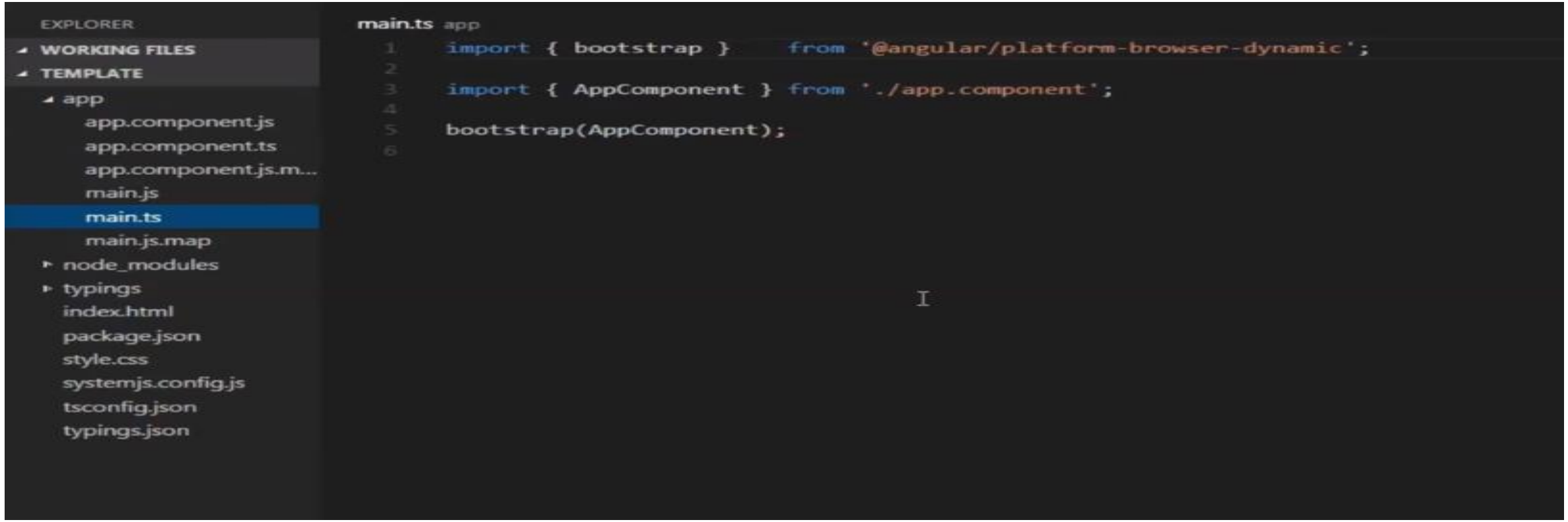
EXPLORER
- WORKING FILES
- TEMPLATE
  - app
    - app.component.js
    - app.component.ts
    - app.component.js.m...
    - main.js
    - main.ts
    - main.js.map
  - node_modules
  - typings
  - index.html
  - package.json
  - style.css
  - systemjs.config.js
  - tsconfig.json
  - typings.json

index.html

```html
<html>
  <head>
    <title>Angular 2 QuickStart</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/es6-shim/es6-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>
  </head>

  <!-- 3. Display the application -->
  <body>
    <my-app>Loading...</my-app>
  </body>
</html>
```

