ANGULAR 4 BASICS

What is Angular (Angular 2)?

- Next version of most successful AngularJS 1.x
- Finally released on **14th Sep, 2016**. It is called Angular 2.0.0.
- 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 Angular2

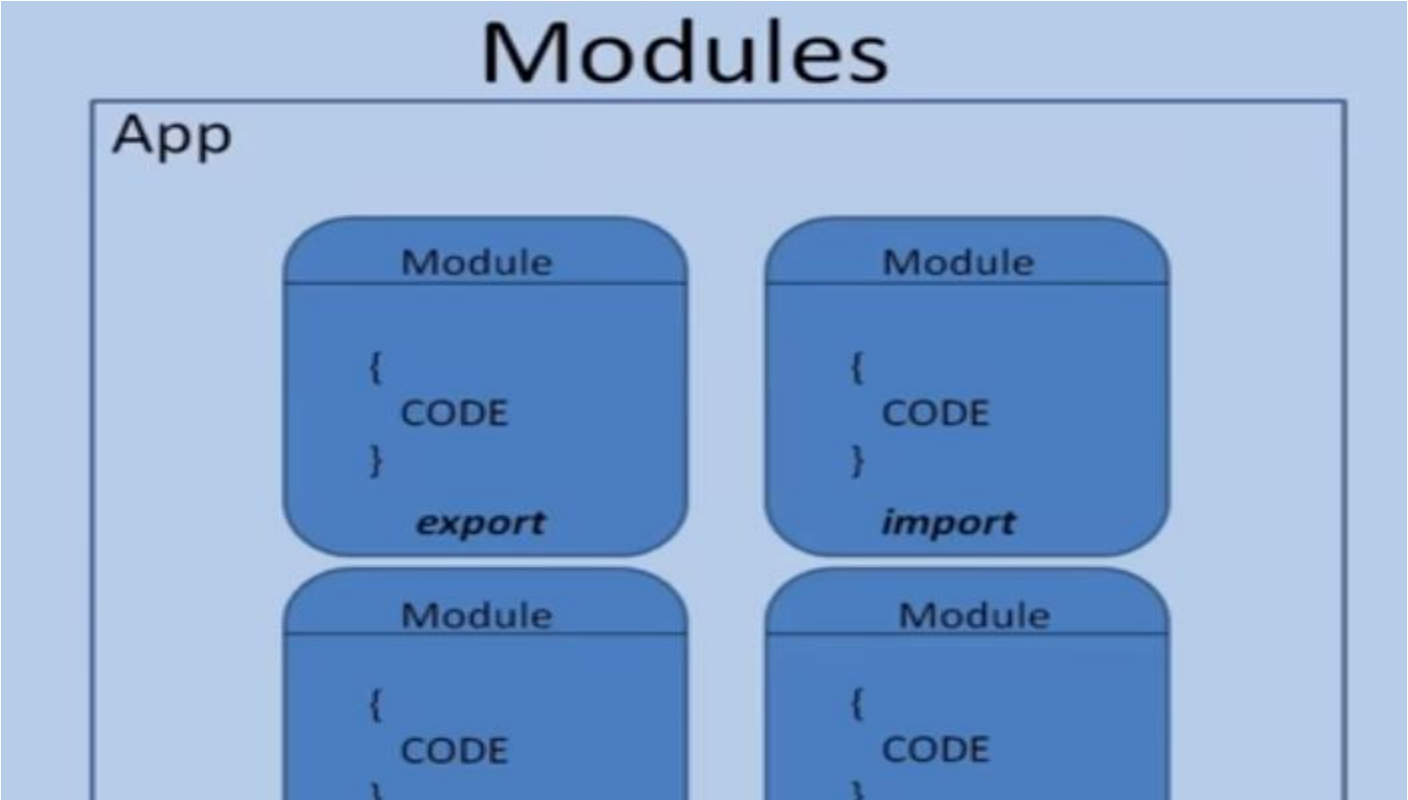
- Called as AngularJS 1.x and Angular.
- Components are used instead of Controllers and $scope. A component is a class with its own data and methods.
- Option to write code in different languages.
- Designed for Speed. Supposed to be 5 times faster than Angular 1.
- Designed for Mobile development also.
- 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 models.
- 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.

Module Example

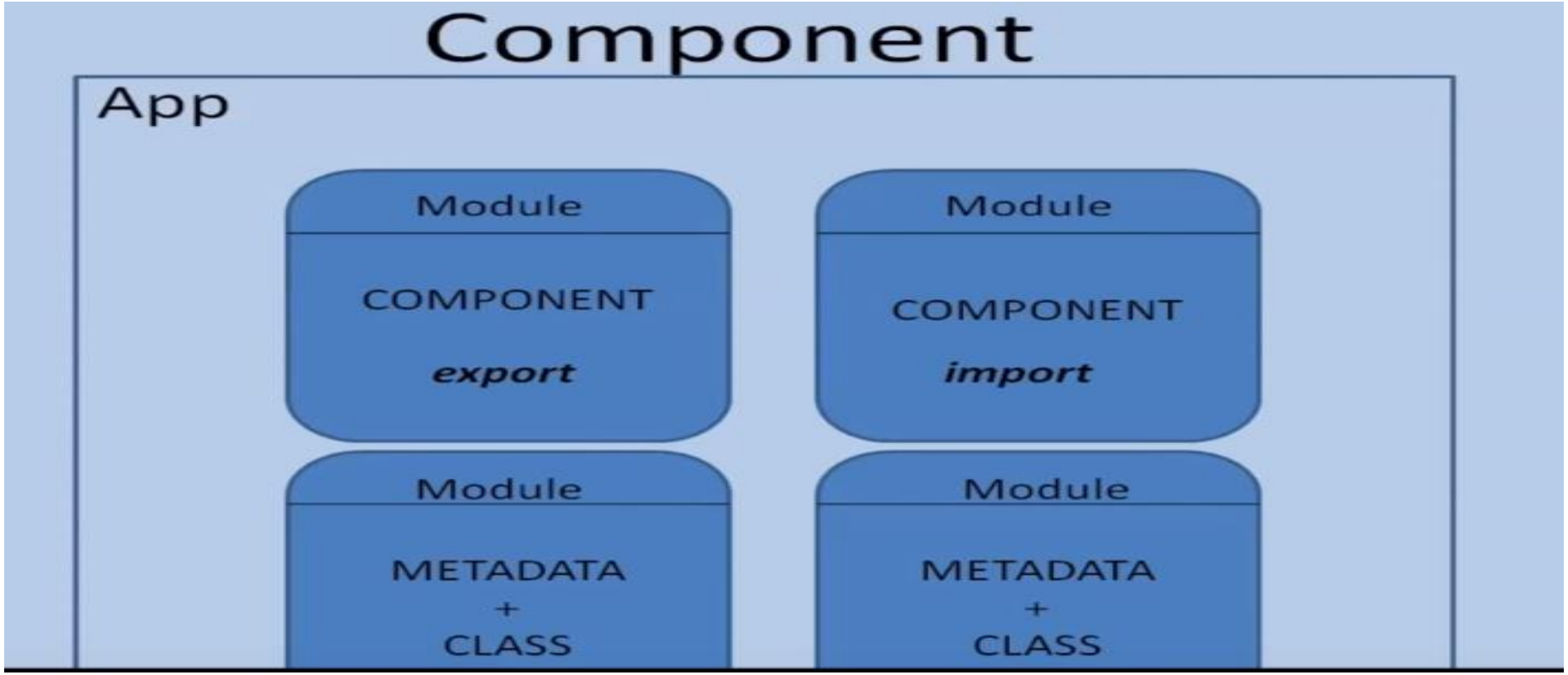
- The following code shows how to create a simple module:

