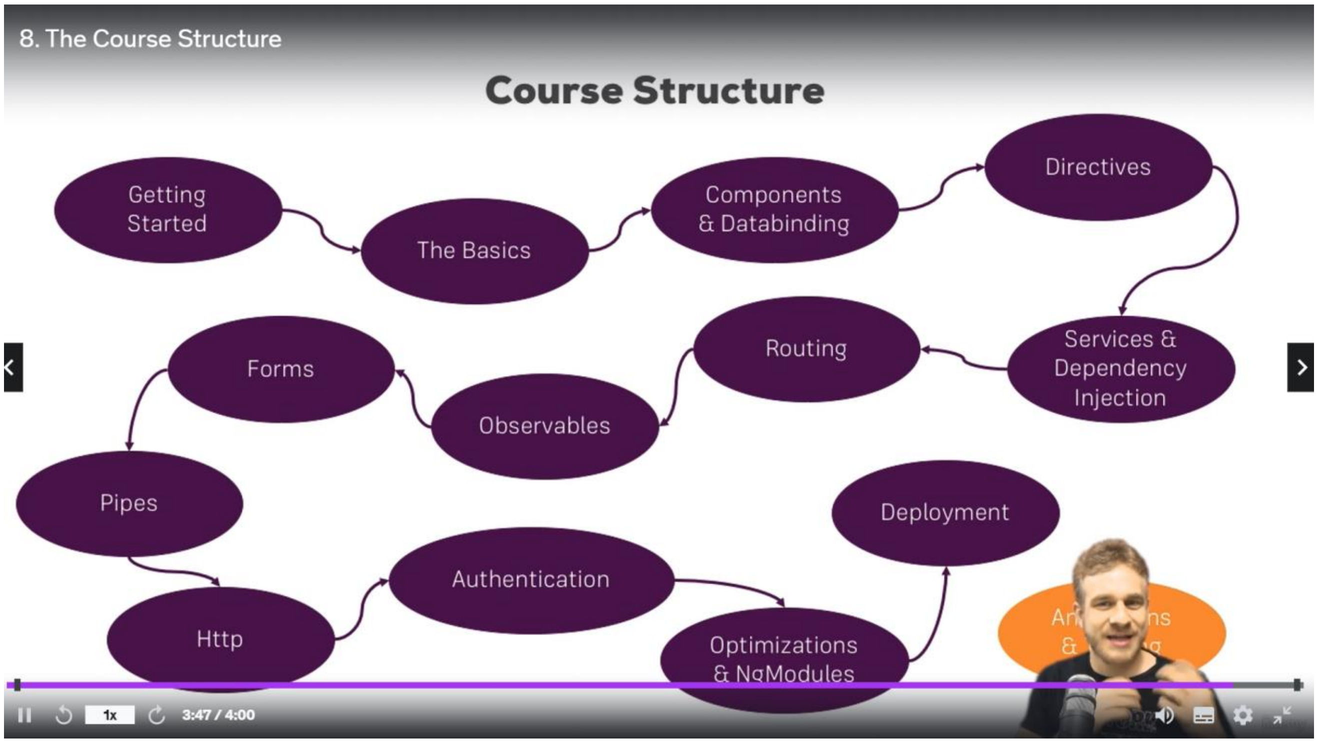
Angular



Angular syllabus links: -

https://www.acil.in/angular-syllabus/

A. Angular Architecture

1. Templates: - The HTML View of Angular.
2. Component: - Binds the view and model.
3. Modules: - Groups components logically.
4. Binding: - Defines how view and component communicate.
5. Directives: - changes the HTML DOM behavior.
6. Services: - Help to share common logic across the project.
7. DI: - Dependency injection helps to inject the instance across constructor.

1. Templates

https://www.c-sharpcorner.com/article/templates-in-angular/

Overview

We are going to discuss the below points:

- Templates basics
- Types of Templates
- Elements of Templates
- Implementation of Event

What is Template

Templates in Angular represent a view whose role is to display data and change the data whenever an event occurs. Its default language for templates is HTML.

How to create template

There are two ways of defining templates,

1. Inline Template
2. External Template

First, we are going to discuss about Inline template. So, let's get started.

