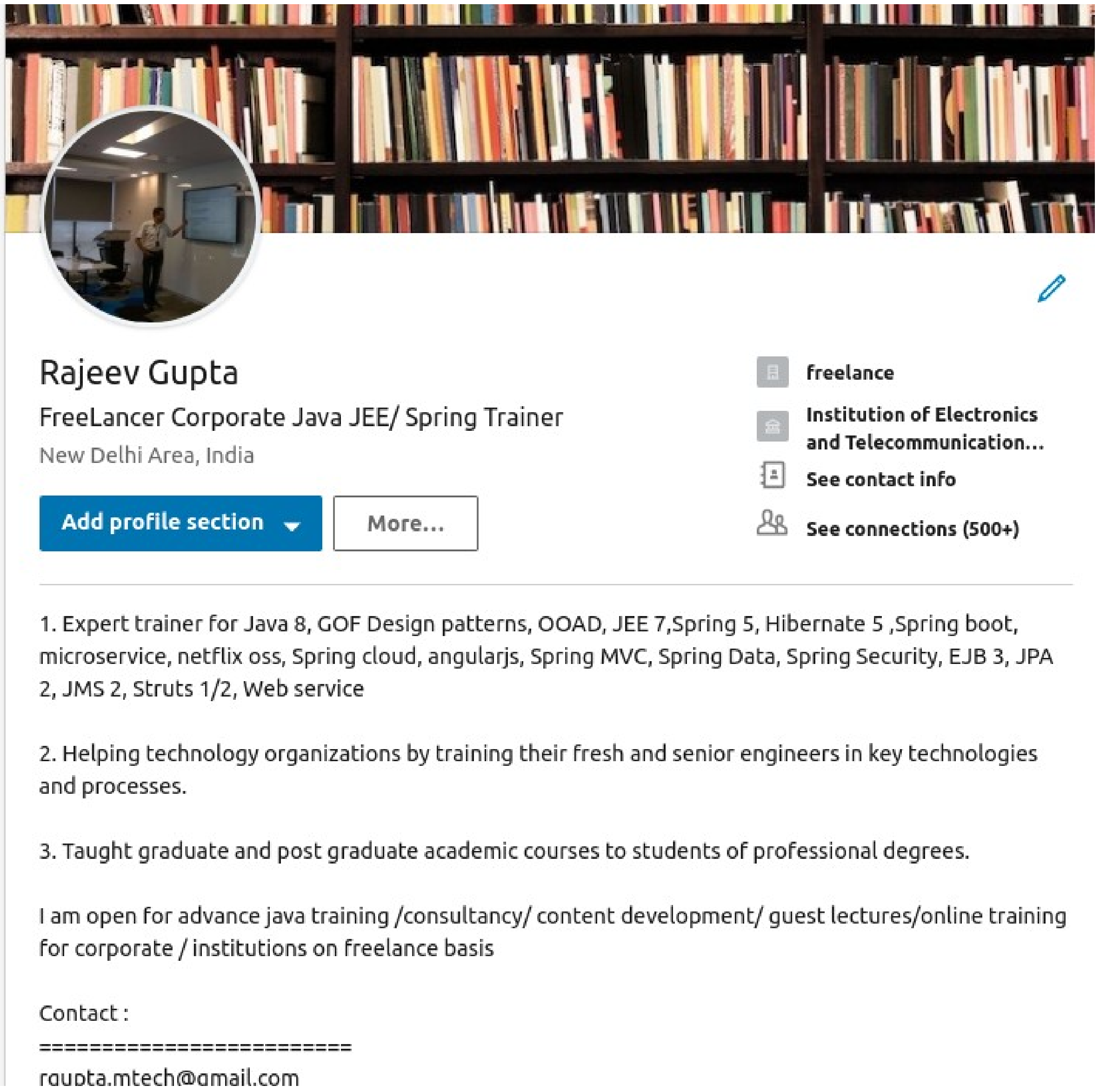
Rajeev Gupta
Trainer & Consultant



Rajeev Gupta

FreeLancer Corporate Java JEE/ Spring Trainer

New Delhi Area, India

freelance

Institution of Electronics and Telecommunication...

See contact info

See connections (500+)

1. Expert trainer for Java 8, GOF Design patterns, OOAD, JEE 7,Spring 5, Hibernate 5 ,Spring boot, microservice, netflix oss, Spring cloud, angularjs, Spring MVC, Spring Data, Spring Security, EJB 3, JPA 2, JMS 2, Struts 1/2, Web service

2. Helping technology organizations by training their fresh and senior engineers in key technologies and processes.

3. Taught graduate and post graduate academic courses to students of professional degrees.

I am open for advance java training /consultancy/ content development/ guest lectures/online training for corporate / institutions on freelance basis

Contact :

=========================

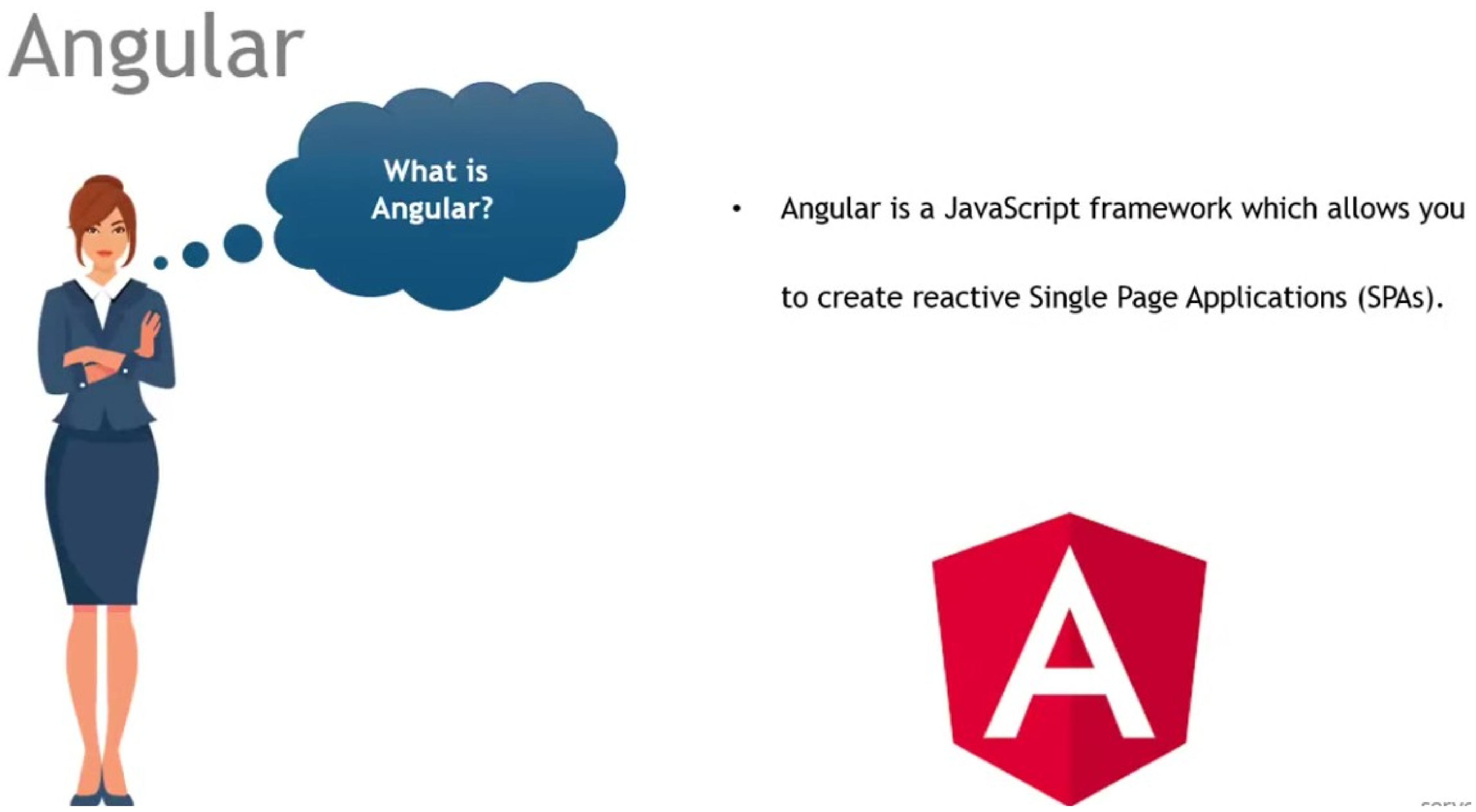
rgupta.mtech@gmail.com

Clients:

================

Gemalto, Noida
Cyient Ltd, Noida
Fidelity Investment Ltd
Blackrock, Gurgaon
Mahindra comviva
Steria
Bank Of America
incedo gurgaon
MakeMyTrip
Capgemini India
HCL Technologies
CenturyLink
Deloitte consulting
Nucleus Software
Ericsson Gurgaon
Avaya gurgaon
Kronos Noida
NEC Technologies
A.T. Kearney
ust global
TCS
North Shore Technologies Noida
IBM
Sapient
Accenture
Incedo
Genpact
Indian Air force
Indian railways
Vidya Knowledge Park

What is Angular?



Angular

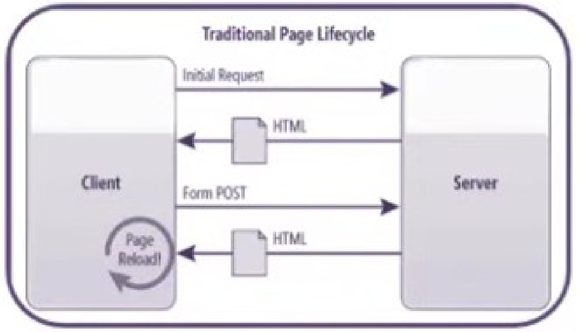
- Angular is a JavaScript framework which allows you to create reactive Single Page Applications (SPAs).

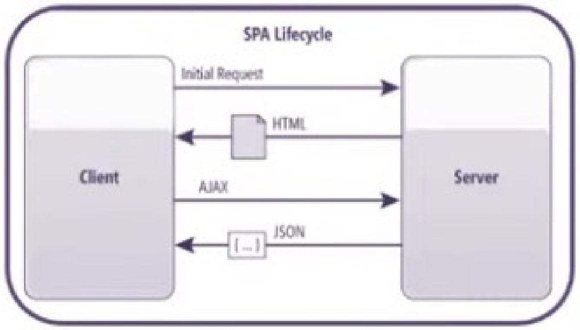
SPA

Single page application SPA

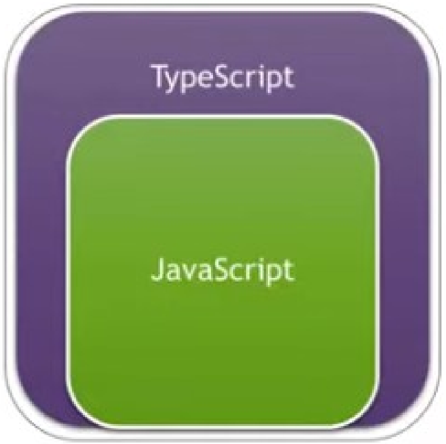
An web application that intract with user by dynamically rewriting the current page rather then loading entire new page from a server...

Best example gmail





TypeScript

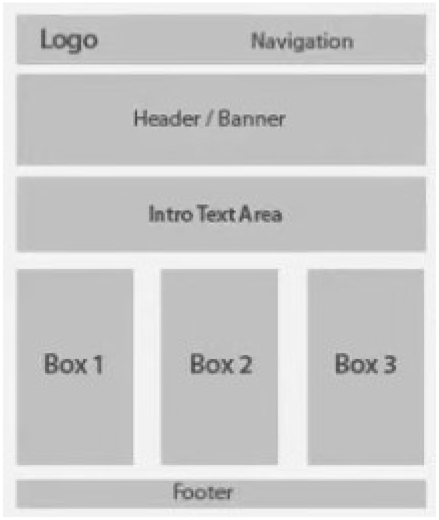


TypeScript is just JavaScript Code

TypeScript supports other JS libraries

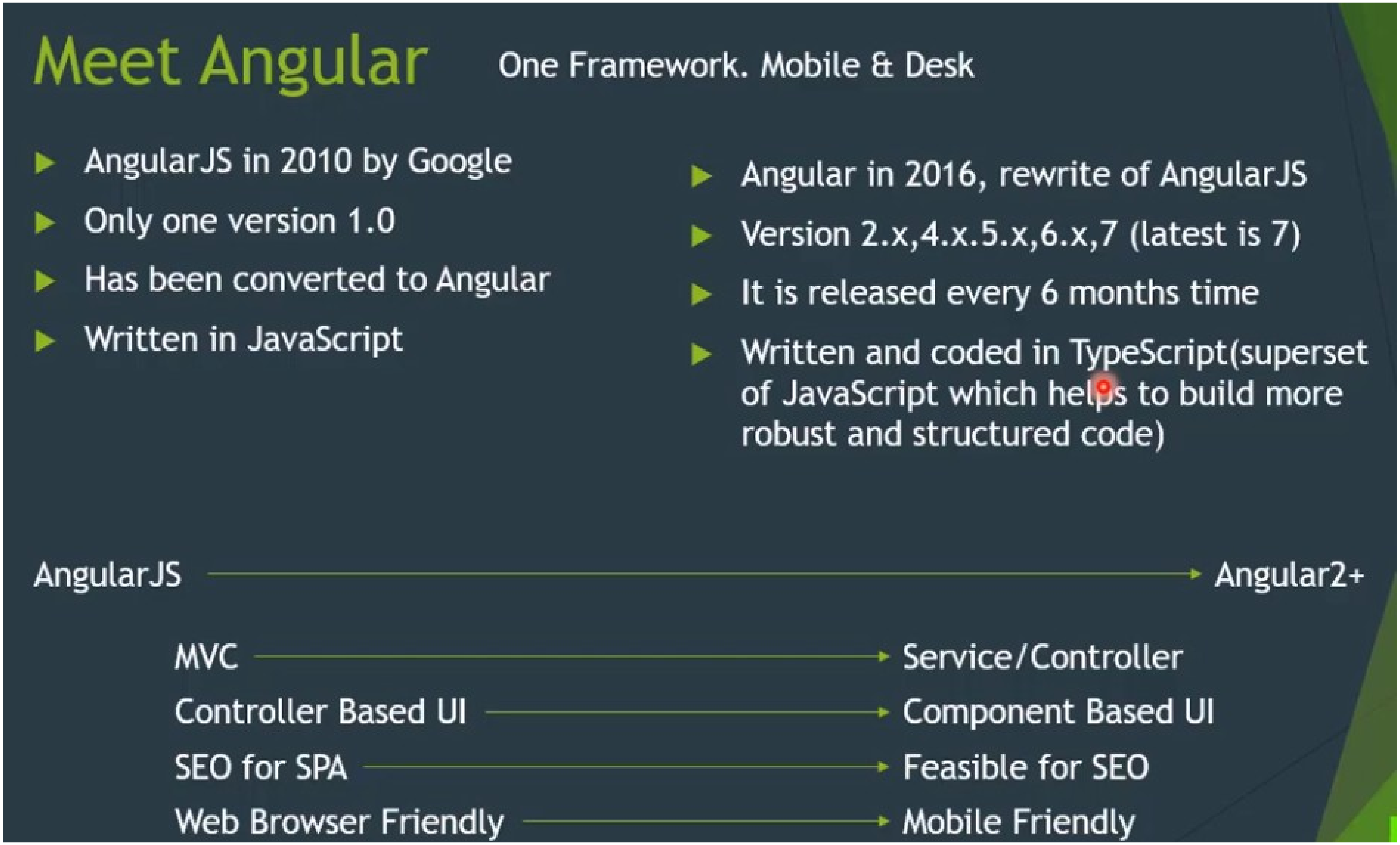
JavaScript files can be renamed as TypeScript

Why angular?



- Modularity
- Consistancy
- Maintainability
- Productivity
- Early handling, suggestion of error using typescript
- Better Mobile support
- **In angular everything is a component, gives better code reuse**
- Cross plateform, SPA application
- Angular is component based framework, and components are resuable

AngularJS	Angular
JavaScript is a programming language that is used with AngularJS.	Typescript is the programming language that is used with Angular 2 and above. Also, languages like JS, Dart, PureScript can also be used.
Uses scope and controller for variables.	Uses components and has no terms like scope and controllers.
Repetition is achieved using ng-repeat which is a directive.	Repetition is achieved using *ngFor, which is a directive.
Two Way Binding is done using ng-model.	Two Way Binding is done using [(ngModel)]
Dedicated directives are provided for event (ng-click) and proper binding (ng-bind).	Dedicated syntax are provided for event (())and property binding ([]).
Used third party mobile libraries as there was no native mobile support.	No need to use third-party libraries for mobile support as angular comes with native mobile support.
Route configuration is done using $routeProvider.when()	Route configuration is done using routes array.
Less manageable when compared to angular	Easy to create a project and functional structure for any kind (large/complex) of a project and well maintainable.



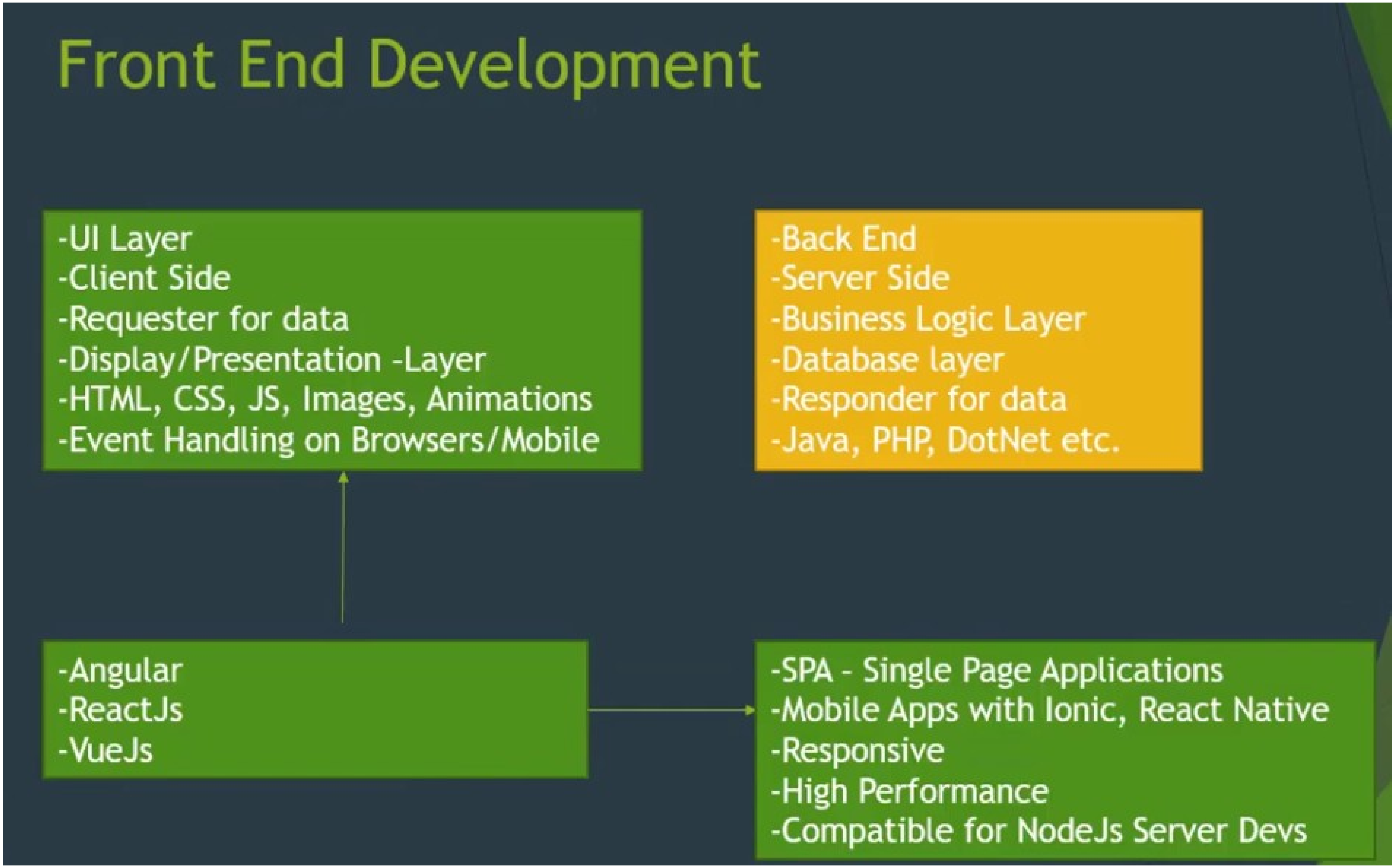
Meet Angular

One Framework. Mobile & Desk

- AngularJS in 2010 by Google
- Only one version 1.0
- Has been converted to Angular
- Written in JavaScript

- Angular in 2016, rewrite of AngularJS
- Version 2.x,4.x.5.x,6.x,7 (latest is 7)
- It is released every 6 months time
- Written and coded in TypeScript(superset of JavaScript which helps to build more robust and structured code)

AngularJS	Angular2+
MVC	Service/Controller
Controller Based UI	Component Based UI
SEO for SPA	Feasible for SEO
Web Browser Friendly	Mobile Friendly



Front End Development

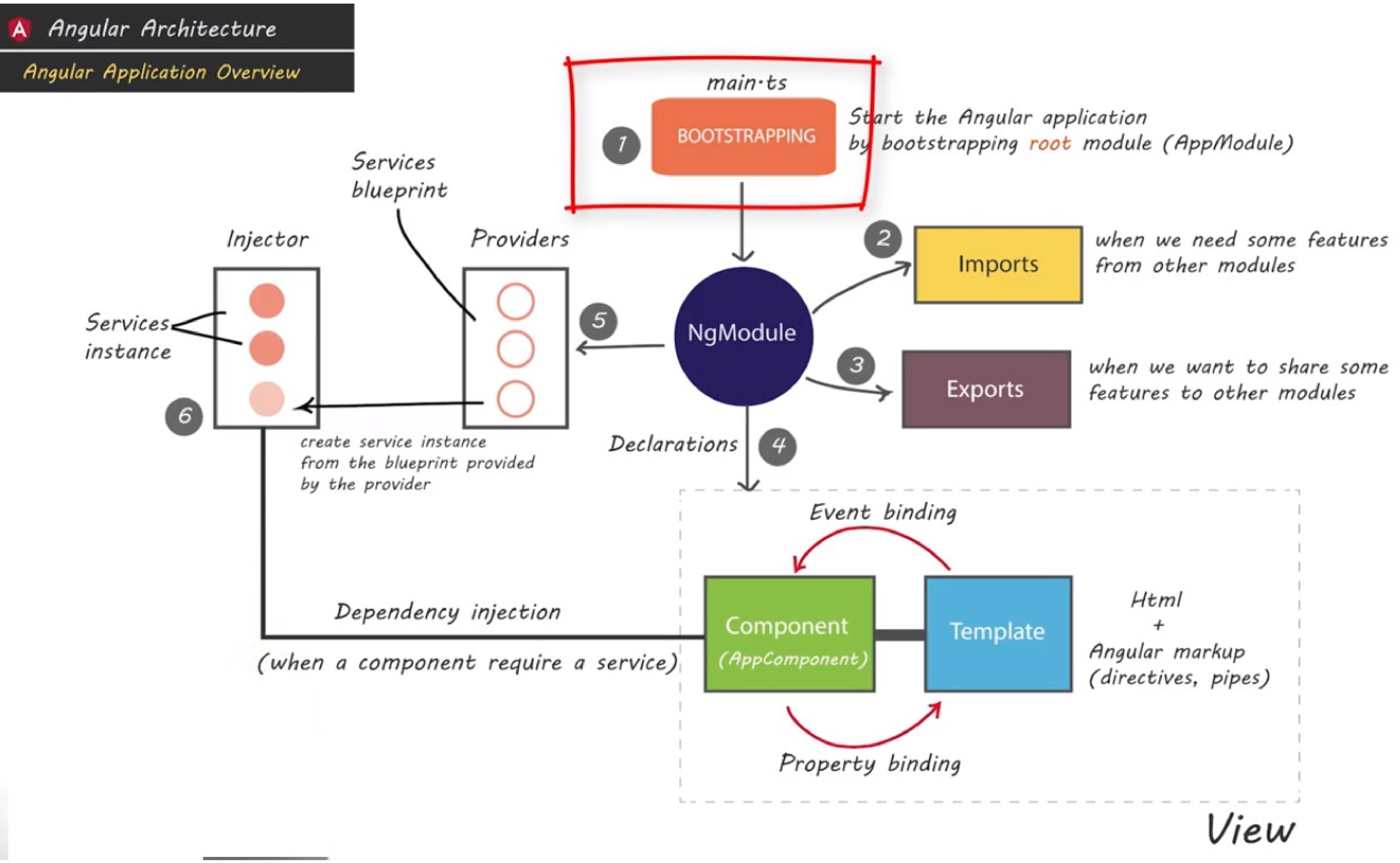
-UI Layer
-Client Side
-Requester for data
-Display/Presentation -Layer
-HTML, CSS, JS, Images, Animations
-Event Handling on Browsers/Mobile

-Back End
-Server Side
-Business Logic Layer
-Database layer
-Responder for data
-Java, PHP, DotNet etc.