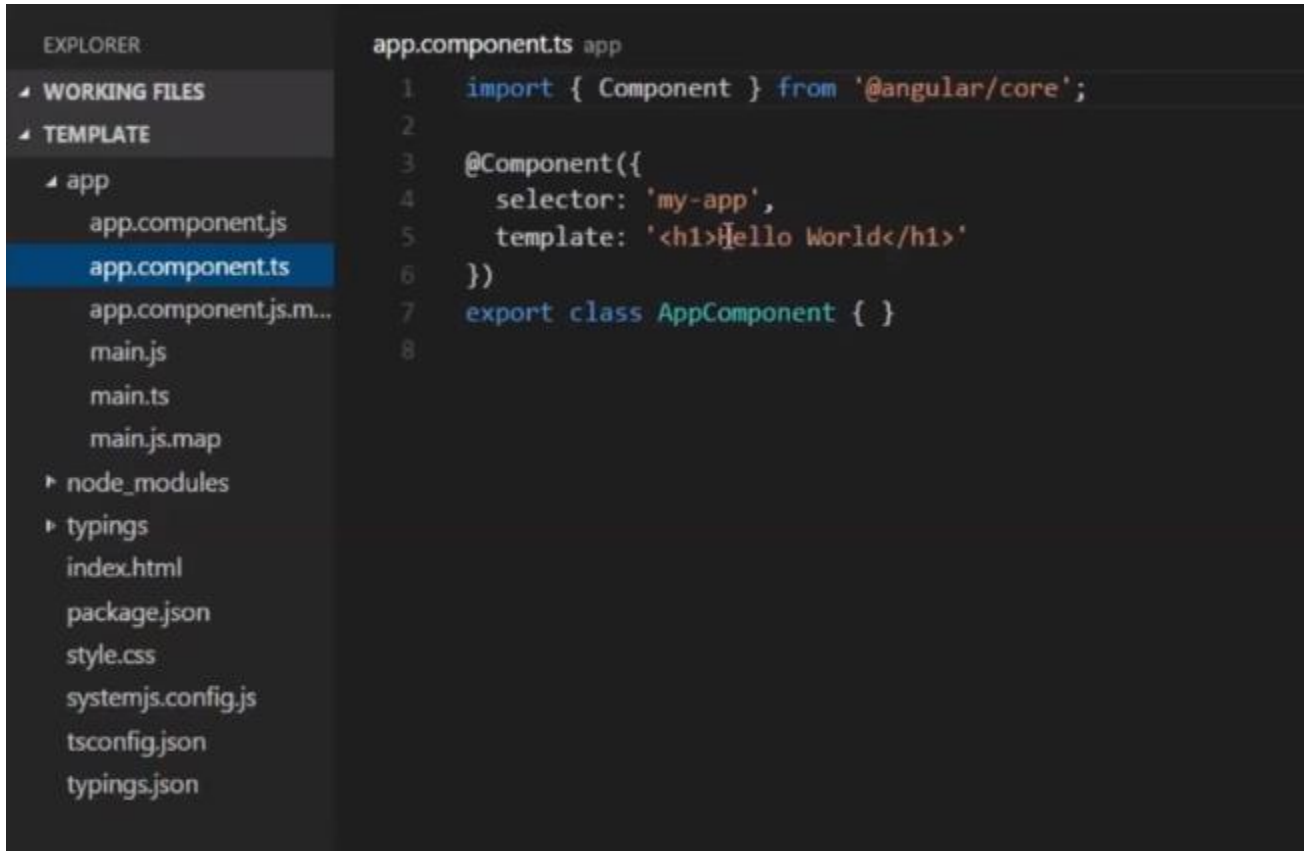
# App folder

- This folder contains all files required for our application
- It contains the files with extension .js and .ts but ignore .js files
- These are generated file. .ts file we will write and modify.



