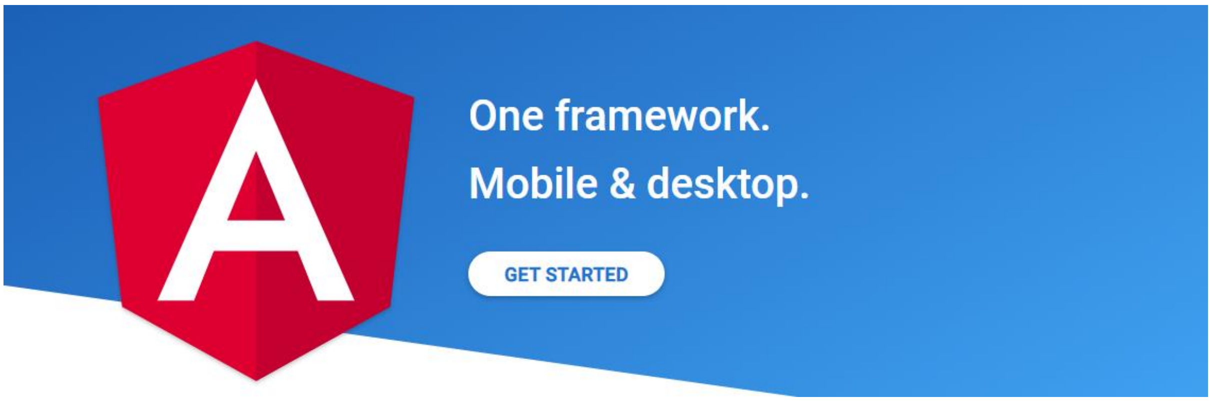


Angular is one of the most popular Web application development framework given by google to develop a rich client side applications specially SPA's.

Angular follows a modular approach, means the entire application is divided into various modules.

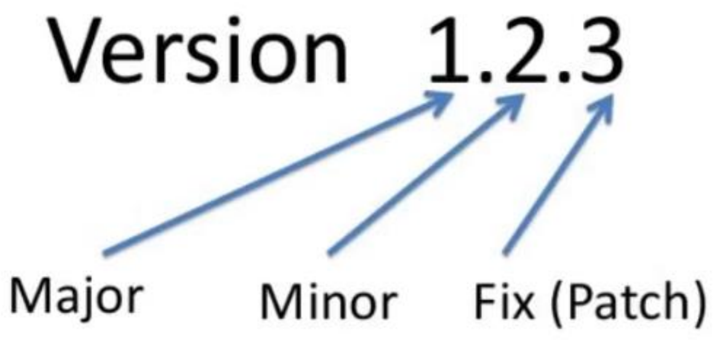
Angular have its own Routing System and HTTP Clients to connect with Server side Business Logics.

With Angular we can write a re-usable code which is better to test the application very easily.

Angular Version History

Year	Version
2012 JUNE	**Angular JS**
2016 SEP	**Angular 2**
2017 MAR	**Angular 4**
2017 NOV	**Angular 5**
2018 MAY	**Angular 6**
2018 OCT	**Angular 7**
2019 MAY	**Angular 8**
2020 FEB	**Angular 9**
2020 Jun	**Angular 10**

Angular is one of the most dynamic framework developed and maintained by Google, which agreed to release 2 versions per year with semantic versioning system.



FIX : for any bug fixing release

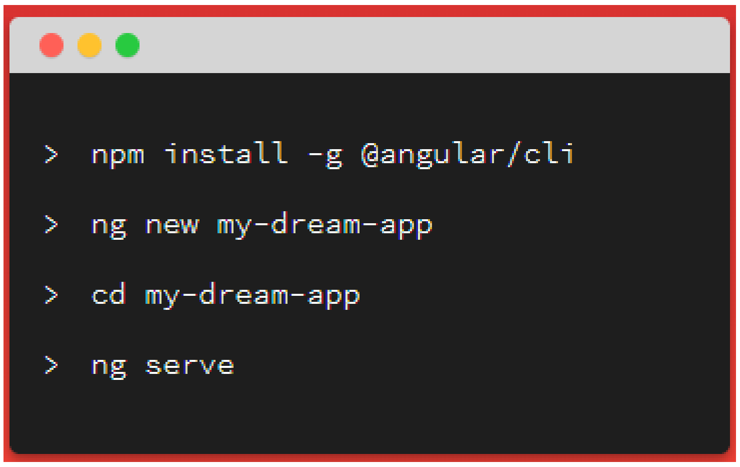
Minor : for minor changes , which will not break the application

Major : Major Feature added to a software (May break application)

NOTE: even though these versioning are changing rapidly, but these are not contains reinvent of application, we can still continue with the older versions and we can easily upgrade to newer versions and these are almost backward compatible.

Installation of Angular CLI

Angular CLI is a command line interface tool to create an angular project structure with all the required files, and the project artefacts like components, services, pipes etc.



```
> npm install -g @angular/cli

> ng new my-dream-app

> cd my-dream-app

> ng serve
```

NOTE: For more information visit https://cli.angular.io/

Few Commands

Purpose	Command
To install angular CLI	**npm install -g @angular/cli**
to check the CLI version	**ng --version**
to create Angular project	**ng new <project-name>**
to start angular application	**cd <project-name>** **ng serve**

url to access the application	**http://localhost:4200**
To Generate a Component	**ng generate component <name>**
To Generate a Service	**ng generate service <name>**

First Angular Application

To Create First Application we use Angular CLI Tool

Ex: ng new <project-name>

Ex: ng new first-app

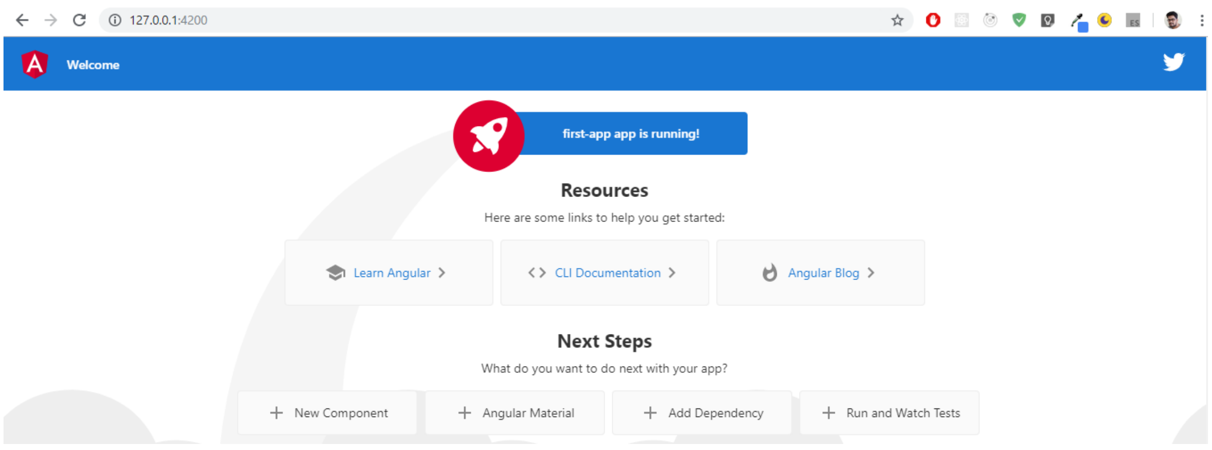
```
E:\Project_Work\MyProjects\Angular_8_Class>ng new first-app
? Would you like to add Angular routing? Yes
? Which stylesheet format would you like to use? CSS
```

This will take a couple of minutes and create an Angular Application. We can access the application using the following URL.

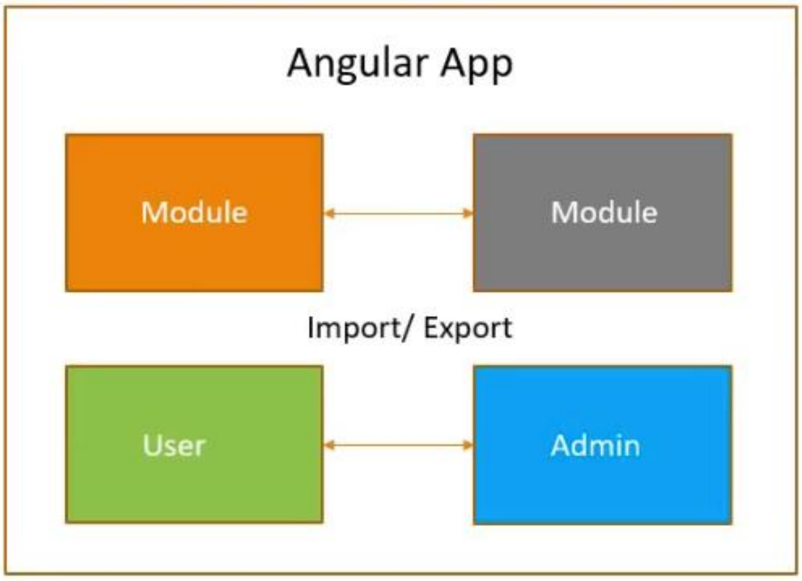
Cd first-app

Ng serve

http://127.0.0.1:4200/



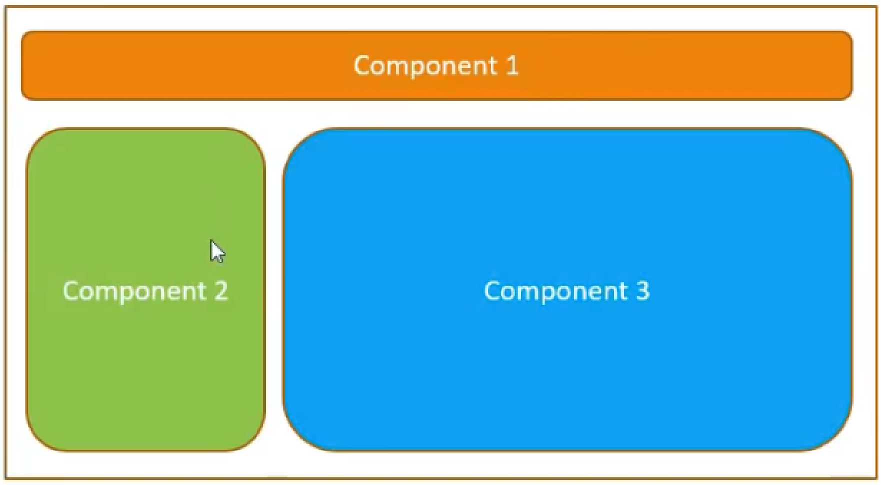
Angular builds an application into various modules,



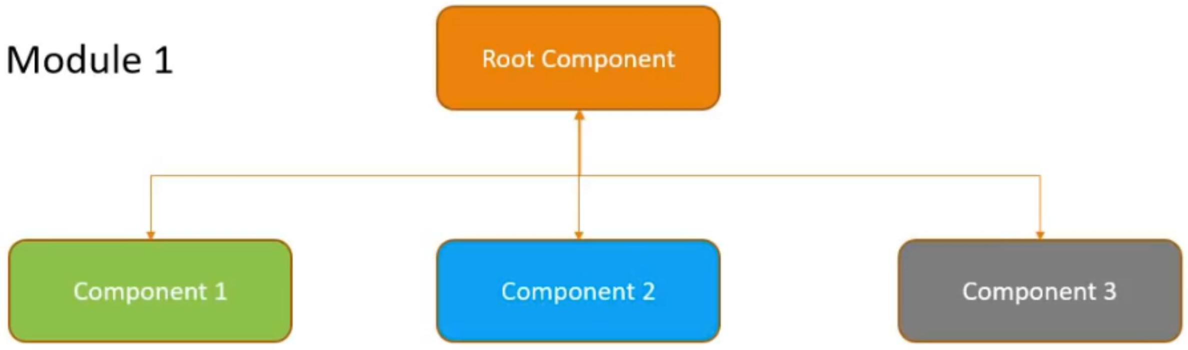
Every Angular Application is made up of different set of modules, for all module for each Angular application we have a Root Module that is called as “APP MODULE”.

Each module contains a set of components and services.

Each component holds portion of the view in the browser. This is acts like a controller for each of the portion of UI.

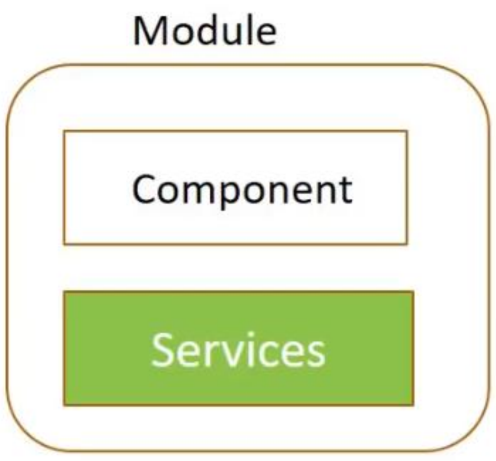


Every Angular Application will have one ROOT Component called “App Component”. All other components are nested inside this ROOT Component.

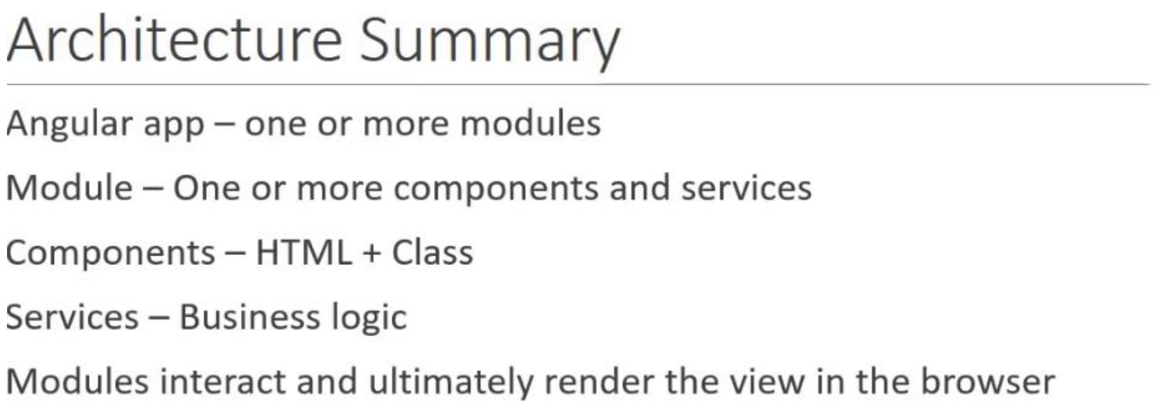


Each component has a HTML and a class which acts as controller for the view with HTML.

Finally Each Module contains a Components and Services as follows,



The Angular Service is also a TypeScript Class, but this is used to maintain the application data and the main business logic of the application and this service is responsible to connects with server side Logic or web services.