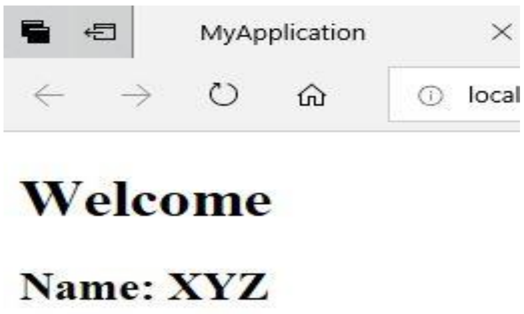
Inline Template

We can create template inline into component class itself using template property of @Component decorator.

Open HiComponent.ts file and add the below line of code into @Component decorator,

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

@Component({
  selector: 'app-hi',
  template: `
  <h1> Welcome </h1>
  <h2> Name: {{ Name }}</h2>
`,
  styleUrls: ['./hi.component.css']
     })
     export class HiComponent implements OnInit {
       Name : string = "XYZ";
       constructor() { }

       ngOnInit(): void {
       }
     }
```

Now run the application and you will see the below output screen,



Welcome

Name: XYZ

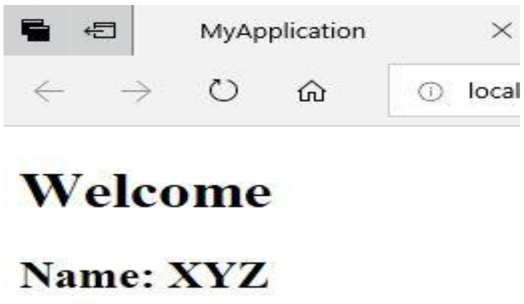
External Template

By default, Angular CLI uses the external template. It binds template with a component using templateUrl option. TemplateUrl is used in External format whereas in case of intline Template we use template instead of templateurl.

Now open Hi.component,html and add the below line of code:

```
<h1>Hello</h1>
<h2>Name : {{Name}}</h2>
```

Observe the output screen and you will see the below output on the screen:



Welcome

Name: XYZ

Elements of Templates

1. HTML
2. Interpolation
3. Template Expressions
4. Template Statements

Let's start with the explanation of each one of the template elements 👉

HTML

Angular uses HTML as a template language.

Interpolation

Interpolation is one of the forms of data binding where we can access a component's data in a template. For interpolation, we use double curly braces {{ }}.

Template Expressions

The text inside {{ }} is called as template expression.
Ex,

```
{{Expression}}
```

Angular first evaluates the expression and returns the result as a string. The scope of a template expression is a component instance. That means, if we write {{ Name }}, Name should be the property of the component to which this template is bound to.

Template Statements

Template Statements are the statements which respond to a user event.

```
(event) = {{Statement}}
```

Ex : lets create an click event - Add changename() method inside hi.component.ts file as below,

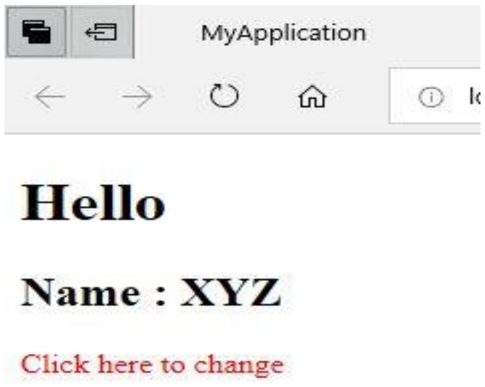
```
import { Component, OnInit } from '@angular/core';

@Component({
  selector: 'app-hi',
  templateUrl: `./hi.component.html`,
  styleUrls: ['./hi.component.css']
})
export class HiComponent implements OnInit {
  Name : string = "XYZ";
       changeName() {
         this.Name = "ABC";
     }
       constructor() { }
       ngOnInit(): void {
       }
     }
```

Now open hi.component.html and add the below line of code inside it,