# App.component.ts

- This is our root component.
- All other components will be included here



# Run the code

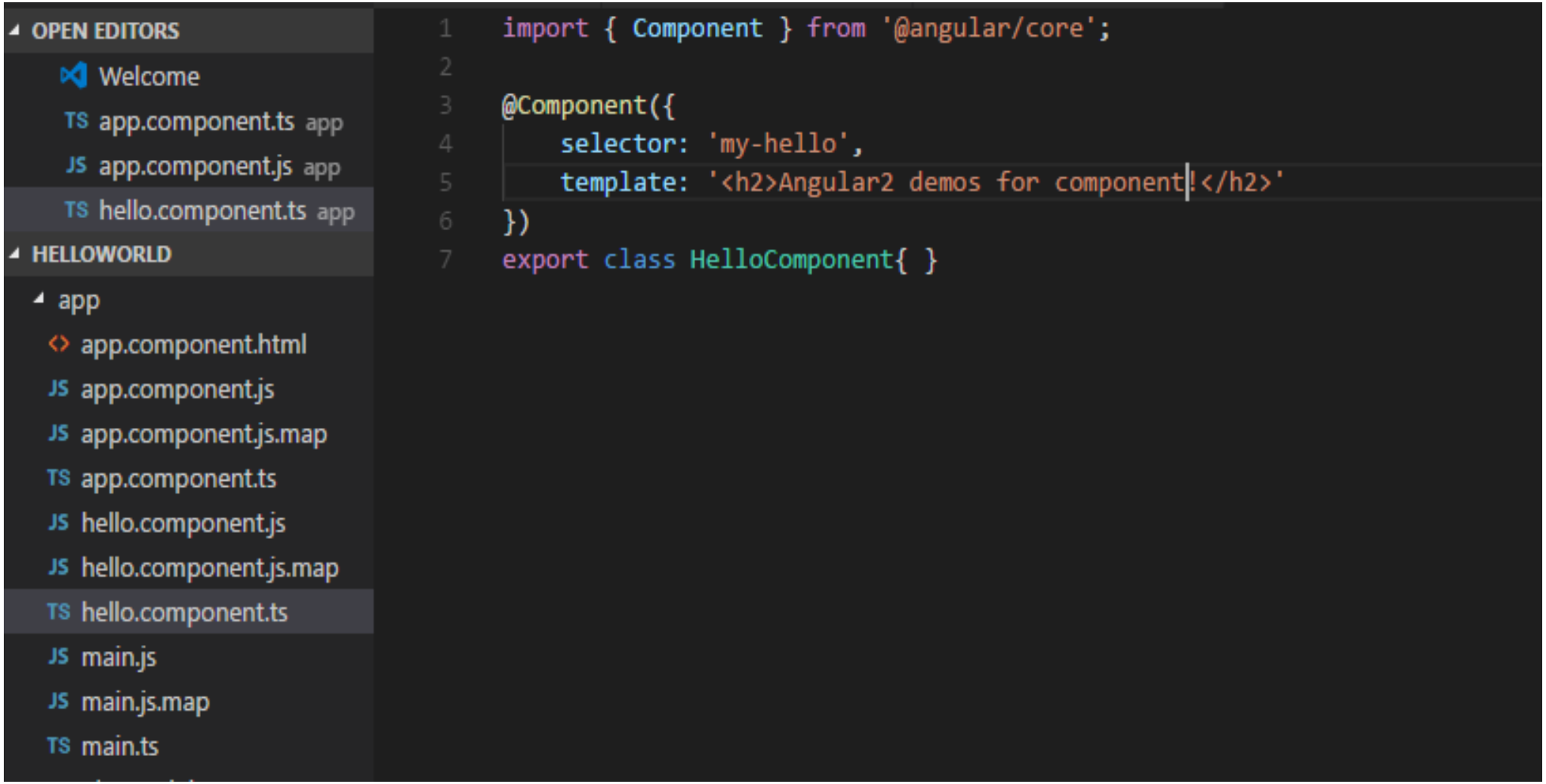
- To run the code
- Change the folder to demo

```
npm start
```