Architecture Summary

Angular app – one or more modules

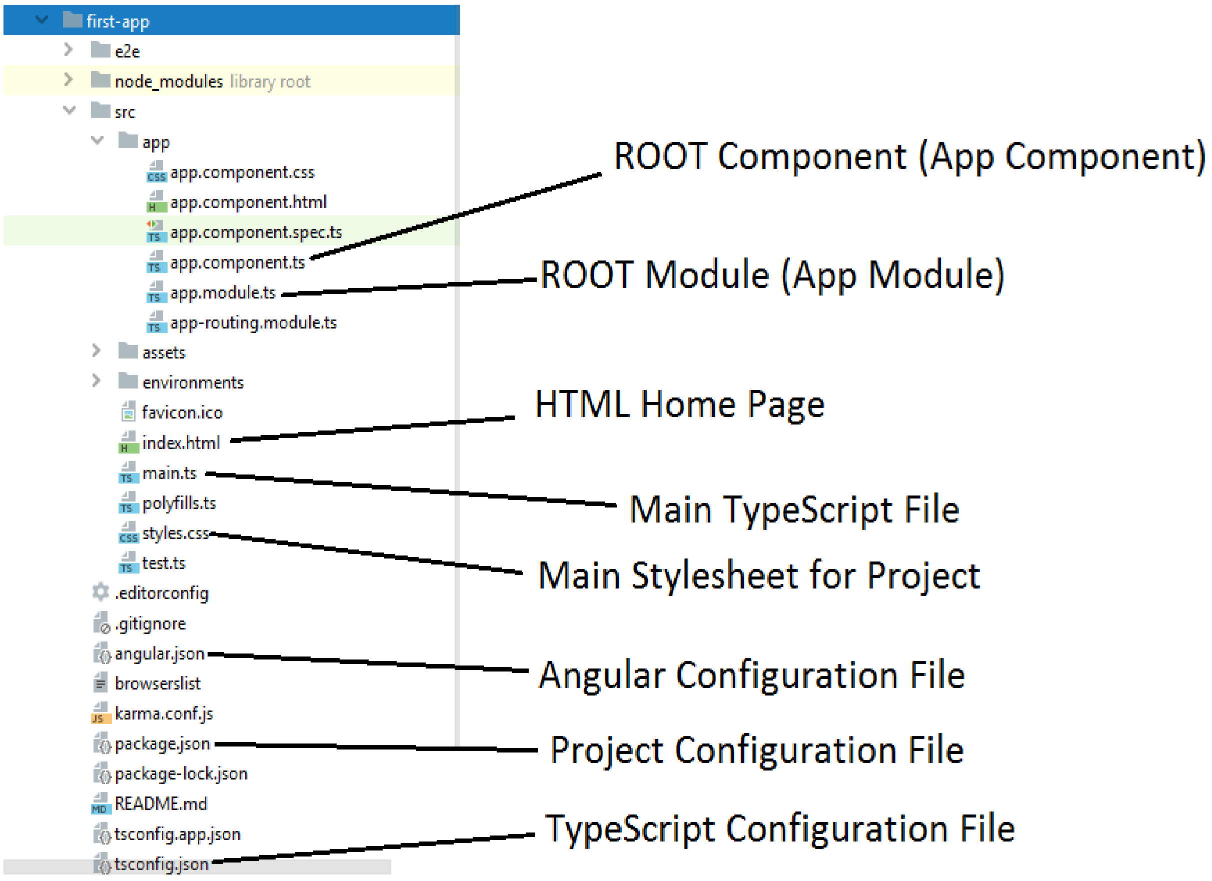
Module – One or more components and services

Components – HTML + Class

Services – Business logic

Modules interact and ultimately render the view in the browser

Project Structure of Angular



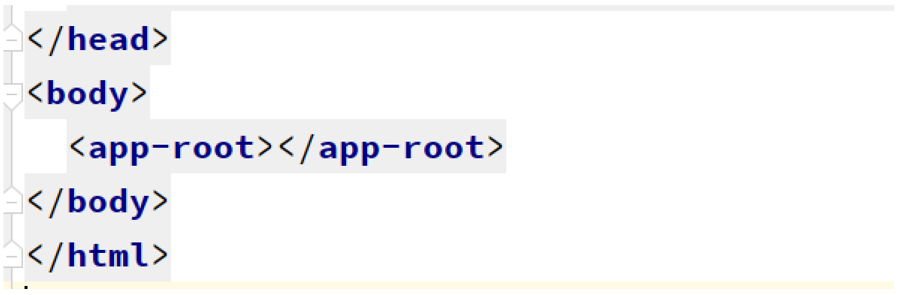
Application Flow of Execution

We can start an Angular Application by executing the command in the command line is,

ng serve

The url to access is : http://127.0.0.1:4200/

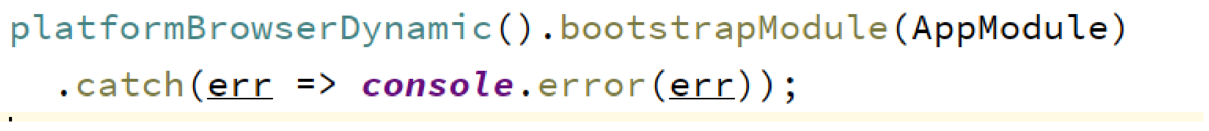
When we enter this URL in the browser, it looks of a home page html “index.html”.



```
</head>
<body>
  <app-root></app-root>
</body>
</html>
```

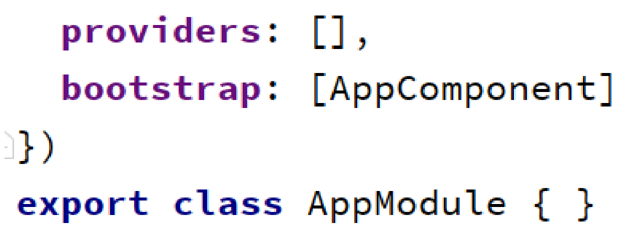
This Contains a custom HTML Tag called <app-root>.

The Main file which the Application looks for startup is “main.ts”



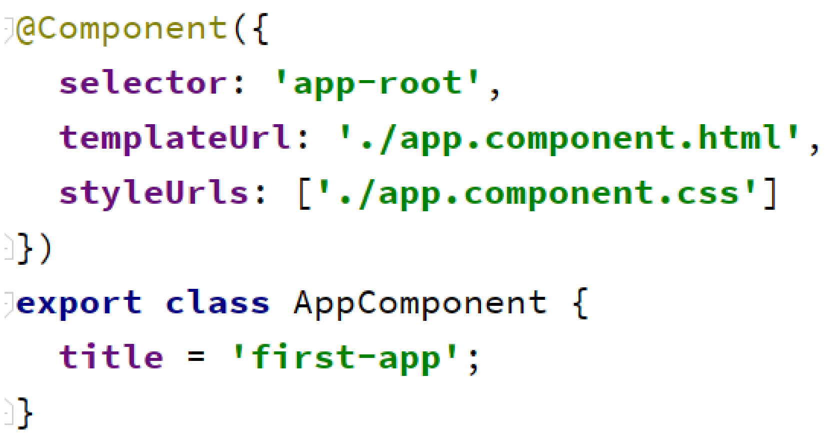
```
platformBrowserDynamic().bootstrapModule(AppModule)
  .catch(err => console.error(err));
```

In this file, The App Module is configured as a bootstrap Module. This the command called “ng serve” this file will kick starts the Mail Application Module “App Module”.



```
  providers: [],
  bootstrap: [AppComponent]
})
export class AppModule { }
```

The Main App Module is configured with “App Component” to kick Start.



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

The App Component is configured with a Component Decorator which contains the following Entries.

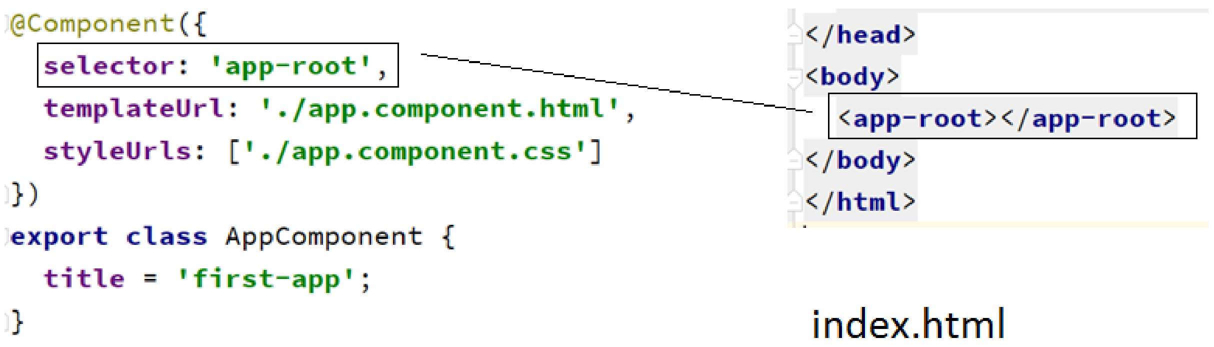
Selector : this selector can be used as a custom tag in any HTML doc to represent the view part of this component

Template URL : this is the html document responsible for preparing the view part of the component.

styleUrls : this is a style sheet file responsible for display the view portion of this component.

This Component is acts as controller for the view part.

Finally this “Selector” is added as a custom tag in the HTML to represent the view part of this component.



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

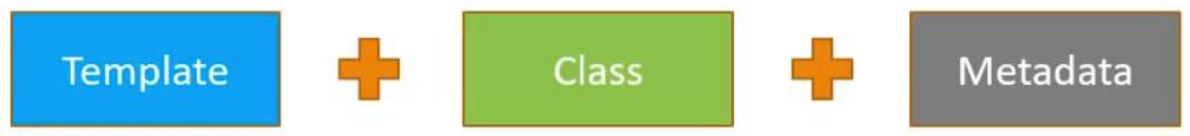
```
</head>
<body>
  <app-root></app-root>
</body>
</html>
```

index.html

app.component.ts

Angular Component

Each Angular Component is a logical piece of code which is responsible and acts as controller for any View Part of Angular Application User Interface. This is the collection of 3 elements such as,

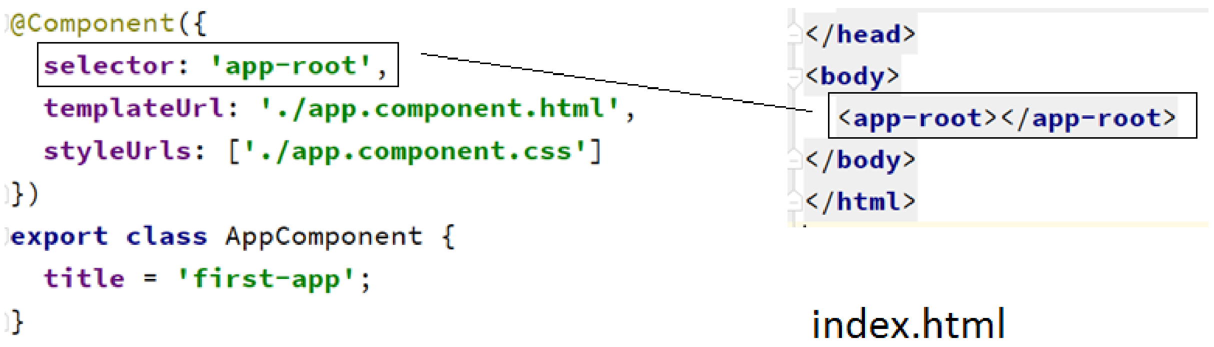


A Template is a normal HTML Document + CSS to represent the View Part.

A Class is generally a TypeScript Class which supports and control the view Part.

Metadata is the extra information Provider to a Component which Configures this class with the Template. This Contains a Decorator which informs the Angular App like this is a Component or a Service of the application. This intern having a template URL , Selector , Style URL's information.

Every Angular Component is acts as Custom HTML Tag, which can be used in any main HTML file or any other nested component template to render the view.



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

```
</head>
<body>
  <app-root></app-root>
</body>
</html>
```

index.html

app.component.ts

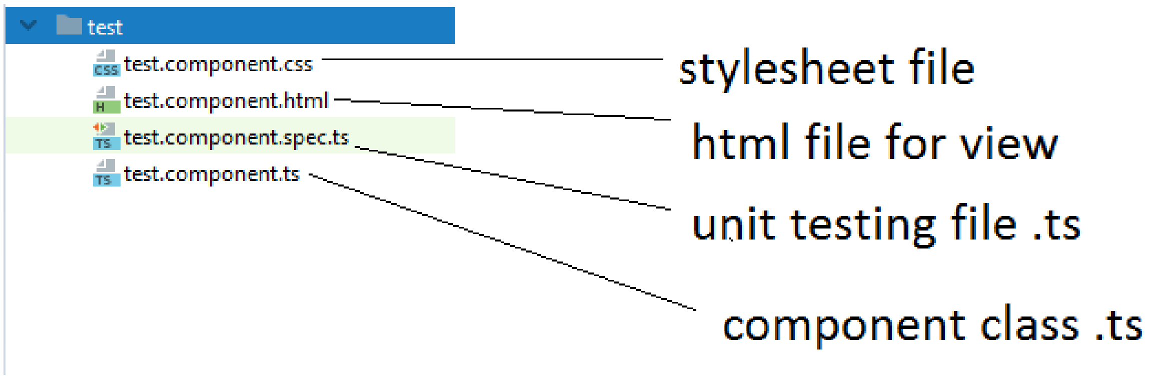
To create a new component in Angular we use Angular CLI commands,

```
ng generate component <component-name>
```

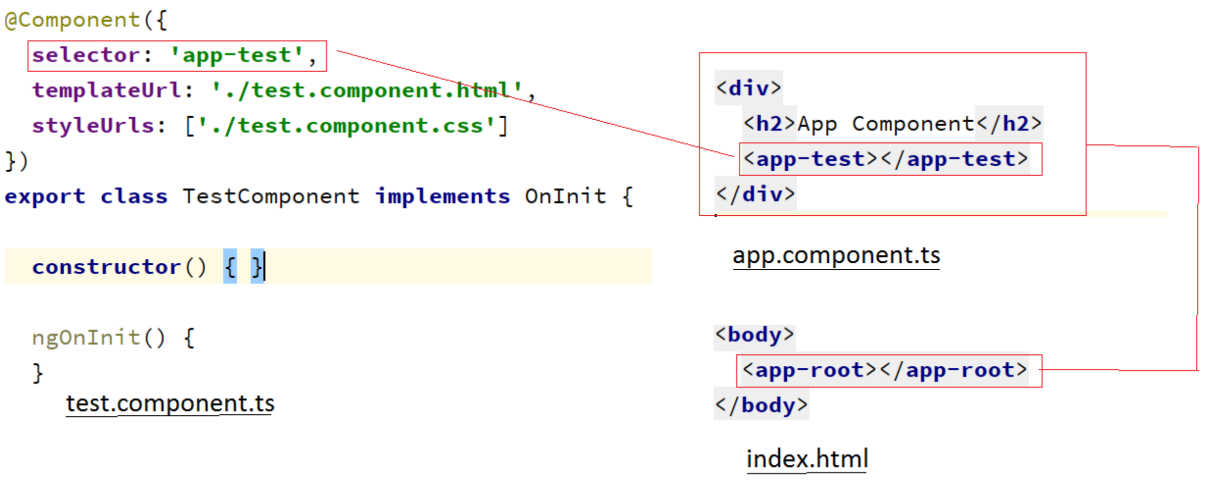
```
Ex: ng generate component test
```

This command will generate a set of files the test folder.

```
E:\Project_Work\MyProjects\Angular_8_Class\first-app>ng generate component test
CREATE src/app/test/test.component.html (19 bytes)
CREATE src/app/test/test.component.spec.ts (614 bytes)
CREATE src/app/test/test.component.ts (261 bytes)
CREATE src/app/test/test.component.css (0 bytes)
UPDATE src/app/app.module.ts (467 bytes)
```



Now we can use this component specific selector or custom tag in any other component like App Component to render the view.



```
@Component({
  selector: 'app-test',
  templateUrl: './test.component.html',
  styleUrls: ['./test.component.css']
})
export class TestComponent implements OnInit {

  constructor() { }

  ngOnInit() {
  }
```

test.component.ts

```
<div>
  <h2>App Component</h2>
  <app-test></app-test>
</div>
```