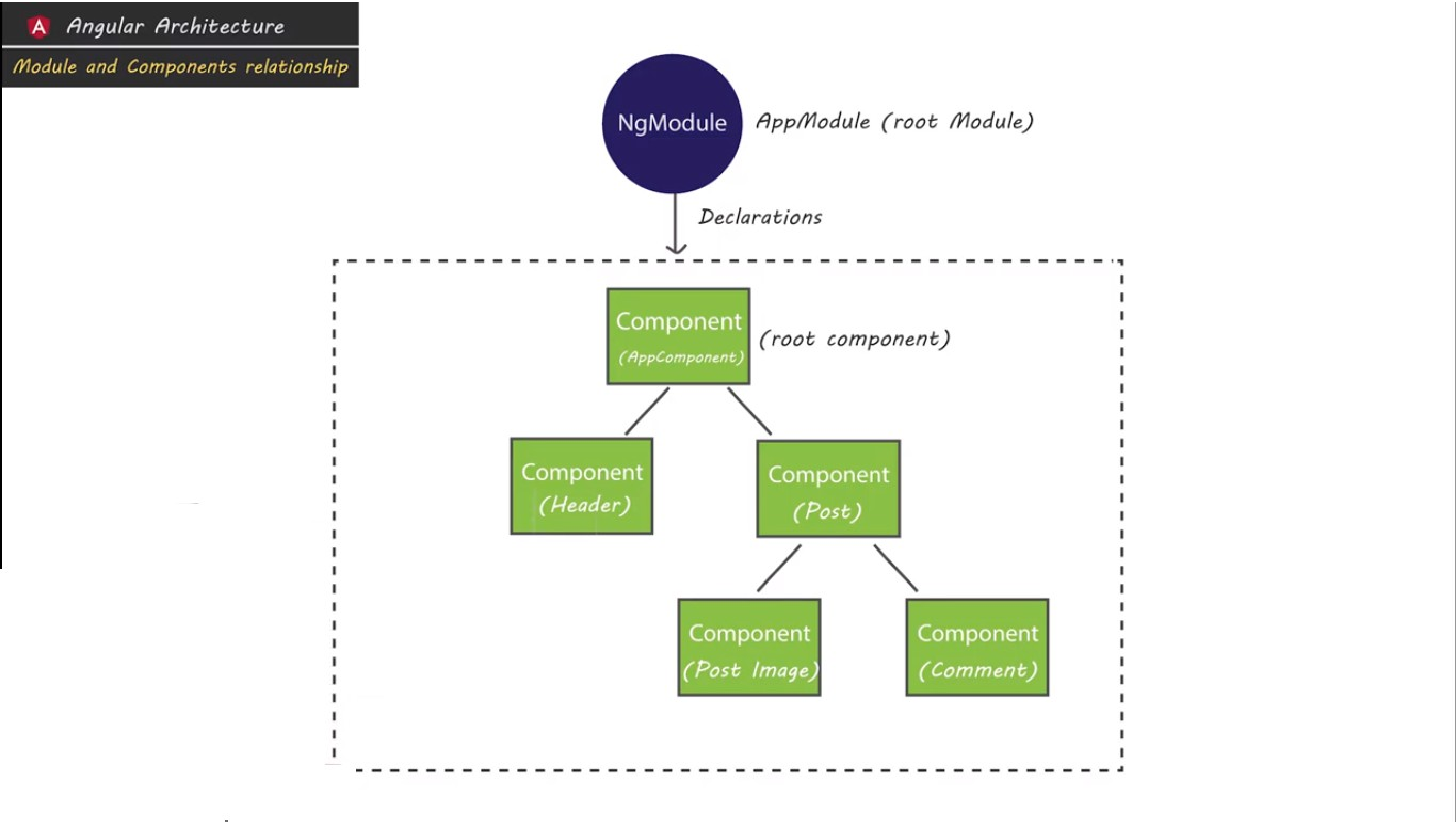
-Angular
-ReactJs
-VueJs

-SPA - Single Page Applications
-Mobile Apps with Ionic, React Native
-Responsive
-High Performance
-Compatible for NodeJs Server Devs

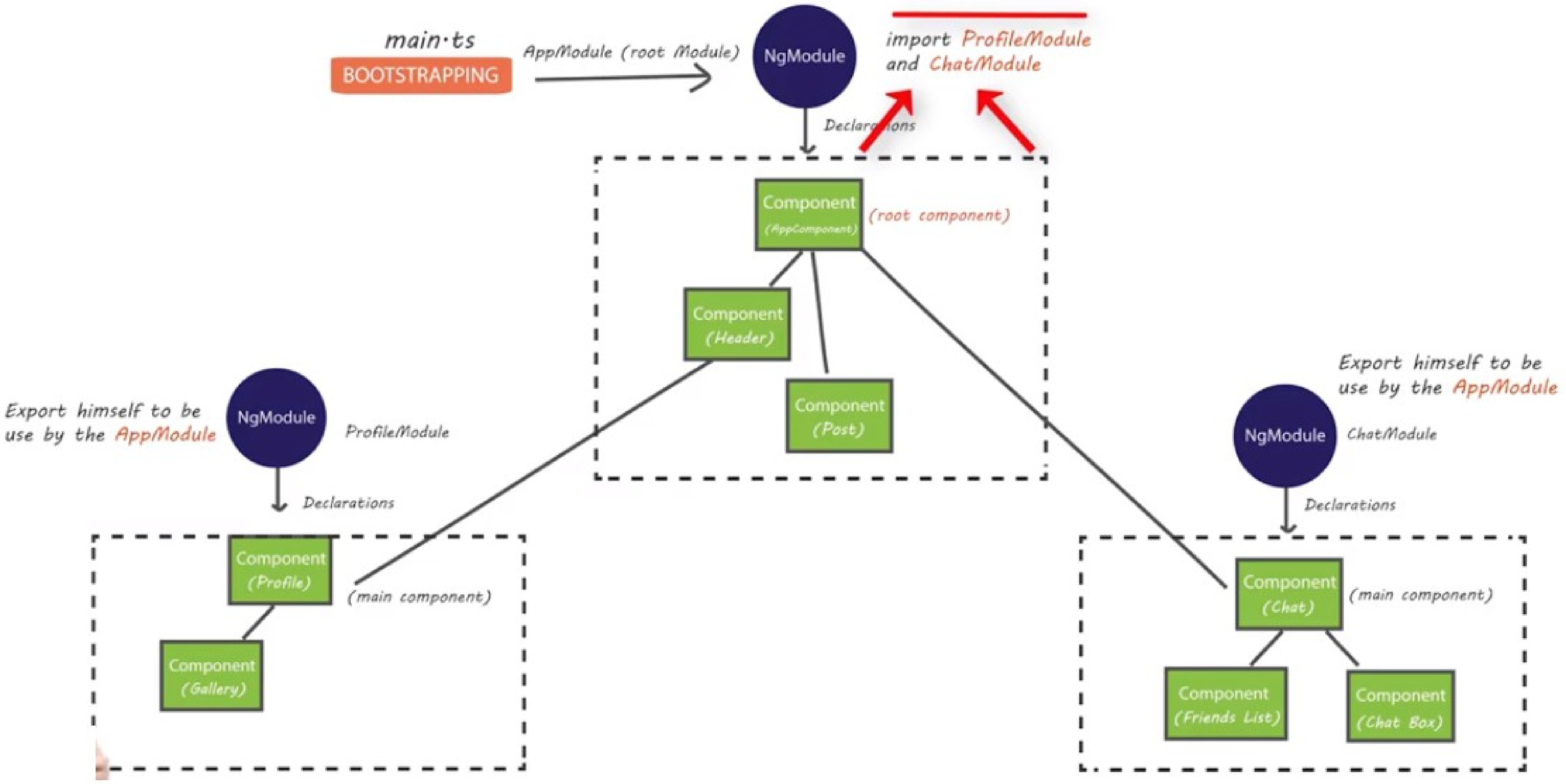
Angular Architecture


Angular Architecture
Angular Application Overview
main.ts
1
BOOTSTRAPPING
Start the Angular application by bootstrapping root module (AppModule)
Services blueprint
Injector
Providers
2
Imports
when we need some features from other modules
Services instance
5
NgModule
3
Exports
when we want to share some features to other modules
6
create service instance from the blueprint provided by the provider
Declarations
4
Event binding
Dependency injection
Component
(AppComponent)
Template
Html
+
Angular markup
(directives, pipes)
(when a component require a service)
Property binding
View

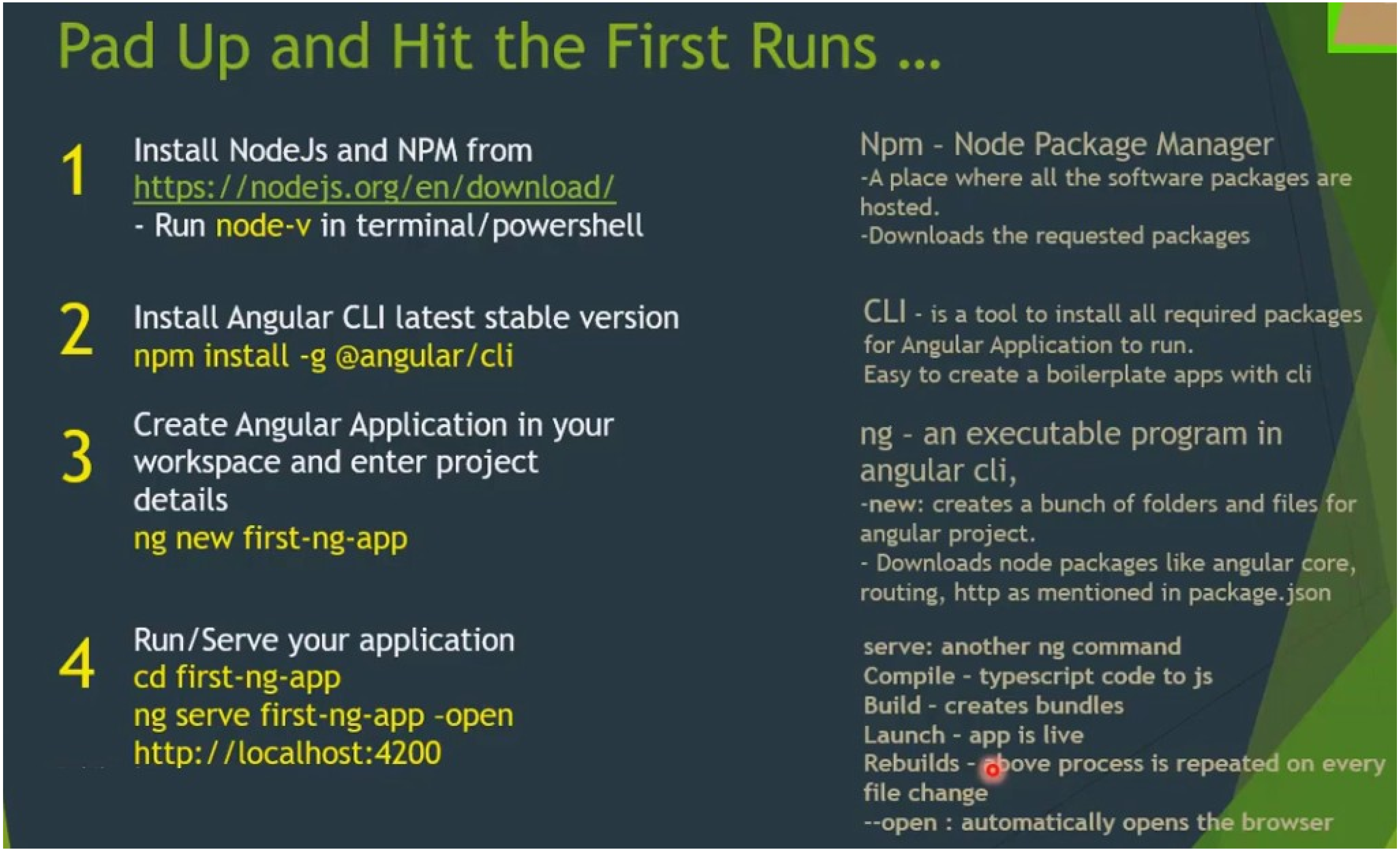
Module and component intraction



Angular modules intraction



Angular 8 hello world



Pad Up and Hit the First Runs ...

1 Install NodeJs and NPM from
https://nodejs.org/en/download/
- Run node-v in terminal/powershell

2 Install Angular CLI latest stable version
npm install -g @angular/cli

3 Create Angular Application in your workspace and enter project details
ng new first-ng-app

4 Run/Serve your application
cd first-ng-app
ng serve first-ng-app -open
http://localhost:4200

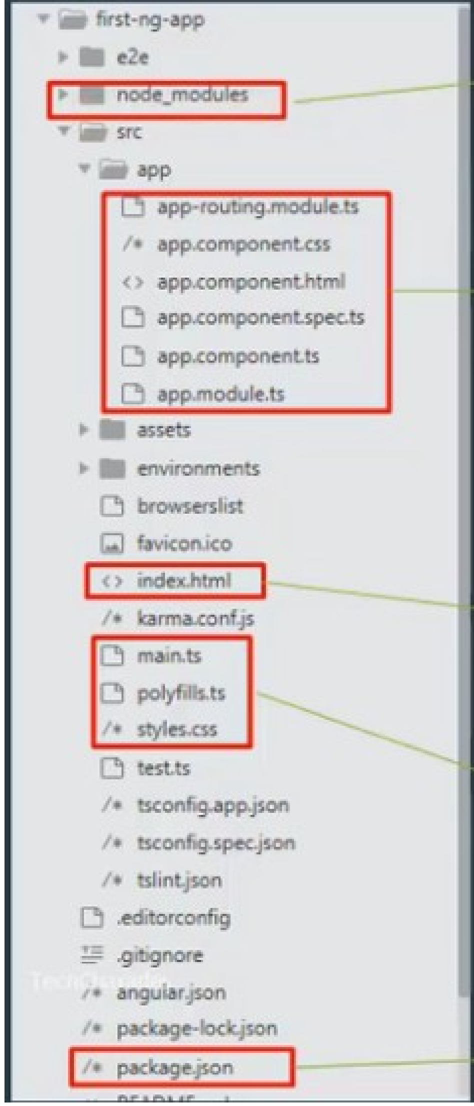
Npm - Node Package Manager
-A place where all the software packages are hosted.
-Downloads the requested packages

CLI - is a tool to install all required packages for Angular Application to run.
Easy to create a boilerplate apps with cli

ng - an executable program in angular cli,
-new: creates a bunch of folders and files for angular project.
- Downloads node packages like angular core, routing, http as mentioned in package.json

serve: another ng command
Compile - typescript code to js
Build - creates bundles
Launch - app is live
Rebuilds - above process is repeated on every file change
--open : automatically opens the browser

Let’s Open the Angular Project Box …



```
first-ng-app
  e2e
  node_modules
  src
    app
      app-routing.module.ts
      app.component.css
      app.component.html
      app.component.spec.ts
      app.component.ts
      app.module.ts
    assets
    environments
    browserslist
    favicon.ico
    index.html
    karma.conf.js
    main.ts
    polyfills.ts
    styles.css
    test.ts
    tsconfig.app.json
    tsconfig.spec.json
    tslint.json
  .editorconfig
  .gitignore
  angular.json
  package-lock.json
  package.json
```

npm downloads and installs the packages in package.json, in this folder

Root component of the app

app.component.ts : Defines the logic and root View for the app's root component, named AppComponent.

app.module.ts : Defines the root module, named AppModule, that tells Angular how to assemble the application.

app.component.spec.ts : Defines a unit test for the root AppComponent

app.component.css and app.component.html are html and css part of root component

index.html: The main HTML page that is served when someone visits your site. The CLI automatically adds all JavaScript and CSS files when building your app, so you typically don't need to add any <script> or<link> tags here manually.

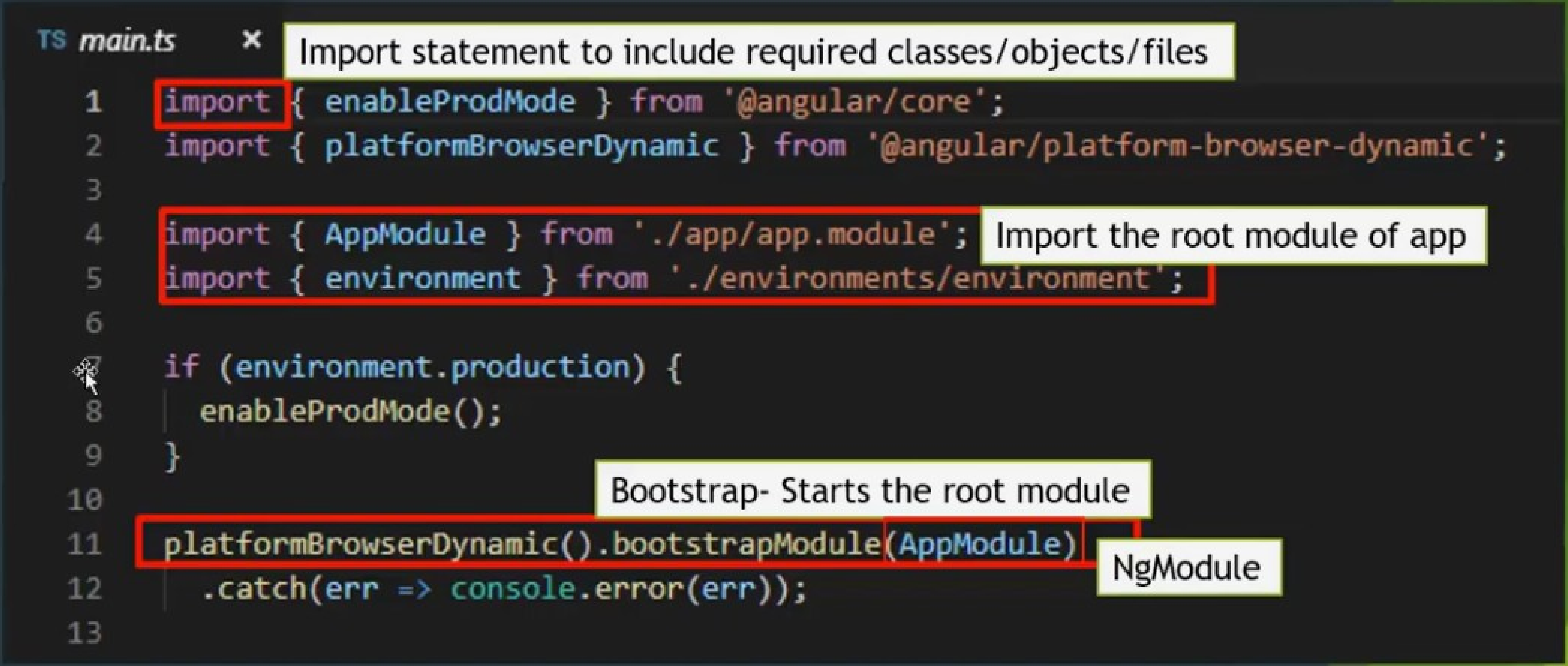
main.ts: 1. Main entry point of the app 2. Compiles the application using JIT | 3. Boots app’s root module to run it in browser

polyfills.js: Provides polyfill scripts for browser support

style.css: Lists CSS files that supply styles for a project

package.json: defines the list of dependencies and packages required to be installed for he app

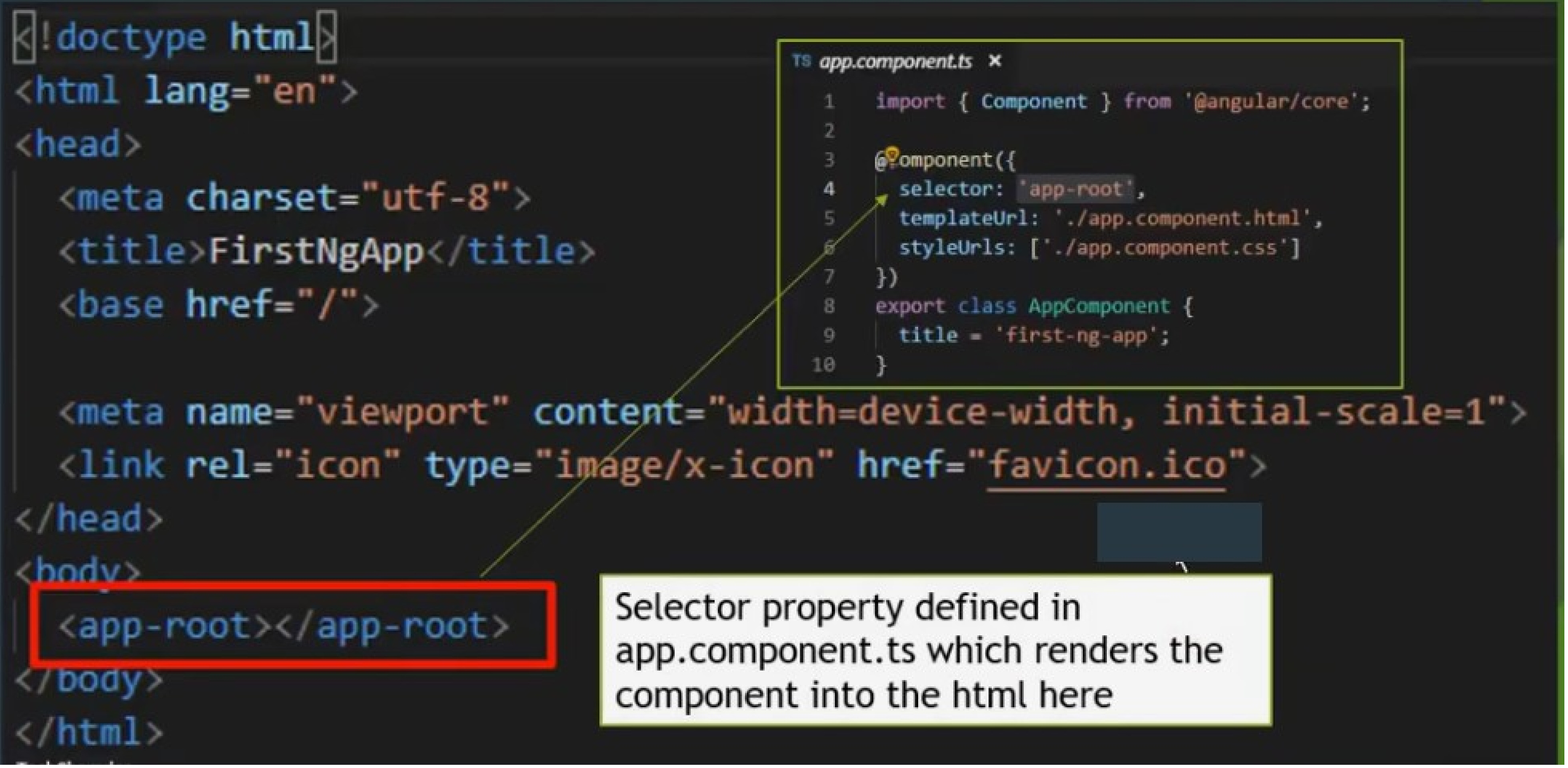
main.ts - understand the entry point



```
import { enableProdMode } from '@angular/core';
import { platformBrowserDynamic } from '@angular/platform-browser-dynamic';

import { AppModule } from './app/app.module';
import { environment } from './environments/environment';

if (environment.production) {
  enableProdMode();
}

platformBrowserDynamic().bootstrapModule(AppModule)
  .catch(err => console.error(err));
```

main.ts
Import statement to include required classes/objects/files
Import the root module of app
Bootstrap- Starts the root module
NgModule

index.html – The master Html



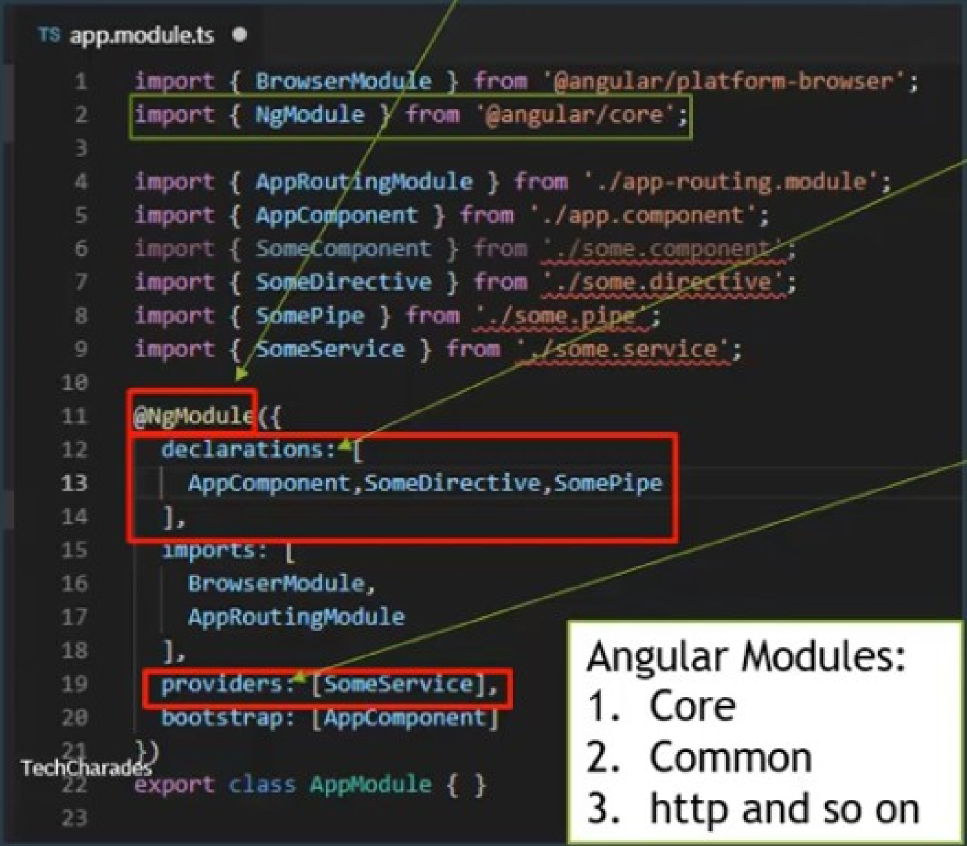
```html
<!doctype html>
<html lang="en">
<head>
  <meta charset="utf-8">
  <title>FirstNgApp</title>
  <base href="/">

  <meta name="viewport" content="width=device-width, initial-scale=1">
  <link rel="icon" type="image/x-icon" href="favicon.ico">
</head>
<body>
  <app-root></app-root>
</body>
</html>
```