Or

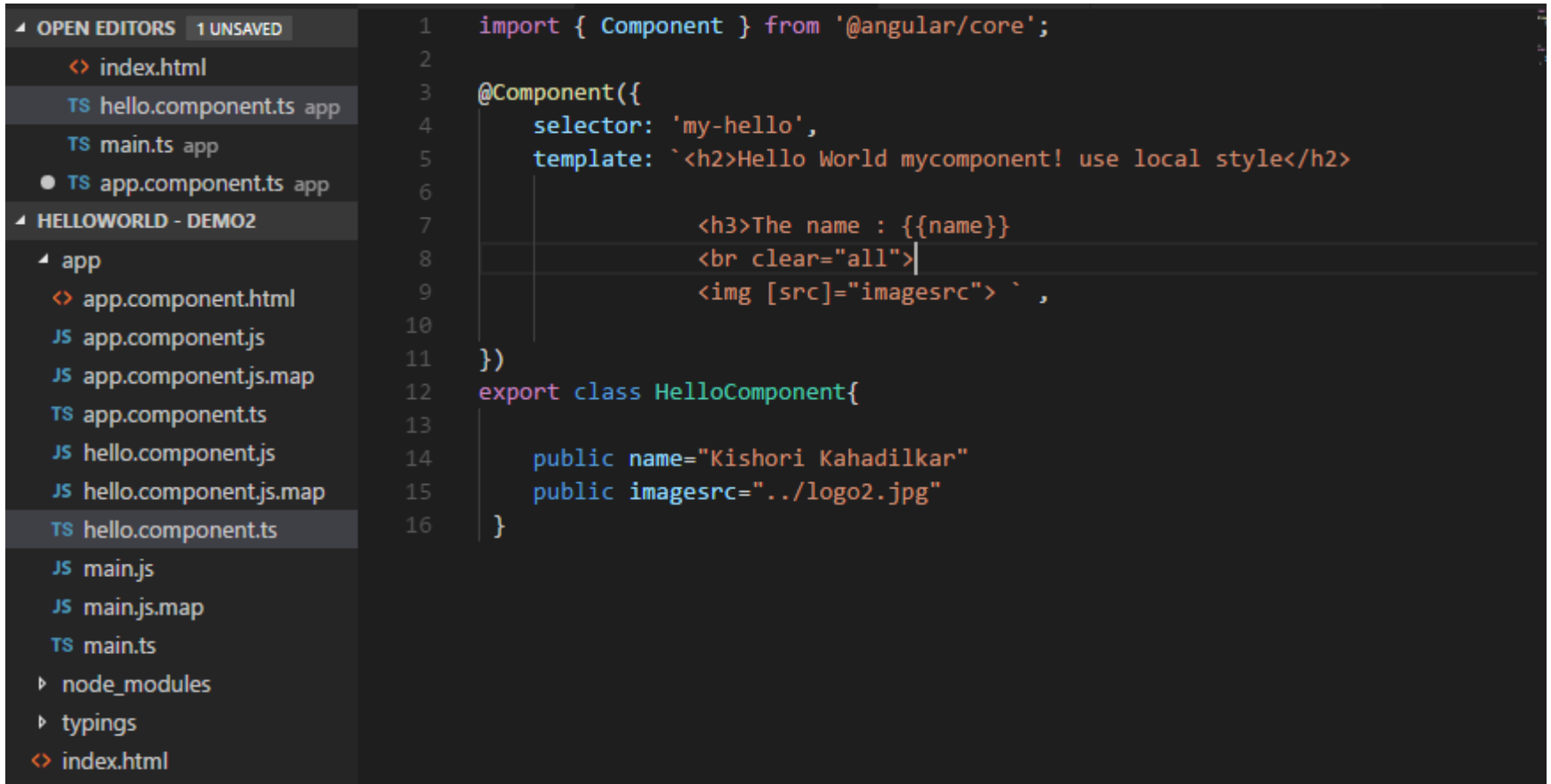
- ng serve

# Add new component



# Interpolation

- Intepolation can be done by using {{ name }}
- Or by using [ ]  example src attribute of img tag  // note : don’t add end tag



# Difference between property and Attribute

- The value of property can be changed
- But the value of attribute cannot be changed

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

@Component({
    selector: 'my-hello',
    template: `<h2>Hello World mycomponent! use local style</h2>

                <h3>The name : {{name}}
                <br clear="all">
                <img [src]="imagesrc">
                <input type="text" value="angular  test">` ,

})
export class HelloComponent{

    public name="Kishori Kahadilkar"
    public imagesrc="../logo2.jpg"
 }
```

- In the given code src is property and not attribute. Mostly there is one to one mapping in property and attributes. But there is difference in both

```
@Component({
    selector: 'my-hello',
    template: `<h2>Hello World mycomponent! use local style</h2>

                <h3>The name : {{name}}
                <br clear="all">
                <img [src]="imagesrc">
                <input type="text" value="angular  test">` ,

})
export class HelloComponent{

    public name="Kishori Kahadilkar"
    public imagesrc="../logo2.jpg"
 }
```

# Event Handling

- Click event handled in the code below

TS hello.component.ts ✕ | TS main.ts | TS app.component.ts ●

```
1   import { Component } from '@angular/core';
2
3   @Component({
4       selector: 'my-hello',
5       template: `<h2>Hello World mycomponent! use local style</h2>
6
7                   <h3>The name : {{name}}
8                   <button type="button" (click)="MyClick()">click me</button>
9                   ` ,
10
11  })
12  export class HelloComponent{
13
14      public name="Kishori Kahadilkar";
15      public imagesrc="../logo2.jpg";
16      public MyClick(){
17          alert("Button clicked");
18      }
19   }
```

# Using references

- #mytext is a reference to text box value can be used in event

hello.component.ts | TS main.ts | TS app.component.ts

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

@Component({
    selector: 'my-hello',
    template: `<h2>Hello World mycomponent! use local

                <h3>The name : {{name}}
                <button type="button" (click)                    lick me
                <input type="text" #mytext>` ,

})
export class HelloComponent{

    public name="Kishori Kahadilkar";
    public imagesrc="../logo2.jpg";
    public MyClick(myval){
        alert("Button clicked"+myval);
    }
}
```