  AppModule.ts

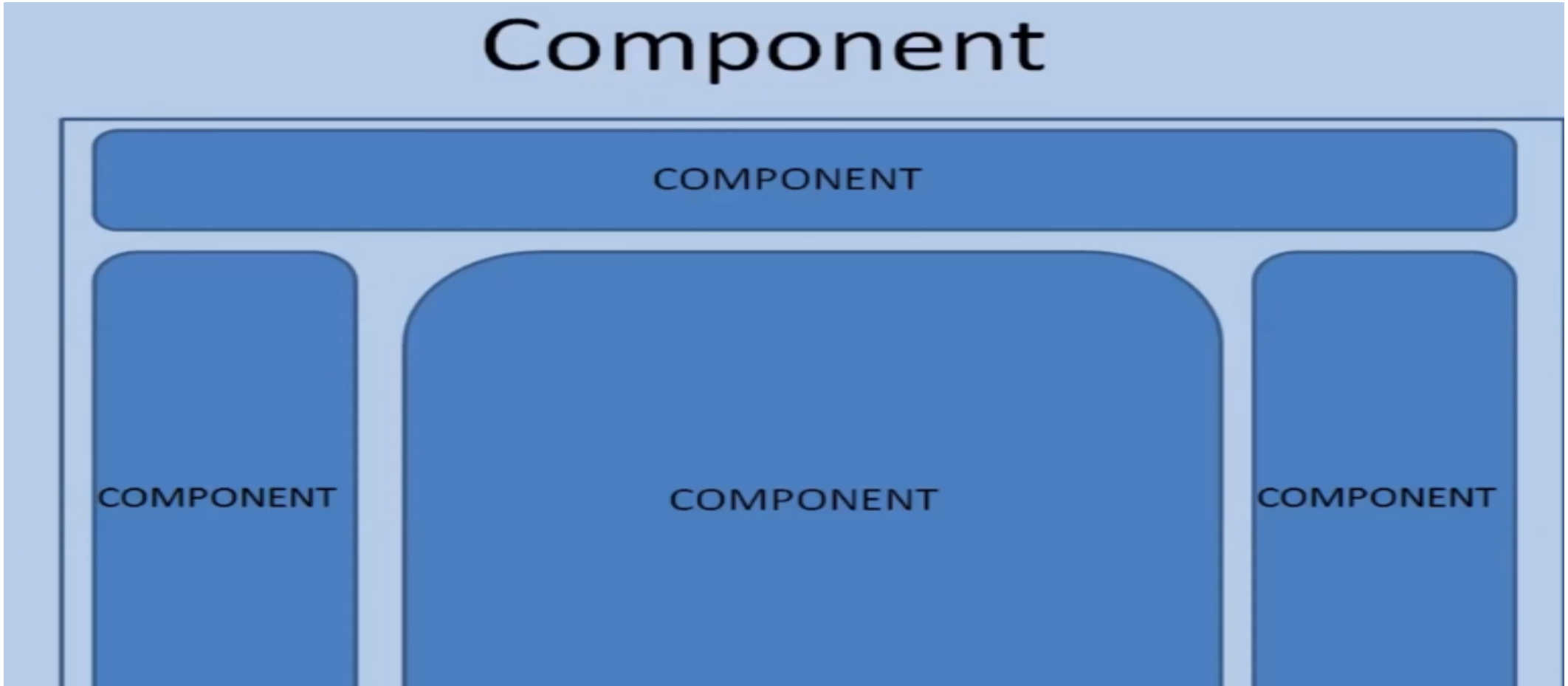
```
import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-
browser';
import { FirstComponent } from './first.component';
@NgModule({
imports: [ BrowserModule], declarations: [ FirstComponent], bootstrap: [ FirstComponent ]
})
export class AppModule { }
```