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

@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent {
  title = 'first-ng-app';
}
```

Selector property defined in app.component.ts which renders the component into the html here

Lets meet @NgModule - decorator

- The purpose of a NgModule is to declare each thing you create in Angular, and group them together (like Java packages or PHP / C# namespaces).



app.module.ts

```typescript
import { BrowserModule } from '@angular/platform-browser';
import { NgModule } from '@angular/core';

import { AppRoutingModule } from './app-routing.module';
import { AppComponent } from './app.component';
import { SomeComponent } from './some.component';
import { SomeDirective } from './some.directive';
import { SomePipe } from './some.pipe';
import { SomeService } from './some.service';

@NgModule({
  declarations: [
    AppComponent,SomeDirective,SomePipe
  ],
  imports: [
    BrowserModule,
    AppRoutingModule
  ],
  providers: [SomeService],
  bootstrap: [AppComponent]
})
export class AppModule { }
```

"declarations" is for things you'll use in your templates: mainly components (~ views: the classes displaying data)

LOCAL TO the MODULE

Only usable in current module

"providers" is for services (~ models: the classes getting and handling data).

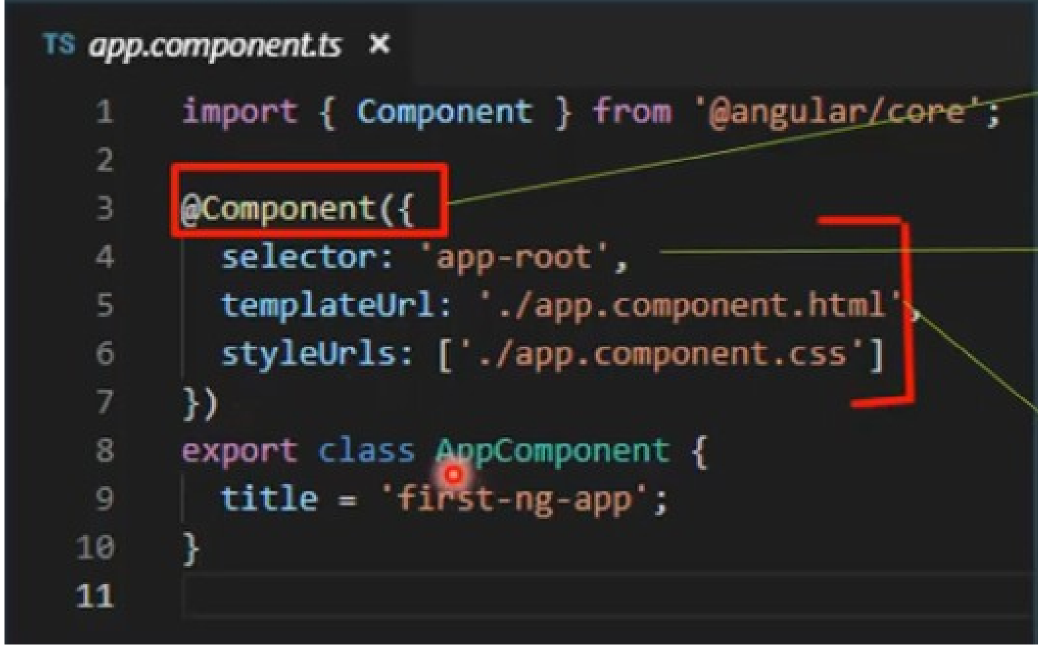
GLOBAL TO ALL MODULES

available / injectable anywhere in your app

Angular Modules:

1. Core
2. Common
3. http and so on

@Component - decorator



app.component.ts

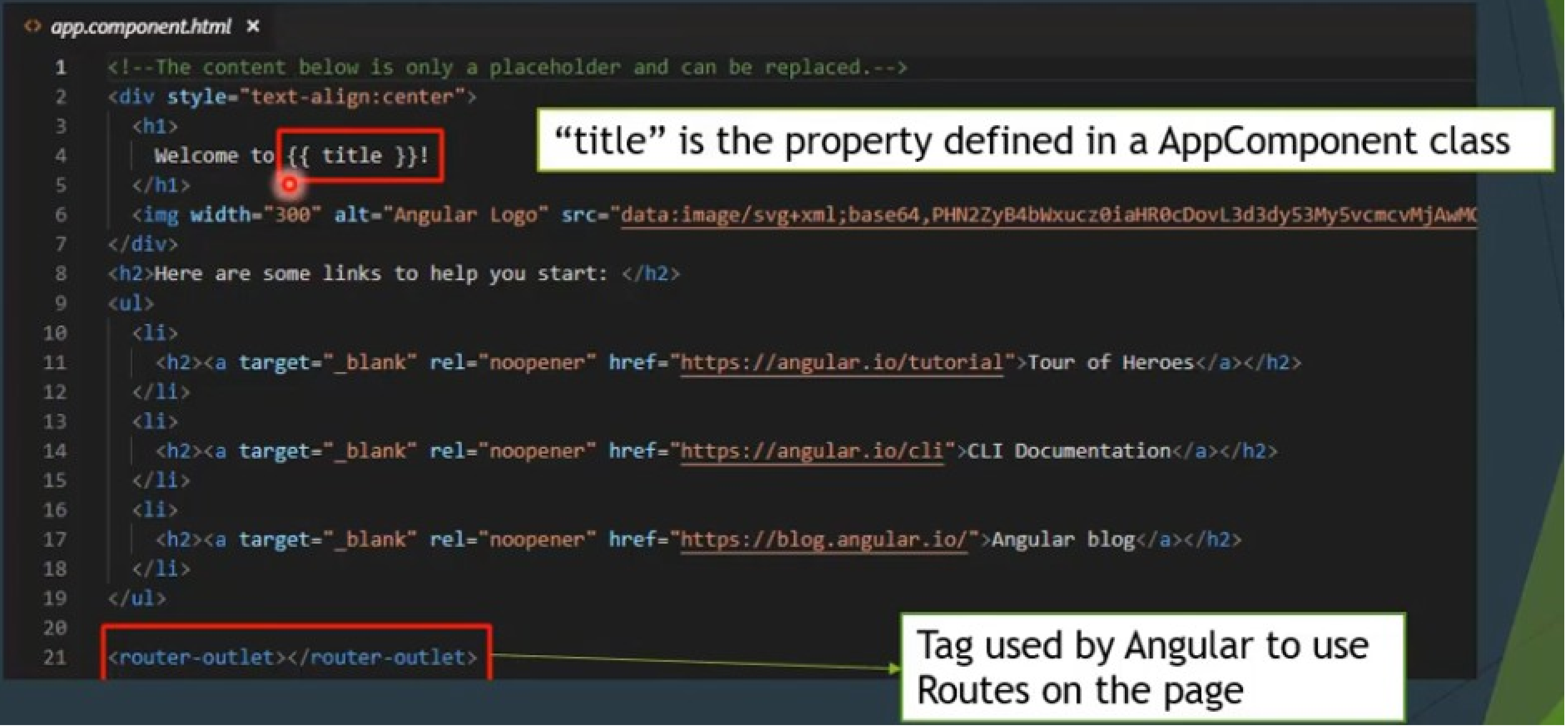
```
import { Component } from '@angular/core';

@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent {
  title = 'first-ng-app';
}
```

that marks a class as an Angular component and provides configuration metadata that determines how the component should be processed, instantiated, and used at runtime.

Tag element in template html file

Template html and css file paths to be associated with this component

App Component Html



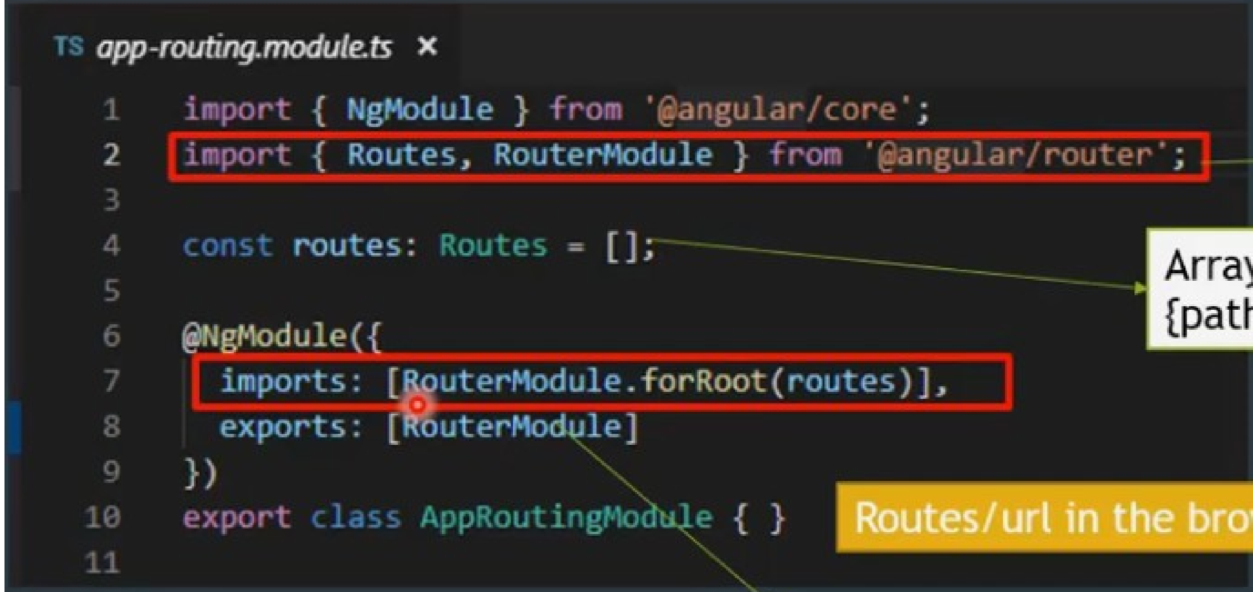
app.component.html

```html
<!--The content below is only a placeholder and can be replaced.-->
<div style="text-align:center">
  <h1>
    Welcome to {{ title }}!
  </h1>
  <img width="300" alt="Angular Logo" src="data:image/svg+xml;base64,PHN2ZyB4bWxucz0iaHR0cDovL3d3dy53My5vcmcvMjAwMC
</div>
<h2>Here are some links to help you start: </h2>
<ul>
  <li>
    <h2><a target="_blank" rel="noopener" href="https://angular.io/tutorial">Tour of Heroes</a></h2>
  </li>
  <li>
    <h2><a target="_blank" rel="noopener" href="https://angular.io/cli">CLI Documentation</a></h2>
  </li>
  <li>
    <h2><a target="_blank" rel="noopener" href="https://blog.angular.io/">Angular blog</a></h2>
  </li>
</ul>

<router-outlet></router-outlet>
```

“title” is the property defined in a AppComponent class

Tag used by Angular to use Routes on the page

Routing Module



app-routing.module.ts

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

const routes: Routes = [];

@NgModule({
  imports: [RouterModule.forRoot(routes)],
  exports: [RouterModule]
})
export class AppRoutingModule { }
```

Import the api for routing

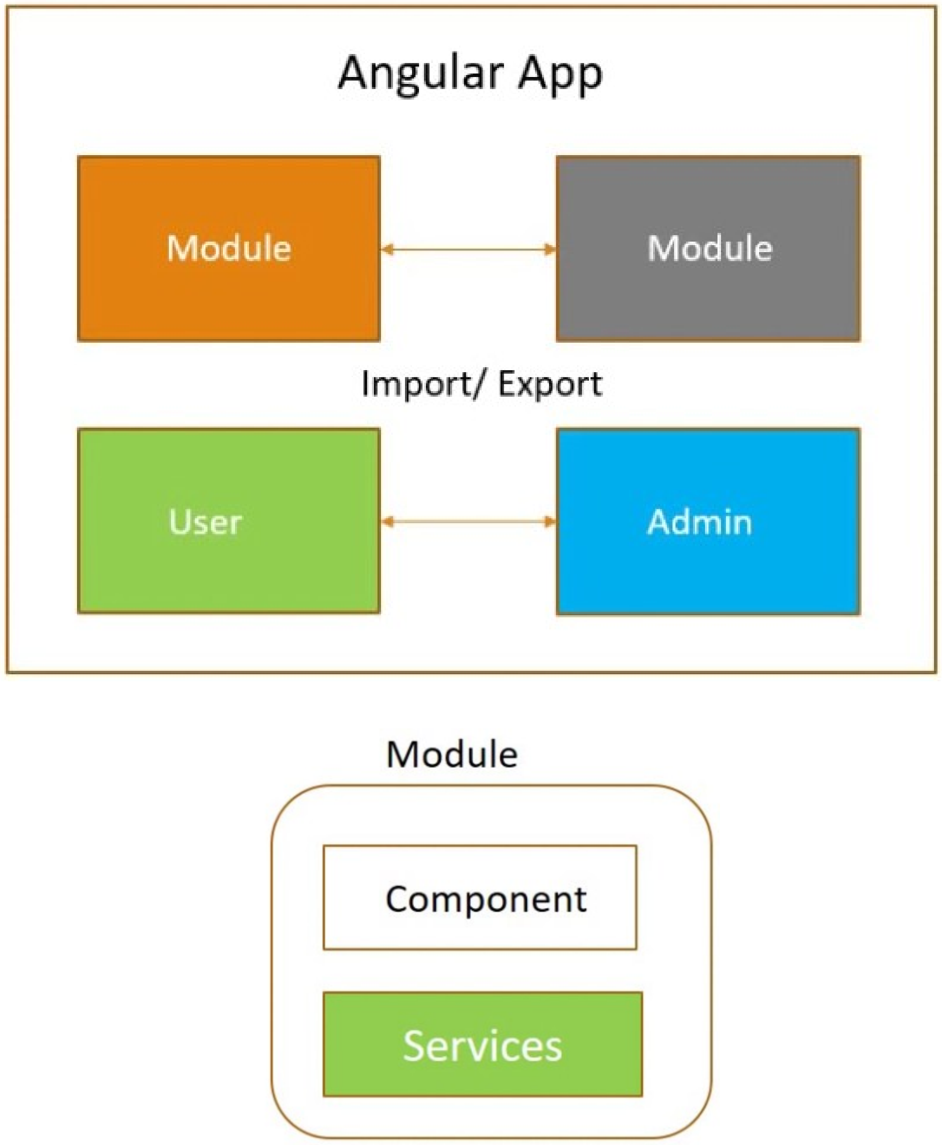
Array of route objects like
{path:'books',component:BooksListComponent}

Routes/url in the browser is mapped with the html Component

To import modules in root module use 'forRoot'.
For other submodules, use 'forChild'

Angular module

module

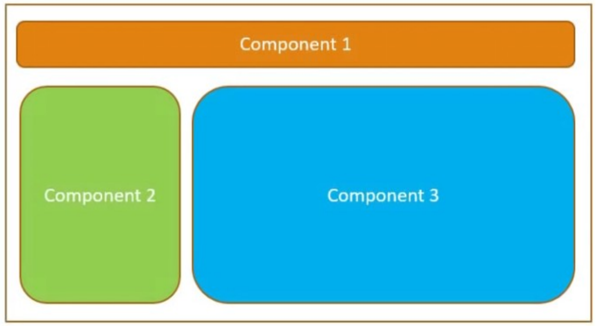


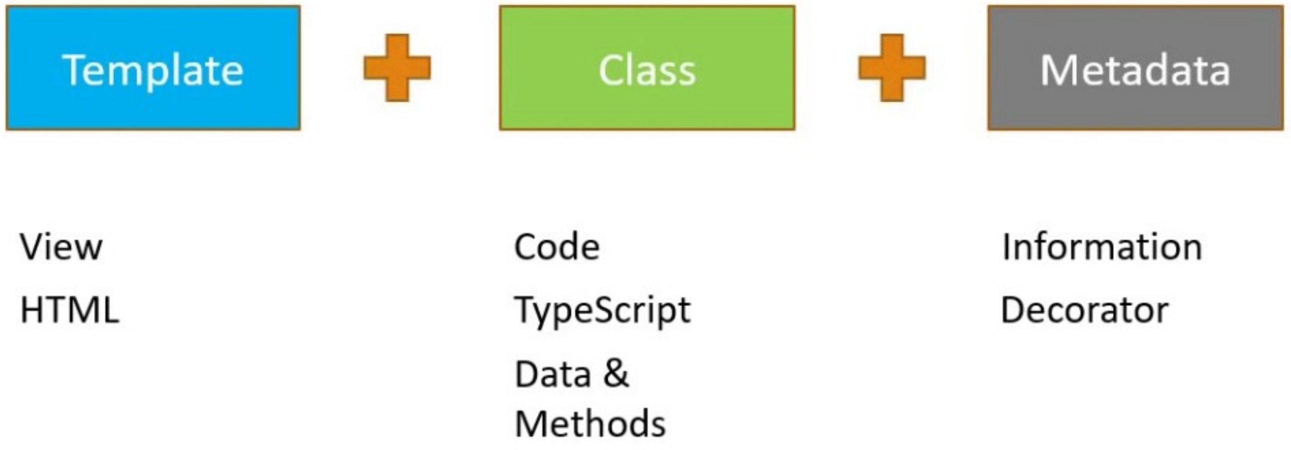
- Angular app is modular in nature
- Module hold other components.
- Every angular project must have at least one module ie root module called **AppModule**
- Each module consist of components and services, pipes etc

Angular components

Angular components

- Components are the most basics building block of UI in angular application
- An component must belong to ngModule
- **A component is a class with Decorators**





Module	Component
A module instead is a collection of components, services, directives, pipes and so on.	A component in Angular is a building block of the Application with an associated template.
Denoted by @ngModule	Denoted by @Component
The Angular apps will contain many modules, each dedicated to the single purpose.	Each component can use other components, which are declared in the same module. To use components declared in other modules, they need to be exported from this module and the module needs to be imported.

Angular app – one or more modules

Module – One or more components and services

Components – HTML + Class

Services – Business logic

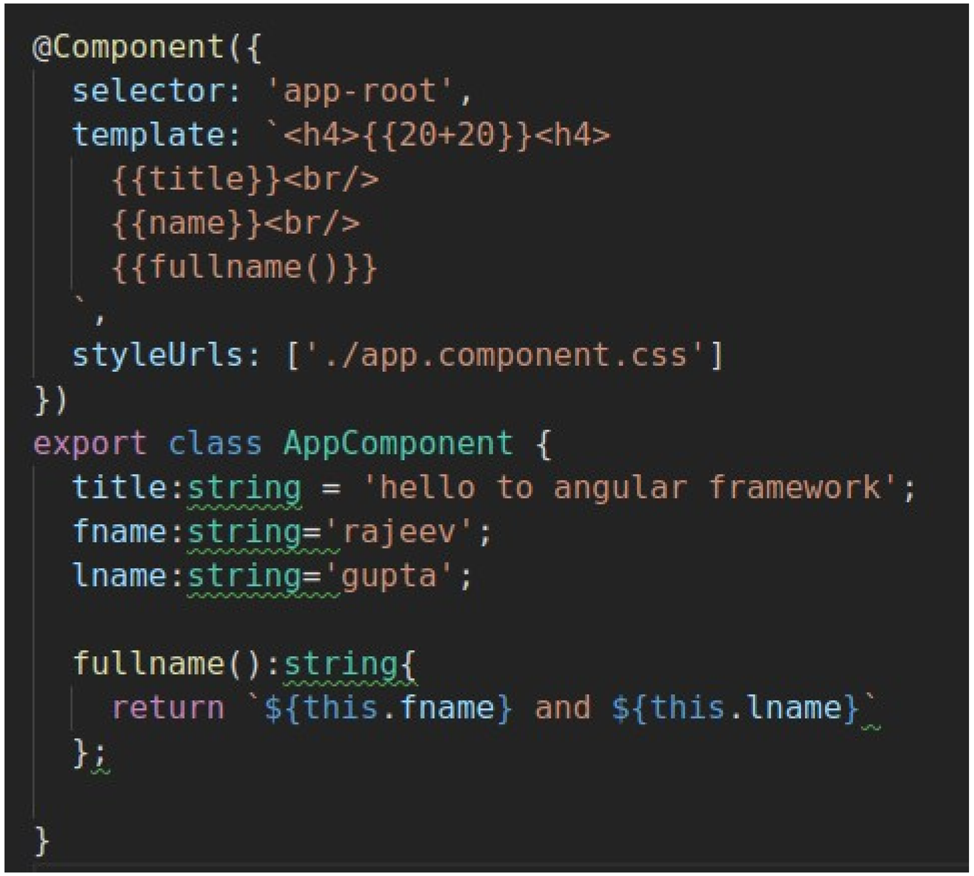
Modules interact and ultimately render the view in the browser

Angular Interpolation

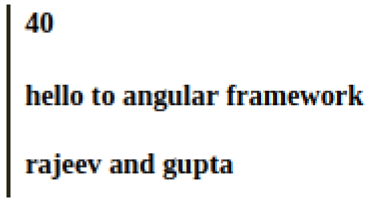
Angular Interpolation

Interpolation use template expression in {{ }} curly braces to display data from component class and template reference variable

It contain templete expression that allow us to read primitive or object or functions return value from component properties



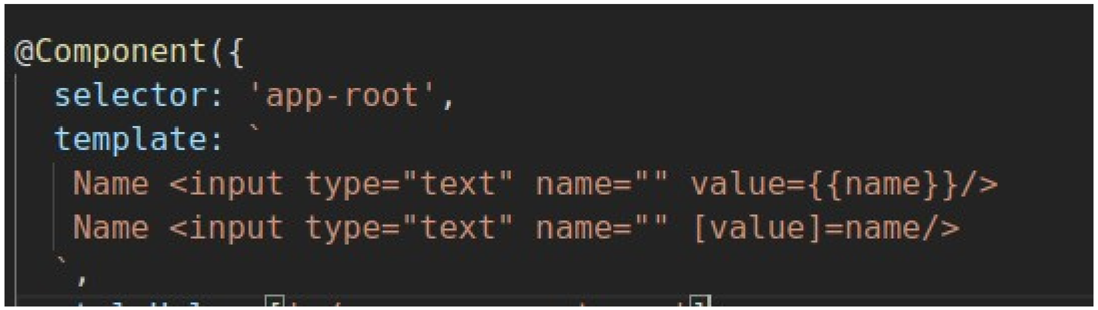
```
@Component({
  selector: 'app-root',
  template: `<h4>{{20+20}}<h4>
    {{title}}<br/>
    {{name}}<br/>
    {{fullname()}}
  `,
  styleUrls: ['./app.component.css']
})
export class AppComponent {
  title:string = 'hello to angular framework';
  fname:string='rajeev';
  lname:string='gupta';

  fullname():string{
    return `${this.fname} and ${this.lname}`
  };

}
```



```
40

hello to angular framework

rajeev and gupta
```