#mytext is a refernce to text box

# To refer event

- <button type="button" (click)="MyClick($event)">click me</button>
- Use $ event to refer event

# Attribute directive

- ngClass

```
@Component({
    selector: 'my-tutorials',
    template: `<h2>{{title}}</h2>
              <p [ngClass]="{classOne:cone,classTwo:ctwo}">ngClass paragraph</p>
              <button (click)="toggle()">Toggle</button>`,
    styles: [`.classOne{color:white}
              .classTwo{background-color:black}`]
})
export class TutorialsComponent{
    public title="Tutorials from Joatmon Channel";
    public cone=true;
    public ctwo=true;
    toggle(){
        this.cone=!this.cone;
        this.ctwo=!this.ctwo;
    }
}
```

# Pipe operator

- String

```
@Component({
    selector: 'my-tutorials',
    template: `<h2>{{name}}</h2>
              <h2>{{name | uppercase}}</h2>
              <h2>{{name | lowercase}}</h2>
              <h2>{{name | slice:'2':}}</h2>`
})
export class TutorialsComponent{
```

- Number transformation

- Number:1.2-3 – indicates 1 digit before decimal min 2 after decimal maximum 3 after decimal if number of digits are more rounding of number will be done

```
@Component({
    selector: 'my-tutorials',
    template: `<h2>{{8.567}}</h2>
               <h2>{{8.567 | number:'1.2-3'}}</h2>
               <h2>{{8.567 | number:'2.2-3'}}</h2>
               <h2>{{8.567 | number:'2.4-4'}}</h2>
               <h2>{{8.567 | number:'2.2-2'}}</h2>`
})
export class TutorialsComponent{

}
```

- Currency transformation
- {{8.99 | currency : ‘EUR’}} -----  EUR 8.99
- {{8.99 | currency : ‘EUR’:true}} – Euro symbol will be displayed

```
@Component({
    selector: 'my-tutorials',
    template: `<h2>{{8.567}}</h2>
               <h2>{{8.567 | number:'1.2-3'}}</h2>
               <h2>{{8.567 | number:'2.2-3'}}</h2>
               <h2>{{8.567 | number:'2.4-4'}}</h2>
               <h2>{{8.567 | number:'2.2-2'}}</h2>

               <h2>{{8.99 | currency:'GBP':true}}</h2>`
})
export class TutorialsComponent{

}
```

- Date transformation
- You may use mediumTime

```
@Component({
    selector: 'my-tutorials',
    template: `<h2>{{date}}</h2>
               <h2>{{date | date:'fullDate'}}</h2>
               <h2>{{date | date:'shortDate'}}</h2>
               <h2>{{date | date:'shortTime'}}</h2>`
})
export class TutorialsComponent{
    date = new Date();
}
```

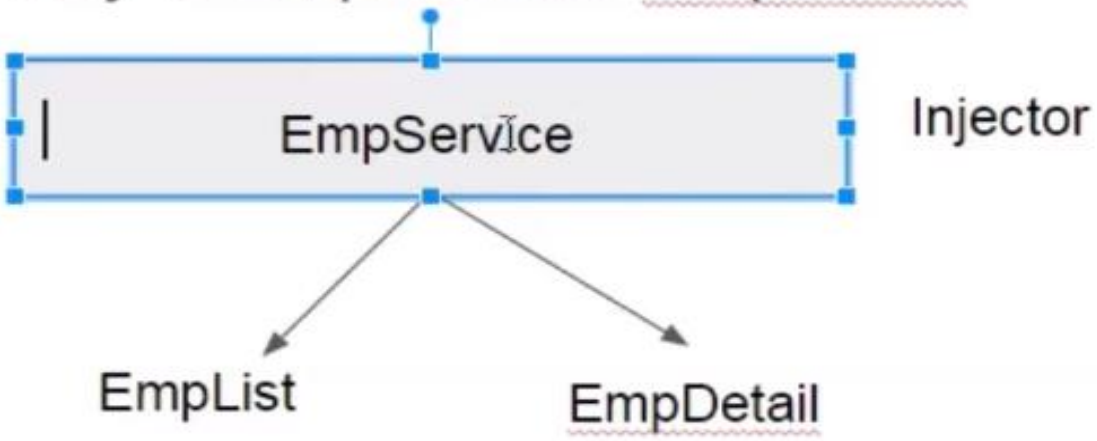
## DI as a design pattern

DI is a coding pattern in which a class receives its dependencies from external sources rather than creating them itself.