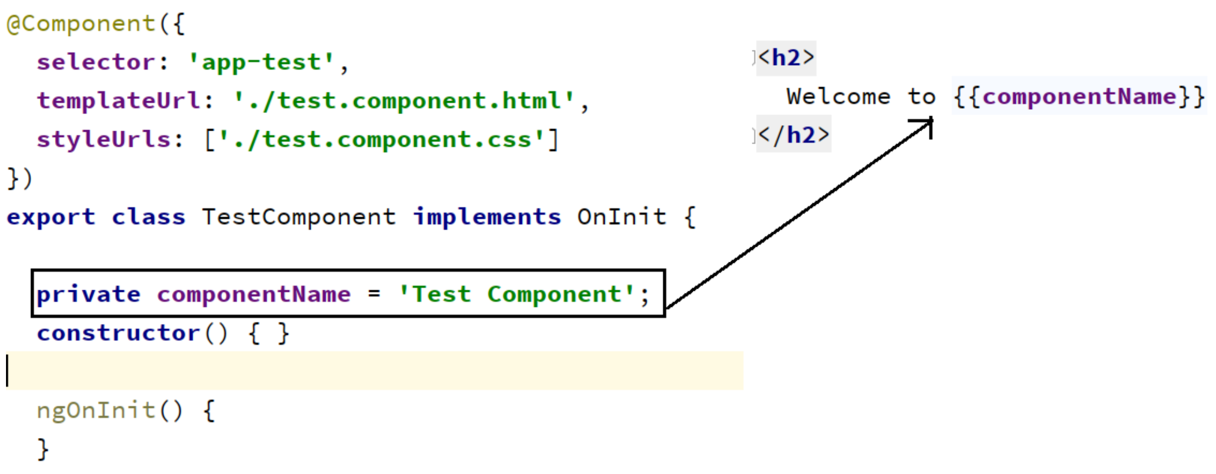
app.component.ts

```
<body>
  <app-root></app-root>
</body>
```

index.html

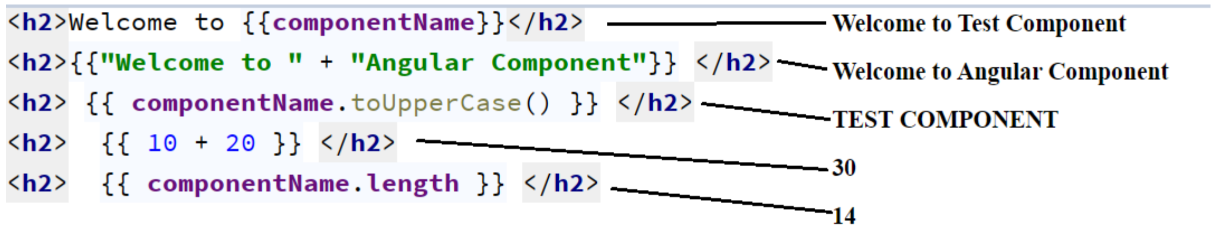
Angular Interpolation

Angular Interpolation is a special syntax to bind the component properties to html template with Angular Expressions.

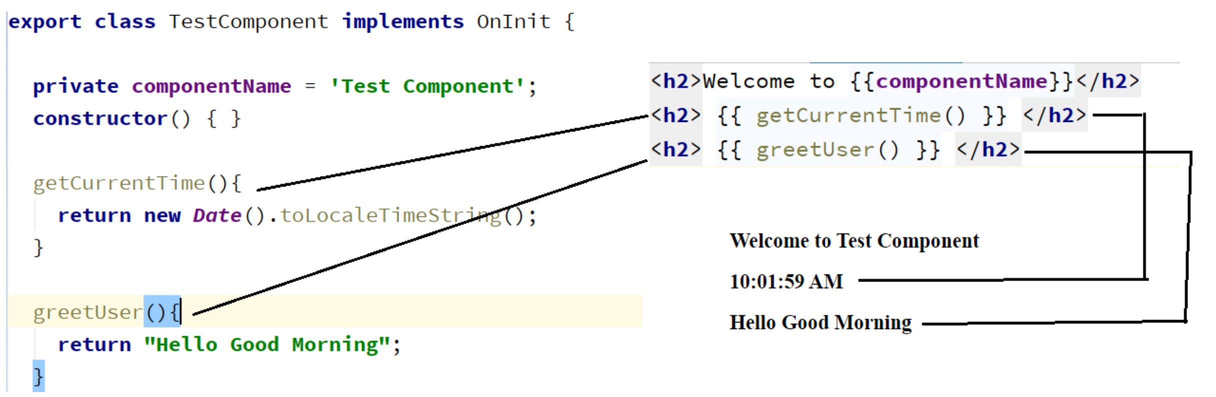


```
@Component({
  selector: 'app-test',
  templateUrl: './test.component.html',
  styleUrls: ['./test.component.css']
})
export class TestComponent implements OnInit {

  private componentName = 'Test Component';
  constructor() { }

  ngOnInit() {
  }
```

```
<h2>
  Welcome to {{componentName}}
</h2>
```



```
<h2>Welcome to {{componentName}}</h2>                    Welcome to Test Component
<h2>{{"Welcome to " + "Angular Component"}} </h2>        Welcome to Angular Component
<h2> {{ componentName.toUpperCase() }} </h2>             TEST COMPONENT
<h2>  {{ 10 + 20 }} </h2>                                30
<h2>  {{ componentName.length }} </h2>                   14
```



```
export class TestComponent implements OnInit {

  private componentName = 'Test Component';
  constructor() { }

  getCurrentTime(){
    return new Date().toLocaleTimeString();
  }

  greetUser(){
    return "Hello Good Morning";
  }
```

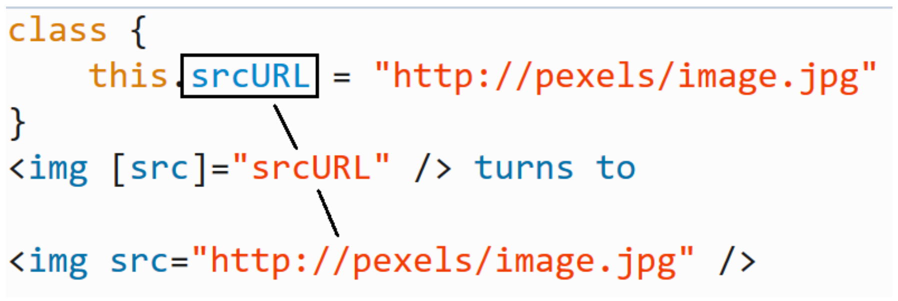
```
<h2>Welcome to {{componentName}}</h2>
<h2> {{ getCurrentTime() }} </h2>
<h2> {{ greetUser() }} </h2>
```

```
Welcome to Test Component
10:01:59 AM
Hello Good Morning
```

NOTE: Using Angular Interpolation we cannot access the JavaScript Global variables, instead we can create a method inside a class and use / call that method from the template using Interpolation.

Property Binding in Angular

Property binding is the primary way of binding data in Angular. The square braces are used to bind data to a property of an element, the trick is to put the property onto the element wrapped in brackets: [property].



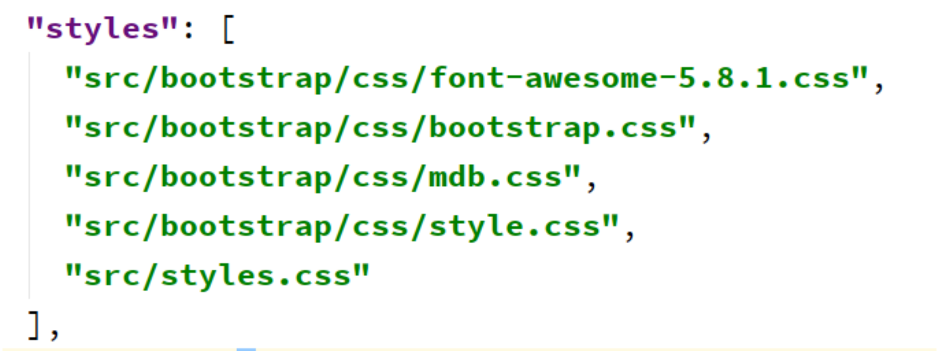
```
class {
    this.srcURL = "http://pexels/image.jpg"
}
<img [src]="srcURL" /> turns to

<img src="http://pexels/image.jpg" />
```

The src property of the HTMLElement img is bound to the srcURL property of the class. Whenever the srcURL property changes the src property of the img element changes.

Add Bootstrap to Angular

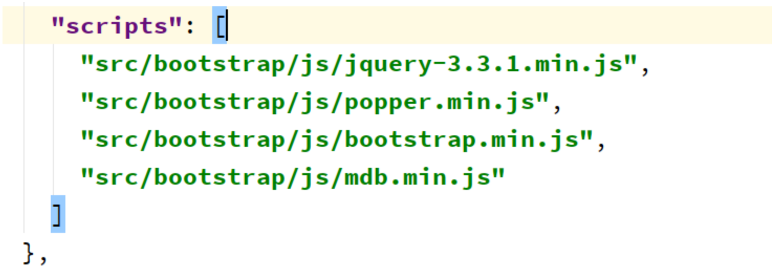
We can add the most popular Bootstrap library with Angular. It required various steps as follows,

Connect the Bootstrap specific '.css' files in the main "angular.json" files.



```
"styles": [
  "src/bootstrap/css/font-awesome-5.8.1.css",
  "src/bootstrap/css/bootstrap.css",
  "src/bootstrap/css/mdb.css",
  "src/bootstrap/css/style.css",
  "src/styles.css"
],
```

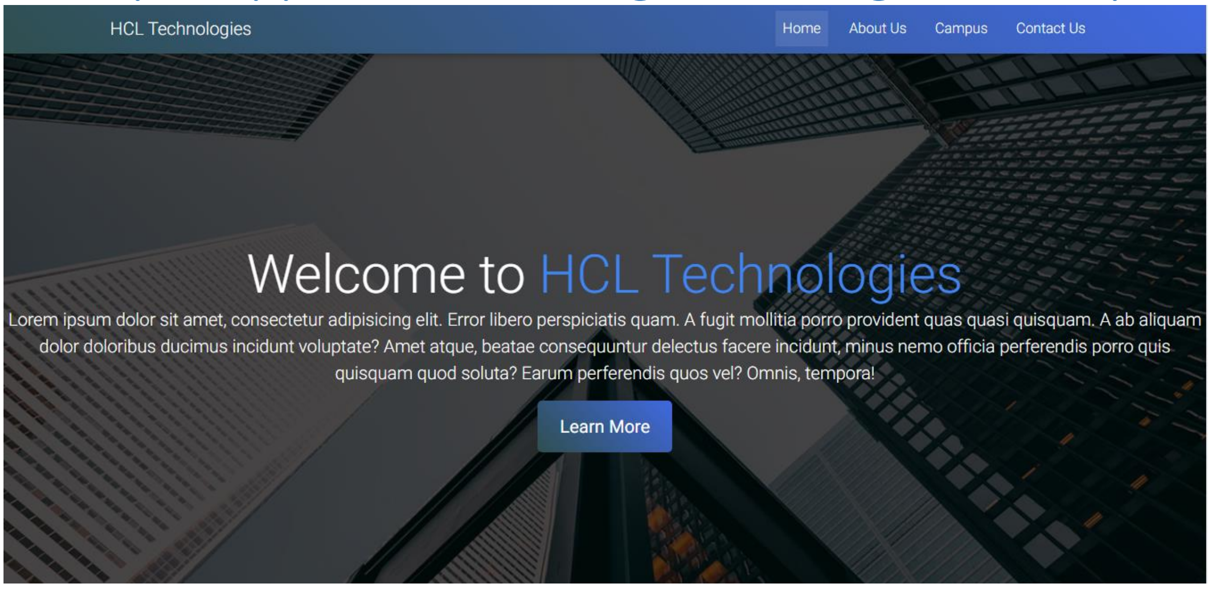
For JavaScript files of Bootstrap, make an entry in "angular.json" file as an array of scripts.



```
  "scripts": [
    "src/bootstrap/js/jquery-3.3.1.min.js",
    "src/bootstrap/js/popper.min.js",
    "src/bootstrap/js/bootstrap.min.js",
    "src/bootstrap/js/mdb.min.js"
  ]
},
```

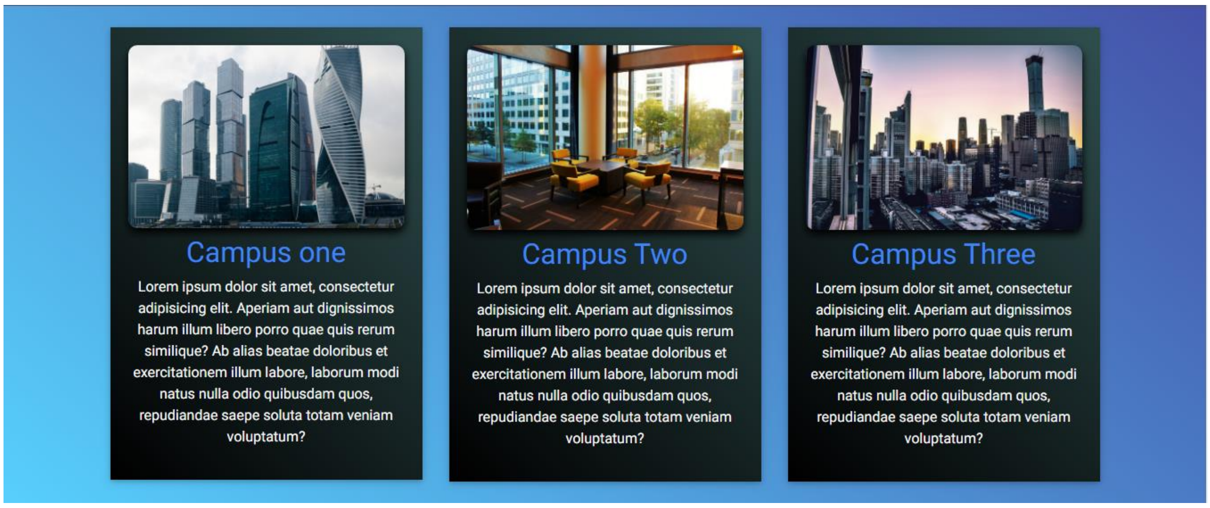
Now, we are ready to start using Bootstrap in Angular.

Example Application on Angular Using Bootstrap



Welcome to HCL Technologies

Lorem ipsum dolor sit amet, consectetur adipisicing elit. Error libero perspiciatis quam. A fugit mollitia porro provident quas quasi quisquam. A ab aliquam dolor doloribus ducimus incidunt voluptate? Amet atque, beatae consequuntur delectus facere incidunt, minus nemo officia perferendis porro quis quisquam quod soluta? Earum perferendis quos vel? Omnis, tempora!



Campus one

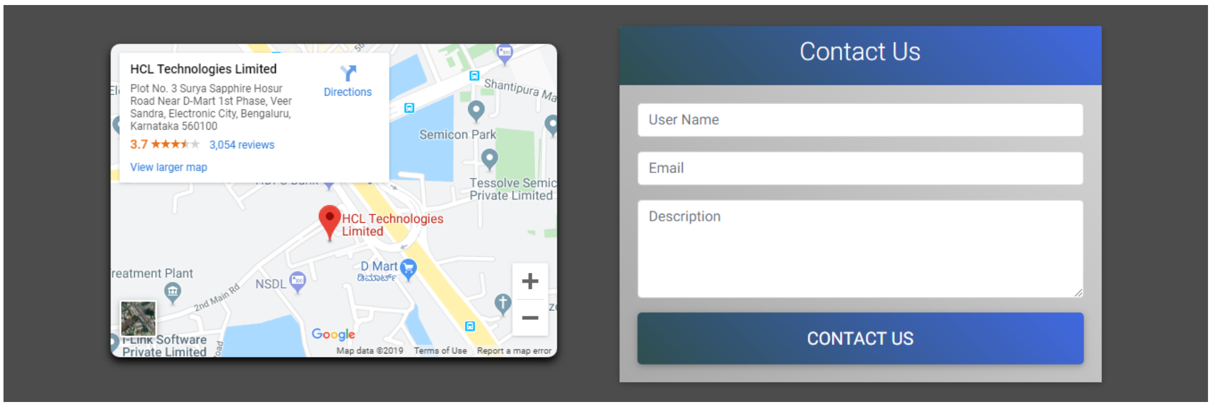
Lorem ipsum dolor sit amet, consectetur adipisicing elit. Aperiam aut dignissimos harum illum libero porro quae quis rerum similique? Ab alias beatae doloribus et exercitationem illum labore, laborum modi natus nulla odio quibusdam quos, repudiandae saepe soluta totam veniam voluptatum?