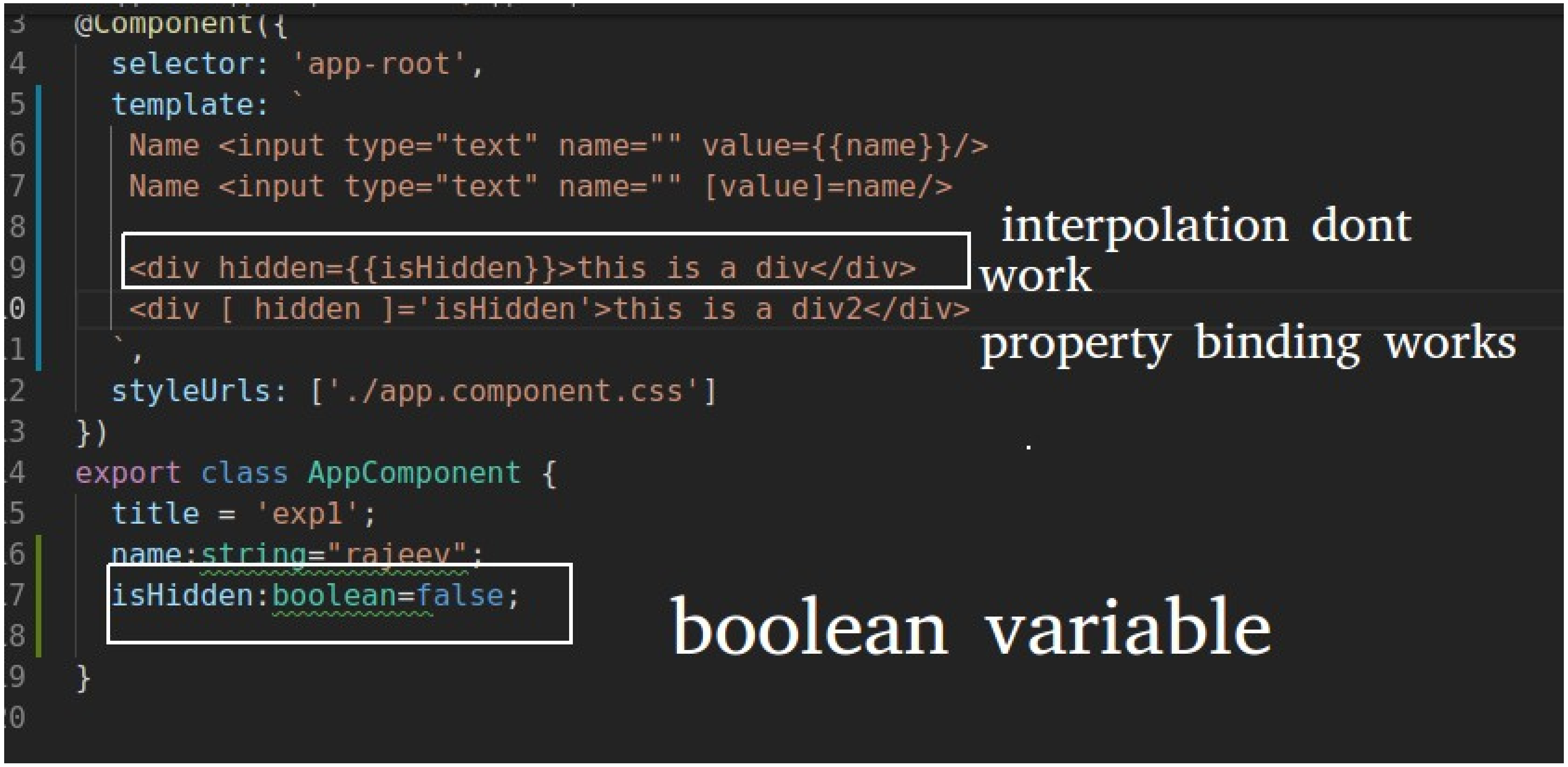
Property binding

- We can use {{}} or property binding [value]=name for one way binding
- Component --> view one way
- Difference:
  - Interpolation is an alternative of Property binding only work for string (not for boolean etc:, hence property binding is better
  - Ex:



```
@Component({
  selector: 'app-root',
  template: `
   Name <input type="text" name="" value={{name}}/>
   Name <input type="text" name="" [value]=name/>
  `,
```

Property binding vs Interpolation

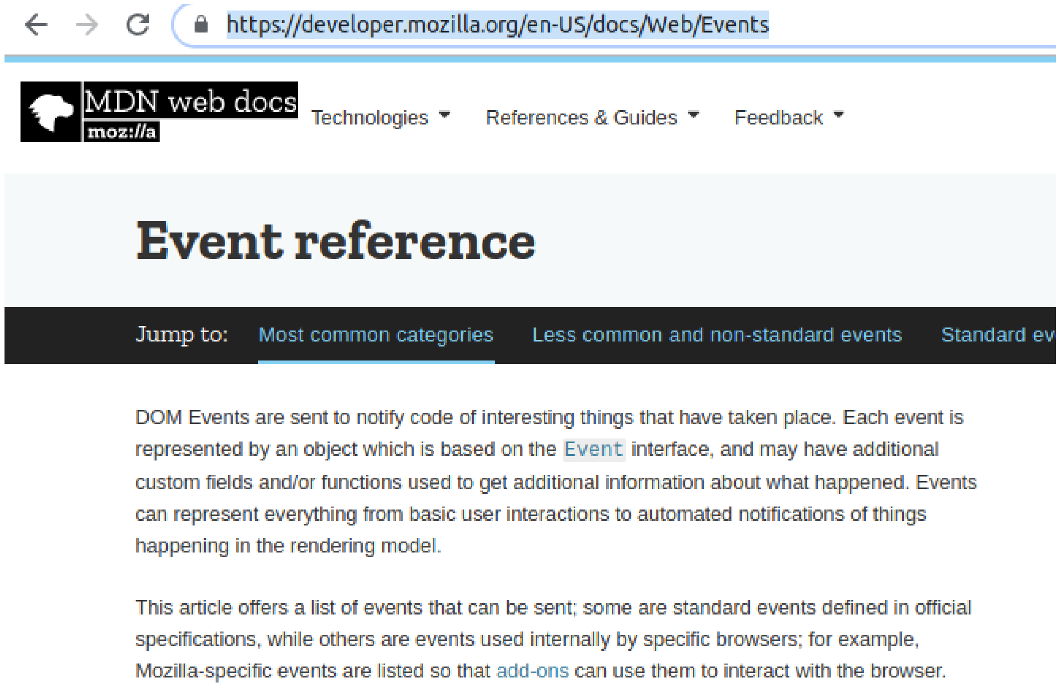


```
@Component({
  selector: 'app-root',
  template: `
   Name <input type="text" name="" value={{name}}/>
   Name <input type="text" name="" [value]=name/>

   <div hidden={{isHidden}}>this is a div</div>
   <div [ hidden ]='isHidden'>this is a div2</div>
  `,
  styleUrls: ['./app.component.css']
})
export class AppComponent {
  title = 'exp1';
  name:string="rajeev";
  isHidden:boolean=false;

}
```

Event binding in Angular

Event binding in Angular

- Handling to user input events using component class is called event handling
- Events supported in angular are enclosed in parenthesis ()
  - https://developer.mozilla.org/en-US/docs/Web/Events

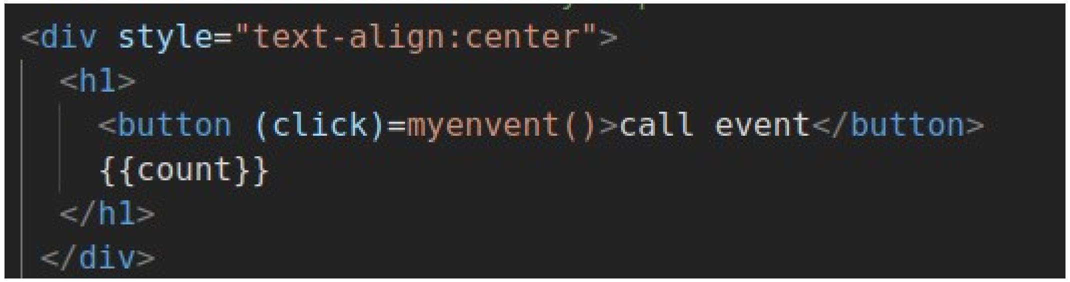


DOM Events are sent to notify code of interesting things that have taken place. Each event is represented by an object which is based on the Event interface, and may have additional custom fields and/or functions used to get additional information about what happened. Events can represent everything from basic user interactions to automated notifications of things happening in the rendering model.

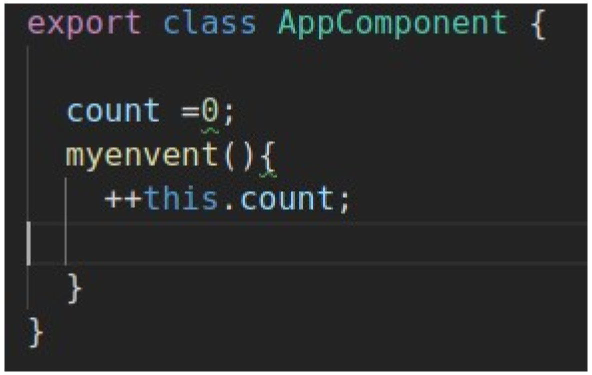
This article offers a list of events that can be sent; some are standard events defined in official specifications, while others are events used internally by specific browsers; for example, Mozilla-specific events are listed so that add-ons can use them to interact with the browser.

Event binding in Angular

- How to handle DOM events?



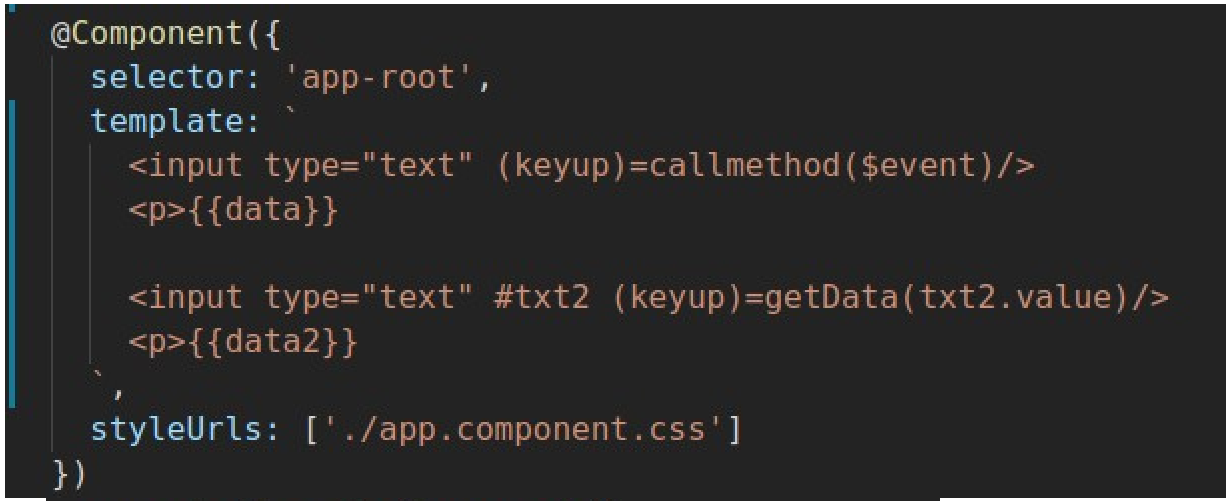
```
<div style="text-align:center">
  <h1>
    <button (click)=myenvent()>call event</button>
    {{count}}
  </h1>
 </div>
```



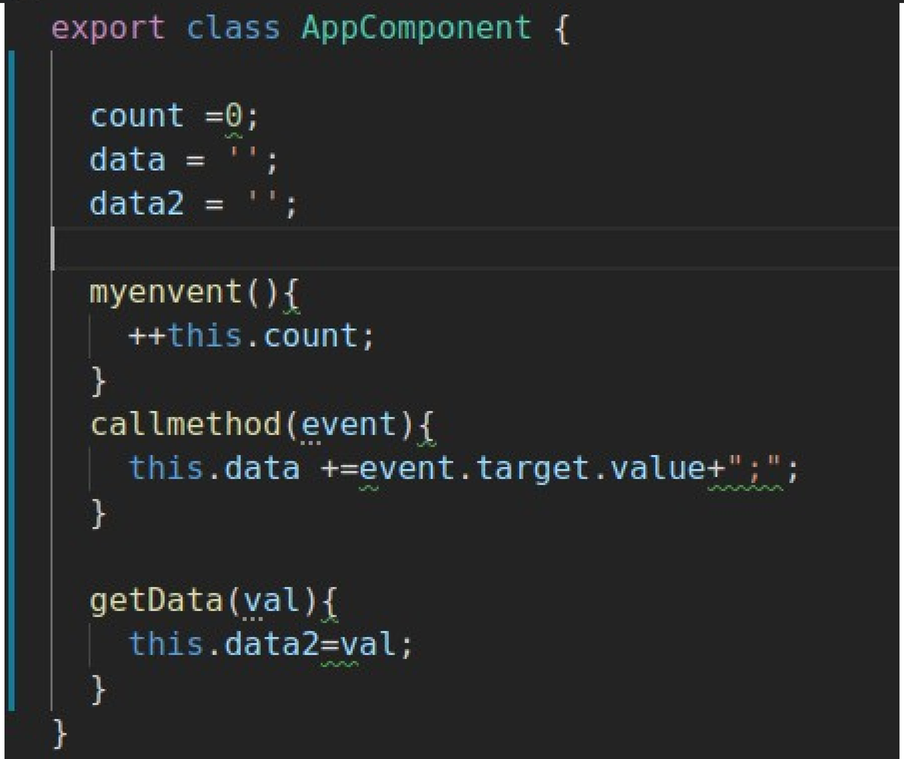
```
export class AppComponent {

  count =0;
  myenvent(){
    ++this.count;

  }
}
```

- How to use $event object?
  - $event is object to get information from view related to DOM in component class

•$event object

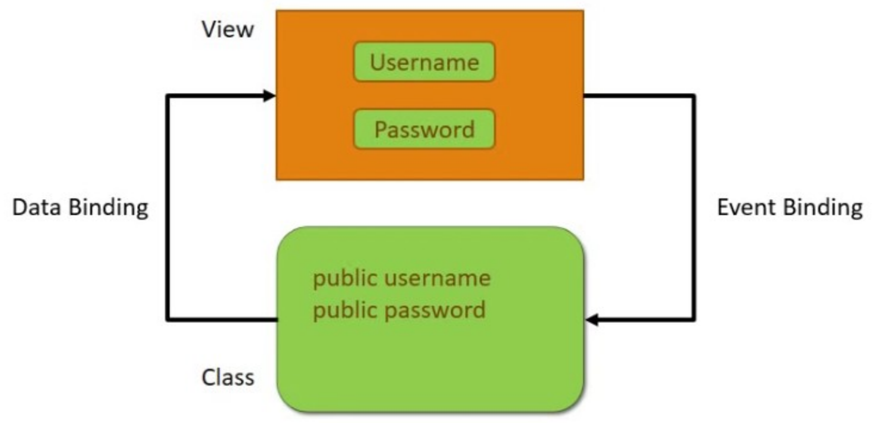


```
@Component({
  selector: 'app-root',
  template: `
    <input type="text" (keyup)=callmethod($event)/>
    <p>{{data}}

    <input type="text" #txt2 (keyup)=getData(txt2.value)/>
    <p>{{data2}}
  `,
  styleUrls: ['./app.component.css']
})
```

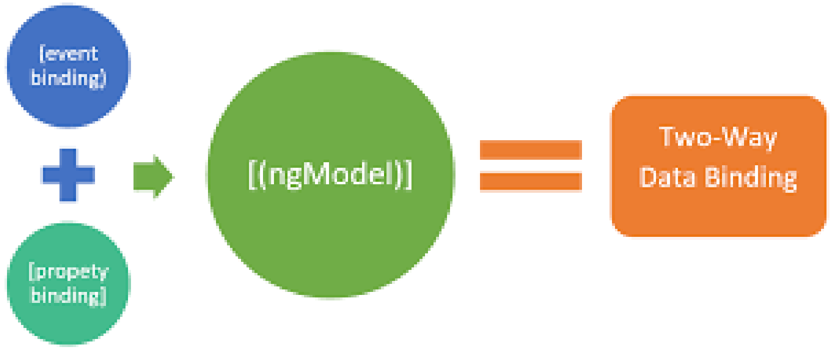


```
export class AppComponent {

  count =0;
  data = '';
  data2 = '';

  myenvent(){
    ++this.count;
  }
  callmethod(event){
    this.data +=event.target.value+";";
  }

  getData(val){
    this.data2=val;
  }
}
```

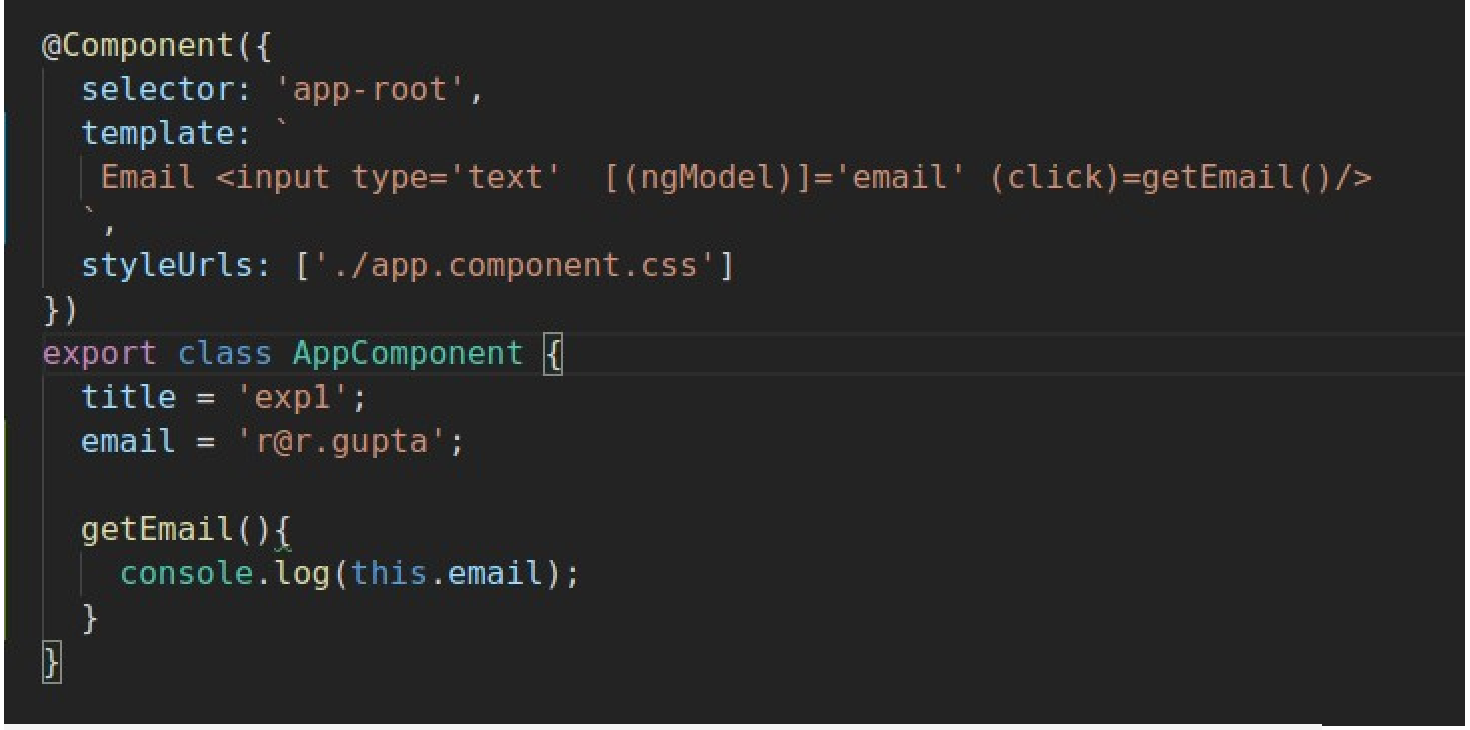
2 Way data binding Angular



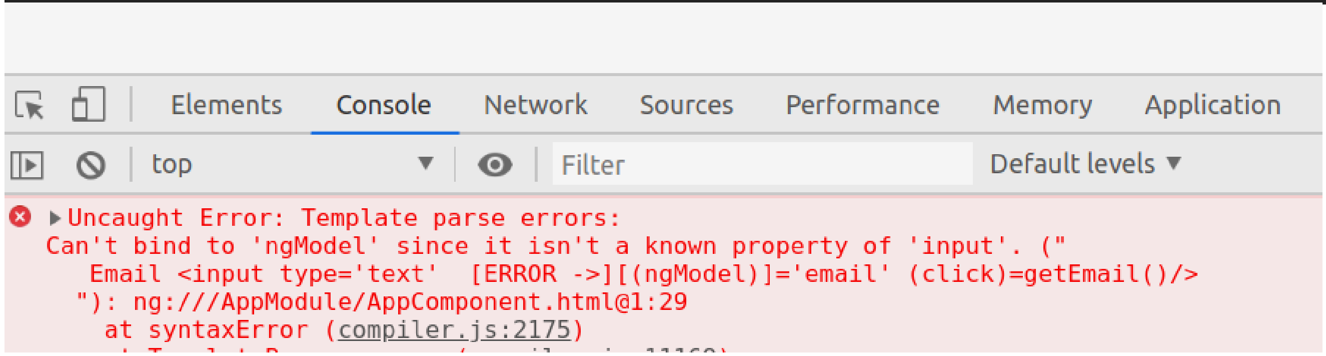
Type of databinding



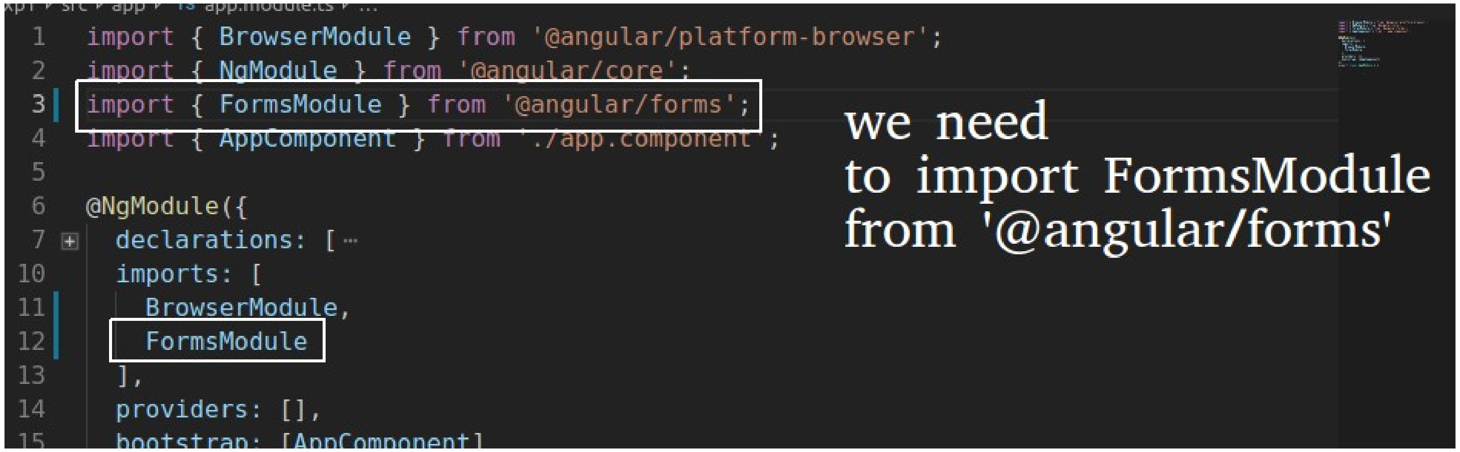
- No binding
  - <input name=”fname”/>
- One way
  - <input name=”fname” [ngModel]=”fname”/>
- Two way
  - <input name=”fname” [ngModel]=”fname” (ngModelChange)=”fname=$event”/>
  - <input name=”fname” [(ngModel)]=”fname”/>



```
@Component({
  selector: 'app-root',
  template: `
   Email <input type='text'  [(ngModel)]='email' (click)=getEmail()/>
  `,
  styleUrls: ['./app.component.css']
})
export class AppComponent {
  title = 'expl';
  email = 'r@r.gupta';

  getEmail(){
    console.log(this.email);
  }
}
```



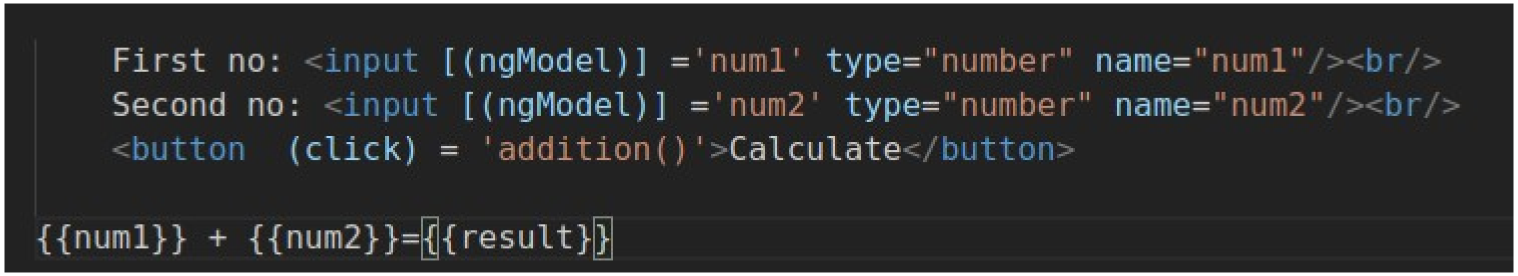
```
▶Uncaught Error: Template parse errors:
Can't bind to 'ngModel' since it isn't a known property of 'input'. ("
   Email <input type='text'  [ERROR ->][(ngModel)]='email' (click)=getEmail()/>
  "): ng:///AppModule/AppComponent.html@1:29
    at syntaxError (compiler.js:2175)
```