# Services

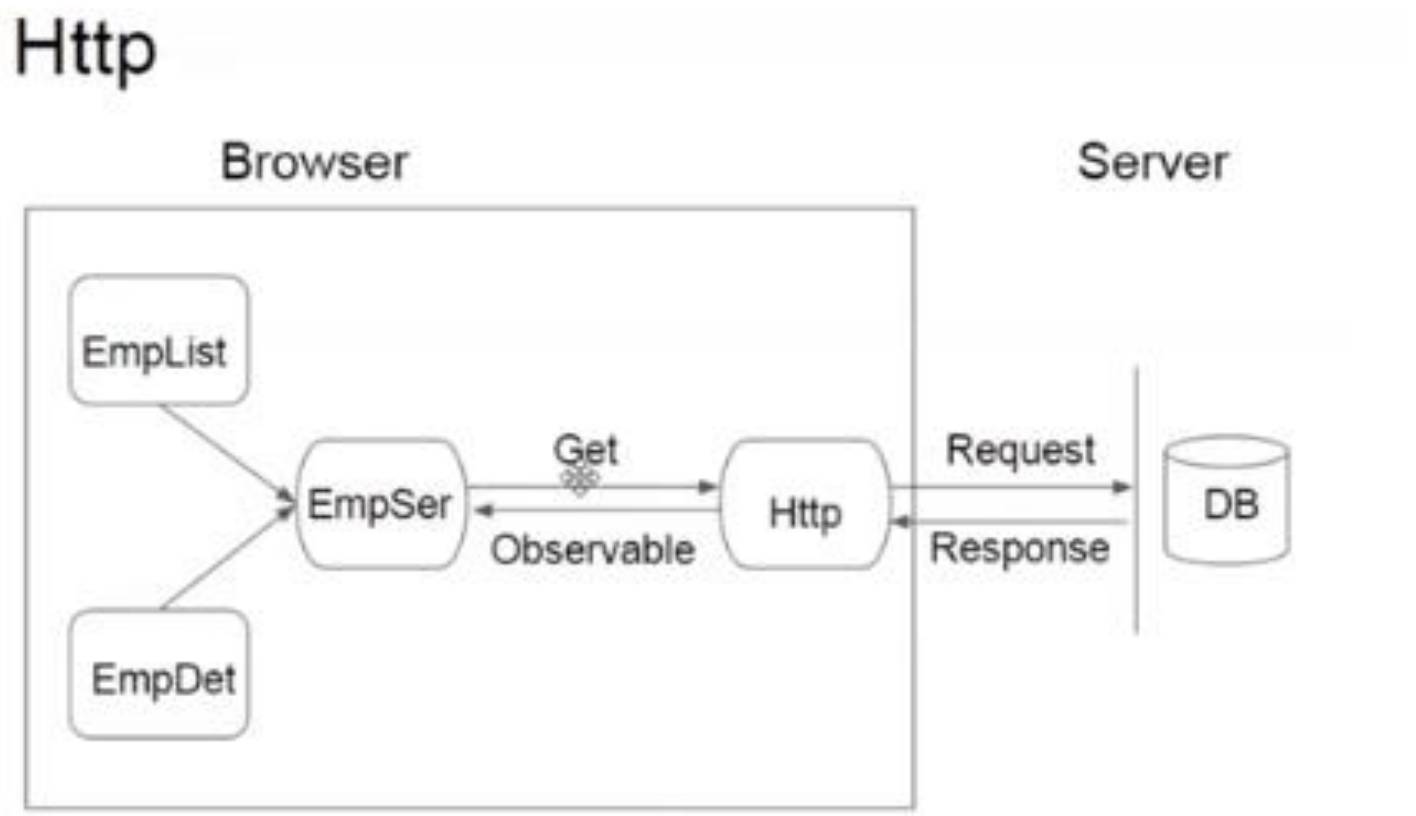
## DI as a framework

1) Define the EmployeeService class
2) Register with Injector
3) Declare as dependency in EmpList and EmpDetail



- Services are injectable hierarchically.