```
<h1>Hello</h1>
<h2>Name : {{Name}}</h2>
<p (click)="changeName()">Click here to change</p>

```

Here, changeName() method is bound to click event which will be invoked on click at run time.
This is called event binding

Now observe the output screen in the browser,

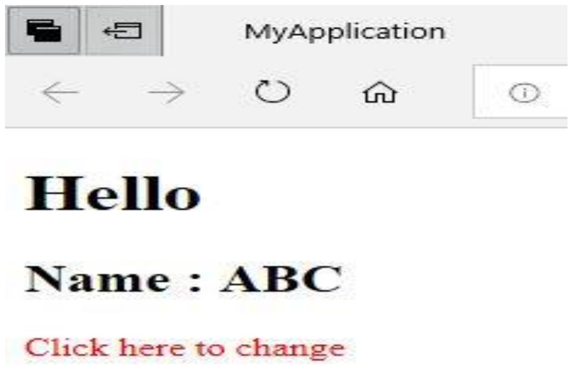


Hello

Name : XYZ

Click here to change

Output after clicking. When user clicks on the paragraph, course name will be changed to 'ABC'



Hello

Name : ABC

Click here to change

Summary

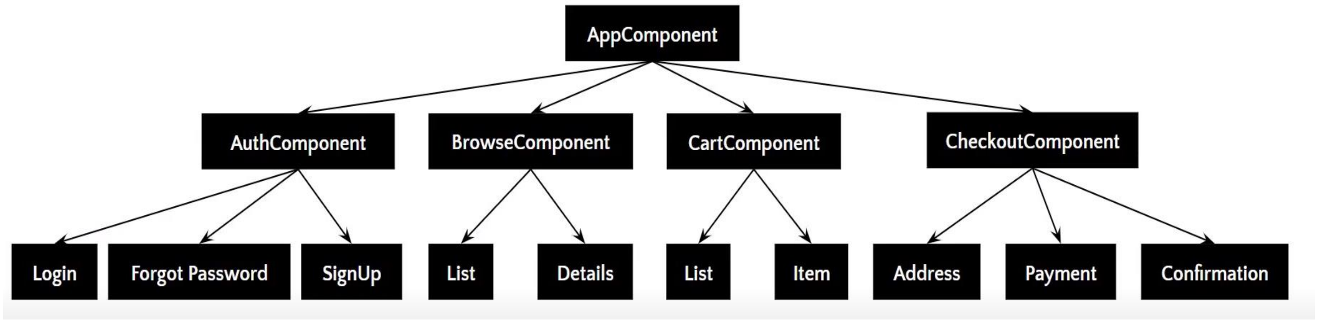
In this article, we have discussed about templates basics, types of templates, elements of templates and implementation of the events.

2. Component

https://www.tutorialsteacher.com/angular/angular-component

- Components are the main building block for Angular applications. Each component consists of: An HTML template that declares what renders on the page. A TypeScript class that defines behavior. A CSS selector that defines how the component is used in a template.
- Components control their runtime behavior by implementing Life-Cycle hooks.

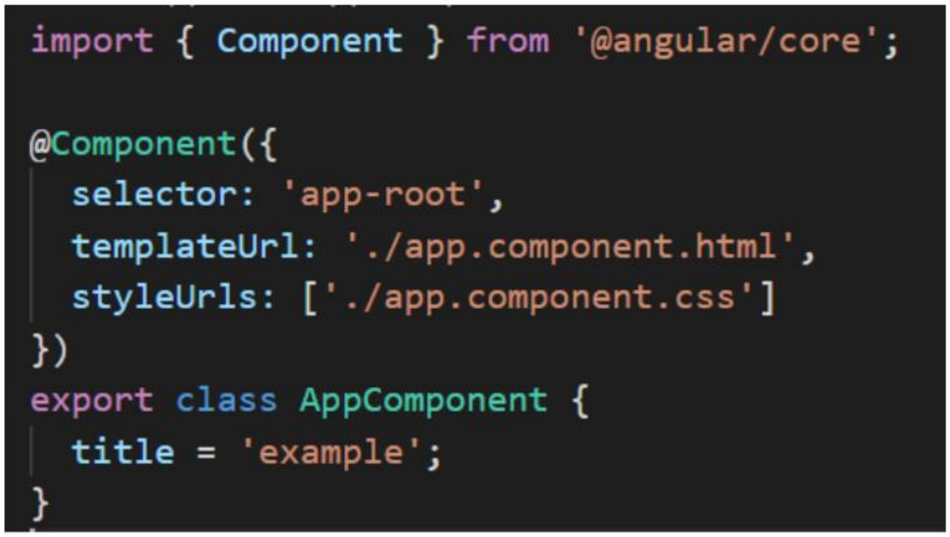
Angular



The above image gives the tree structure of classification. There's a root component, which is the AppComponent, that then branches out into other components creating a hierarchy.

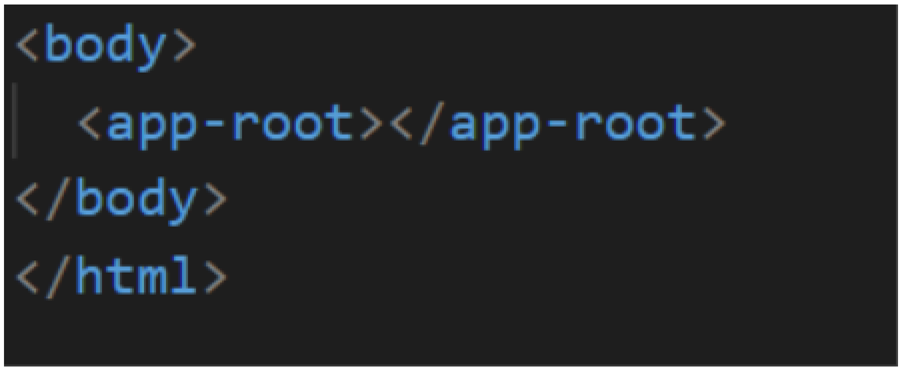
- Each component consists of: -

1. HTML Template
2. TypeScript
3. CSS Selector



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

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

In the index.html file, <app-root> tag corresponds to component's selector. By doing so, Angular will inject the corresponding template of the component.

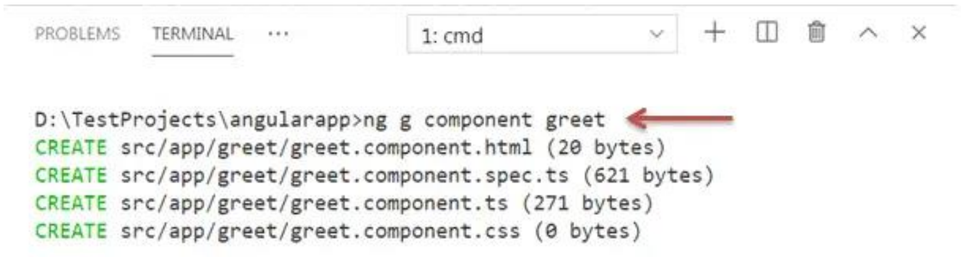