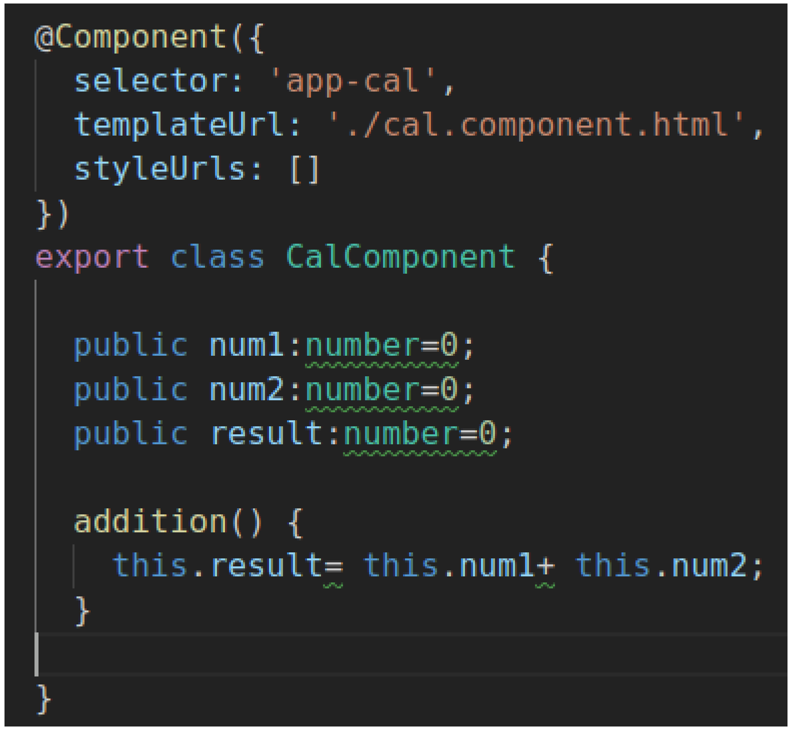
```
import { BrowserModule } from '@angular/platform-browser';
import { NgModule } from '@angular/core';
import { FormsModule } from '@angular/forms';
import { AppComponent } from './app.component';

@NgModule({
  declarations: [⋯
  imports: [
    BrowserModule,
    FormsModule
  ],
  providers: [],
  bootstrap: [AppComponent]
```

we need
to import FormsModule
from '@angular/forms'

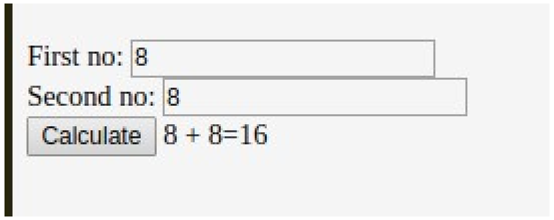
Calculator Example



```
    First no: <input [(ngModel)] ='num1' type="number" name="num1"/><br/>
    Second no: <input [(ngModel)] ='num2' type="number" name="num2"/><br/>
    <button  (click) = 'addition()'>Calculate</button>

{{num1}} + {{num2}}={{result}}
```

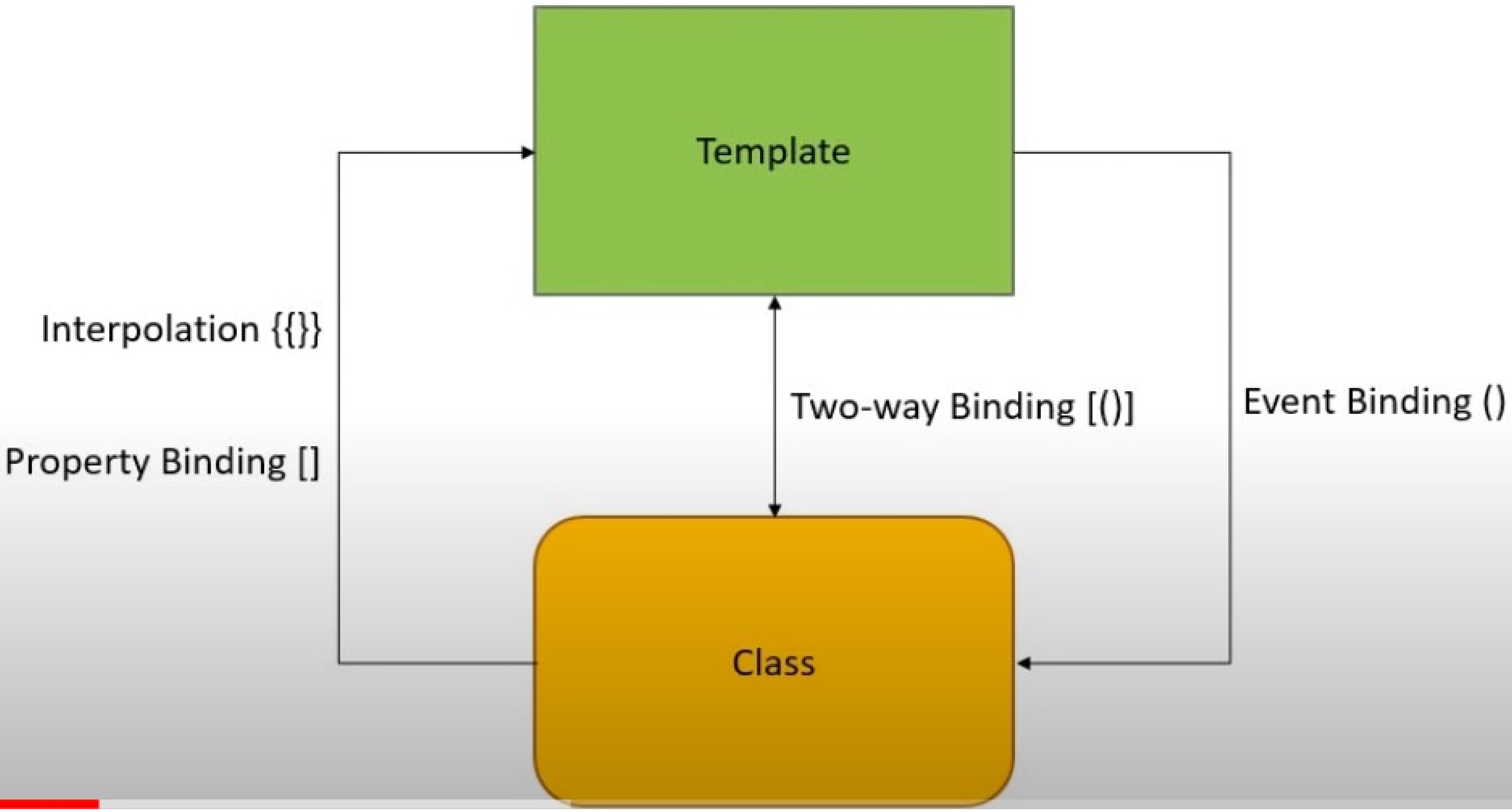


```
@Component({
  selector: 'app-cal',
  templateUrl: './cal.component.html',
  styleUrls: []
})
export class CalComponent {

  public num1:number=0;
  public num2:number=0;
  public result:number=0;

  addition() {
    this.result= this.num1+ this.num2;
  }

}
```



```
First no: 8
Second no: 8
Calculate 8 + 8=16
```

Binding


Template
Interpolation {{}}
Property Binding []
Two-way Binding [()]
Event Binding ()
Class

Component communication

Component communication

- **Parent to child communication**
  - How a child component can send data to the parent component.
  - using @Input decorator,It is used to Passing data from parent to child comp
  - it is done using property binding

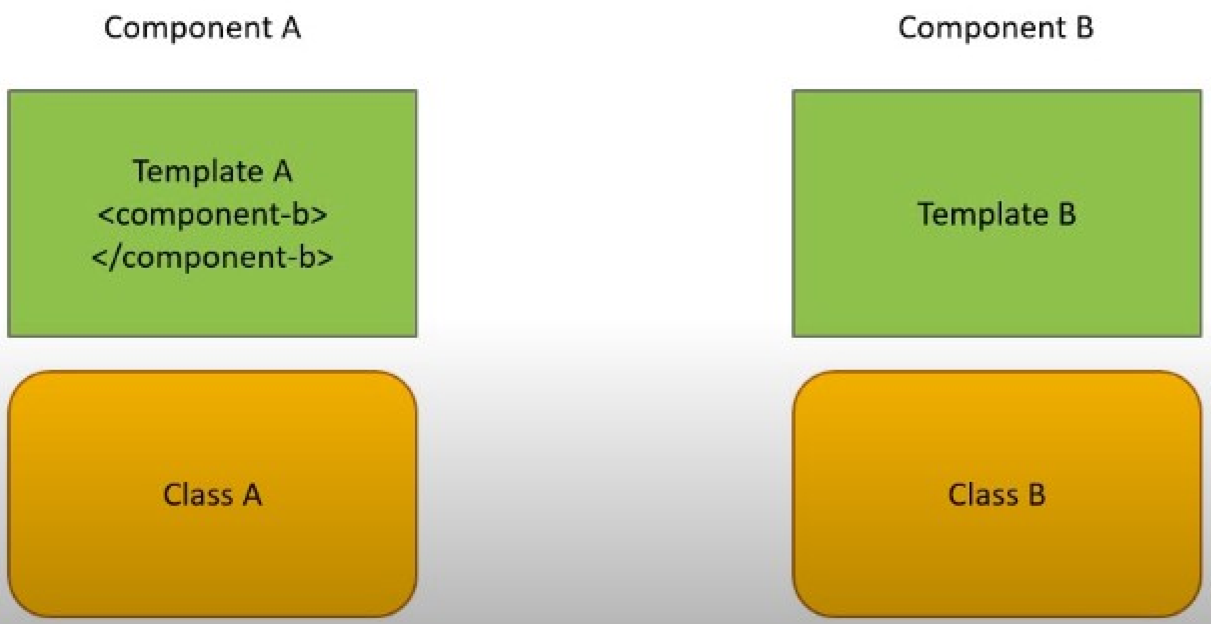
Component communication

- **Child to Parent communication**
  - @Output() Decorator is used to pass data from child to parent via event
  - it is done using event

Intermodule communication

- Create new module that is required by app.module **ng g m mod1**
- Create a component in that module **ng g c mod1/comp1**
- Import mod1 to app.module
- Ask mod1 to exports comp1

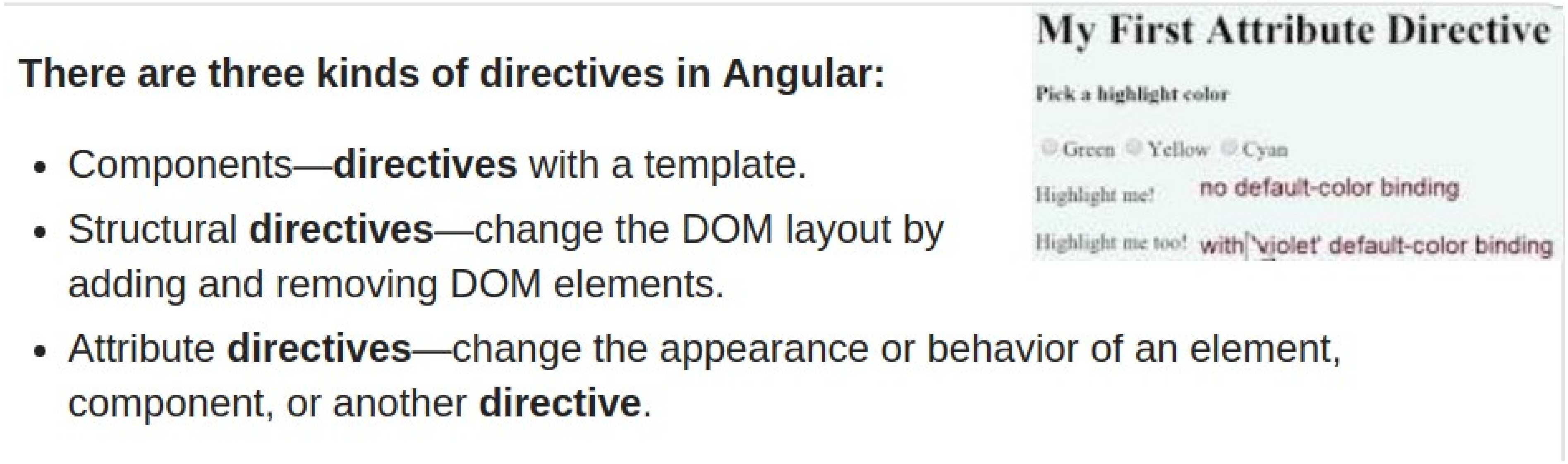
Parent - Child



Directives

Directives

- Directives are instructions in the DOM.
- They specify how to place your components and business logic in the Angular.
- You can define your own directives to attach custom behavior to elements in the DOMa



There are three kinds of directives in Angular:

- Components—**directives** with a template.
- Structural **directives**—change the DOM layout by adding and removing DOM elements.
- Attribute **directives**—change the appearance or behavior of an element, component, or another **directive**.

My First Attribute Directive

Pick a highlight color

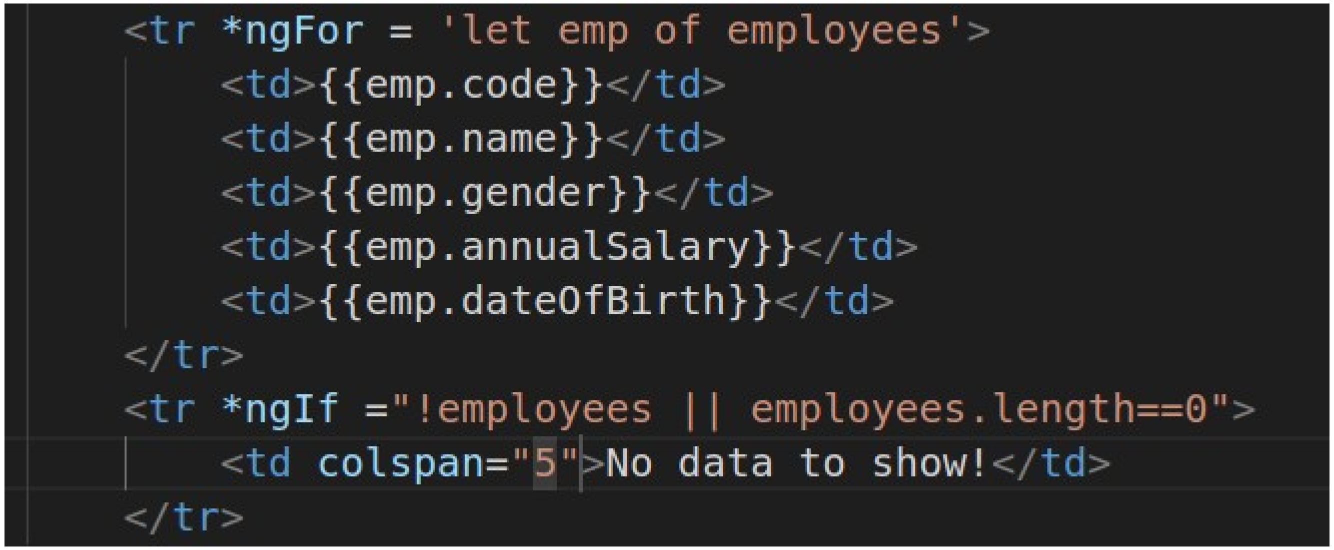
○ Green ○ Yellow ○ Cyan

Highlight me! no default-color binding

Highlight me too! with 'violet' default-color binding

Structural directives

- Add or remove html elements
  - ngIf
  - ngSwitch
  - ngFor



```
<tr *ngFor = 'let emp of employees'>
    <td>{{emp.code}}</td>
    <td>{{emp.name}}</td>
    <td>{{emp.gender}}</td>
    <td>{{emp.annualSalary}}</td>
    <td>{{emp.dateOfBirth}}</td>
</tr>
<tr *ngIf ="!employees || employees.length==0">
    <td colspan="5">No data to show!</td>
</tr>
```

Pipes in angular

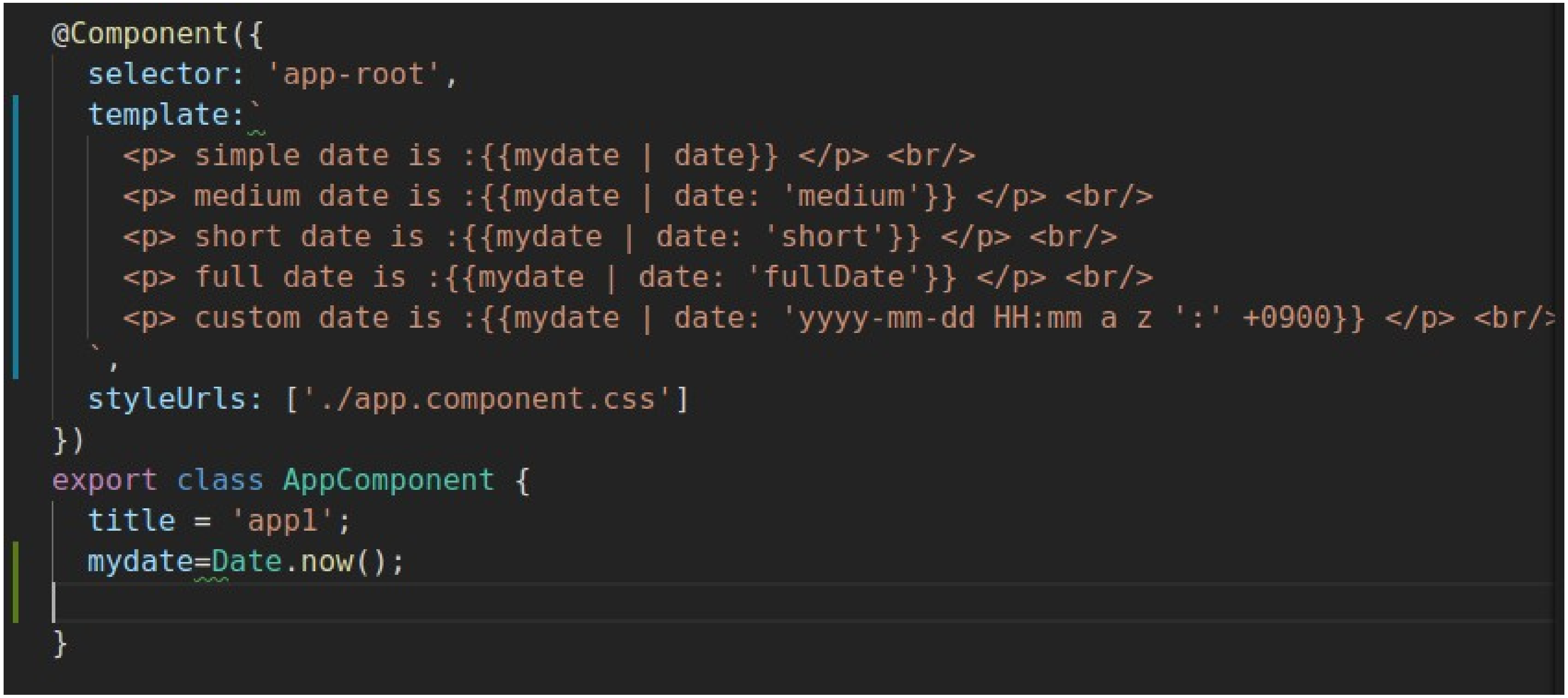
Pipes in angular

- A pipe is used to format the value of an expression displayed in the view
- Angular comes with multiple predefined pipes such as date curreny, lowercase, upper case and others

  Syntex :{{expression | pipeName:inputParam1}}
- If pipe takes multipe inputs then they can be placced one after another separated by a colon :
  - Ex: {{fullname | slice:0:20}}
- pipes can be chained, ie output from one pipe can serve as the input to another pipe
- Common pipes:
  - date: date filter is used to formate the date in a specified manner
  - uppercase and lowercase: chagne the case of input string
  - decimal and percent: based on current brower locale
  - currency: used to format numeric values as a currency based on the current brower locale

  {{14.22 | currency:"USD"}} {{14.2 |currency:"USD":true}}

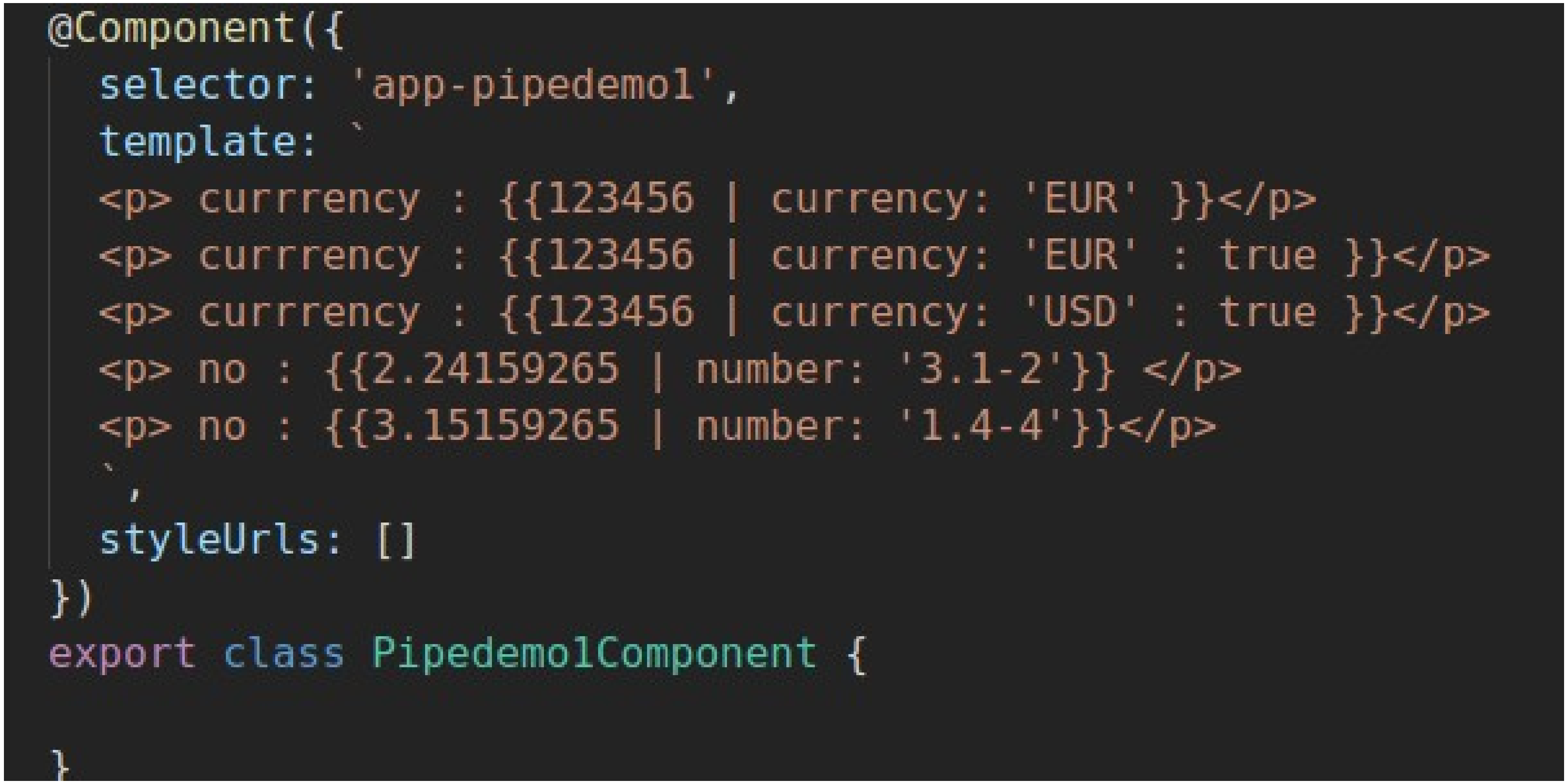
  - json: handy pipe that convert any input into a json string using JSON.stringify



```
@Component({
  selector: 'app-root',
  template:`
    <p> simple date is :{{mydate | date}} </p> <br/>
    <p> medium date is :{{mydate | date: 'medium'}} </p> <br/>
    <p> short date is :{{mydate | date: 'short'}} </p> <br/>
    <p> full date is :{{mydate | date: 'fullDate'}} </p> <br/>
    <p> custom date is :{{mydate | date: 'yyyy-mm-dd HH:mm a z ':' +0900}} </p> <br/>
  `,
  styleUrls: ['./app.component.css']
})
export class AppComponent {
  title = 'app1';
  mydate=Date.now();

}
```