# HTTP


Http
Browser
Server
EmpList
EmpSer
EmpDet
Get
Observable
Http
Request
Response
DB

# Observables and Rxjs

1) Make http call from EmpService
2) Receive the observable and map it
3) Subscribe to the observable
4) Assign the Emp Data to local variable in view

Rxjs - Reactive Extensions for Javascript

- External Library to work with Observables

# Routing

**app.module.ts**

```
const appRoutes: Routes = [
  { path: '', component: HomeComponent },
  { path: 'users', component: UsersComponent },
  { path: 'servers', component: ServersComponent },
];

@NgModule({
  declarations: [
    AppComponent,
    HomeComponent,
    UsersComponent,
    ServersComponent,
    UserComponent,
    EditServerComponent,
    ServerComponent
  ],
  imports: [
    BrowserModule,
    FormsModule,
    HttpModule,
    RouterModule.forRoot(appRoutes)
```

Index.html
-router-outer is ng directive used for routing.
The view will get loaded at this position based on url

```
</div>
<div class="row">
  <div class="col-xs-12 col-sm-10 col-md-8 col-sm-offset-1 col-md-offset-2">
    <router-outlet></router-outlet>
  </div>
</div>
/div>
```

# Routing steps

- step 1. ng new myroutingapp
- do you want to add routing ---yes

- step2- add PlistComponent
- step 3- Add PtabComponent
- step 4- add PlistComponent,PtabComponent in declarations array in app.module.ts
- step 5- add import PlistComponent,PtabComponent statements in app.module.ts
- step6 - in app.component.html

- <h1>Angular Router</h1>
- <nav>
- <a routerLink="/list" routerLinkActive="active">Person List</a>
- <a routerLink="/ptab" routerLinkActive="active">Person table</a>
- </nav>
- <router-outlet></router-outlet>

- step 7- add following entry in app.module.ts

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

const appRoutes: Routes = [
  { path: 'list', component: PlistComponent },
  { path: 'ptab',      component: PtabComponent },

];
@NgModule({
  imports: [
    RouterModule.forRoot(
      appRoutes,
      { enableTracing: true } // <-- debugging purposes only
    )
    // other imports here
  ],
  ...
})
export class AppModule { }
```

# Component Communication

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

 @Component({
  selector: 'app-root',
  template: `<h1 [ngClass]={myclass:one,myclass2:two}>hello</h1>
  {{name}} {{sal}} {{3+6}} {{getName()}}
  From Child Component {{message.name}}

  <app-person [parentData]="pData" (myevent)="message=$event"></app-person>
  <input type="text"  [(ngModel)]="cdata"/>
  <button type="button" (click)="changePdata()">pass data to child</button>
  <div *ngIf="test">this is test for ngif</div>
  `,
  styleUrls: ['./app.component.css']

})
export class AppComponent {
```

```
export class AppComponent {
  cdata:string;
  pData="Testing data from parent";
  public message;
  name="kishori";
  sal=1234.45;
  test:boolean=false;
  one=false;
  two=true;
  getName(){
    return 'abcd';
  }
  changePdata(){
    this.pData="abc change from parent";
  }
```

# Component Communication- Child Component (person.component.ts)

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


@Component({
    selector:'app-person',
    template:`
        <div>This is person Compnent {{strmsg}}</div>
        <button (click)=onclick()>click me</button>
    `
})
export class PersonComponent{
per={name:"kishori",address:"Aundh"}
@Input('parentData') public strmsg;
@Output() public myevent=new EventEmitter();
onclick(){
    this.myevent.emit(this.per);
}

}
```

# TDF Validation

| State | Class if true | Class if false |
| --- | --- | --- |
| Control has been visited | `ng-touched` | `ng-untouched` |
| Control's value has changed | `ng-dirty` | `ng-pristine` |
| Control's value is valid | `ng-valid` | `ng-invalid` |

# Form Handling(Template driven Forms)