Campus Two

Lorem ipsum dolor sit amet, consectetur adipisicing elit. Aperiam aut dignissimos harum illum libero porro quae quis rerum similique? Ab alias beatae doloribus et exercitationem illum labore, laborum modi natus nulla odio quibusdam quos, repudiandae saepe soluta totam veniam voluptatum?

Campus Three

Lorem ipsum dolor sit amet, consectetur adipisicing elit. Aperiam aut dignissimos harum illum libero porro quae quis rerum similique? Ab alias beatae doloribus et exercitationem illum labore, laborum modi natus nulla odio quibusdam quos, repudiandae saepe soluta totam veniam voluptatum?


HCL Technologies Limited
Plot No. 3 Surya Sapphire Hosur Road Near D-Mart 1st Phase, Veer Sandra, Electronic City, Bengaluru, Karnataka 560100
Directions
3.7 ★★★★★ 3,054 reviews
View larger map
Shantipura Ma
Semicon Park
Tessolve Semic Private Limited
HCL Technologies Limited
reatment Plant
NSDL
D Mart
2nd Main Rd
T-Link Software Private Limited
Google
Map data ©2019 Terms of Use Report a map error
Contact Us
User Name
Email
Description
CONTACT US

Events Binding In Angular

When a user interacts with your app, it's sometimes necessary to know when this happens. A click, hover, or a keyboard action are all events that you can use to call component logic within Angular.

That's what Angular event binding is all about. It's one-way data binding, in that it sends information from the view to the component class. This is opposite from property binding, where data is sent from the component class to the view.

Whenever you wish to capture an event from the view, you do so by wrapping the event in question with ( ) parenthesis.

Angular provides you with a variety of events from which you can call component logic.

Here are examples of the most common events that you can use for event binding

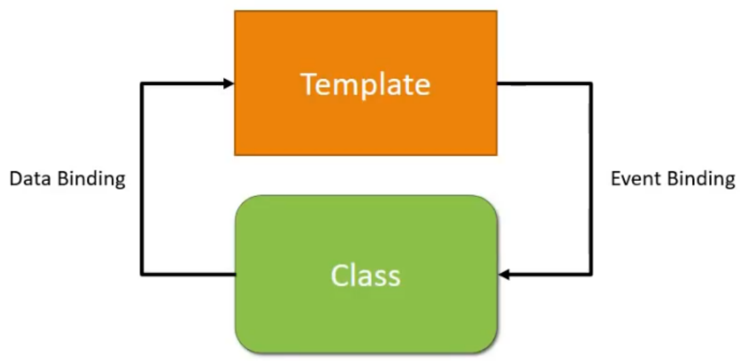
```
(focus)="myMethod()"  // An element has received focus
(blur)="myMethod()"   // An element has lost focus
(submit)="myMethod()"  // A submit button has been pressed
(click)="myMethod()"
(dblclick)="myMethod()"
(scroll)="myMethod()"
(cut)="myMethod()"
(copy)="myMethod()"
(paste)="myMethod()"
```

In our component template, let's create a button with a click event binding:

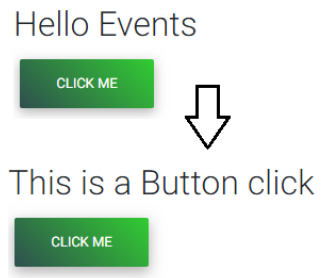
```
<button (click)="myEvent($event)">My Button</button>
```

Notice $event? It's optional, and it will pass along a variety of event properties associated with that particular event. In the component class, let's create the myEvent() method:

```
export class AppComponent {

  myEvent(event) {
    console.log(event);
  }

}
```



```
<div>
  <h2>{{title}}</h2>
  <button class="btn btn-success" (click)="getMessage($event)">Click Me</button>
</div>

 export class AppComponent {
   private title = 'Hello Events';

   public  getMessage(e){
     this.title = "This is a Button click";
   }
 }
```



In the above example, before we click on the button, title was “Hello Events” Once we click on the button, Title changed to “This is a Button click”.

Template Reference Variable in Angular

A template reference variable is often a reference to a DOM element within a template. It can also be a reference to an Angular component or directive or a web component.

That means you can easily access the variable anywhere in the template.

You declare a reference variable by using the hash symbol (#). The #firstNameInput declares a firstNameInput variable on an <input> element.

```
<input type="text" #firstNameInput>
<input type="text" #lastNameInput>
```

After that, you can access the variable anywhere inside the template. For example, I pass the variable as a parameter on an event.

```
<button (click)="show(lastNameInput)">Show</button>
```

Remember that the lastNameInput belongs to HTMLInputElement type.

```
show(lastName: HTMLInputElement){
    console.log(lastName.value);
}
```

Two Way Data Binding in Angular

Two way data binding means, it will sync the data from view to component, and from component to view part.

Two-way data binding is pretty rare in angular. But there is one commonly used directive that makes two-way data binding possible. This directive is called ngModel.

NgModel is part of the angular "FormsModule" and has to be imported into your module manually.

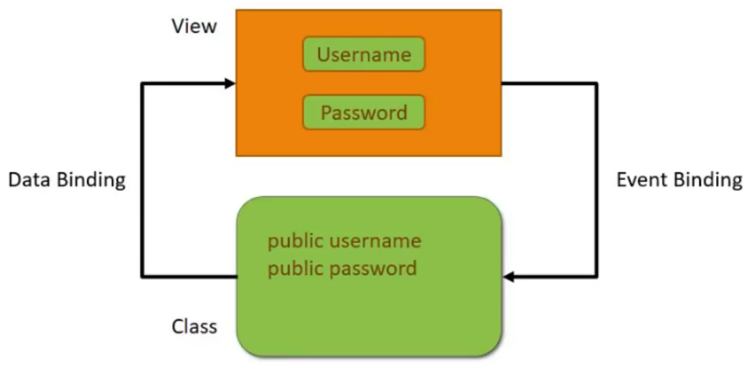
```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-
browser'
import { FormsModule } from '@angular/forms'

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

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

NgModel can be used with form-elements like inputs to implement two-way data binding. To do that, we have to use a pretty special syntax: [(ngModel)]. Its a combination of the one-way- and the event binding syntax.

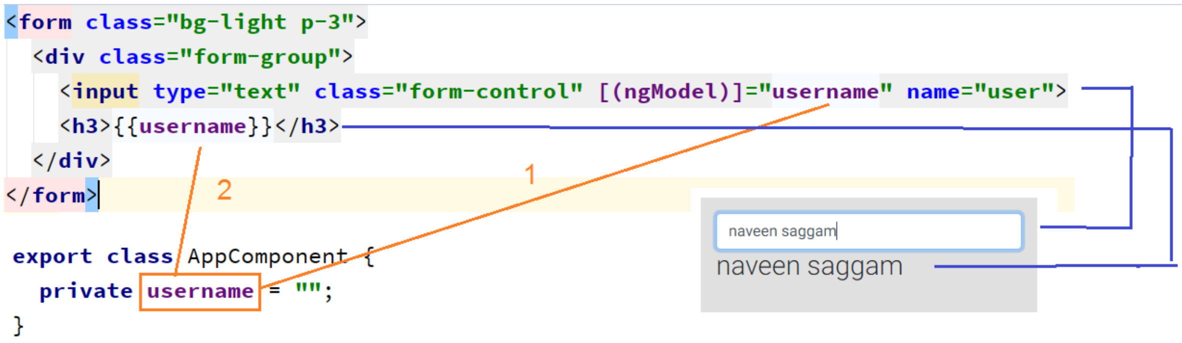
It is used like so:

```
<input [(ngModel)]="name" />
```

Using this syntax the value of the variable "name" is not only shown as the value of the input, but both values change when the user types into the input field.



Show User Name



```
<form class="bg-light p-3">
  <div class="form-group">
    <input type="text" class="form-control" [(ngModel)]="username" name="user">
    <h3>{{username}}</h3>
  </div>
</form>

export class AppComponent {
  private username = "";
}
```

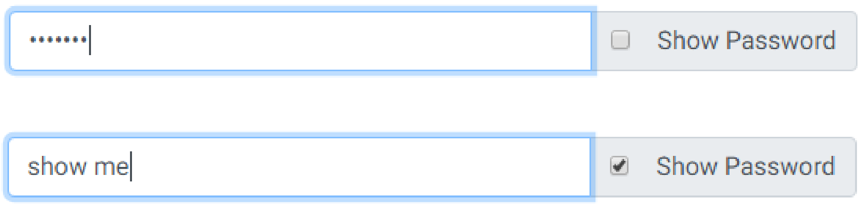
naveen saggam

naveen saggam

Show Password Functionality

```
<form>
  <div class="input-group mb-3">
    <input [type]="password" class="form-control" #passwordElem>
    <div class="input-group-append">
      <div class="input-group-text">
        <input type="checkbox" class="mr-3" (change)="changeCheckBox(passwordElem)"> Show Password
      </div>
    </div>
  </div>
</form>
```

```
export class ChangePasswordComponent implements OnInit {
  private password = "password";

  changeCheckBox(inputElement){
    if(inputElement.getAttribute( attributeName: 'type') === 'password'){
      this.password = 'text';
    }
    else{
      this.password = 'password';
    }
  }
```



•••••••

Show Password

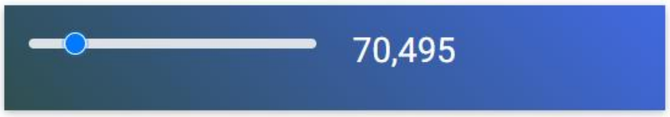
show me

Show Password

Range Selection in Angular

```
<form>
  <div class="row">
    <div class="col">
      <div class="form-group">
        <input min="1000" max="500000" [value]="selectedValue" type="range"
                                       class="custom-range" (change)="getRangeValue($event)">
      </div>
    </div>
    <div class="col">
      <h3>{{selectedValue | number}}</h3>
    </div>
  </div>
</form>
```

```
export class ChangeRangeComponent implements OnInit {
  private selectedValue = 0;

  getRangeValue(event){
    this.selectedValue = event.target.value;
  }
```



Structural Directive (ngIf , ngSwitch , ngFor)

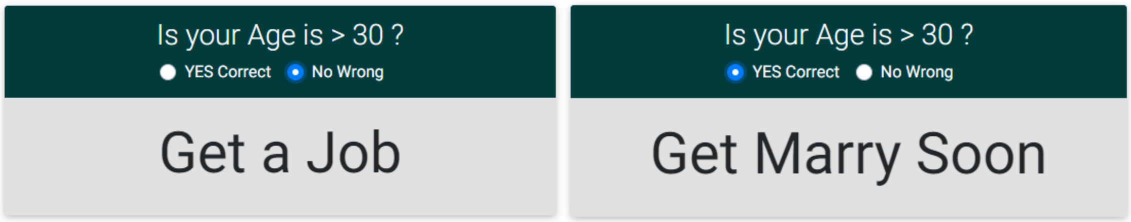
The Structural directives are the directives that lets you add or remove the html elements from the DOM.

The ngIf and ngSwitch are to conditionally render HTML elements.

The ngFor directive is used to render a list of HTML elements.

```
<div class="card-header bg-teal text-white text-center">
  <h3>Is your Age is > 30 ?</h3>
  <div class="custom-control custom-radio custom-control-inline">
    <input type="radio" id="customRadioInline1" [(ngModel)]="ageRadio" value="yes" name="ageRadio" class="custom-control-input">
    <label class="custom-control-label" for="customRadioInline1">YES Correct</label>
  </div>
  <div class="custom-control custom-radio custom-control-inline">
    <input type="radio" id="customRadioInline2" [(ngModel)]="ageRadio" value="no" name="ageRadio" class="custom-control-input">
    <label class="custom-control-label" for="customRadioInline2">No Wrong</label>
  </div>
</div>
<div class="card-body text-center bg-light">
    <h1 *ngIf="ageRadio === 'yes'" class="display-4">Get Marry Soon</h1>
    <h1 *ngIf="ageRadio === 'no'" class="display-4">Get a Job</h1>
</div>
```

```
export class AgeSelectorComponent implements OnInit {
  private ageRadio = null;
  constructor() { }

  ngOnInit() {
  }
```



ngSwitch

In Angular 8, ngSwitch is a structural directive which is used to Add/Remove DOM Element. It is similar to switch statement of C#. The ngSwitch directive is applied to the container element with a switch expression.

```
<container_element [ngSwitch]="switch_expression">
    <inner_element *ngSwitchCase="match_expresson_1">...</inner_element>
    <inner_element *ngSwitchCase="match_expresson_2">...</inner_element>
    <inner_element *ngSwitchCase="match_expresson_3">...</inner_element>
    <inner_element *ngSwitchDefault>...</element>
</container_element>
```

ngSwitchCase

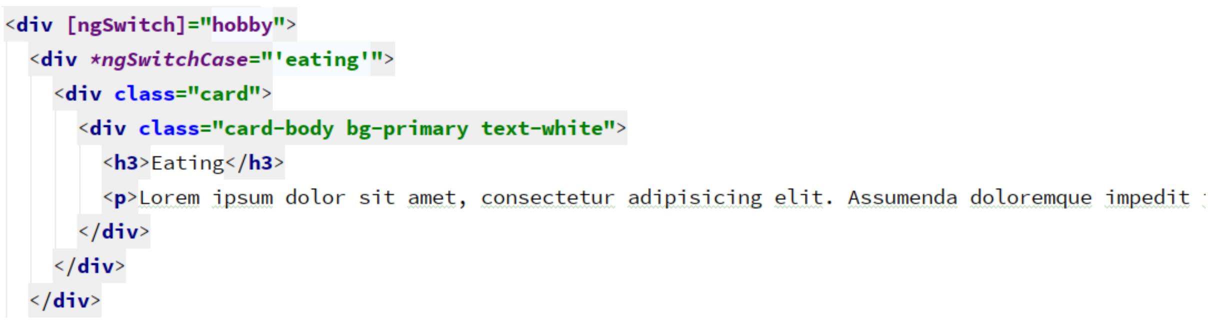
In Angular ngSwitchCase directive, the inner elements are placed inside the container element. The ngSwitchCase directive is applied to the inner elements with a match expression. Whenever the value of the match expression matches the value of the switch expression, the corresponding inner element is added to the DOM. All other inner elements are removed from the DOM

If there is more than one match, then all the matching elements are added to the DOM.

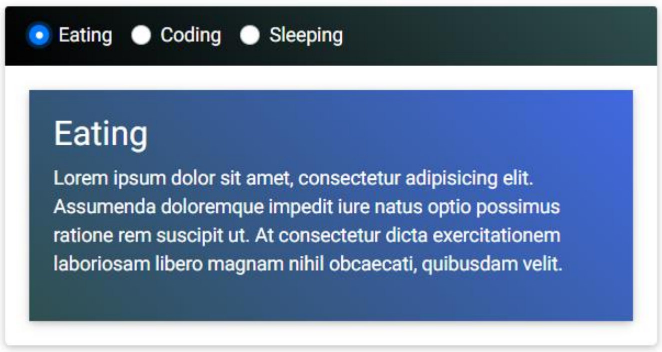
ngSwitchDefault

You can also apply the ngSwitchDefault directive in Angular 8. The element with ngSwitchDefault is displayed only if no match is found. The inner element with ngSwitchDefault can be placed anywhere inside the container element and not necessarily at the bottom. If you add more than one ngSwitchDefault directive, all of them are displayed.