Component

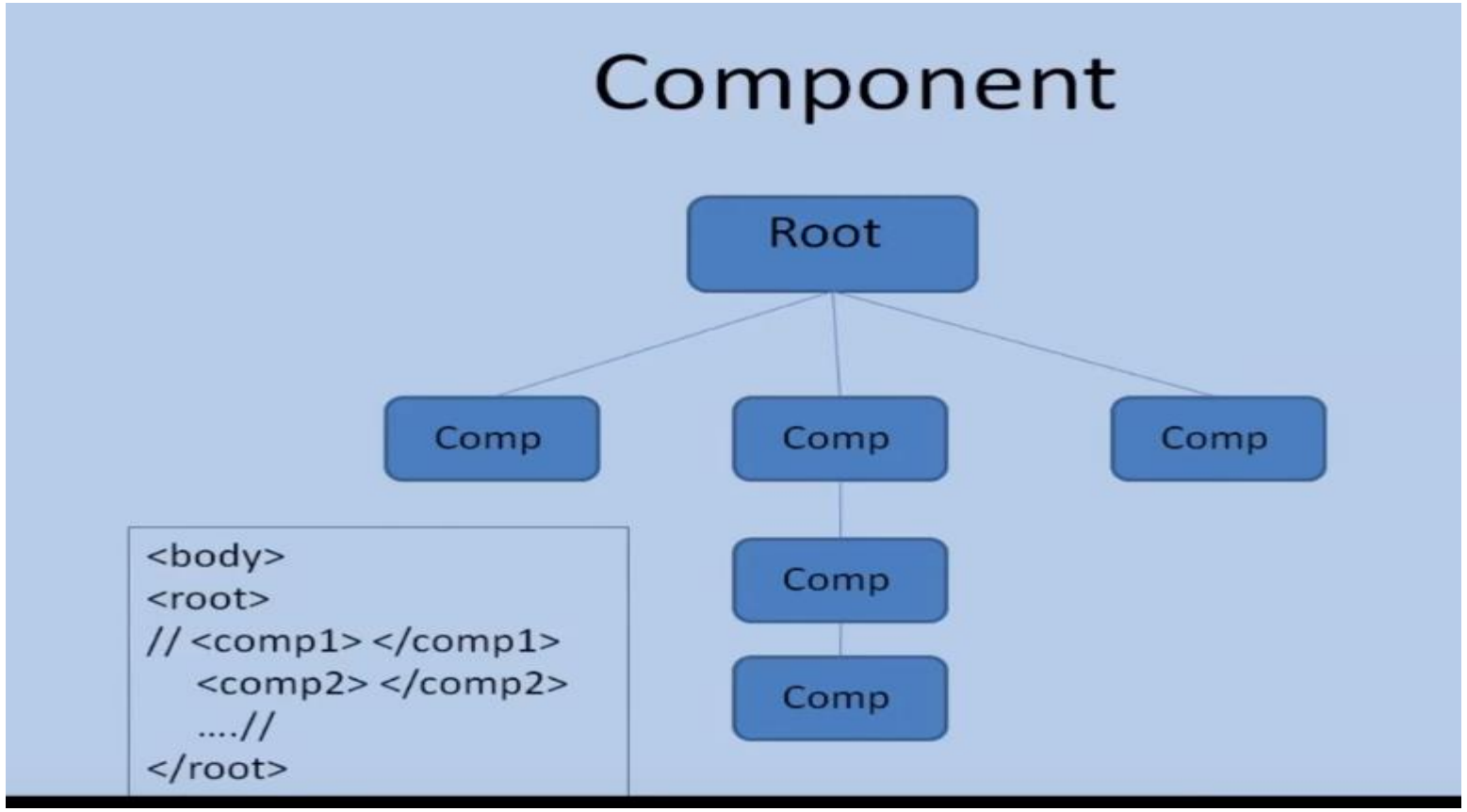
- A component controls a part of the screen called view.
- Every component is a class with its own data and code.
- A component may depend on services that are injected using dependency injection.
- The template, metadata, and component together describe a view.
- Components are decorated with @Component decorator through which we specify **template** and **selector** (tag) related to component.
- Properties like **templateUrl** 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 atleast one component which is root component and other componets 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";
- }

- **Templates**
- 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.

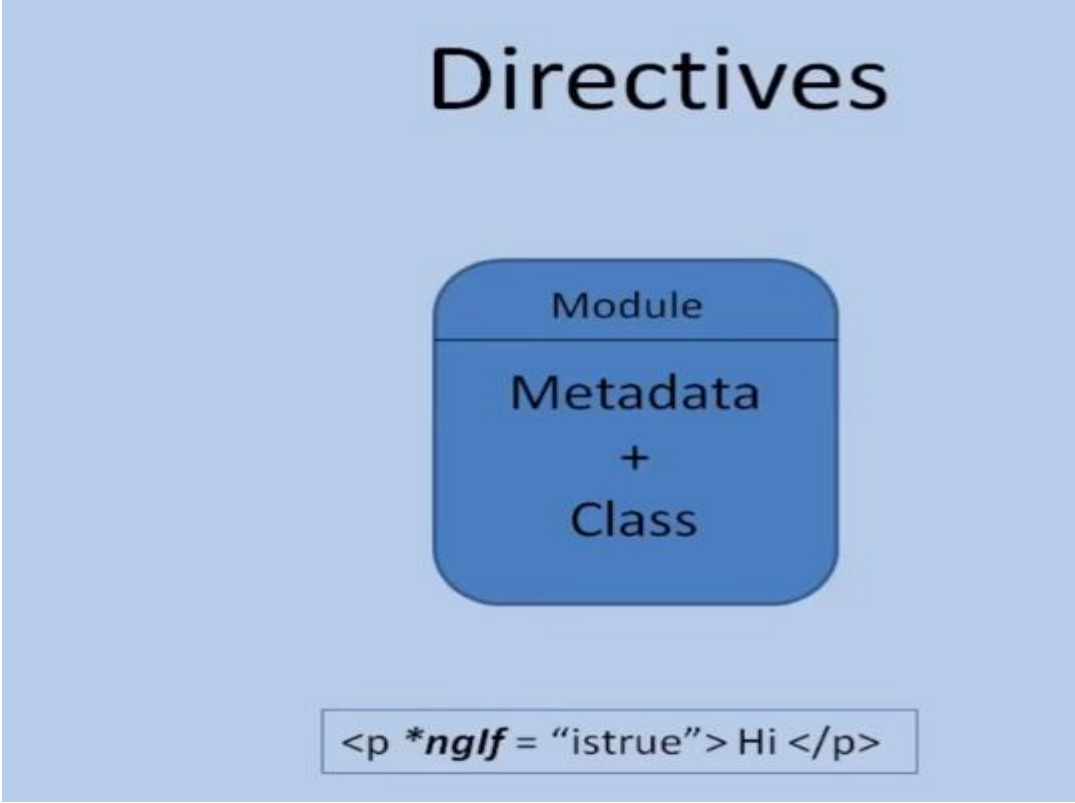
- For example, @Component decorator tells Angular to treat a class as a component and also provides additional information through attributes of decorator (like selector, template etc.)

- **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) copies value to property.
- Enclosing event in parentheses () will assign event handler to event.
- Interpolation allows value of an expression to be used in HTML
- The ng-model is used to for two way 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, hence the name.
- *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



- **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.