```
{{title|uppercase}}
<div class="form-group">
<form #pform="ngForm"  (ngSubmit)="onsubmit(pform.value)" >
 Person Id : <input #myidRef="ngModel" class="form-control" type="text" name="pid" required minlength="3" ngModel><br/>
 <div *ngIf="myidRef.errors && (myidRef.dirty || myidRef.touched)">

 <div [hidden]="!myidRef.errors.required">
   Pls enter valid data
</div>
<div [hidden]="!myidRef.errors.minlength">
    Pls enter min length to 3
 </div>
</div>


 <br><br> Person Name :<input type="text" name="name" ngModel>
 Person Designation <input type="text" class="form-control" name="designation" ngModel>
  <input type="submit" name="btn1" value="Submit">

</form>
</div>
```

# Retrieve data from checkbox

- <div *ngFor="let data of emails">
- <input type="checkbox" (change)="onChange(data.email, $event.target.checked,$event)">
{{data.email}}<br>
- 
- And onChange:

- onChange(email:string, isChecked: Boolean,e:any) {
- const emailFormArray = <FormArray>this.myForm.controls.useremail;

- if(isChecked) {
- emailFormArray.push(new FormControl(email));
- } else {
- let index = emailFormArray.controls.findIndex(x => x.value == email)
- emailFormArray.removeAt(index);
- }
- }

- To install @angular/http in angular version 7
- npm install --save-dev @angular/http

# WebServiceDemo – app.module.ts

```
import { HttpClientModule, HttpClient } from '@angular/common/http’;
import { BrowserModule } from '@angular/platform-browser';
import { NgModule } from '@angular/core';
import { PtabComponent } from './person/person.component';
import { AppRoutingModule } from './app-routing.module';
import { AppComponent } from './app.component';
import { HttpModule } from '@angular/http';
import {PerService} from './person.service';
import { Person } from './Person';
import {FormsModule } from '@angular/forms';
@NgModule({
  declarations: [
    AppComponent,PtabComponent
  ],
  imports: [
    BrowserModule,
    AppRoutingModule,HttpClientModule,FormsModule
  ],
  providers: [PerService],
  bootstrap: [AppComponent]
})
export class AppModule { }
```

# Ptab.component.ts

```
import { PerService } from "../person.service";
import { Component } from "@angular/core";
import { Person } from "../Person"
@Component({
    selector:'ptab',
    templateUrl:'./person.component.html',
    styleUrls:['./person.component.css']
})
export class PtabComponent{
perarr:Person[];
showform=false;
p:Person={pid:0,pname:"",mobile:""};
choice:string="add";
constructor(private pservice:PerService){}
ngOnInit(){
    this.pservice.getPersons()
    .subscribe((r)=>this.perarr=<Person[]>r);

}

getPersonById(){
    this.pservice.getPersonById(this.p.pid)
    .subscribe((r)=>this.p=<Person>r);

}

```

# Ptab.component.ts

```
updatePerson(p1:Person,ch:string){
    //to display form
    this.showform=true;
    //no need to call getPersonById we can pass the object from HTML
   //assign u to choice to call update function in onsubmit
   this.choice="ch";
   //display person in form for updation
   this.p={pid:p1.pid,pname:p1.pname,mobile:p1.mobile};
}

onsubmit(){
  //hide the form
  this.showform=false;
  if(this.choice=="add"){
    this.pservice.addPerson(this.p)
    .subscribe((r)=>this.perarr=<Person[]>r);
}else{
   //reset choice to add
   this.choice="add";
   this.pservice.updatePerson(this.p)
    .subscribe((r)=>this.perarr=<Person[]>r);

}

}
}
```

# WebService Communication-person.service.ts

- import { Observable } from 'rxjs';
- import { HttpClient } from '@angular/common/http';
- import { Response, Headers, RequestOptions,  } from '@angular/http';
- import { Person } from './Person';
- import { Injectable } from '@angular/core';
- import 'rxjs/add/operator/map’;  ///add  "rxjs-compat": "^6.3.3",
- import 'rxjs/add/operator/catch’;
- @Injectable()
- export class PerService{
-
-  personUrl="http://localhost:9090/PersonWebService123/persons";
-
-  constructor(private http: HttpClient){
-
- }

# Person.ts

- export class Person{
- pid;
- pname;
- mobile;

- }