Decimal pipe

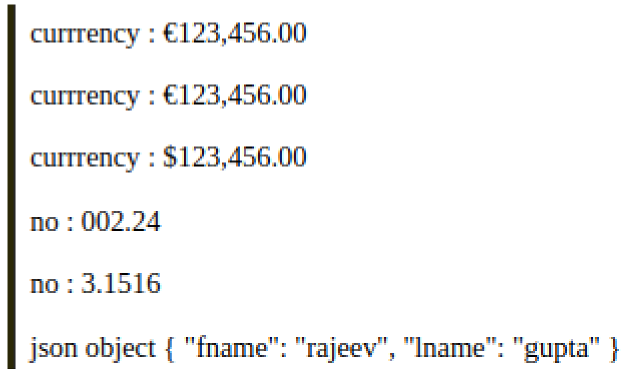
- This pipe is uesd for transformation of decimal numbers
- The first arguement is a formate string of the form
  - "{minIntegerDigits}.{minFractionDigits}{maxFractionDigits}"
  - Ex: {{2.24159265 | number: '3.1-2'}} Output: 002.24
  - Ex: {{3.15159265 | number: '1.4-4'}} Output: 3.1416



```
@Component({
  selector: 'app-pipedemo1',
  template: `
  <p> currrency : {{123456 | currency: 'EUR' }}</p>
  <p> currrency : {{123456 | currency: 'EUR' : true }}</p>
  <p> currrency : {{123456 | currency: 'USD' : true }}</p>
  <p> no : {{2.24159265 | number: '3.1-2'}} </p>
  <p> no : {{3.15159265 | number: '1.4-4'}}</p>
  `,
  styleUrls: []
})
export class Pipedemo1Component {

}
```

JsonPipe

- This transforms a javascript object into a JSON string
  - object : Object ={fname:'rajeev',lname:'gupta'};
  - <p> json objet: {{object | json }} </p>

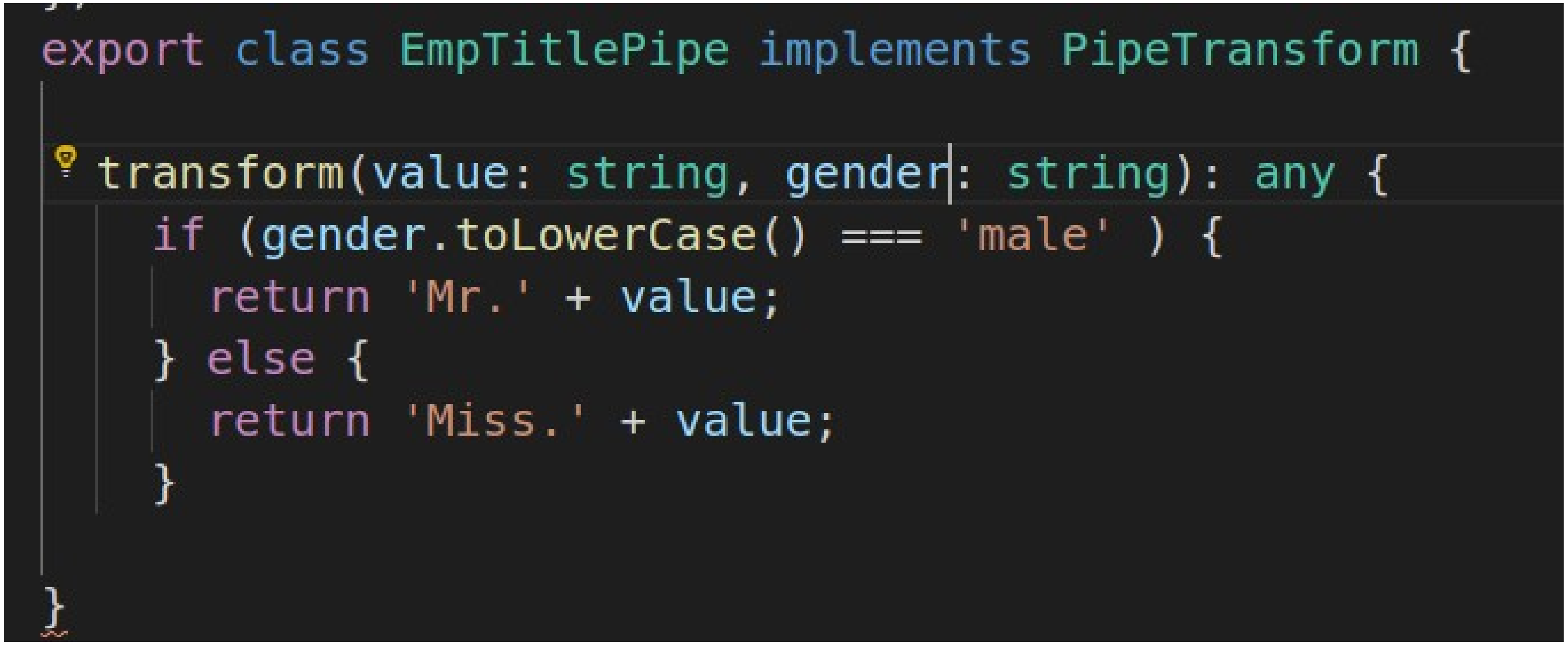


```
currrency : €123,456.00

currrency : €123,456.00

currrency : $123,456.00

no : 002.24

no : 3.1516

json object { "fname": "rajeev", "lname": "gupta" }
```

Custom pipe

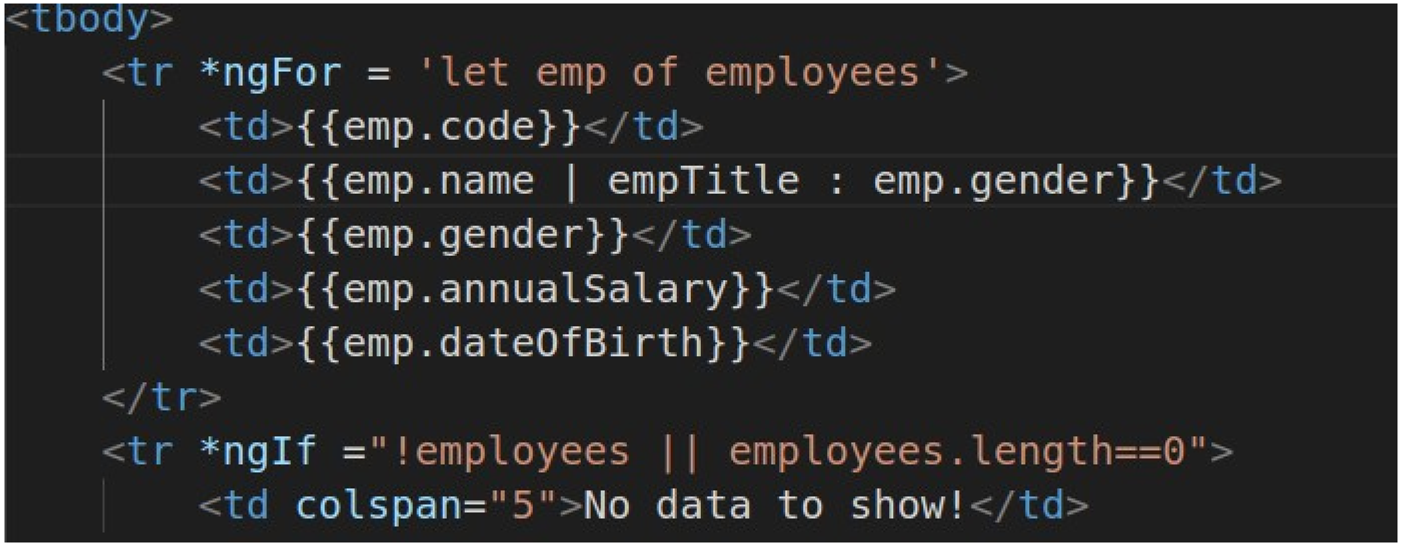
- The pipe is a class that contain "pipe" metadata
- The pipe class contains method "transform", which is implememented from "PipeTransform" interface. This method accepts the value and optionally accept the arguments and convert it to the required formate
- we can add the required argument to the "transform" method
- The @pipe decorator is used to declare pipe and it is defined in core angular library, we we need to import this library. This decorator allow us to define the name of the pipe, which is used in html markup
- The 'transform' method can return any type of value.
- If our pipe's return type is decided on run time, we can use 'any' otherwise we can defind specific type such as number or string

How to define custom pipe?



```
export class EmpTitlePipe implements PipeTransform {

  transform(value: string, gender: string): any {
    if (gender.toLowerCase() === 'male' ) {
      return 'Mr.' + value;
    } else {
      return 'Miss.' + value;
    }

}
```

Using custom pipe

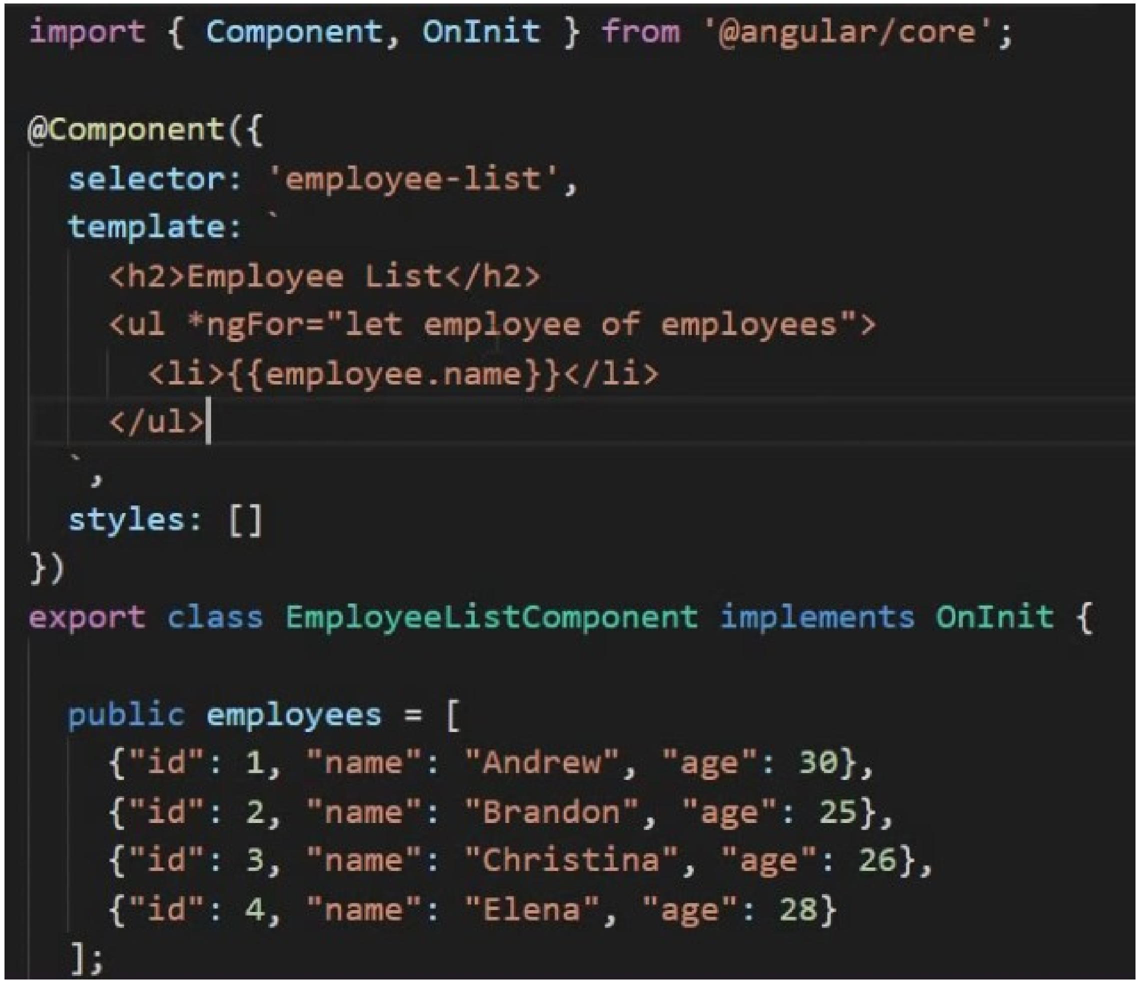


```
<tbody>
    <tr *ngFor = 'let emp of employees'>
        <td>{{emp.code}}</td>
        <td>{{emp.name | empTitle : emp.gender}}</td>
        <td>{{emp.gender}}</td>
        <td>{{emp.annualSalary}}</td>
        <td>{{emp.dateOfBirth}}</td>
    </tr>
    <tr *ngIf ="!employees || employees.length==0">
        <td colspan="5">No data to show!</td>
```

Angular services

Angular services

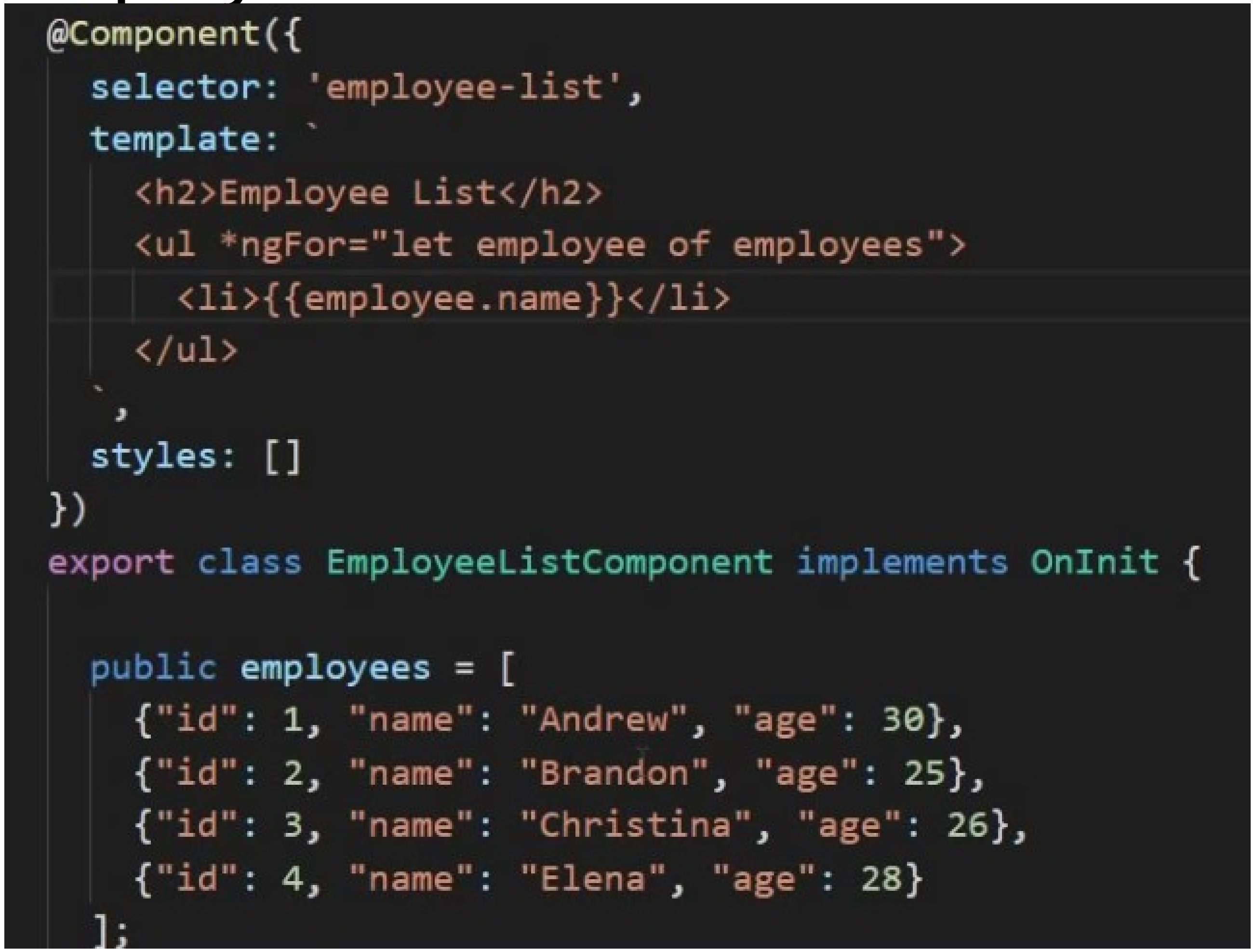
- Let say we need to print emp names on ui



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

@Component({
  selector: 'employee-list',
  template: `
    <h2>Employee List</h2>
    <ul *ngFor="let employee of employees">
      <li>{{employee.name}}</li>
    </ul>
  `,
  styles: []
})
export class EmployeeListComponent implements OnInit {

  public employees = [
    {"id": 1, "name": "Andrew", "age": 30},
    {"id": 2, "name": "Brandon", "age": 25},
    {"id": 3, "name": "Christina", "age": 26},
    {"id": 4, "name": "Elena", "age": 28}
  ];
```

Angular services

- Let say now we need to display whole employee details



```
@Component({
  selector: 'employee-list',
  template: `
    <h2>Employee List</h2>
    <ul *ngFor="let employee of employees">
      <li>{{employee.name}}</li>
    </ul>
  `,
  styles: []
})
export class EmployeeListComponent implements OnInit {

  public employees = [
    {"id": 1, "name": "Andrew", "age": 30},
    {"id": 2, "name": "Brandon", "age": 25},
    {"id": 3, "name": "Christina", "age": 26},
    {"id": 4, "name": "Elena", "age": 28}
  ];
```

Angular services

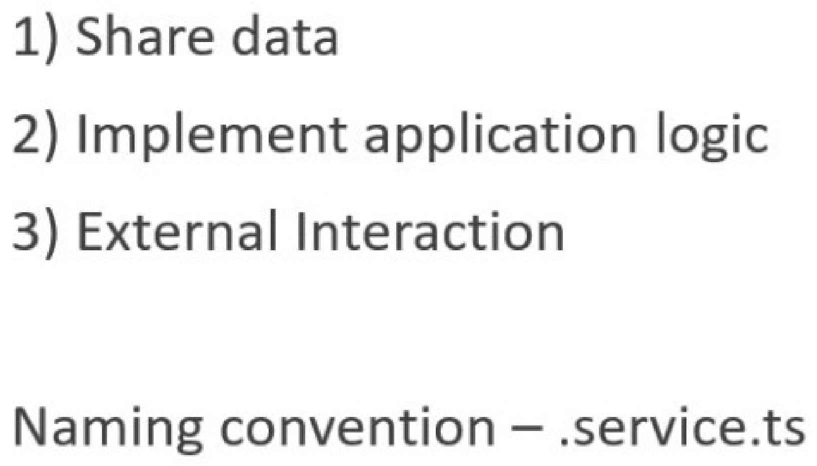
- We are doing copy paste of employee data in both component, that is not good.

Do Not Repeat Yourself (DRY)

Single Responsibility Principle

Angular service

- Problem with previous code is that employee array is local to both component
- Service is a class a specific classs with specific purpose, service can provide data to components
- We put application logic to angular service layer
- We want to seprate concern of component and service layer



1) Share data

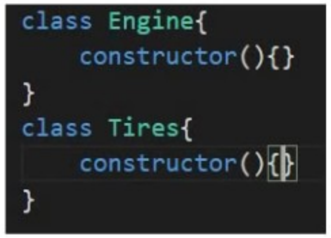
2) Implement application logic

3) External Interaction

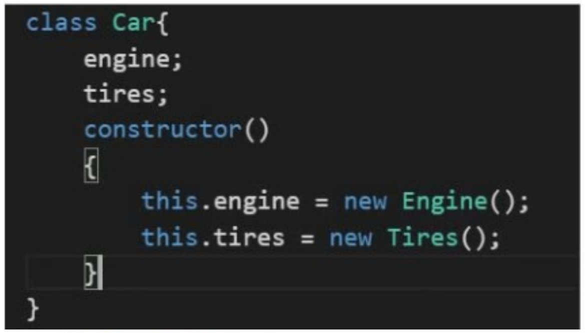
Naming convention – .service.ts

Dependency Injection

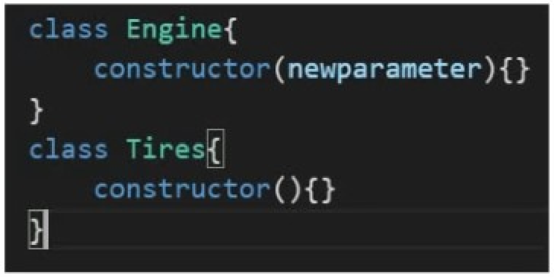
- Code without DI
  - Any time Engine, Tires class changes we need to change Car class
  - Testing is difficult if we want to chane the type of Engine and Tires
  - DI help us to solve this problem



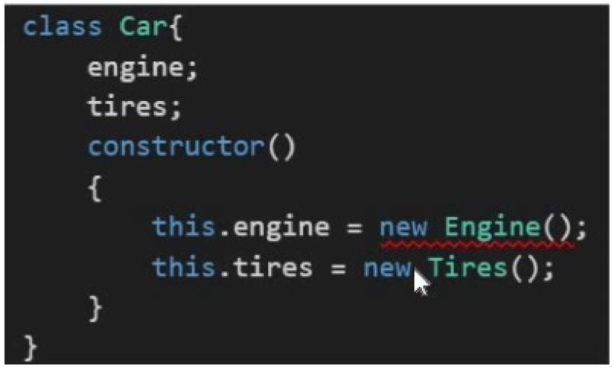
```
class Engine{
    constructor(){}
}
class Tires{
    constructor(){}
}
```



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



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

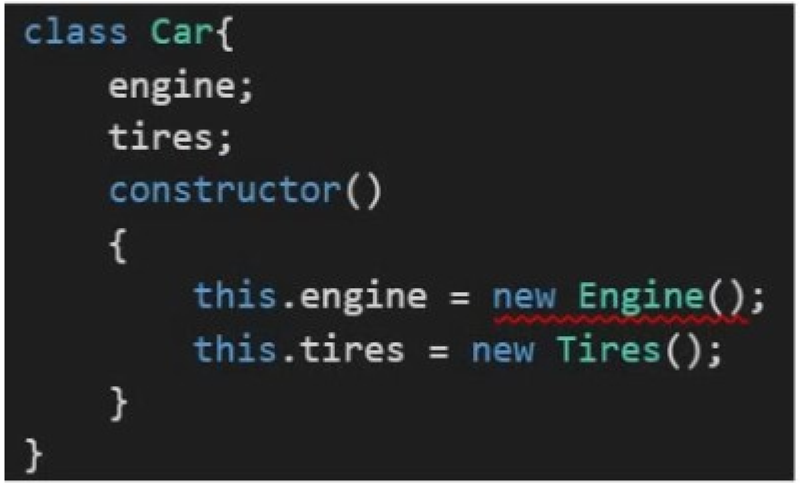


```
class Car{
    engine;
    tires;
    constructor()
    {
        this.engine = new Engine();
        this.tires = new Tires();
    }
}
```

Dependency Injection

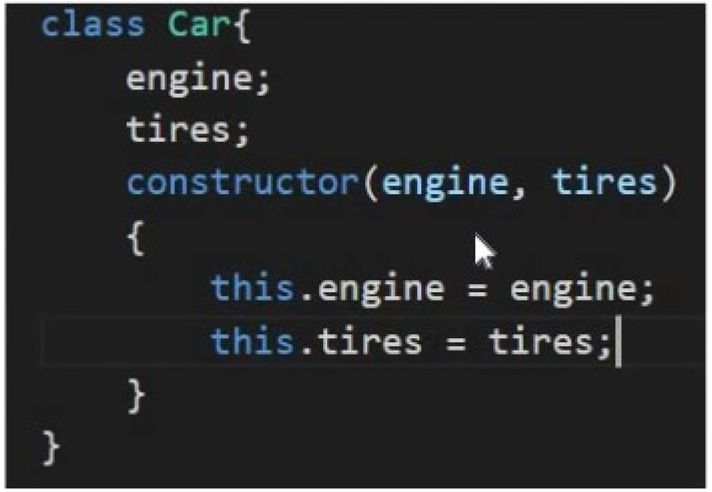
DI is a coding pattern in which a class receives its dependencies from external sources rather than creating them itself.

Without DI



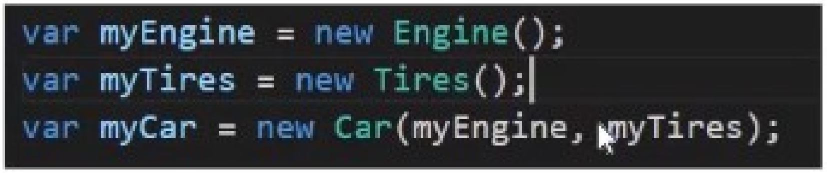
```
class Car{
    engine;
    tires;
    constructor()
    {
        this.engine = new Engine();
        this.tires = new Tires();
    }
}
```

With DI

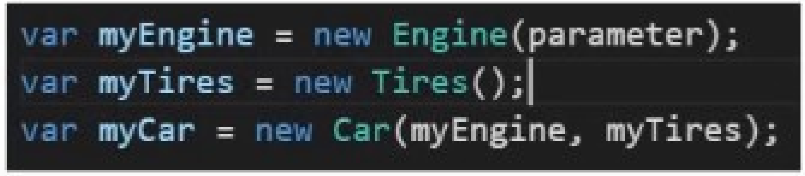