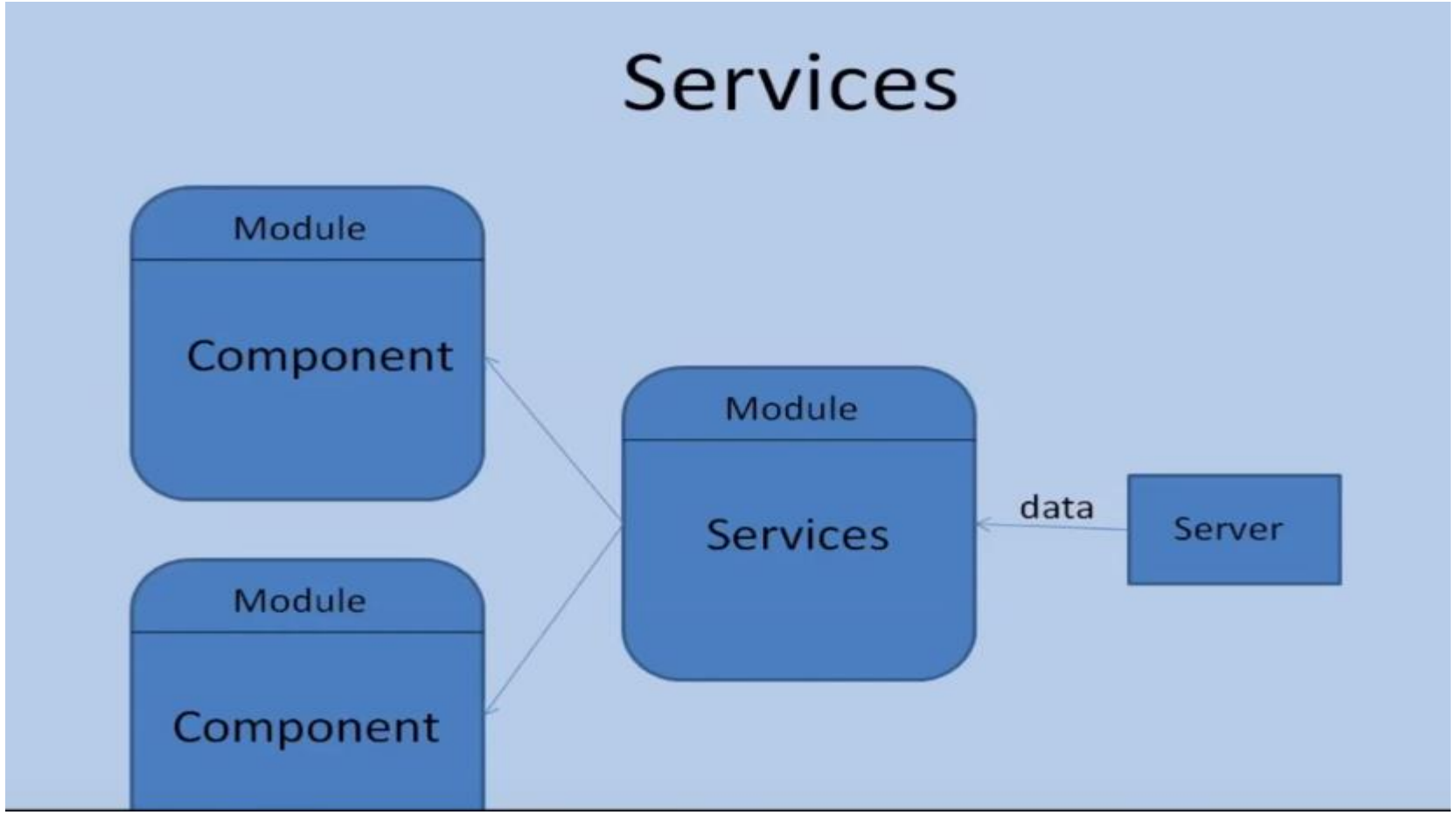
- Services are injected into Component that use them.

- **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.

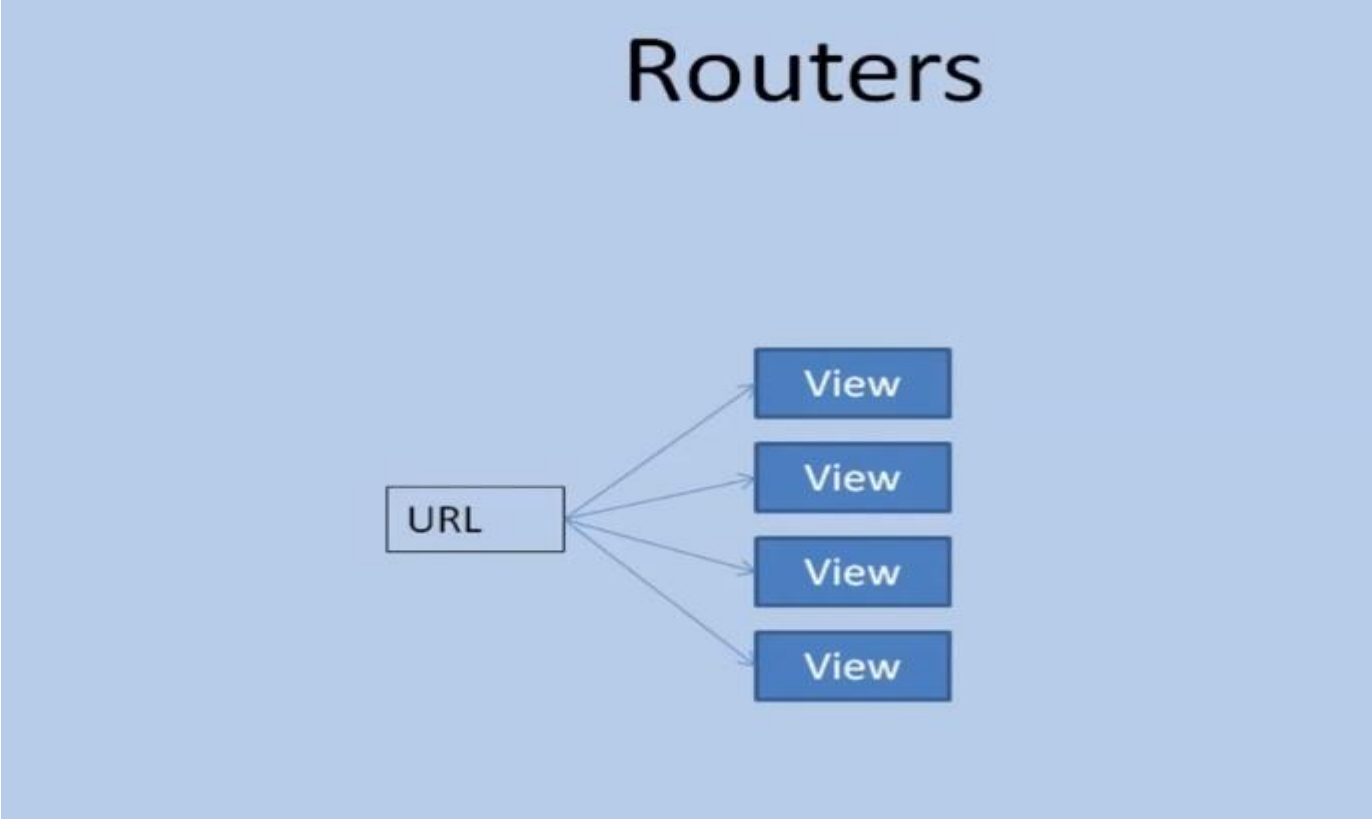
- An injector contains instances of services. It injects them into other services and components as and where they are needed.

Services


Services
Module
Component
Module
Services
data
Server
Module
Component

Router

- It decides which view should appear based on URL



Angular 4 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.