```
<div class="custom-control custom-radio custom-control-inline">
  <input type="radio" id="hobbiesRadioInline1" [(ngModel)]="hobby" value="eating"
         name="hobbiesRadio" class="custom-control-input">
  <label class="custom-control-label" for="hobbiesRadioInline1">Eating</label>
</div>
```



```
<div [ngSwitch]="hobby">
  <div *ngSwitchCase="'eating'">
    <div class="card">
      <div class="card-body bg-primary text-white">
        <h3>Eating</h3>
        <p>Lorem ipsum dolor sit amet, consectetur adipisicing elit. Assumenda doloremque impedit
      </div>
    </div>
  </div>
```



Eating

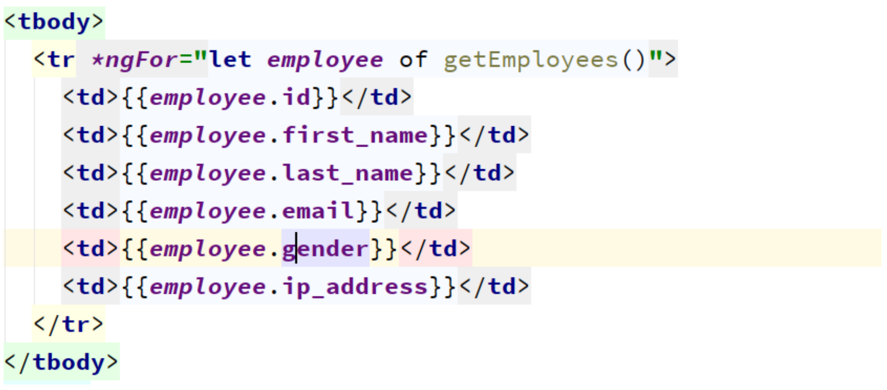
Lorem ipsum dolor sit amet, consectetur adipisicing elit. Assumenda doloremque impedit iure natus optio possimus ratione rem suscipit ut. At consectetur dicta exercitationem laboriosam libero magnam nihil obcaecati, quibusdam velit.

ngFor Directive

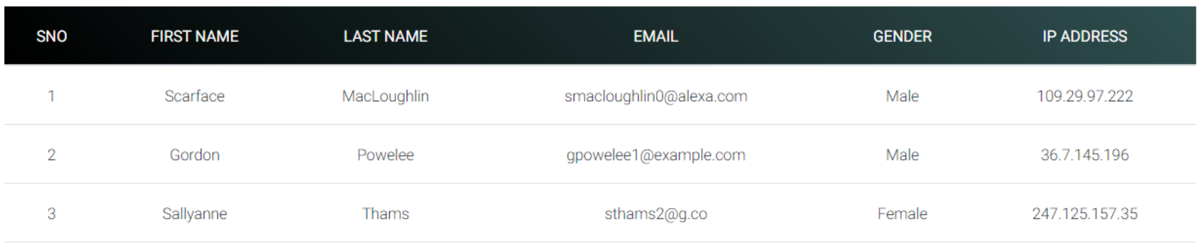
The *ngFor directive is used to repeat a portion of HTML template once per each item from an iterable list (Collection). The ngFor is an Angular structural directive and is similar to ngRepeat in AngularJS. Some local variables like Index, First, Last, odd and even are exported by *ngFor directive.

```
<li *ngFor="let item of items;"> .... </li>
```

To Use ngFor directive, you have to create a block of HTML elements, which can display a single item of the items collection. After that you can use the ngFor directive to tell angular to repeat that block of HTML elements for each item in the list.



```
<tbody>
  <tr *ngFor="let employee of getEmployees()">
    <td>{{employee.id}}</td>
    <td>{{employee.first_name}}</td>
    <td>{{employee.last_name}}</td>
    <td>{{employee.email}}</td>
    <td>{{employee.gender}}</td>
    <td>{{employee.ip_address}}</td>
  </tr>
</tbody>
```



SNO	FIRST NAME	LAST NAME	EMAIL	GENDER	IP ADDRESS
1	Scarface	MacLoughlin	smacloughlin0@alexa.com	Male	109.29.97.222
2	Gordon	Powelee	gpowelee1@example.com	Male	36.7.145.196
3	Sallyanne	Thams	sthams2@g.co	Female	247.125.157.35

Angular Pipes

Pipes allow us to transform the data before we display them on the view. Angular pipes are also called as filters in the earlier versions of Angular JS. It is denoted by symbol |

Pipe takes integers, strings, arrays, and date as input separated with |. It transforms the data in the format as required and displays the same in the browser.

Ex : pipes : lowercase , uppercase , title case , slice , json, number , currency ,percent , date

```
<h2>{{courseName | lowercase}}</h2>
<h2>{{courseName | uppercase}}</h2>
<h2>{{courseName | titlecase}}</h2>
<h2>{{message | titlecase}}</h2>
```

```
<h2>{{courseName | slice: start: 3: end: 5}}</h2>
```

```
<h2>{{student | json}}</h2>
```

```
<h2>{{ 10.2562 | number : digitsInfo: '1.2-3'}}</h2>
<h2>{{ 10.2562 | number : digitsInfo: '3.4-5'}}</h2>
<h2>{{ 10.2562 | number : digitsInfo: '3.2-3'}}</h2>
```

```
<h2>{{0.25 | percent}}</h2>
```

```
<h2>{{today | date : format: 'short'}}</h2>
<h2>{{today | date : format: 'medium'}}</h2>
<h2>{{today | date : format: 'shortDate'}}</h2>
<h2>{{today | date : format: 'mediumDate'}}</h2>
<h2>{{today | date : format: 'shortTime'}}</h2>
<h2>{{today | date : format: 'mediumTime'}}</h2>
```

Angular Services

We might come across a situation where we need some code to be used everywhere on the page. It can be for data connection that needs to be shared across components, etc. Services help us achieve that. With services, we can access methods and properties across other components in the entire project.

We mostly use this services to share the business logic with other components.

This services are also used to implement reusable business logic or application logic.

Services are also used to connects to external environments like Servers and databases.

Naming convention for service is “.service.ts”

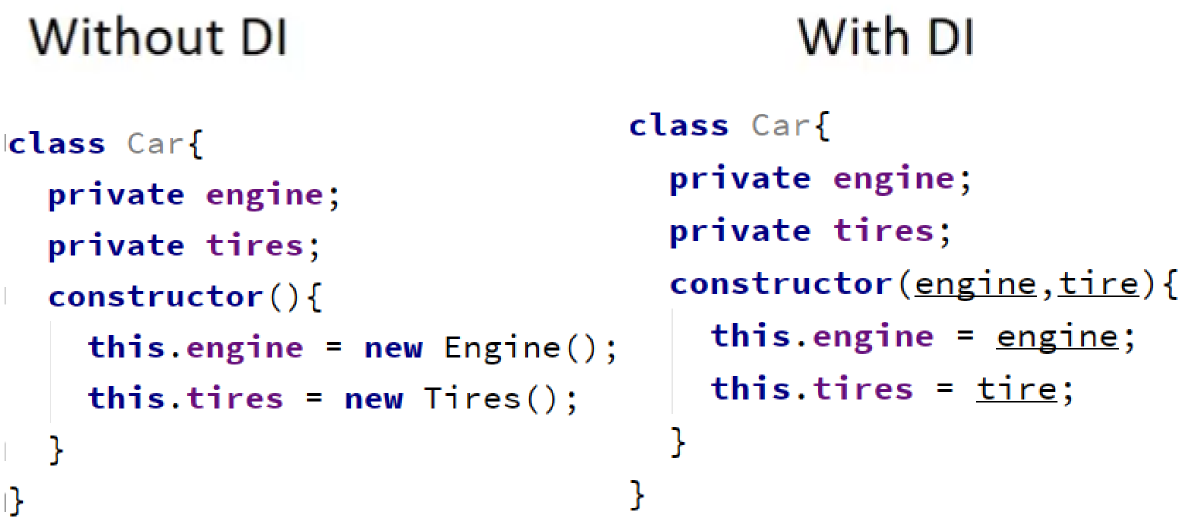
We can create a service in angular application using the angular CLI as follows

Ex: ng g s <services_name>

Services can be available for us as dependency Injection.

Dependency Injection in Angular

Dependency Injection is a coding pattern in which a class receives its dependencies from external sources rather than creating themselves.

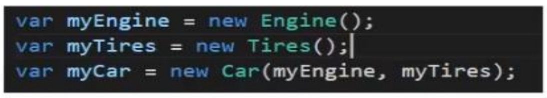


Without DI

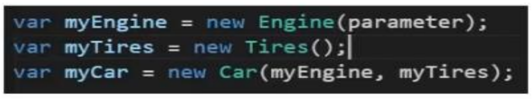
```
class Car{
  private engine;
  private tires;
  constructor(){
    this.engine = new Engine();
    this.tires = new Tires();
  }
}
```

With DI

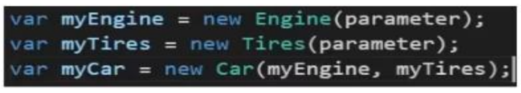
```
class Car{
  private engine;
  private tires;
  constructor(engine,tire){
    this.engine = engine;
    this.tires = tire;
  }
}
```



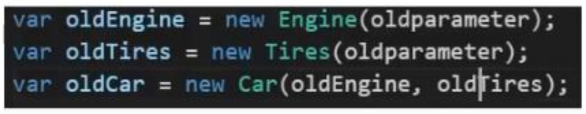
```
var myEngine = new Engine();
var myTires = new Tires();
var myCar = new Car(myEngine, myTires);
```



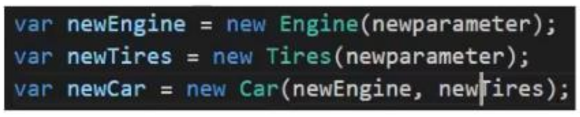
```
var myEngine = new Engine(parameter);
var myTires = new Tires();
var myCar = new Car(myEngine, myTires);
```



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



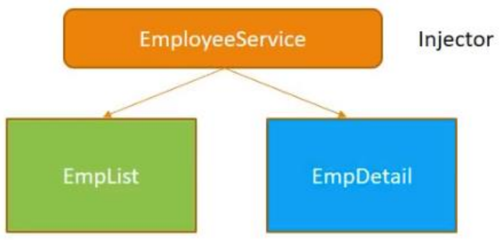
```
var oldEngine = new Engine(oldparameter);
var oldTires = new Tires(oldparameter);
var oldCar = new Car(oldEngine, oldTires);
```



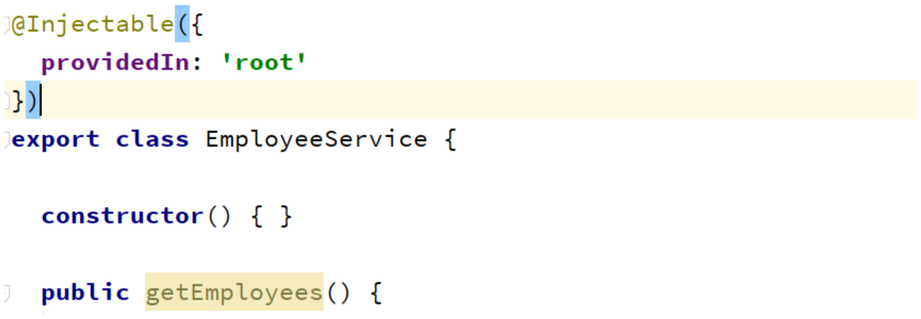
```
var newEngine = new Engine(newparameter);
var newTires = new Tires(newparameter);
var newCar = new Car(newEngine, newTires);
```

How Dependency works with Angular with Angular Services

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

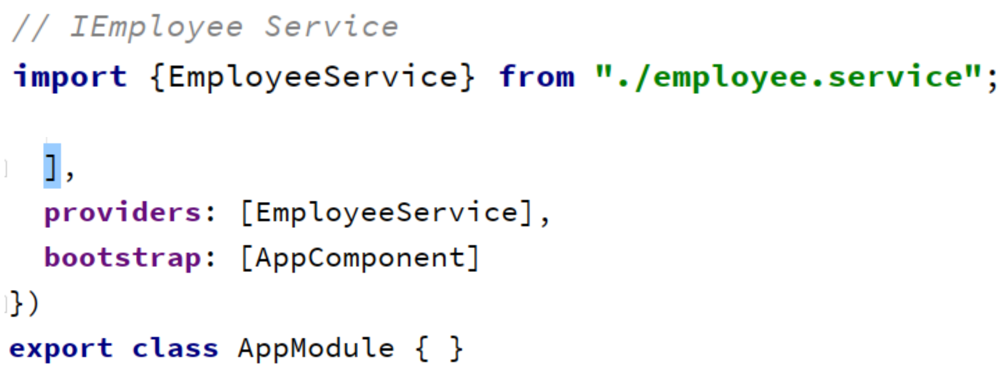


Step 1 : Create an Angular Service using Angualr CLI as “ng g s <Service_name>”



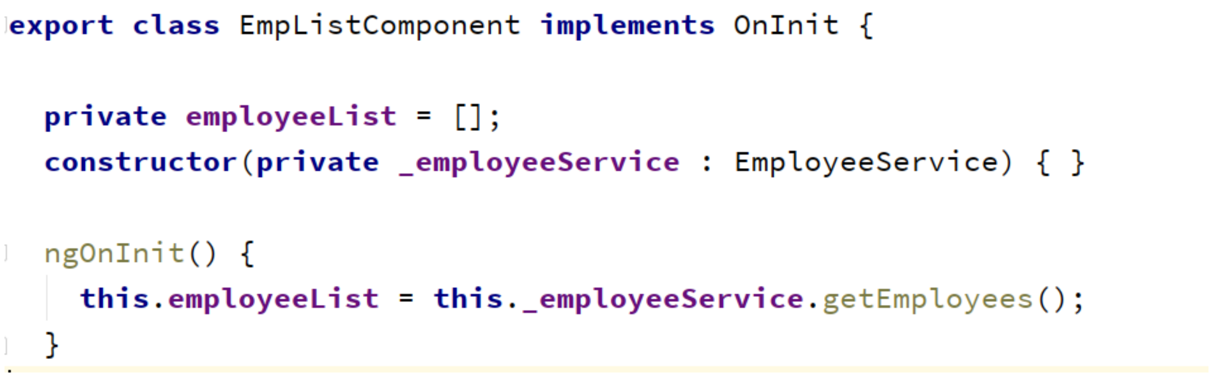
```
@Injectable({
  providedIn: 'root'
})
export class EmployeeService {

  constructor() { }

  public getEmployees() {
```

Step 2 : Declare the service as a provider in app.modules.ts



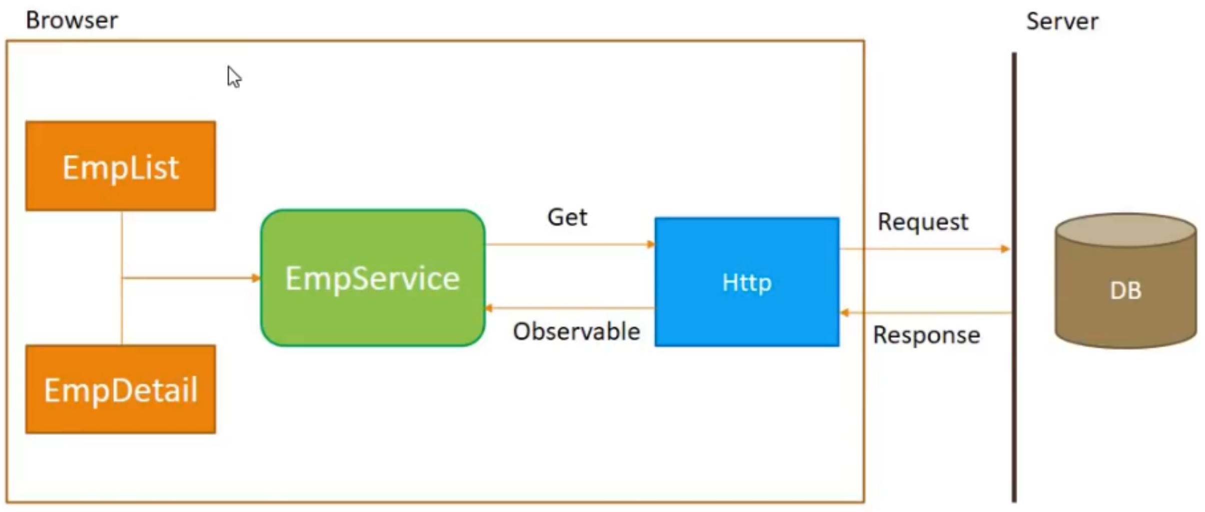
```
// IEmployee Service
import {EmployeeService} from "./employee.service";