```
class Car{
    engine;
    tires;
    constructor(engine, tires)
    {
        this.engine = engine;
        this.tires = tires;
    }
}
```

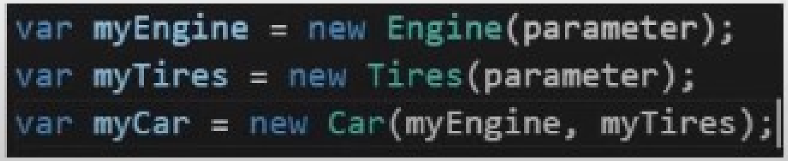
DI as a design pattern contd.



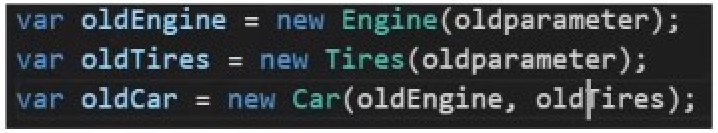
```
var myEngine = new Engine();
var myTires = new Tires();
var myCar = new Car(myEngine, myTires);
```



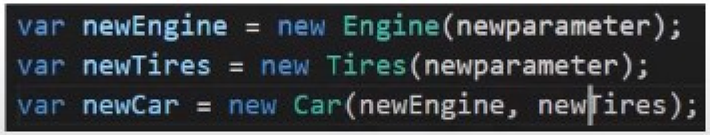
```
var myEngine = new Engine(parameter);
var myTires = new Tires();
var myCar = new Car(myEngine, myTires);
```



```
var myEngine = new Engine(parameter);
var myTires = new Tires(parameter);
var myCar = new Car(myEngine, myTires);
```

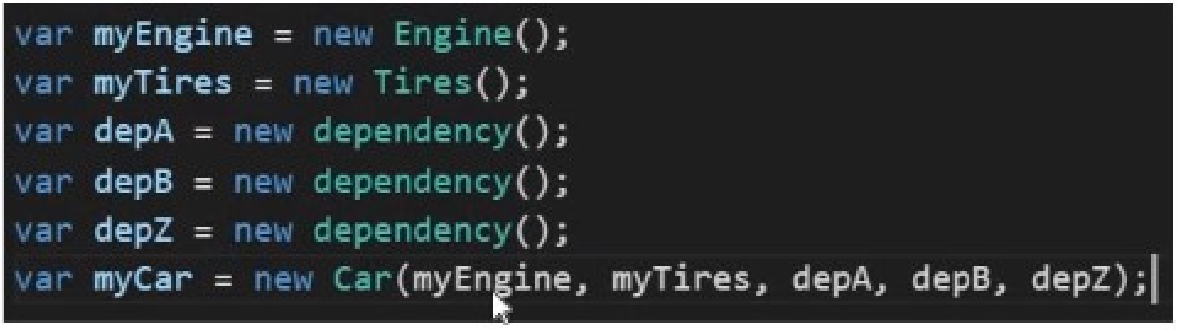


```
var oldEngine = new Engine(oldparameter);
var oldTires = new Tires(oldparameter);
var oldCar = new Car(oldEngine, oldTires);
```

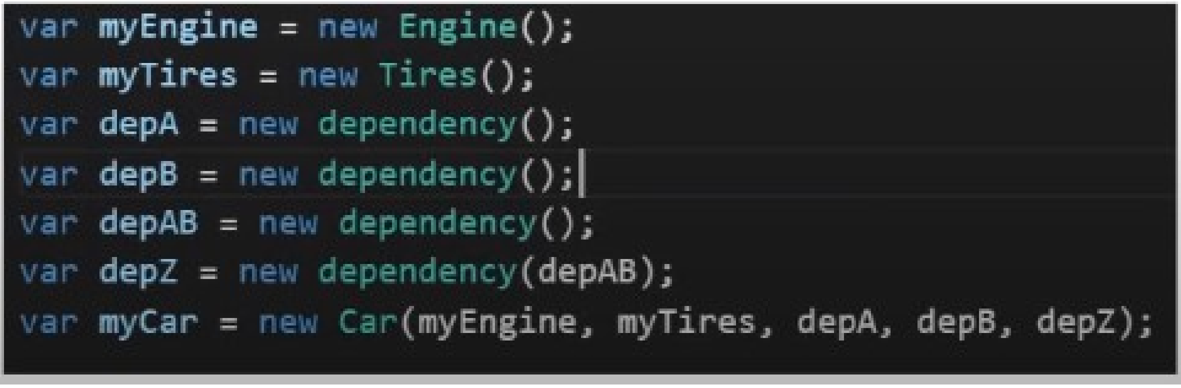


```
var newEngine = new Engine(newparameter);
var newTires = new Tires(newparameter);
var newCar = new Car(newEngine, newTires);
```

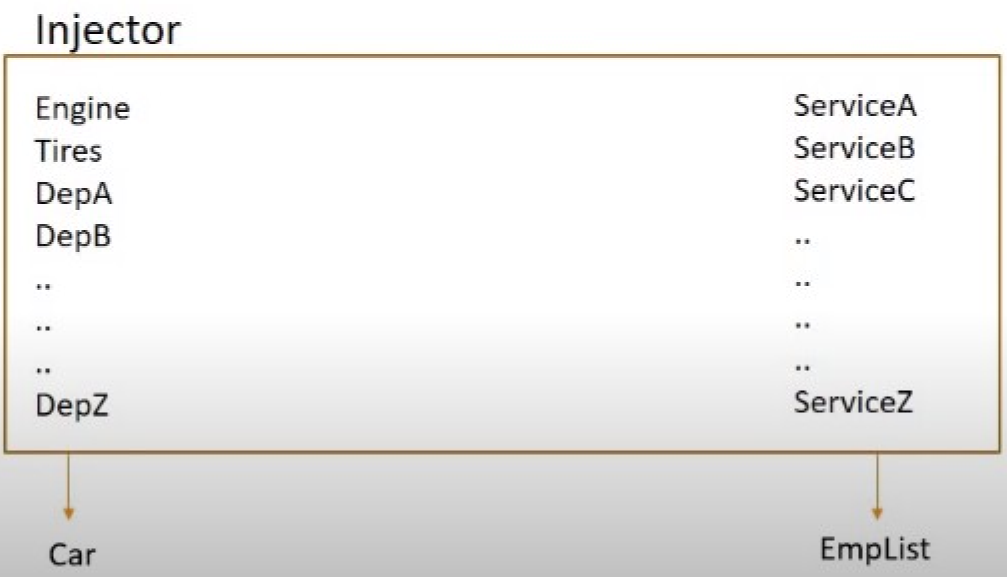
DI as a design pattern contd.



```
var myEngine = new Engine();
var myTires = new Tires();
var depA = new dependency();
var depB = new dependency();
var depZ = new dependency();
var myCar = new Car(myEngine, myTires, depA, depB, depZ);
```



```
var myEngine = new Engine();
var myTires = new Tires();
var depA = new dependency();
var depB = new dependency();
var depAB = new dependency();
var depZ = new dependency(depAB);
var myCar = new Car(myEngine, myTires, depA, depB, depZ);
```



Angular DI

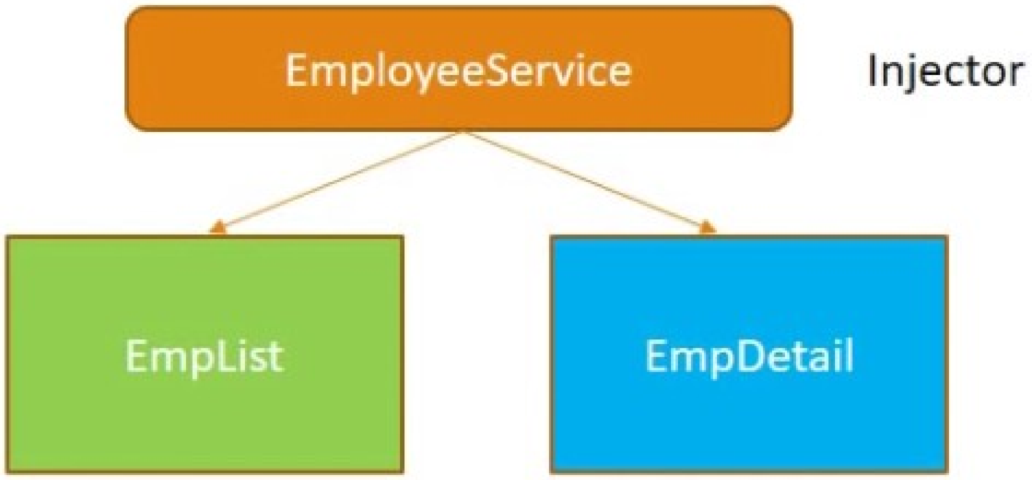
- Angular DI uses an concept called injector where it register all dependencies, framework manages all dependencies
- We register all services EmployeeComponent may depends such as Service1, Service2 etc, with injector, now when we initilize EmployeeComponent injetor provide(inject) all dependencies to EmployeeComponent

Steps for Defining Services

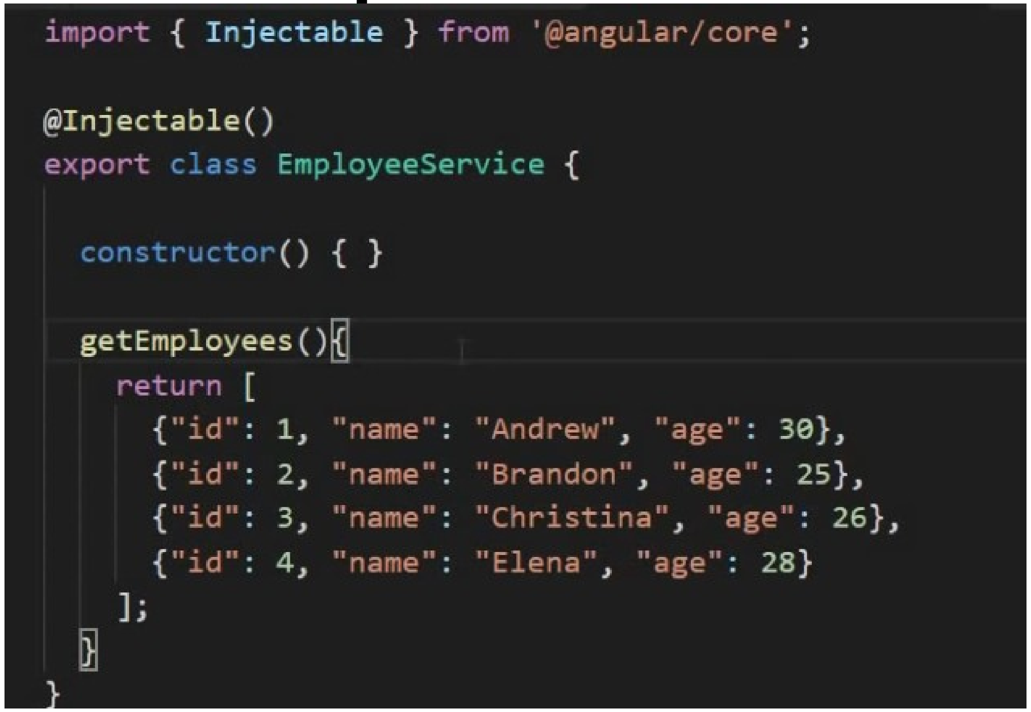
1) Define the EmployeeService class

2) Register with Injector

3) Declare as dependency in EmpList and EmpDetail

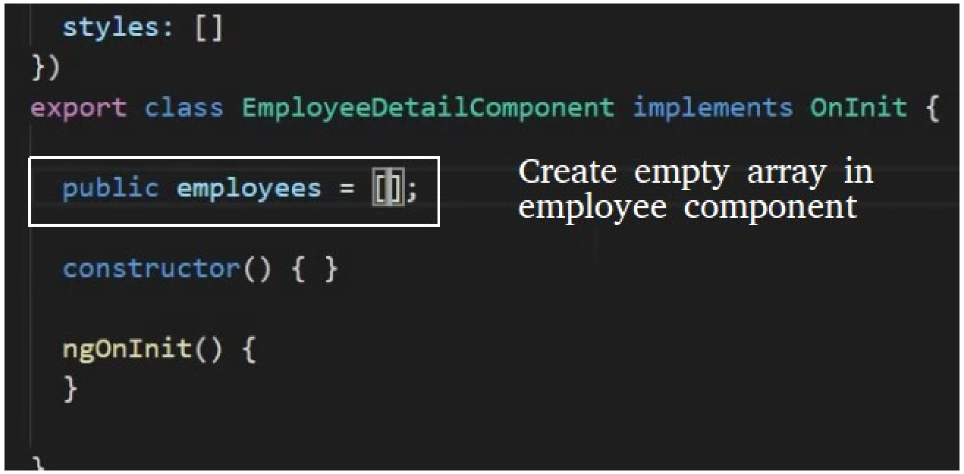


Step 1: Define EmployeeService



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

@Injectable()
export class EmployeeService {

  constructor() { }

  getEmployees(){
    return [
      {"id": 1, "name": "Andrew", "age": 30},
      {"id": 2, "name": "Brandon", "age": 25},
      {"id": 3, "name": "Christina", "age": 26},
      {"id": 4, "name": "Elena", "age": 28}
    ];
  }
}
```

- Create a service that return hardcoded Employees data
- @Injectable() annotation tell that this servie is injectable, it can further have injected services



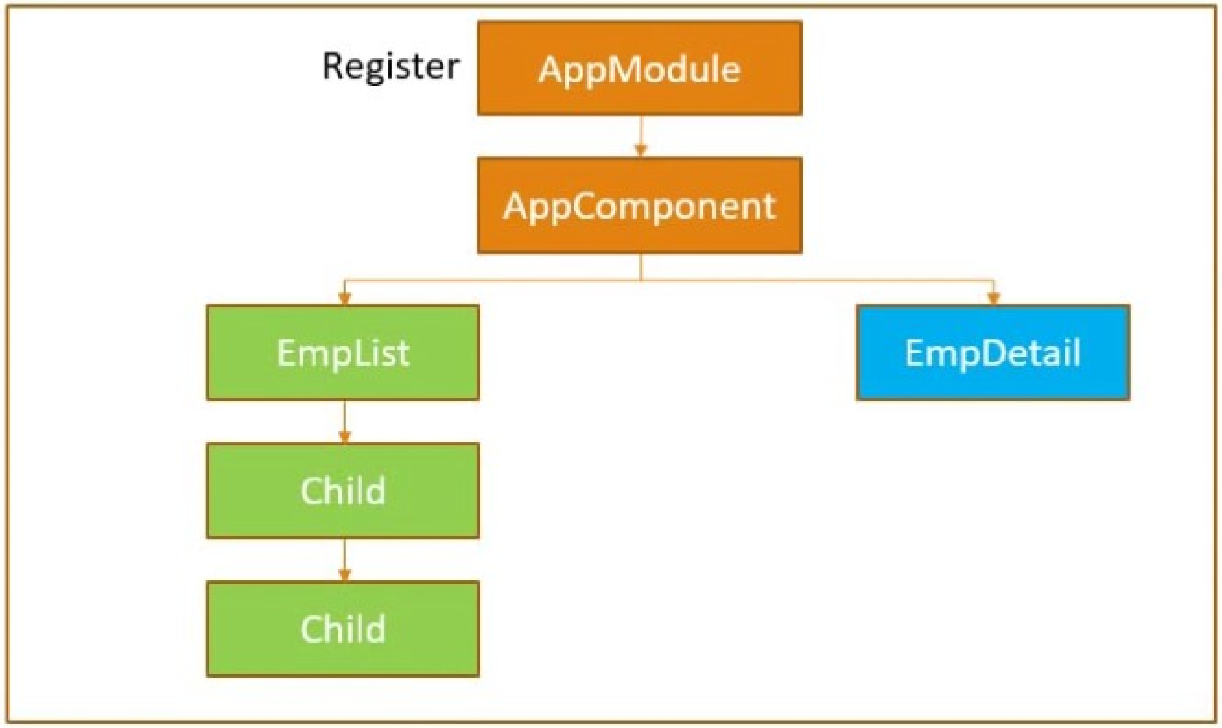
```
  styles: []
})
export class EmployeeDetailComponent implements OnInit {

  public employees = [];

  constructor() { }

  ngOnInit() {
  }

}
```

Create empty array in employee component

- Create an empty array in EmployeeDetailsComponent to accept the data form the service

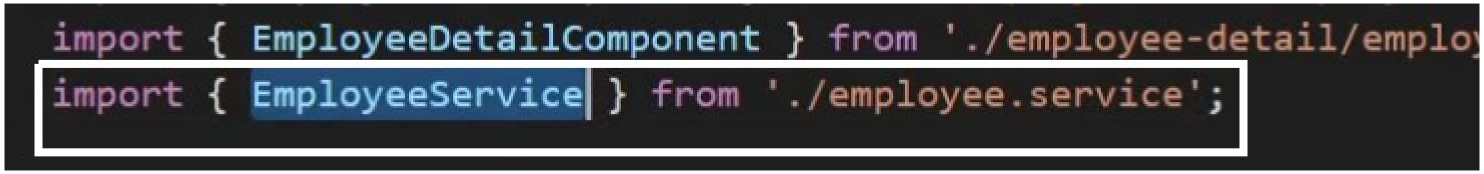
Step 2: Register with Injector

- We need to register our service with the angular injector, if we dont register then service is only a simple class and DI dont work
- It is better if we register service with AppModule so that all compnents can access it.

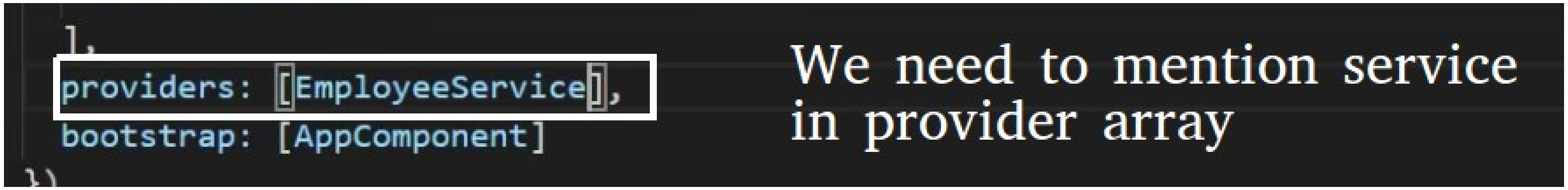


Step 2: Register with Injector

- Register in app.module.js



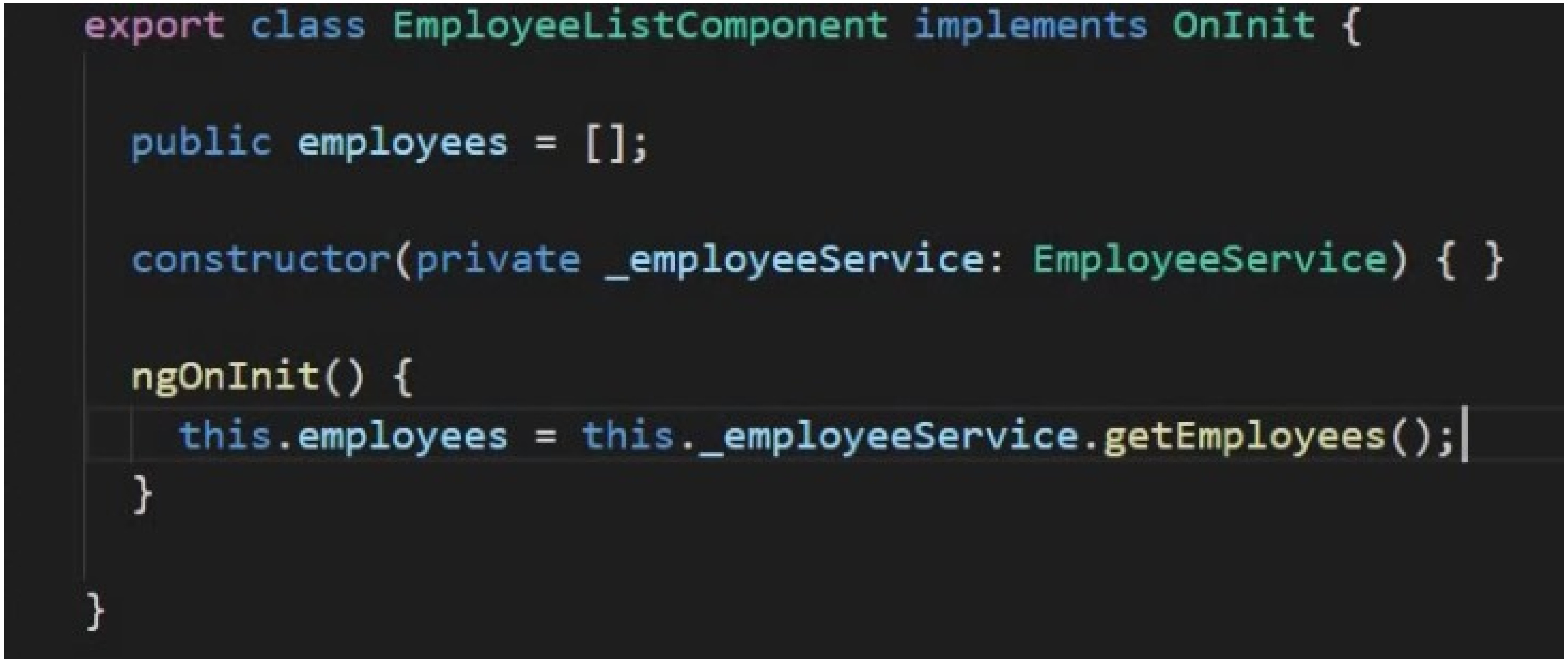
```
import { EmployeeDetailComponent } from './employee-detail/emplo
import { EmployeeService } from './employee.service';
```



```
  ],
  providers: [EmployeeService],
  bootstrap: [AppComponent]
})
```

We need to mention service in provider array

Step 3: Declare as DI in Employee component

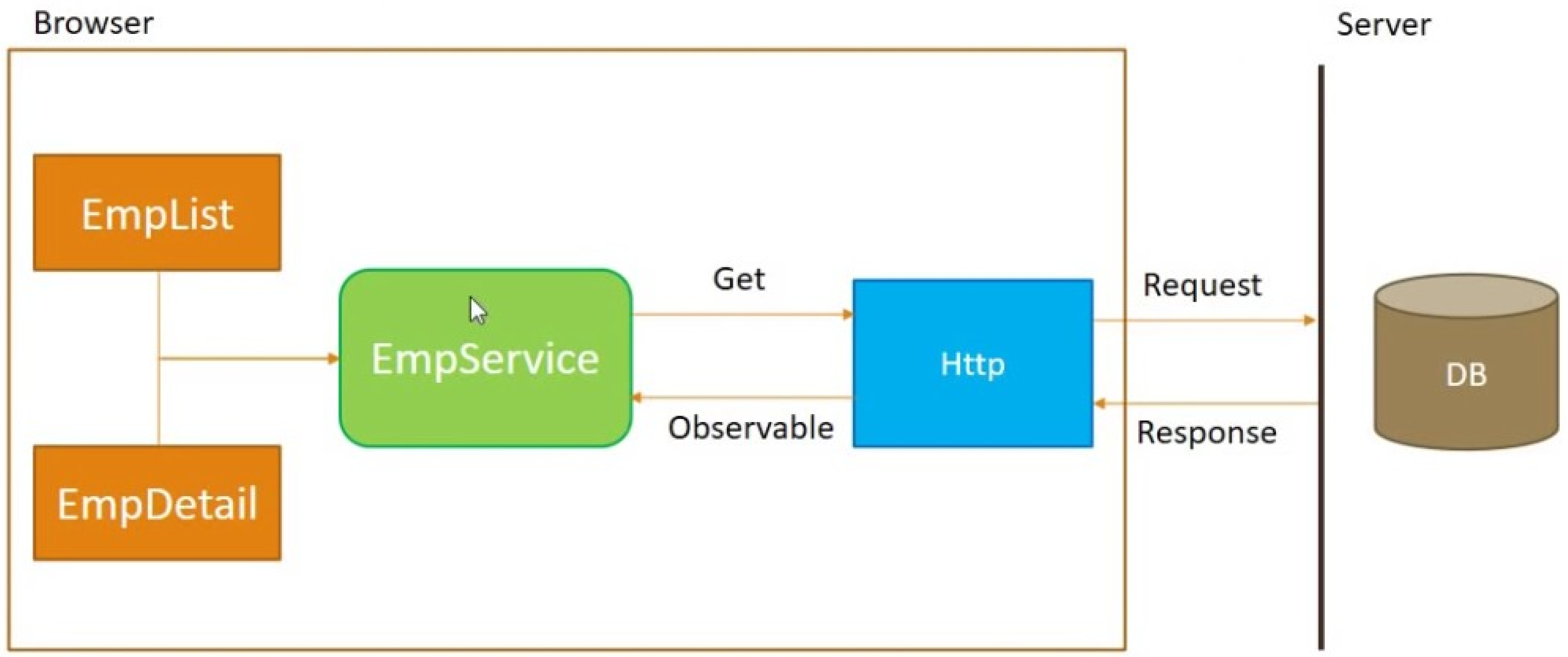


```
export class EmployeeListComponent implements OnInit {

  public employees = [];

  constructor(private _employeeService: EmployeeService) { }

  ngOnInit() {
    this.employees = this._employeeService.getEmployees();
  }

}
```

- Inject service EmployeeService to EmployeeListComponent
- We can inject service to any component that need that
- @Component annoation tell Angular to look for dependencies and inject it if it is registered with angular.