- Quickstart seed, maintained on github, is quick way to get started with Angular local development.
- Follow the steps below to clone and launch quickstart application.
- 1. Create a folder for project. Let us call it demo.
- 2. Download quickstart seed from https://github.com/angular/quickstart/archive/master.zip
- 3. Extract quickstart-master.zip into folder created above.

- 4. Install all packages mentioned in packages.json file using:
- Change the folder to demo give command

  npm install
- After npm downloaded required packages given in package.json,
- start application using the following command. It starts server and monitors for changes in the application.

  npm start

Package.json

```
{
  "name": "form-validation-app",
  "version": "0.0.0",
  "license": "MIT",
  "scripts": {
    "ng": "ng",
    "start": "ng serve",
    "build": "ng build --prod",
    "test": "ng test",
    "lint": "ng lint",
    "e2e": "ng e2e"
  },
  "private": true,
  "dependencies": {
    "@angular/animations": "^5.2.0",
    "@angular/common": "^5.2.0",
    "@angular/compiler": "^5.2.0",
    "@angular/core": "^5.2.0",
    "@angular/forms": "^5.2.0",
    "@angular/http": "^5.2.0",
    "@angular/platform-browser": "^5.2.0",
    "@angular/platform-browser-dynamic": "^5.2.0",
    "@angular/router": "^5.2.0",
    "core-js": "^2.4.1",
    "rxjs": "^5.5.6",
    "zone.js": "^0.8.19"
  },
  "devDependencies": {
    "@angular/cli": "~1.7.4",
    "@angular/compiler-cli": "^5.2.0",
    "@angular/language-service": "^5.2.0",
    "@types/jasmine": "~2.8.3",
    "@types/jasminewd2": "~2.0.2",
```

- It includes dependancies property
- These will get downloaded when you use

  npm install
- All modules will get downloaded in node-modules folder

  npm start

Start command from package.json will get executed

- Will execute typescript compiler
- And start lite server
- Type script compiler will convert code in es5 format
- There are certain component added in ES6 version which is used by browser
- So we need file typings.config

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

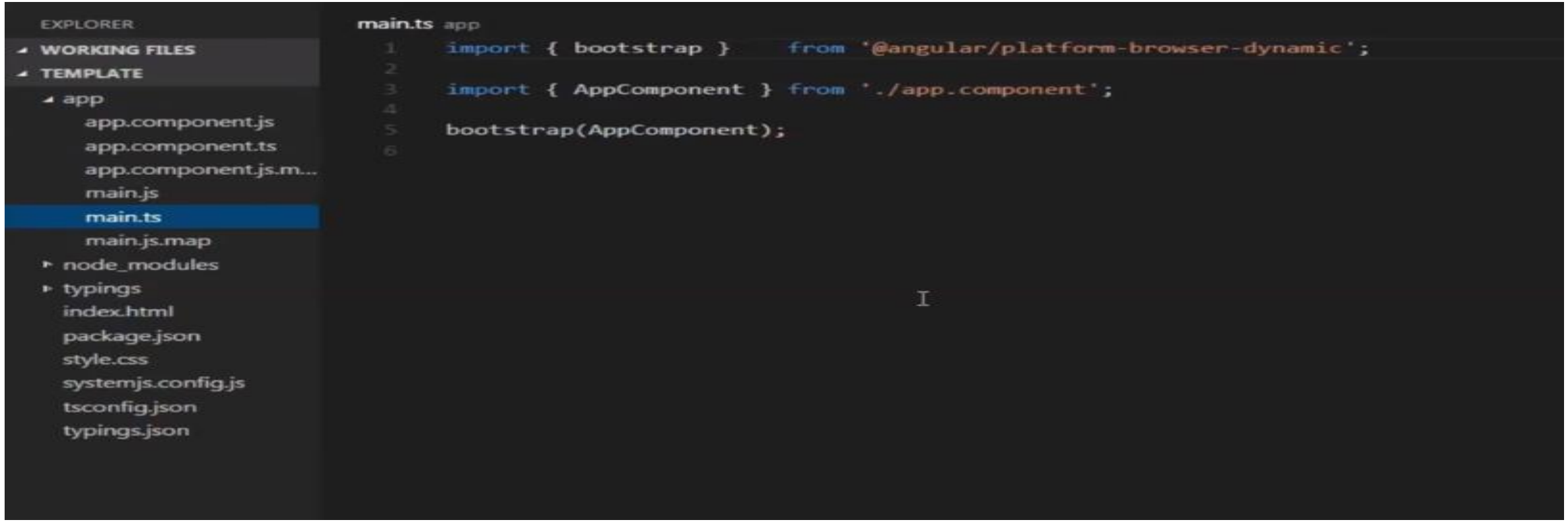
EXPLORER
- WORKING FILES
- TEMPLATE
  - app
    - app.component.js
    - app.component.ts
    - app.component.js.m...
    - main.js
    - main.ts
    - main.js.map
  - node_modules
  - typings
  - index.html
  - package.json
  - style.css
  - systemjs.config.js
  - tsconfig.json
  - typings.json