  ],
  providers: [EmployeeService],
  bootstrap: [AppComponent]
})
export class AppModule { }
```

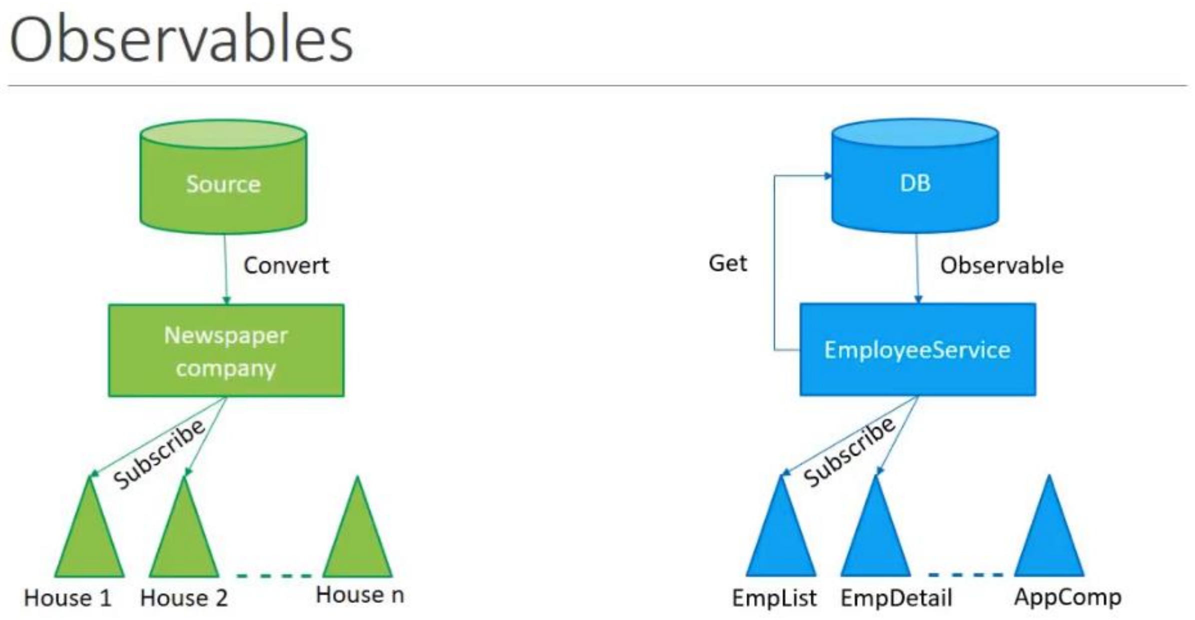
Step 3 : Use the service as a dependency in any of the required component



```
export class EmpListComponent implements OnInit {

  private employeeList = [];
  constructor(private _employeeService : EmployeeService) { }

  ngOnInit() {
    this.employeeList = this._employeeService.getEmployees();
  }
```

HTTP & Observables in Angular





Observables:

An Observable is a sequence of items that arrives asynchronously over the time.

In Angular, Observable mean a HTTP response which arise from the server to our app asynchronously.

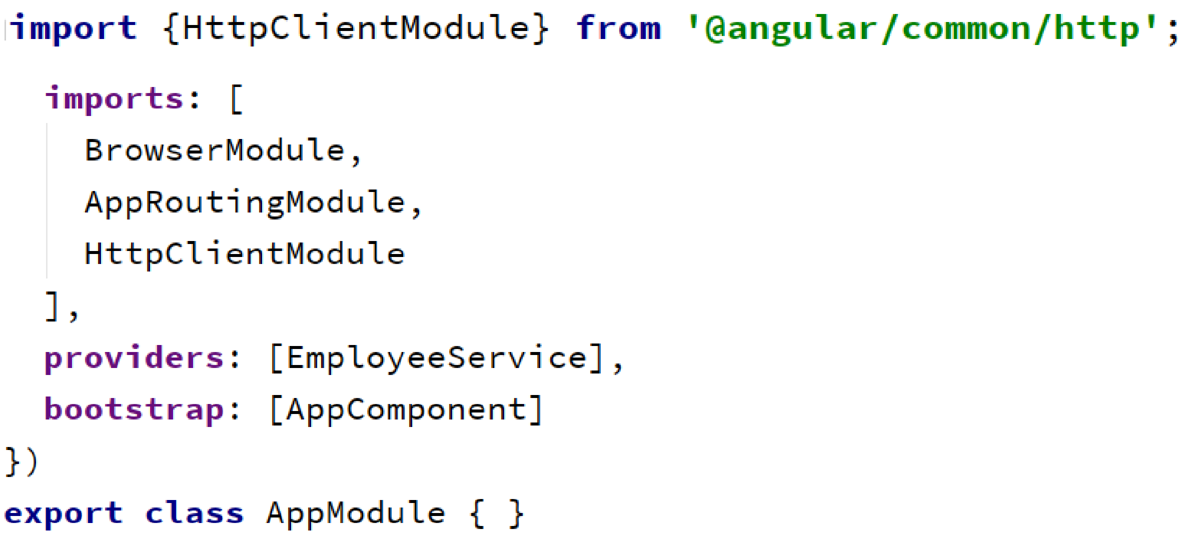
Once we receive the Observables, we need to type cast them to the required format as per the incoming response.

Once we receive the data, which will be available to our components which are subscribed to our Services.

We can make calls using httpClient Module from Angular.

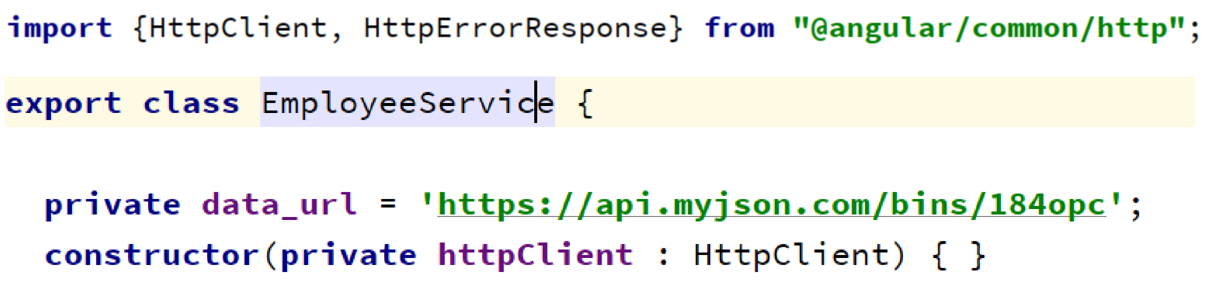
Steps to Fetch the data from a server using httpClient in Angular.

Step 1: Import HttpClientModule in app.module and declare as imports.



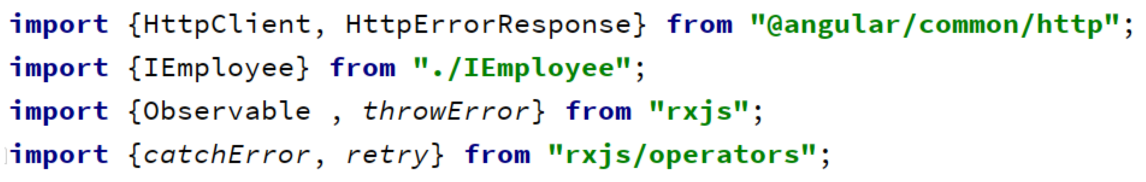
```
import {HttpClientModule} from '@angular/common/http';

  imports: [
    BrowserModule,
    AppRoutingModule,
    HttpClientModule
  ],
  providers: [EmployeeService],
  bootstrap: [AppComponent]
})
export class AppModule { }
```

Step 2 : Inject HttpClient to any Angular Service in the constructor

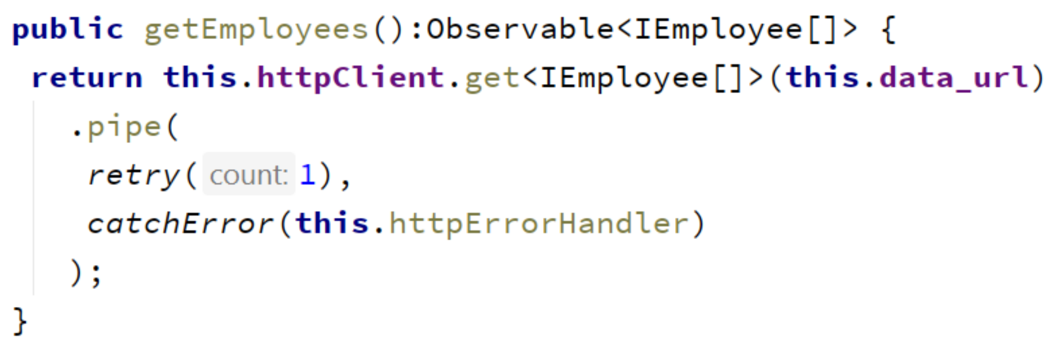


```
import {HttpClient, HttpErrorResponse} from "@angular/common/http";

export class EmployeeService {

  private data_url = 'https://api.myjson.com/bins/184opc';
  constructor(private httpClient : HttpClient) { }
```

Step 3 : Make http calls from the service and get the response as Observables and cast the data to the required type.



```
import {HttpClient, HttpErrorResponse} from "@angular/common/http";
import {IEmployee} from "./IEmployee";
import {Observable , throwError} from "rxjs";
import {catchError, retry} from "rxjs/operators";
```



```
public getEmployees():Observable<IEmployee[]> {
  return this.httpClient.get<IEmployee[]>(this.data_url)
    .pipe(
     retry( count: 1),
     catchError(this.httpErrorHandler)
    );
}
```

```
public httpErrorHandler(error: HttpErrorResponse){
  let errorMessage = '';
  if(error.error instanceof ErrorEvent){
    // client side error
    errorMessage = `Error : ${error.error.message}`;
  }
  else{
    // Server Side Error
    errorMessage = `Error Code: ${error.status}\n Message: ${error.message}`;
  }
  return throwError(errorMessage);
}
```

Step 4 : Use the service in components , we will receive the data, only when we subscribed.

```
import {EmployeeService} from "../employee.service";

export class EmpListComponent implements OnInit {
  private employeeList = [];
  private errorMessage;
  constructor(private _employeeService : EmployeeService) { }
  ngOnInit() {
   this._employeeService.getEmployees().subscribe( next: (data) => {
     this.employeeList = data;
   } ,  error: error => {
     this.errorMessage = error;
   });
  }
}
```

Step 5 : Use the data in Component related Template

```
<tr *ngFor="let employee of employeeList">
  <td>{{employee.id}}</td>
  <td>
    <img src="{{employee.photo}}" alt="" width="25" height="25">
  </td>
  <td>{{employee.first_name | uppercase}} {{employee.last_name |
  <td>{{employee.email}}</td>
  <td>{{employee.gender}}</td>
  <td>{{employee.ip_address}}</td>
</tr>
```

Routing & Navigation in Angular

The Angular Router enables navigation from one view to the next as users perform application tasks.

Overview

The browser is a familiar model of application navigation:

- Enter a URL in the address bar and the browser navigates to a corresponding page.
- Click links on the page and the browser navigates to a new page.
- Click the browser's back and forward buttons and the browser navigates backward and forward through the history of pages you've seen.

The Angular Router ("the router") borrows from this model. It can interpret a browser URL as an instruction to navigate to a client-generated view. It can pass optional parameters along to the supporting view component that help it decide what specific content to present. You can bind the router to links on a page and it will navigate to the appropriate application view when the user clicks a link.

You can navigate imperatively when the user clicks a button, selects from a drop box, or in response to some other stimulus from any source. And the router logs activity in the browser's history journal so the back and forward buttons work as well.

Usage of Routes in Angular

Step 1 : Import AppRoutingModule and connects as imports in app.module

```
import { AppRoutingModule , routingModules } from './app-routing.module';
```

```
  imports: [
    BrowserModule,
    AppRoutingModule
  ],
  providers: [DepartmentService],
  bootstrap: [AppComponent]
})
export class AppModule { }
```

Step 2 : Configure routes in AppRoutingModule with app the required components

```
const routes: Routes = [
  {path : '' , redirectTo : '/' , pathMatch : 'full'} ,
  {path : '' , component : HomePageComponent},
  {path : 'departments' , component : DeptListComponent},
  {path : 'departments/:id' , component : DepartmentDetailComponent},
  {path : 'employees' , component : EmpListComponent},
  {path : '**' , component : PageNotFoundComponent}
];
```

Step 3 : Declare each component in app.module.ts

```
export class AppRoutingModule { }
export const routingModules = [
  DeptListComponent ,
  EmpListComponent ,
  HomePageComponent ,
  PageNotFoundComponent,
  DepartmentDetailComponent];
```

```
import { AppRoutingModule , routingModules } from './app-routing.module';
```

```
@NgModule({
  declarations: [
    AppComponent,
    routingModules
  ],
```

Step 4: Use Routing in app.component.html

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

Step 5 : Use router links even in navigation as well

```
<li class="nav-item">
  <a class="nav-link" routerLink="/departments" routerLinkActive="active">Depa
</li>
<li class="nav-item">
  <a class="nav-link" routerLink="/employees" routerLinkActive="active">Employ
</li>
```

Route Parameters in Angular

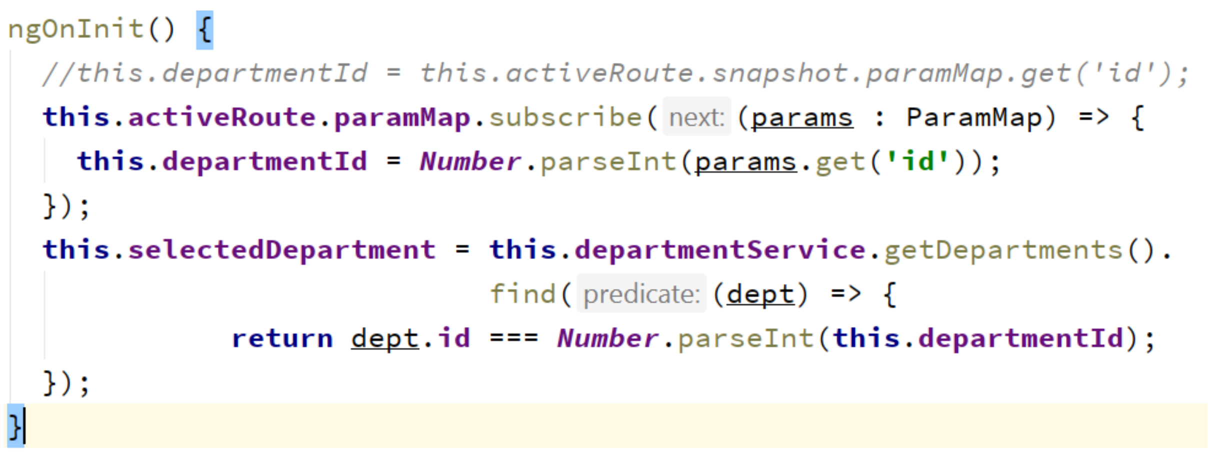
We can even pass few parameters in Angular such as