```
<body>
  <app-root></app-root>
</body>
</html>
```

Generate Angular Component using Angular CLI

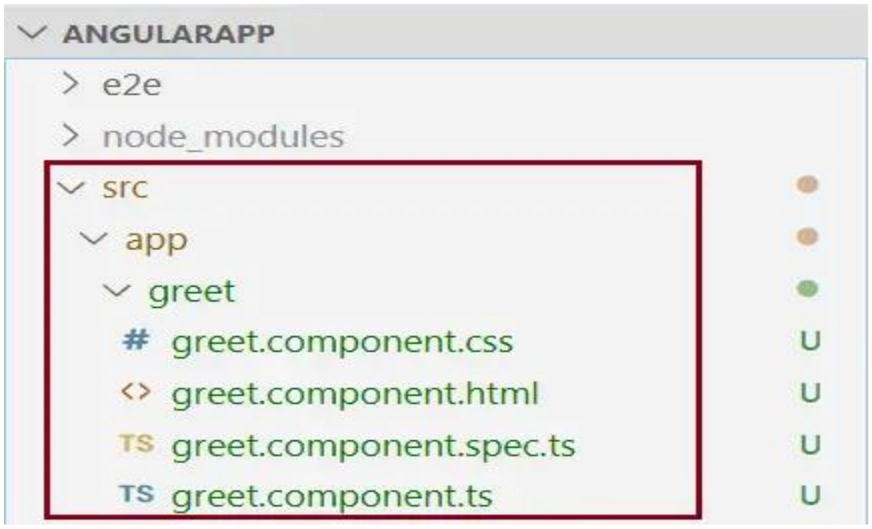
- Creating a Component using Angular CLI : -
  ng generate component <component-name>
  Example: -

  ng g c greet

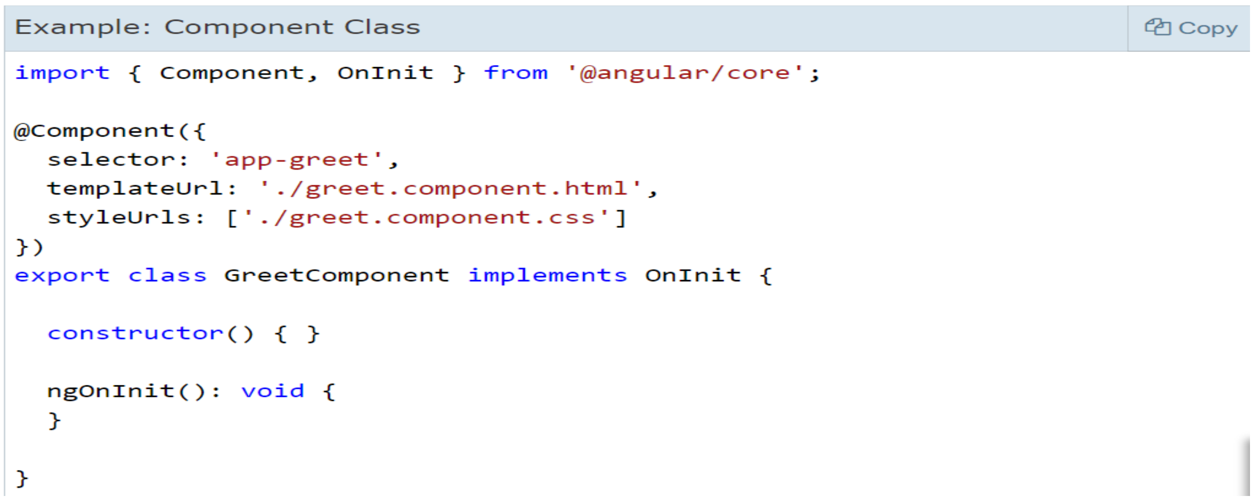