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>
```

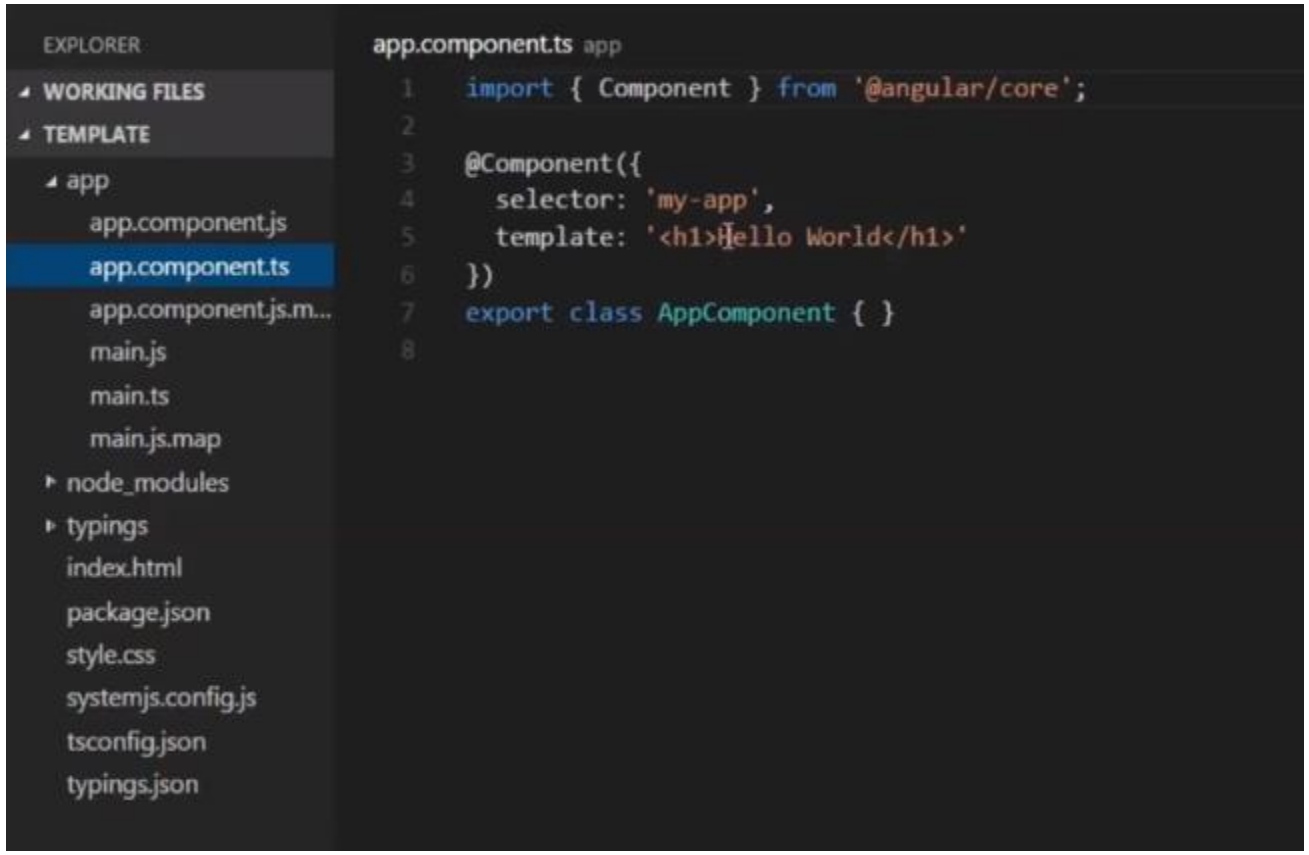
App folder

- 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

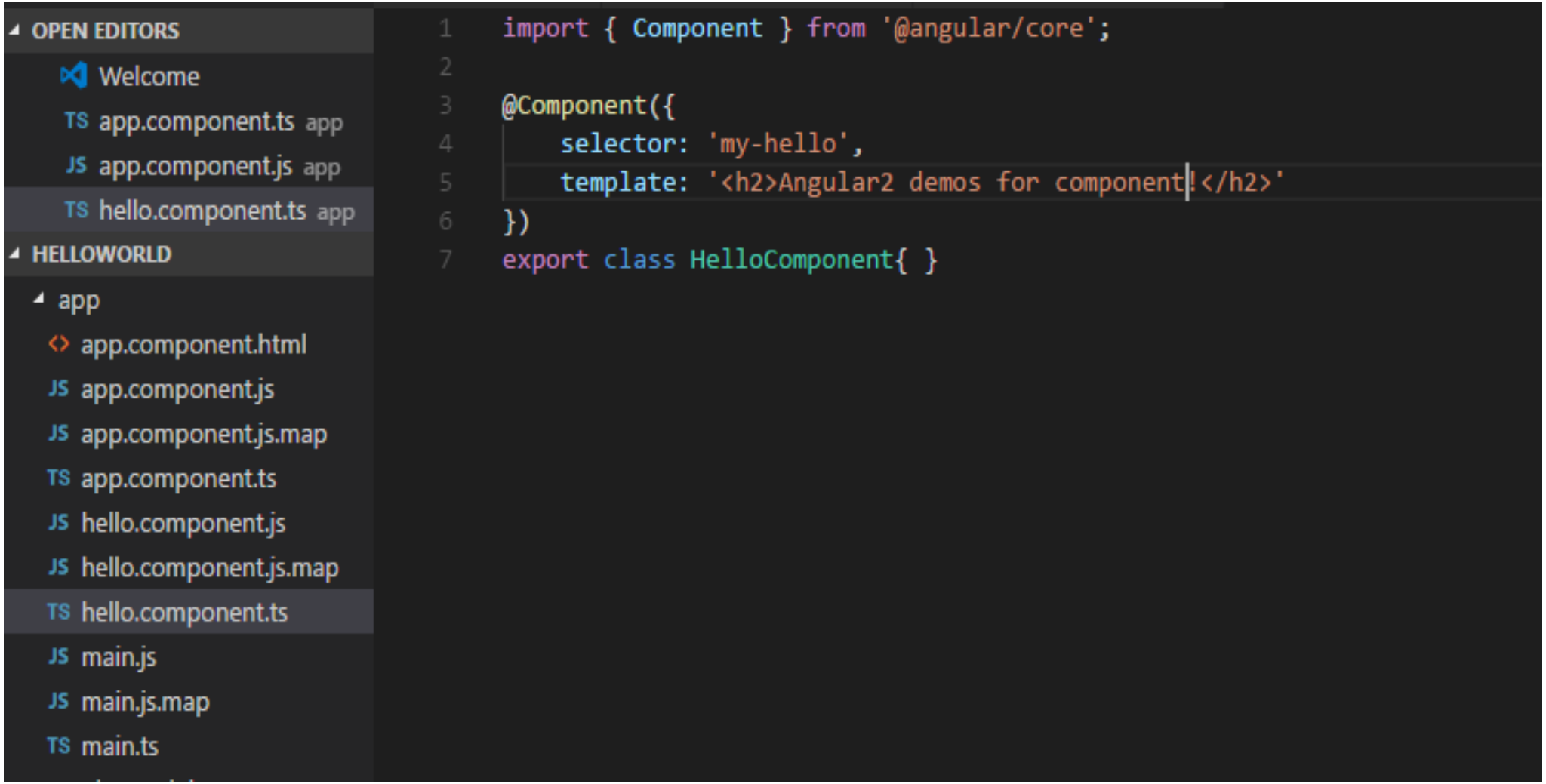


Run the code

- To run the code
- Change the folder to demo

```
npm start
```

Add new component



- Modify app.component.ts

```
OPEN EDITORS
    Welcome
    TS app.component.ts app
    JS app.component.js app
    TS hello.component.ts app
HELLOWORLD
  app
    <> app.component.html
    JS app.component.js
    JS app.component.js.map
    TS app.component.ts
    JS hello.component.js
    JS hello.component.js.map
    TS hello.component.ts
    JS main.js
    JS main.js.map
    TS main.ts
  node_modules
  typings
  <> index.html
  npm-debug.log
  {} package.json
  # style.css
  JS systemjs.config.js
  {} tsconfig.json
  {} typings.json
```