```
<li class="list-group-item list-group-item-action">
  <a routerLink="/departments/{{department.id}}">
    <span class="badge badge-danger">{{department.id}}</span>
    {{department.department}}
  </a>
</li>
```

```
{path : 'departments/:id' , component : DepartmentDetailComponent},
```

```
import { ActivatedRoute , ParamMap } from '@angular/router';
```

```
constructor(private departmentService : DepartmentService ,
            private activeRoute : ActivatedRoute) { }
```



```
ngOnInit() {
  //this.departmentId = this.activeRoute.snapshot.paramMap.get('id');
  this.activeRoute.paramMap.subscribe( next: (params : ParamMap) => {
    this.departmentId = Number.parseInt(params.get('id'));
  });
  this.selectedDepartment = this.departmentService.getDepartments().
                                  find( predicate: (dept) => {
              return dept.id === Number.parseInt(this.departmentId);
  });
}
```

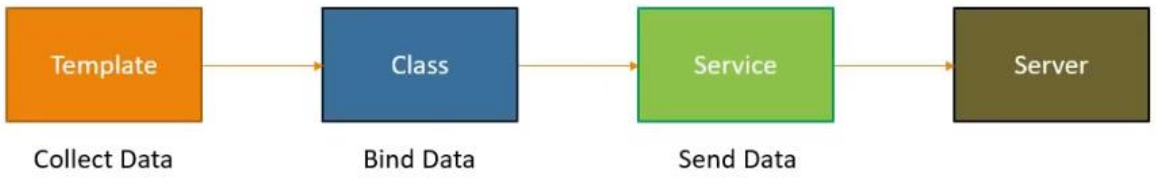
Form Validations in Angular

For every web application, the forms will place a vital role which can be any type of forms like registration, login, ticket booking, Online Payments, booking Appointments and other tasks.

To make the forms more effective and efficient we need to provide a better user experience.

For this as a developed we need to perform the following actions Forms.

- Data Binding in forms
- Change tracking of each form controls
- Provide validations for form fields
- Apply a visual Feedback
- Display error messages
- Form Submission to server side environments



Angular form submission contains the above process. First a Template which collects the form data and binds the form data to a component class.

This collected form data will be sent to the server side environments using Angular Services.

We can achieve the form validation in Angular in two ways

- Template Driven Forms (heavy code on Component Templates)
- Reactive Forms (heavy code on Component Class)

Template Driven Forms (TDF)

In Template Driven forms most of the form validation code resides in components Templates and less code in component classes.

These are easy to use and similar to Angular JS Forms.

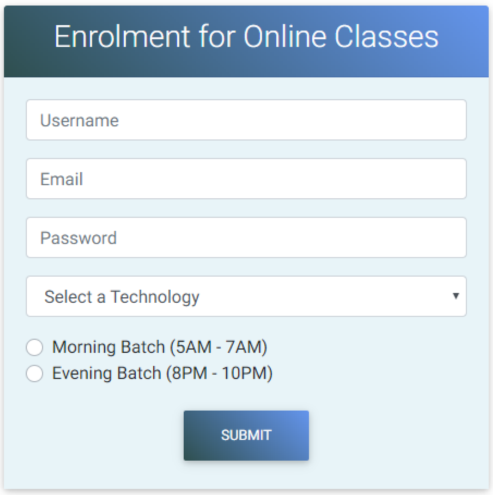
For two-way data binding, we will use ngModel directive along with ngForm Directive to track the form data , each form field state and validity.

Only one small drawback of this approach is Unit Testing is bit Complex.

Here bulky code on html side and less code on component side, this mostly used for the simple forms with few validations and if we don't need of any kind of unit testing.

In this approach we will follow the below steps

- Creating forms with Bootstrap framework
- Binding data with template and component
- Tracking of form control state and validity
- Providing visual feedbacks for validations
- Posting data to server side environments
- Displaying error / success messages on the template


Enrolment for Online Classes
Username
Email
Password
Select a Technology
Morning Batch (5AM - 7AM)
Evening Batch (8PM - 10PM)
SUBMIT

In order to work with forms in this approach, first we need to import 'FormsModule' in app.module.ts file.

```
import { FormsModule} from "@angular/forms";
```

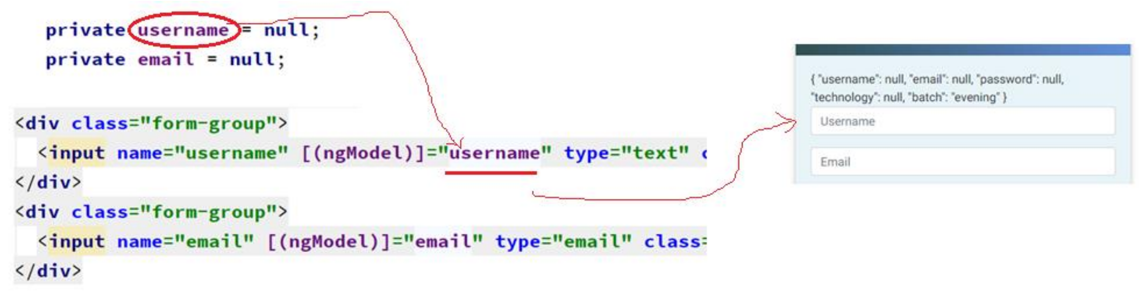
```
  imports: [
    BrowserModule,
    AppRoutingModule,
    FormsModule
  ],
  providers: [],
  bootstrap: [AppComponent]
})
export class AppModule { }
```

Once we import FormsModule, the Angular automatically starts looking into HTML forms and track of each form controls data with an angular Directive ‘ngForm’.

We can see the activities of ngForm directive using a template Reference variable.

```
{{enrolmentForm.value | json}}
<form #enrolmentForm = 'ngForm'>
```

We can track and apply two-way data binding to form fields using ‘ngModel’ directive.



```
  private username = null;
  private email = null;

<div class="form-group">
  <input name="username" [(ngModel)]="username" type="text" c
</div>
<div class="form-group">
  <input name="email" [(ngModel)]="email" type="email" class=
</div>
```

```
{ "username": null, "email": null, "password": null, "technology": null, "batch": "evening" }
```

The best way of declaring forms fields is to attach them to a model class and use those properties of model.

Let’s create an enrolment.ts file for a UI Model.

```
export class Enrolment {
  public username : string;
  public email : string;
  public password : string;
  public technology : string;
  public batch : string;

  constructor(username,email,password,technology,batch){
    this.username = username;
    this.email = email;
    this.password = password;
    this.technology = technology;
    this.batch = batch;
  }
}
```

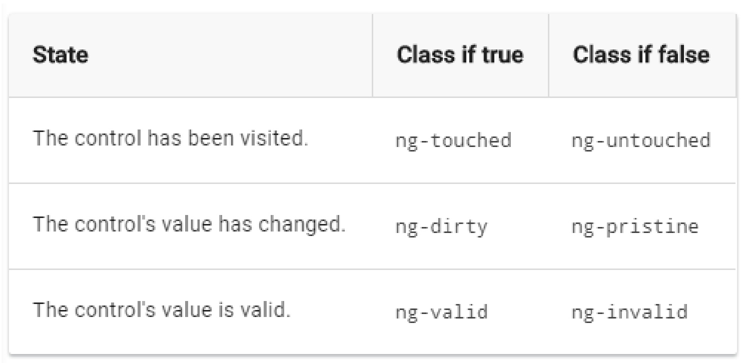
Use this model to create the form data in enrolment component

```
private enrollment = new Enrolment( username: '', email: '', password: '', technology: '', batch: '');
```

And also use this in the template with ngModel directive.

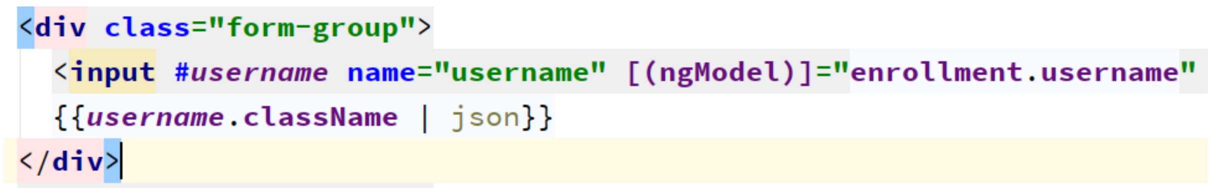
```
<div class="form-group">
  <input name="username" [(ngModel)]="enrollment.username" type="text
</div>
<div class="form-group">
  <input name="email" [(ngModel)]="enrollment.email" type="email" cla
</div>
```

Now its time to track the state and validity of each form field, Angular will apply the following classes based on form control state and its validity.

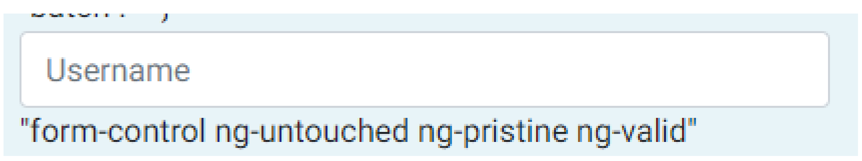


State	Class if true	Class if false
The control has been visited.	`ng-touched`	`ng-untouched`
The control's value has changed.	`ng-dirty`	`ng-pristine`
The control's value is valid.	`ng-valid`	`ng-invalid`

We can test the state of each input field or form control is by using template reference variables.

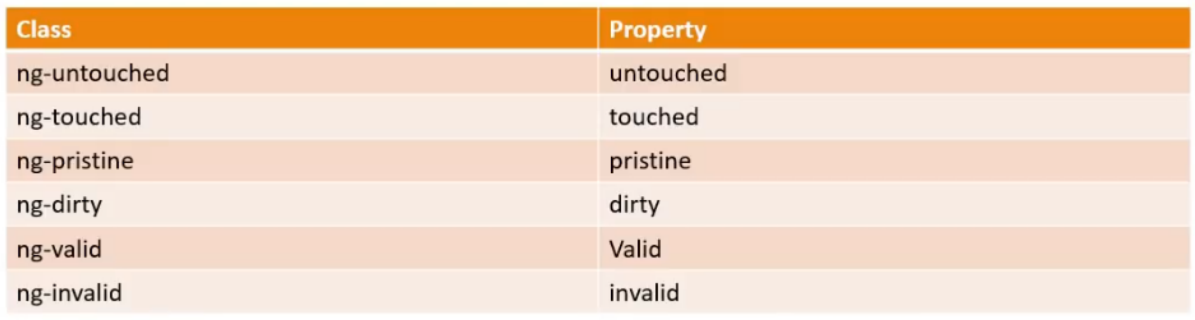


```
<div class="form-group">
  <input #username name="username" [(ngModel)]="enrollment.username"
  {{username.className | json}}
</div>
```



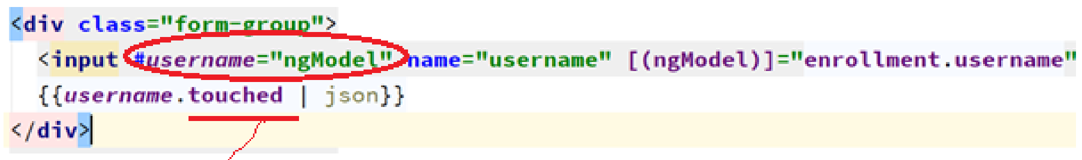
"form-control ng-untouched ng-pristine ng-valid"

We can either use these classes to display a visual feedbacks on pages, or Angular also provides an alternate properties on ngModel directive to display visual feedbacks.

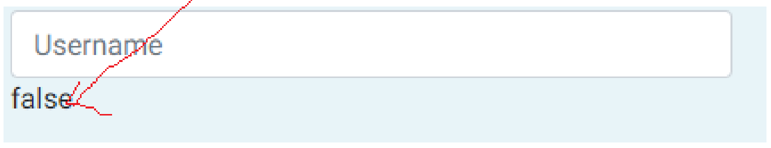


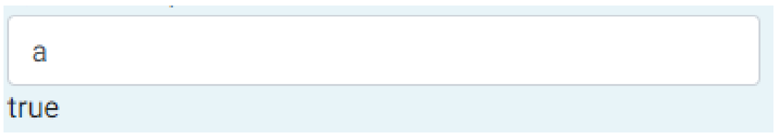
Class	Property
ng-untouched	untouched
ng-touched	touched
ng-pristine	pristine
ng-dirty	dirty
ng-valid	Valid
ng-invalid	invalid

Now we can attach ngModel directive to each form-field and make use these properties to display a visual feedbacks on the pages.

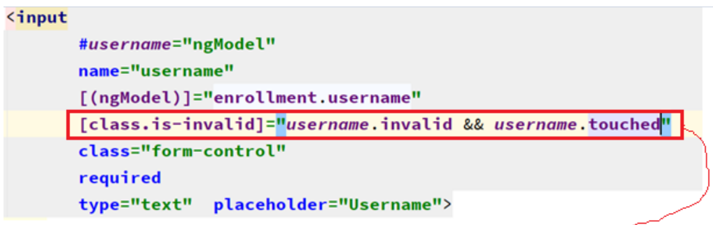


```
<div class="form-group">
  <input #username="ngModel" name="username" [(ngModel)]="enrollment.username"
  {{username.touched | json}}
</div>
```

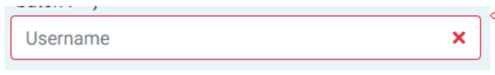




Now we can apply various Bootstrap classes to highlight the form fields in the condition of these properties of ngModel Directives.

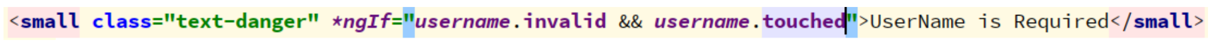


```
<input
      #username="ngModel"
      name="username"
      [(ngModel)]="enrollment.username"
      [class.is-invalid]="username.invalid && username.touched"
      class="form-control"
      required
      type="text"  placeholder="Username">
```

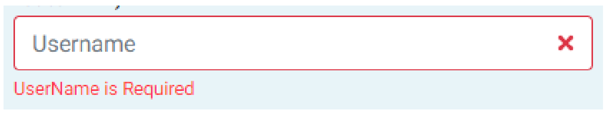


it dynamically adds the class 'is-invalid', when the form-field is invalid.

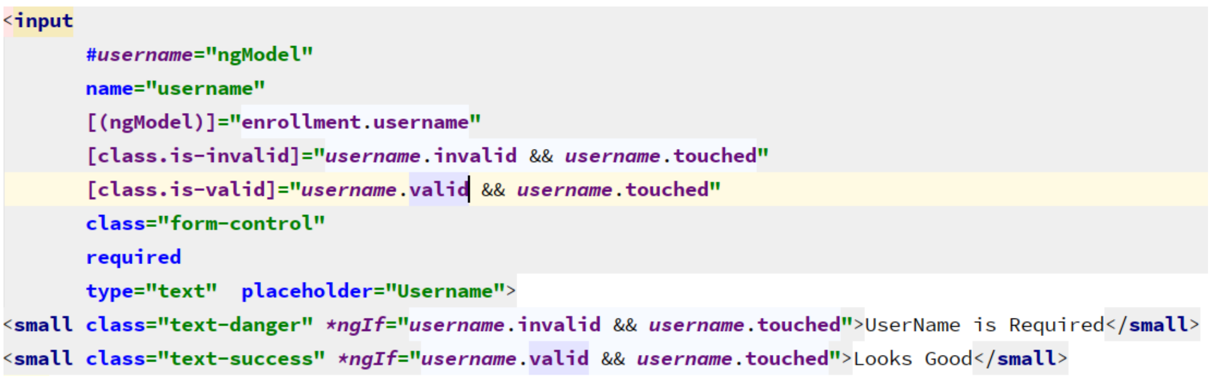
To display error message, we can use again Bootstrap classes based on form validations.



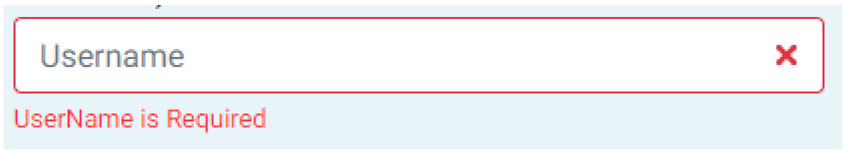
```
<small class="text-danger" *ngIf="username.invalid && username.touched">UserName is Required</small>
```



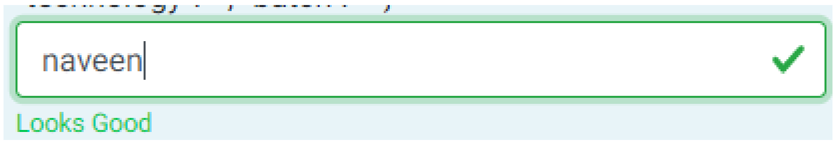
We can even write a more elaborated validation messages with following code.