```
D:\TestProjects\angularapp>ng g component greet
CREATE src/app/greet/greet.component.html (20 bytes)
CREATE src/app/greet/greet.component.spec.ts (621 bytes)
CREATE src/app/greet/greet.component.ts (271 bytes)
CREATE src/app/greet/greet.component.css (0 bytes)
```

The above command will create a new "greet" folder and app folder and create four files, as shown below



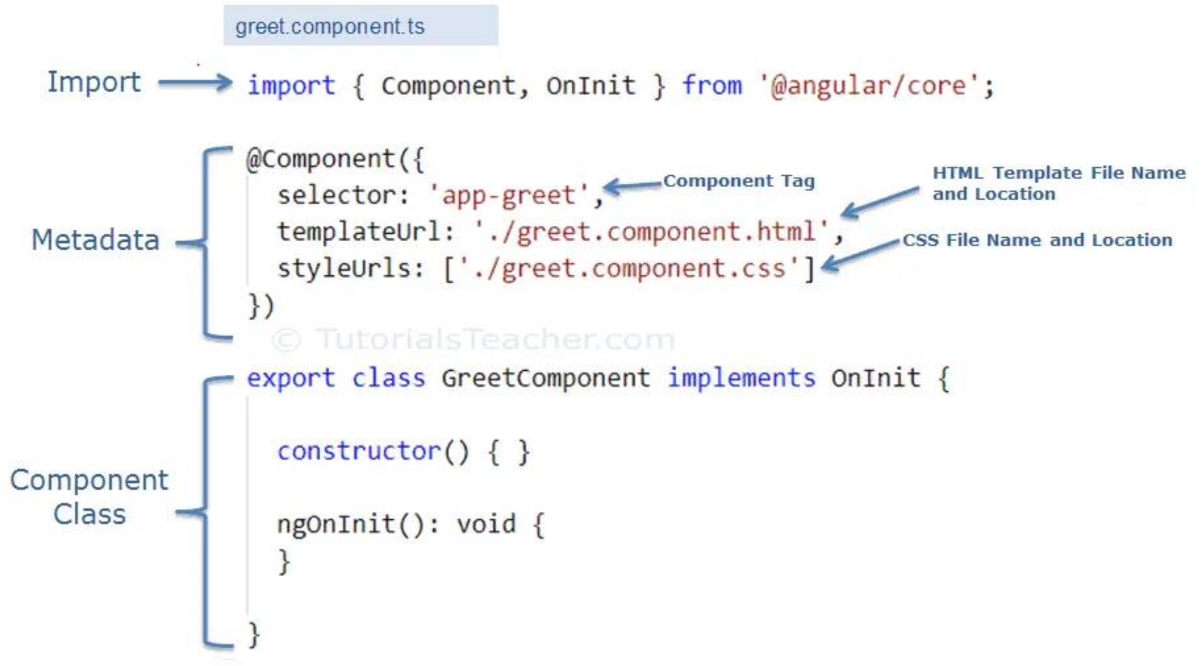
Now, open `greet.component.ts` file in VS Code, and you will see the following code.



Example: Component Class

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

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

  constructor() { }

  ngOnInit(): void {
  }

}
```