HTTP and Obserable

HTTP mechanism

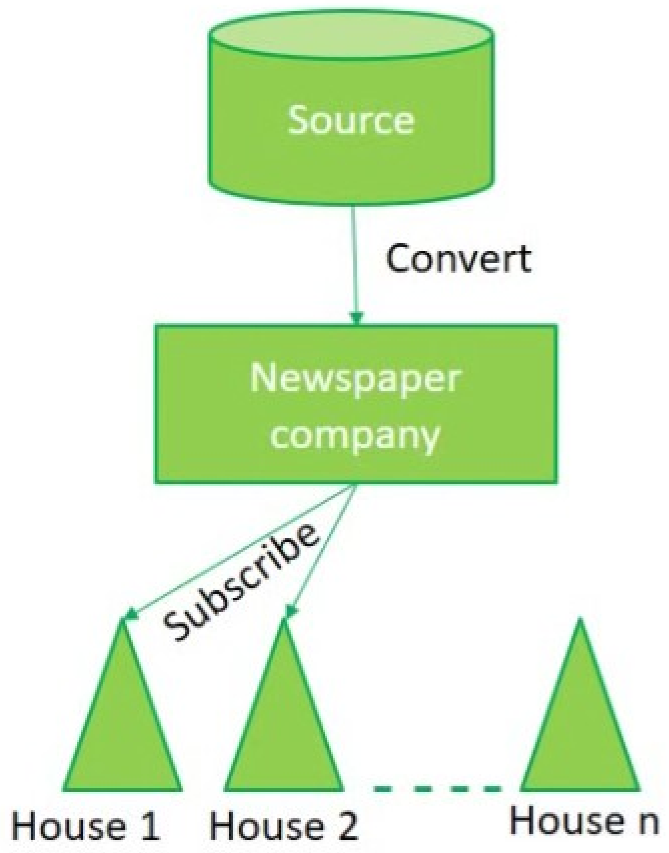


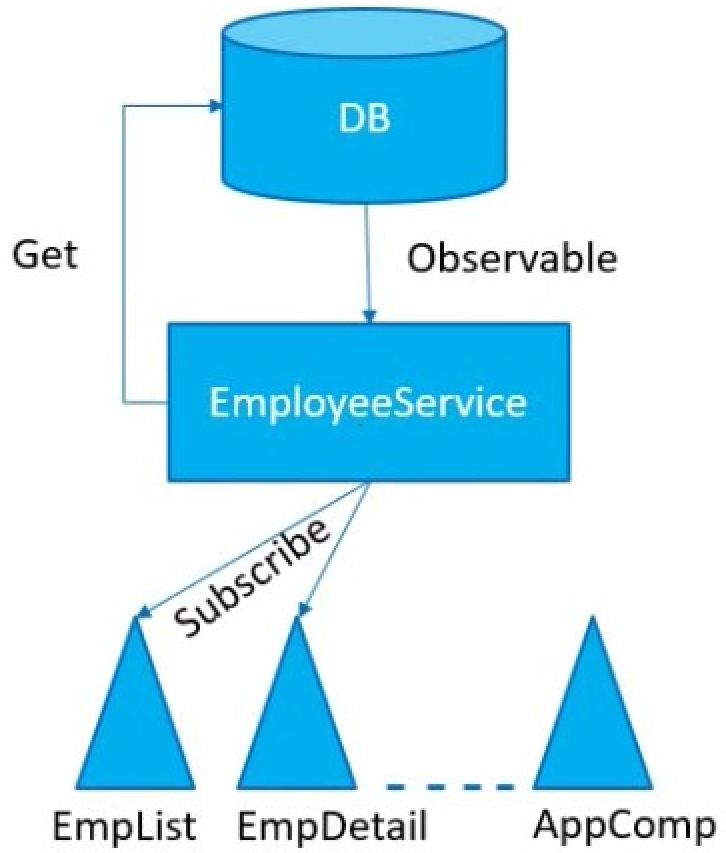
HTTP mechanism

- Http mechanism consist of two steps:
  - First step: Send the http request
  - Second Step: Recieve and process http response

  - What is Obserable?

Obserable





A sequence of items that arrive asynchronously over time.

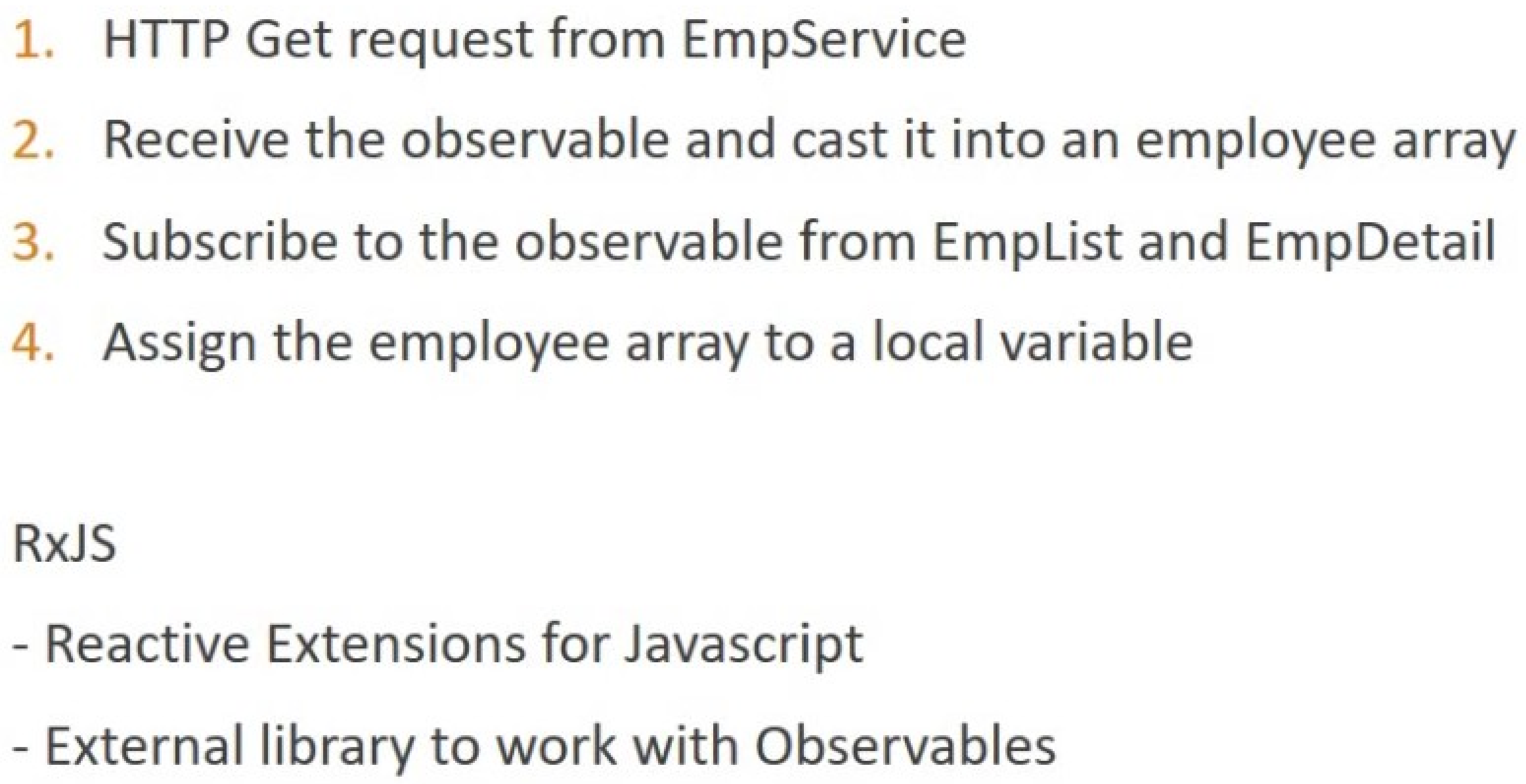
HTTP call – single item

Single item – HTTP response

Obserable

- Obserable is simply http response obtained from REST api
- Obserable is not the formate that is used by component, so we need to convert data into employee Array
- Data is provided to only component that subscribted to it.
- Now it is upto component to display data however they want

Http, Obserable, RxJS



1. HTTP Get request from EmpService
2. Receive the observable and cast it into an employee array
3. Subscribe to the observable from EmpList and EmpDetail
4. Assign the employee array to a local variable

RxJS

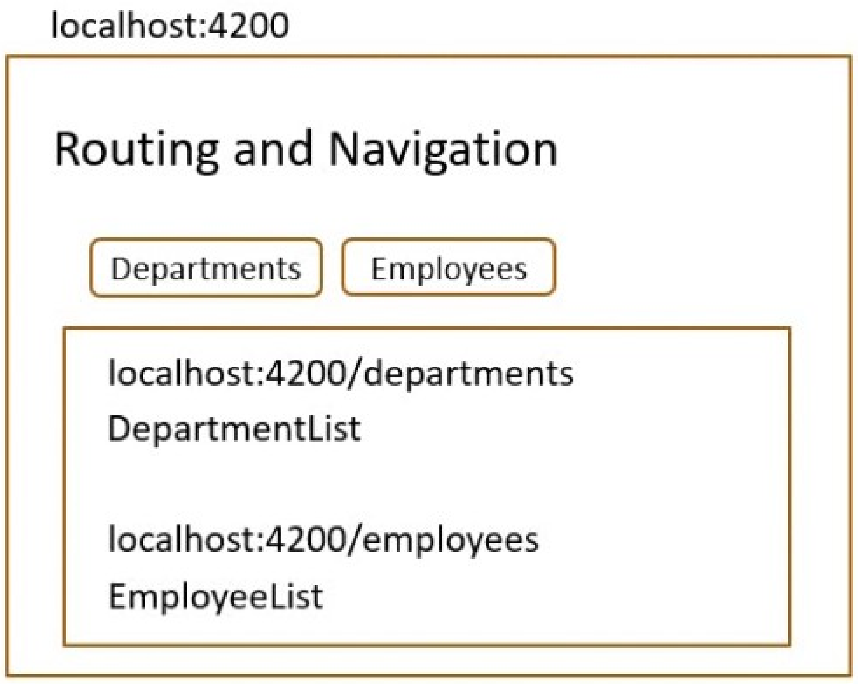
- Reactive Extensions for Javascript

- External library to work with Observables

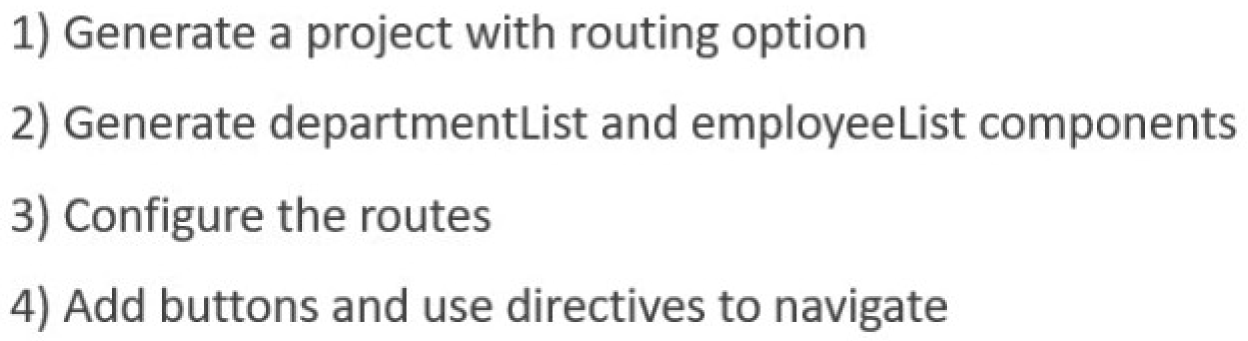
Routing in angular

What is routing?

- web applications can have different sections that corrosponds to different URLs and supporting those section programmatically is called routing
- In simple words, as per url open different angular components is called routing

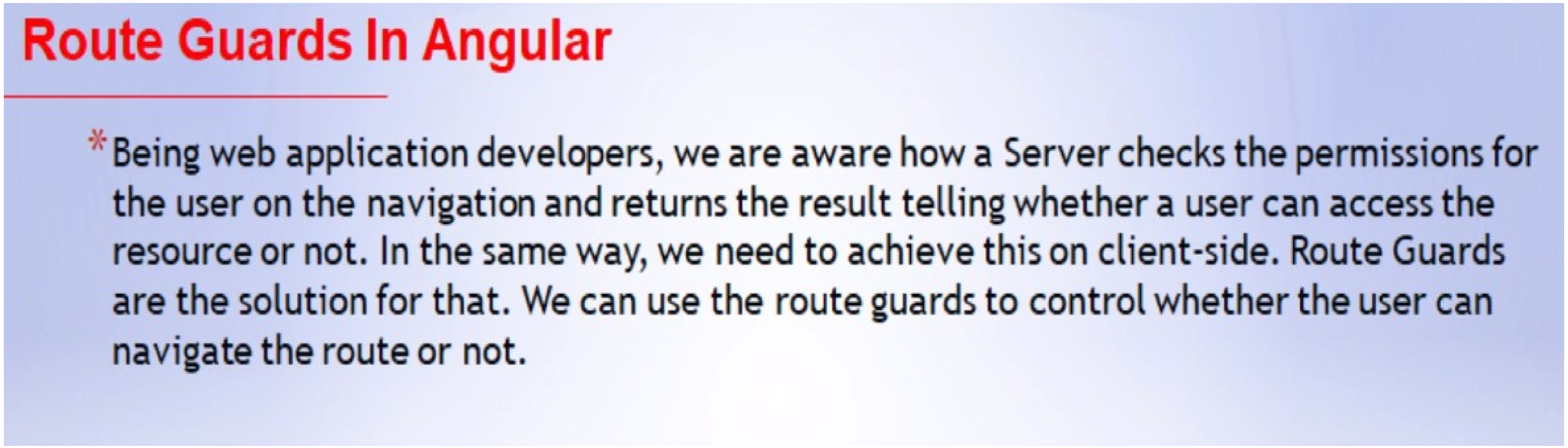


Steps for routing



1) Generate a project with routing option
2) Generate departmentList and employeeList components
3) Configure the routes
4) Add buttons and use directives to navigate

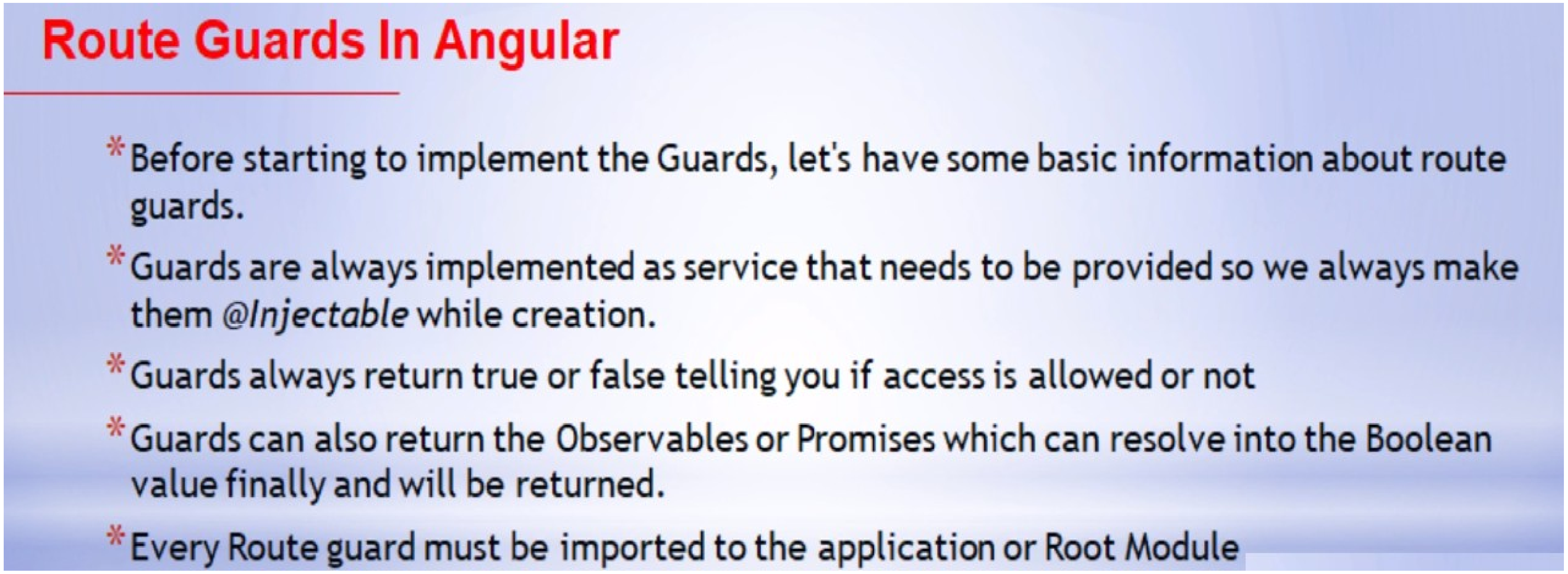
CanActivate route guards



Route Guards In Angular

* Being web application developers, we are aware how a Server checks the permissions for the user on the navigation and returns the result telling whether a user can access the resource or not. In the same way, we need to achieve this on client-side. Route Guards are the solution for that. We can use the route guards to control whether the user can navigate the route or not.

CanActivate route guards

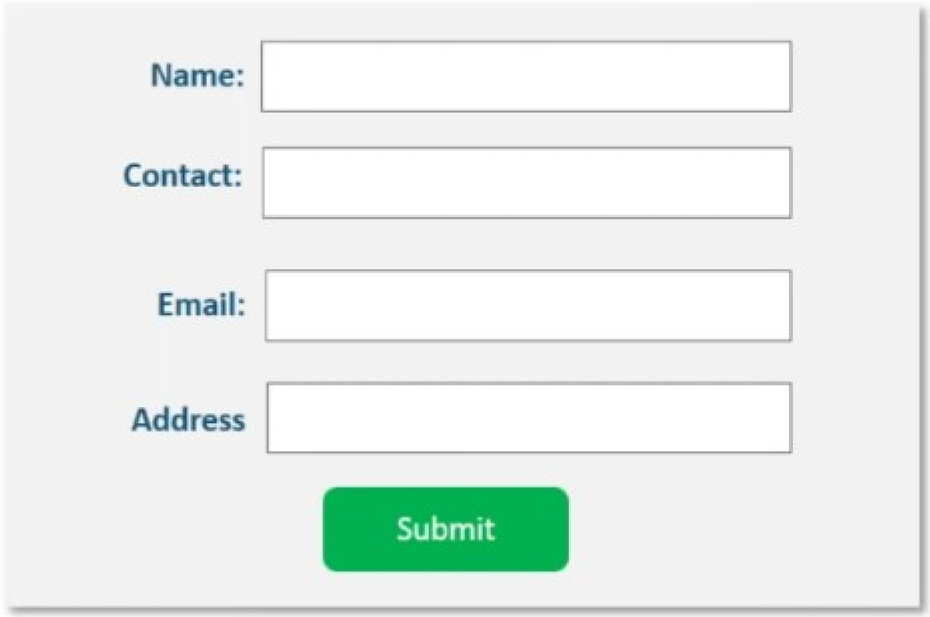


Route Guards In Angular

* Before starting to implement the Guards, let's have some basic information about route guards.
* Guards are always implemented as service that needs to be provided so we always make them *@Injectable* while creation.
* Guards always return true or false telling you if access is allowed or not
* Guards can also return the Observables or Promises which can resolve into the Boolean value finally and will be returned.
* Every Route guard must be imported to the application or Root Module

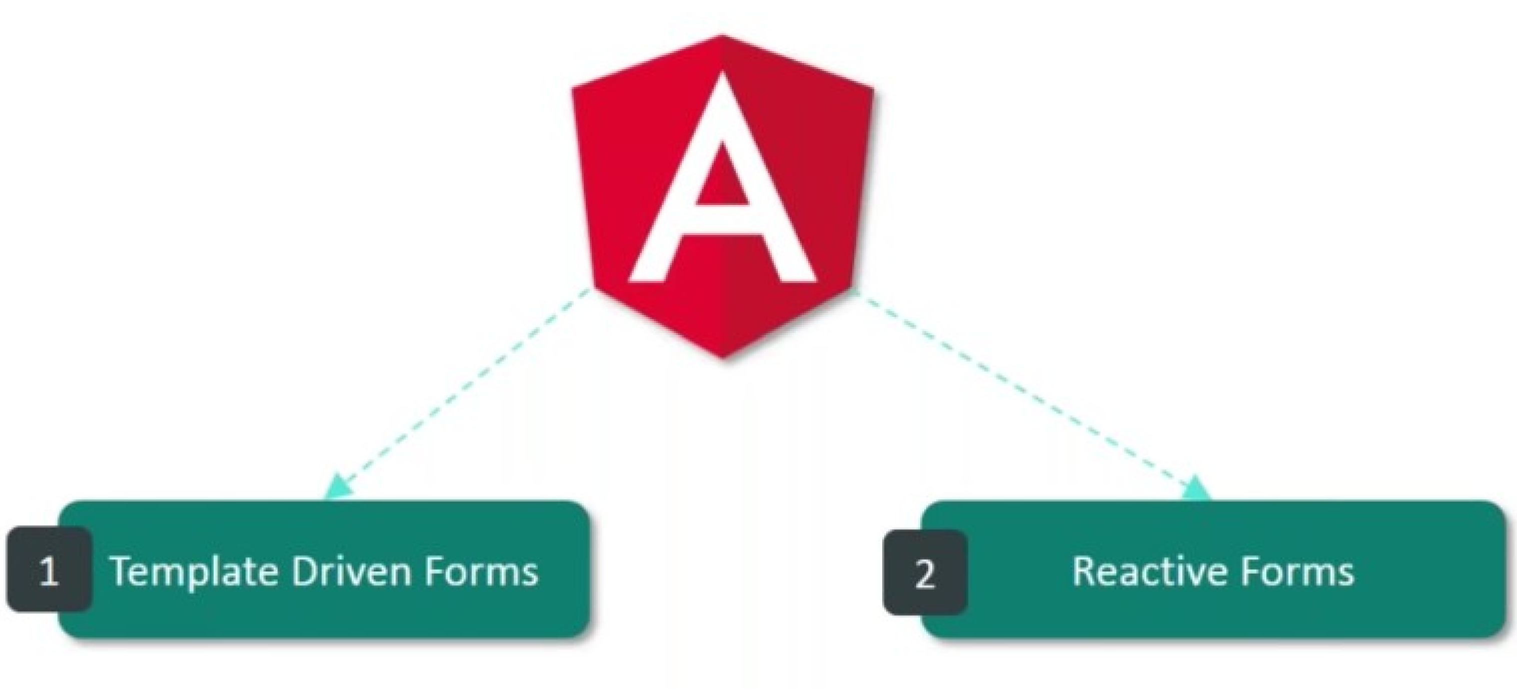
Angular forms

Forms are the most crucial aspect of any web application. It is forms from where we get majority of rich data from users.



Angular forms

- Angular offers two types of forms:-
  - template-driven
  - model-driven



Template Driven

- template-driven forms place the emphasis on developing a form within an HTML template and handling most of the logic for the form-inputs, data validation, saving, and updating-in form directives placed within that template.
- Very less code required because most of the forms related work have been done by the angular framework itself.
- heavy use of the ngModel form directive
- FormModule must be imported into app-module.ts

Directives for form building

- NgForm
- NgModel

- ngForm:
  - It plays a great role in form building, this angular directive concrete values of all HTML elements of form.
  - We can say that by using this directive we are capable to create form object.

- NgModel
  - This directive has used for binding, one-way and two-way data binding.
  - It is responsible to hold values of HTML elements.
  - It is also used to validate HTML elements values.

Updating Angular CLI

If you're using Angular CLI `1.0.0-beta.28` or less, you need to uninstall `angular-cli` package. It should be done due to changing of package's name and scope from `angular-cli` to `@angular/cli`:

```
npm uninstall -g angular-cli
npm uninstall --save-dev angular-cli
```

To update Angular CLI to a new version, you must update both the global package and your project's local package.

Global package:

```
npm uninstall -g @angular/cli
npm cache verify
# if npm version is < 5 then use `npm cache clean`
npm install -g @angular/cli@latest
```