```
1   import { Component } from '@angular/core';
2   import { HelloComponent } from './hello.component';
3
4   @Component({
5     selector: 'my-app',
6     template: `<h1>Hello World</h1>
7               <my-hello>Loading....</my-hello>`,
8     directives: [HelloComponent]
9   })
10  export class AppComponent { }
11
```

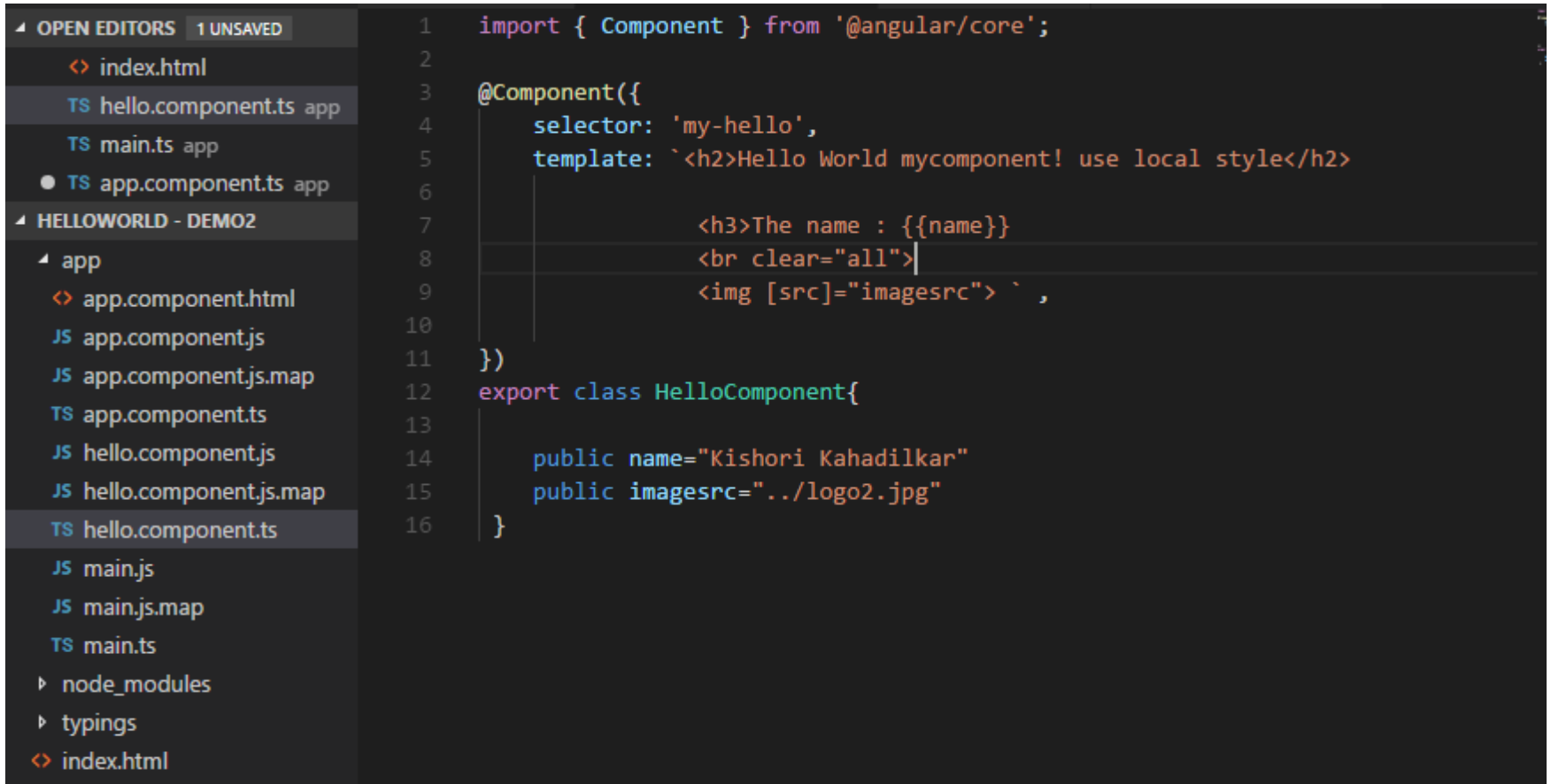
Import the compon ent

Array that contains list of components to be used in in this component

Adding style to the 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"
 }
```

- If you render the page and in console window give command

- Check values in console window

```
> $0.value
<· "angular  test"
> $0.getAttribute("Value")
<· "angular  test"
> |
```

- Change value of text box to Kishori.
- The value of property will change but attribute will not

```
> $0.value
<· "angular  test"
> $0.getAttribute("Value")
<· "angular  test"
> $0.getAttribute("Value")
<· "angular  test"
> $0.value
<· "Kishori"
> |
```

- 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"
 }
```

Applying style using classes

```
• tutorials.component.ts app
import { Component } from '@angular/core';

@Component({
    selector: 'my-tutorials',
    template: `<h2>{{title}}</h2>
                <div [class.myClass]="applyclass">Apply Class</div>`,
    styles: [`.myClass{
        color:red;
    }`]
})
export class TutorialsComponent{
    public title="Tutorials from Joatmon Channel";
    public applyclass = true;
}
```

- Ternary expression for class binding

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

@Component({
    selector: 'my-tutorials',
    template: `<h2>{{title}}</h2>
                <div [class.myClass]="applyclass">Apply Class</div>
                <div [style.color]="applyblue? 'blue' : 'orange'">This is style binding</div>`,
    styles: [`.myClass{
        color:red;
    }`]
})
export class TutorialsComponent{
    public title="Tutorials from Joatmon Channel";
    public applyclass = true;
    public applyblue = true;
}
```

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 style</h2>

                <h3>The name : {{name}}
                <button type="button" (click)="MyClick(mytext.value)">cli
                <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

Two way binding

- For two way binding we use ngModel
- Since we use both event handling and property assigning ngModel need tobe enclosed in [(ngModel)]

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

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

                <h3>The name : {{name}}
                <!--<button type="button" (click)="MyClick(mytext.value)">clic
                <button type="button" (click)="MyClick($event)">click me</butt
                <input type="text" #mytext>
                <input type="text" [(ngModel)]="fname">{{fname}}
                <input type="text" [(ngModel)]="lname">{{lname}}` ,

})
export class HelloComponent{

    public name="Kishori Kahadilkar";
    public imagesrc="../logo2.jpg";
    public MyClick(myval){
        alert("Button clicked"+myval);
        console.log(myval);
    }
    public fname="Rajan";
    public lname="Khadilkar";
```

Directives in Angular 2

- 3 types of directive
  - Component
  - Structural
  - Attribute
- Structural
  - ngIf – based on value of showElement either paragraph will be displayed or hidden

```
@Component({
    selector: 'my-tutorials',
    template: `<h2>{{title}}</h2>
               <p *ngIf="showElement">Show Element<p>`
})
export class TutorialsComponent{
    public title="Tutorials from Joatmon Channel";
    public showElement=false;
}
```

- ngFor – use to repeat elements