```
<input
      #username="ngModel"
      name="username"
      [(ngModel)]="enrollment.username"
      [class.is-invalid]="username.invalid && username.touched"
      [class.is-valid]="username.valid && username.touched"
      class="form-control"
      required
      type="text"  placeholder="Username">
<small class="text-danger" *ngIf="username.invalid && username.touched">UserName is Required</small>
<small class="text-success" *ngIf="username.valid && username.touched">Looks Good</small>
```

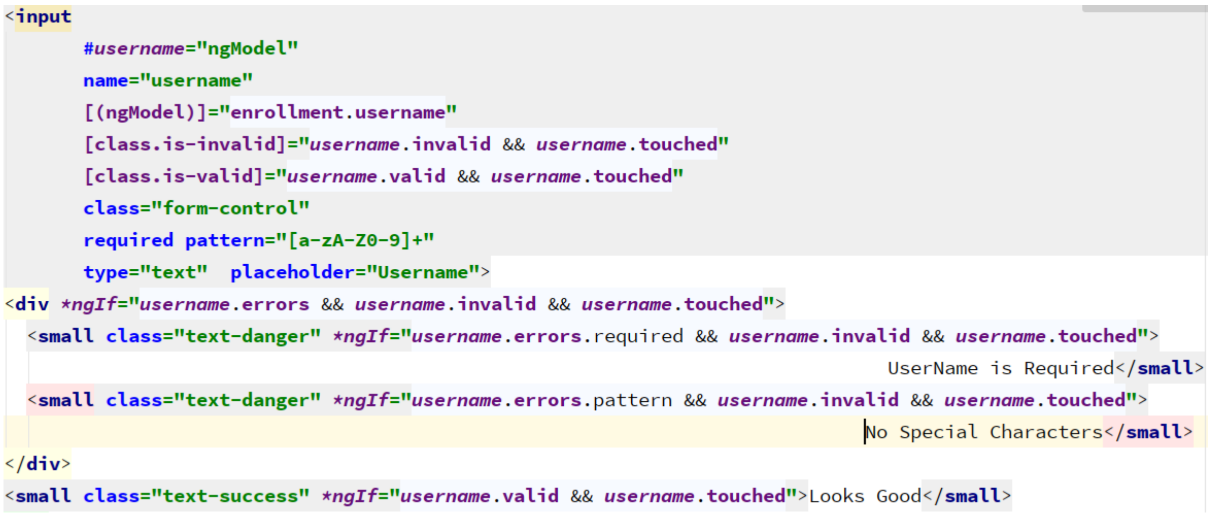

Username

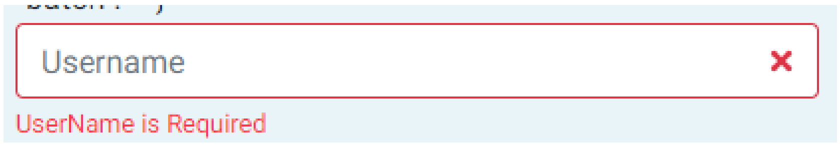
UserName is Required


naveen

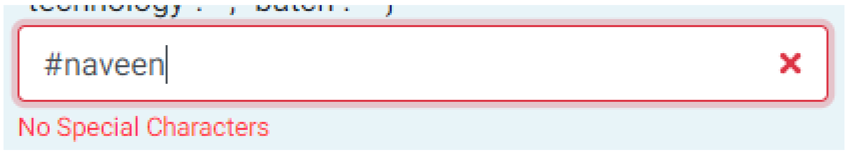
Looks Good

We can even apply multiple validations on each form-field using ‘errors’ property.

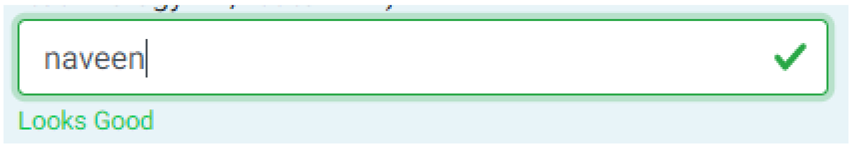

```
<input
       #username="ngModel"
       name="username"
       [(ngModel)]="enrollment.username"
       [class.is-invalid]="username.invalid && username.touched"
       [class.is-valid]="username.valid && username.touched"
       class="form-control"
       required pattern="[a-zA-Z0-9]+"
       type="text"  placeholder="Username">
<div *ngIf="username.errors && username.invalid && username.touched">
  <small class="text-danger" *ngIf="username.errors.required && username.invalid && username.touched">
                                                                          UserName is Required</small>
  <small class="text-danger" *ngIf="username.errors.pattern && username.invalid && username.touched">
                                                                        No Special Characters</small>
</div>
<small class="text-success" *ngIf="username.valid && username.touched">Looks Good</small>
```


Username

UserName is Required


#naveen

No Special Characters


naveen

Looks Good

Now it’s time to send the form data to server through Angular Services.

Reactive Forms

Here in this process, there is no two-way binding with ngModel. The complete code has to be written in a component class.

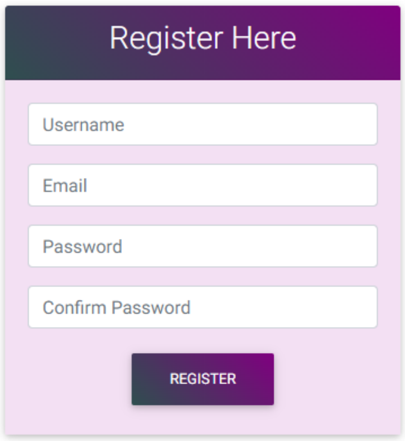
Bulky code in Component class and less code in component Template.

Recommended for larger forms with Complex validations and also best suitable for custom validations.

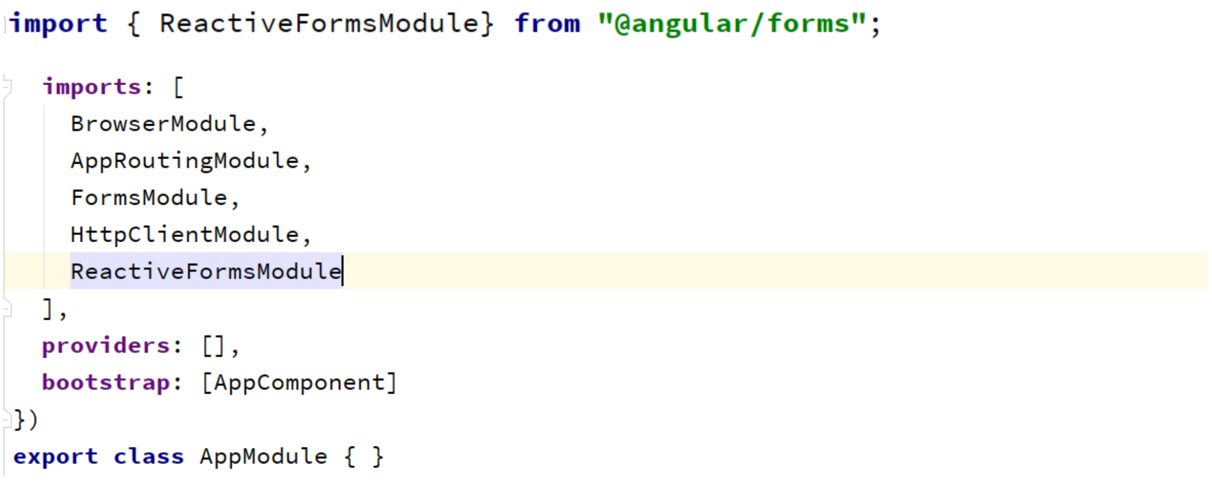
Also supports the unit testing of the forms, because the validation logic resides in component class.

In Reactive forms, we will follow the below steps

- Create a Bootstrap Form
- Create the form model
- Manage the form control values with component class
- Using Form Builder Service to populate form fields / form controls
- Apply various validations
- Display Visual feedback on component template
- Submitting the form data to server
- Displaying validation messages on Template

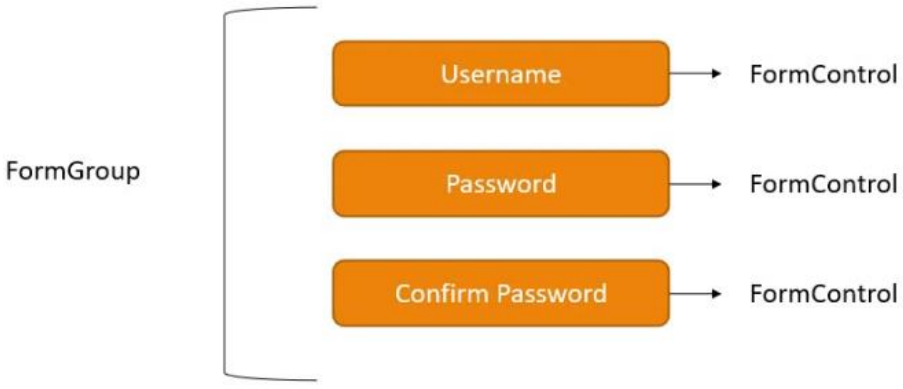


In order to work with Reactive Forms first we have to import ReactiveFormsModule in app.module.ts.

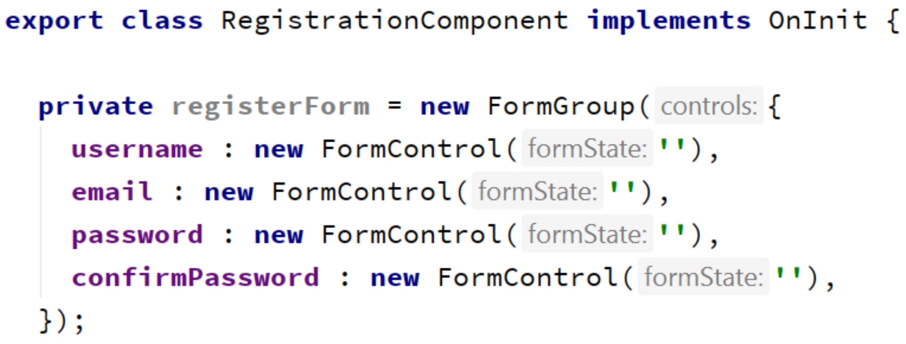


```
import { ReactiveFormsModule} from "@angular/forms";

  imports: [
    BrowserModule,
    AppRoutingModule,
    FormsModule,
    HttpClientModule,
    ReactiveFormsModule
  ],
  providers: [],
  bootstrap: [AppComponent]
})
export class AppModule { }
```

In Reactive forms approach, Each HTML form is represented with a FormGroup and each form-field is represented with FormControl in Angular.

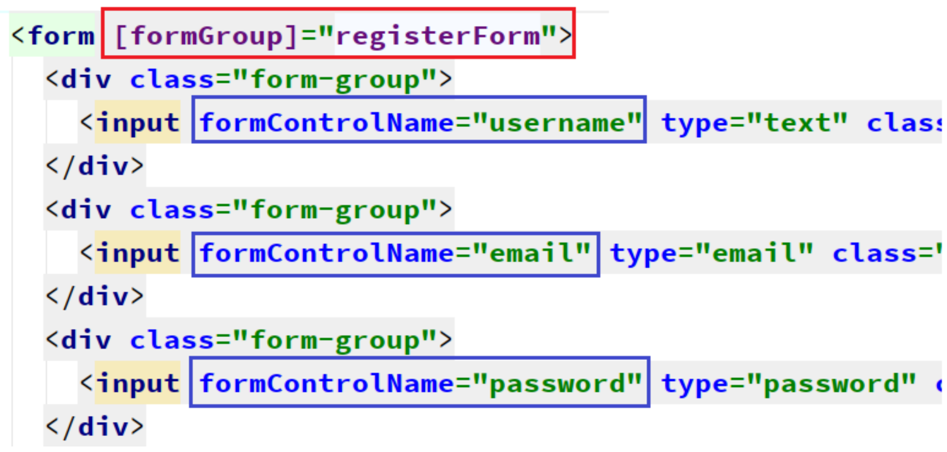


We can represent the whole form in component class is as follows,



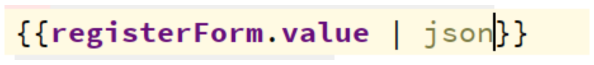
```
export class RegistrationComponent implements OnInit {

  private registerForm = new FormGroup( controls: {
    username : new FormControl( formState: ''),
    email : new FormControl( formState: ''),
    password : new FormControl( formState: ''),
    confirmPassword : new FormControl( formState: ''),
  });
```

Now it’s time attach these data to actual form controls in component template.

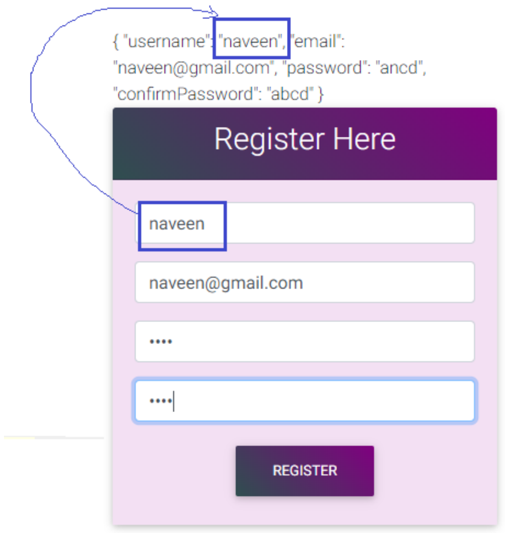


```
<form [formGroup]="registerForm">
  <div class="form-group">
    <input formControlName="username" type="text" clas
  </div>
  <div class="form-group">
    <input formControlName="email" type="email" class='
  </div>
  <div class="form-group">
    <input formControlName="password" type="password" c
  </div>
```

Now we have a connection between component class and component template. We can even test the form control data using interpolation.



```
{{registerForm.value | json}}
```



{ "username": "naveen", "email": "naveen@gmail.com", "password": "ancd", "confirmPassword": "abcd" }

The best way to create the form controls is by using Angular’s **FormBuilder** Service

So, first we need to import FormBuilder to our component class, and start using it create form-controls.

```
// Inject FormBuilder from Angular
constructor(private formBuilder : FormBuilder) { }

// use FormBuilder to create form-controls
private registerForm = this.formBuilder.group( controlsConfig: {
    username : [''],
    email : [''],
    password : [''],
    confirmPassword : [''],
});
```

Now it’s time to add validations to our form.

JWT Authentication in Angular