```
@Component({
    selector: 'my-tutorials',
    template: `<h2>{{title}}</h2>
                <p *ngIf="showElement">Show Element<p>
                <div [ngSwitch]="color">
                    <p *ngSwitchWhen="'red'">Red color is shown</p>
                    <p *ngSwitchWhen="'blue'">Blue color is shown</p>
                    <p *ngSwitchDefault>Invalid color</p>
                </div>
                <ul>
                    <li *ngFor="let color of colors">{{color}}</li>
                </ul>`
})
export class TutorialsComponent{
    public title="Tutorials from Joatmon Channel";
    public showElement=false;
    public color='green';
    public colors=['red','blue','green'];
}
```

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();
}
```

Drawbacks of not using DI

Code without DI

```
class Engine{
    constructor(){}
}
class Tires{
    constructor(){}
}
```

```
class Car{
    engine;
    tires;
    constructor()
    {
        this.engine = new Engine();
        this.tires = new Tires();
    }
}
```

Class car creates engine object
So tight coupling is there. Any change in engine class will lead to change car class
If we add parameter to engine constructor will affect car

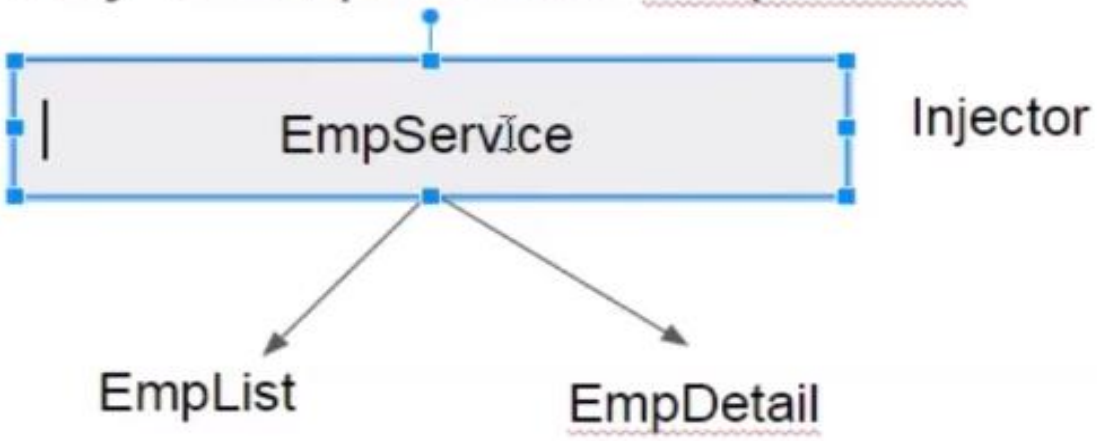
DI as a design pattern

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

Services

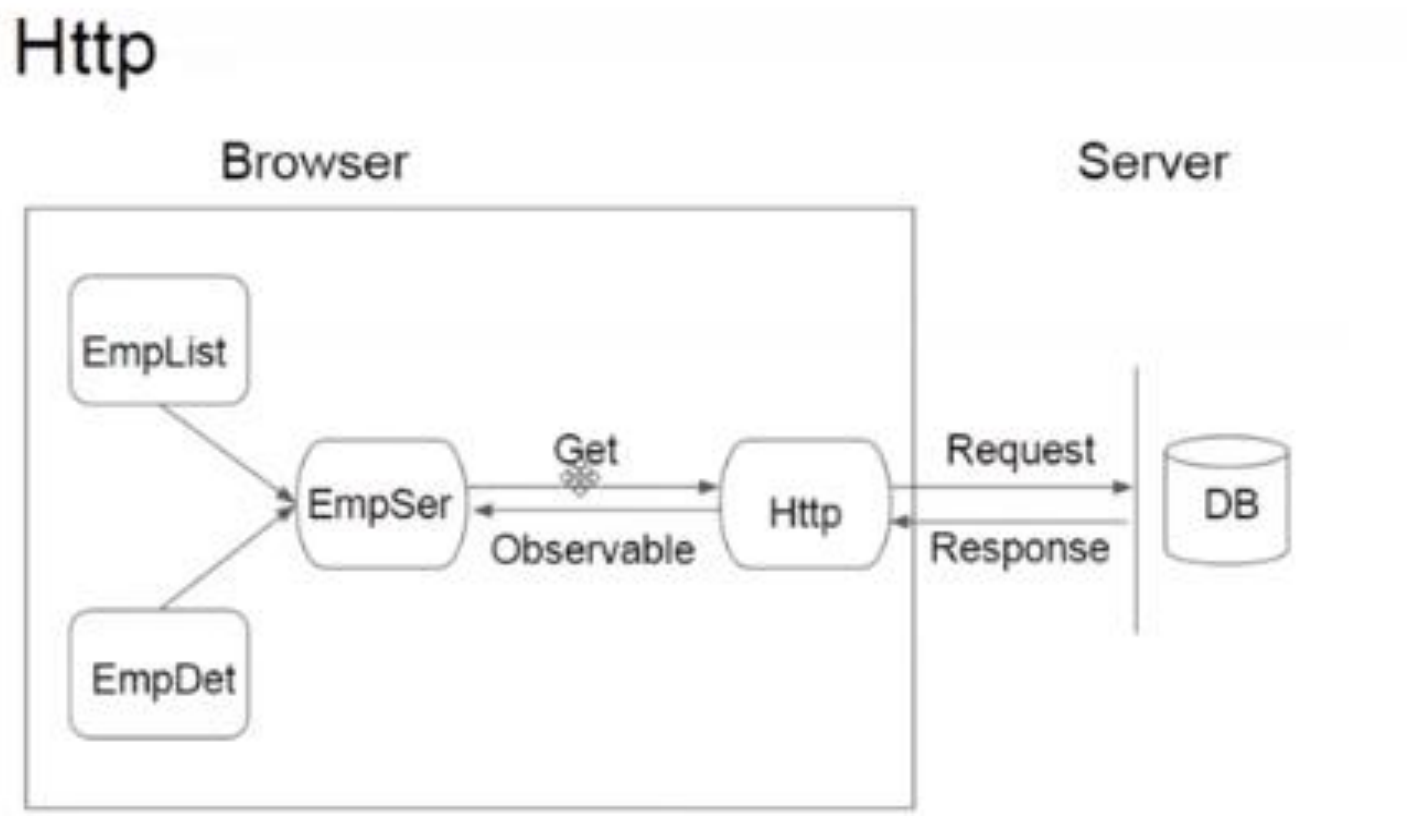
DI as a framework

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>
```