The following figure illustrates the important part of the component class.



greet.component.ts

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

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

  constructor() { }

  ngOnInit(): void {
  }

}
```

★ Types of Components in Angular / Component communication / parent vs child component

https://angular.io/guide/inputs-outputs

There are two types of components in Angular: Parent and Child.

- A common pattern in Angular is sharing data between a parent component and one or more child components. Implement this pattern with the `@Input()` and `@Output()` decorators.

- Consider the following hierarchy:

```
<parent-component>
 <child-component></child-component>
</parent-component>
```

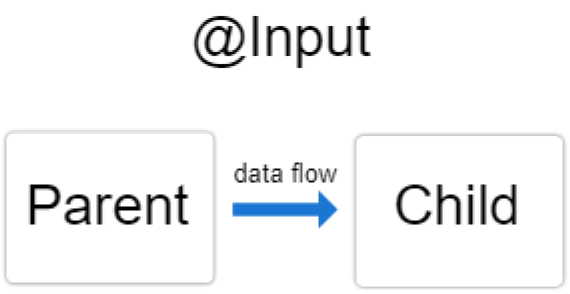
The `<parent-component>` serves as the context for the `<child-component>`.

`@Input()` and `@Output()` give a child component a way to communicate with its parent component.

`@Input()` lets a parent component update data in the child component. Conversely, `@Output()` lets the child send data to a parent component.

1. Sending data to a child component from parent (@input)

The `@Input()` decorator in a child component or directive signifies that the property can receive its value from its parent component.



To use `@Input()`, you must configure the parent and child.

Step1:-Configuring the child component

To use the `@Input()` decorator in a child component class, first import `Input` and then decorate the property with `@Input()`, as in the following example.

src/app/item-detail/item-detail.component.ts

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

```
export class ItemDetailComponent {

  @Input() item = ''; // decorate the property with @Input()

}
```

In this case, `@Input()` decorates the property `item`, which has a type of `string`, however, `@Input()` properties can have any type, such as `number`, `string`, `boolean`, or `object`. The value for `item` comes from the parent component.

Next, in the child component template, add the following:

src/app/item-detail/item-detail.component.html

```
<p>

  Today's item: {{item}}

</p>
```

Step2: Configuring the parent component

The next step is to bind the property in the parent component's template. In this example, the parent component template is `app.component.html`.

1. Use the child's selector, here `<app-item-detail>`, as a directive within the parent component template.
2. Use property binding to bind the `item` property in the child to the `currentItem` property of the parent.

   src/app/app.component.html

   ```
   <app-item-detail [item]="currentItem"></app-item-detail>
   ```

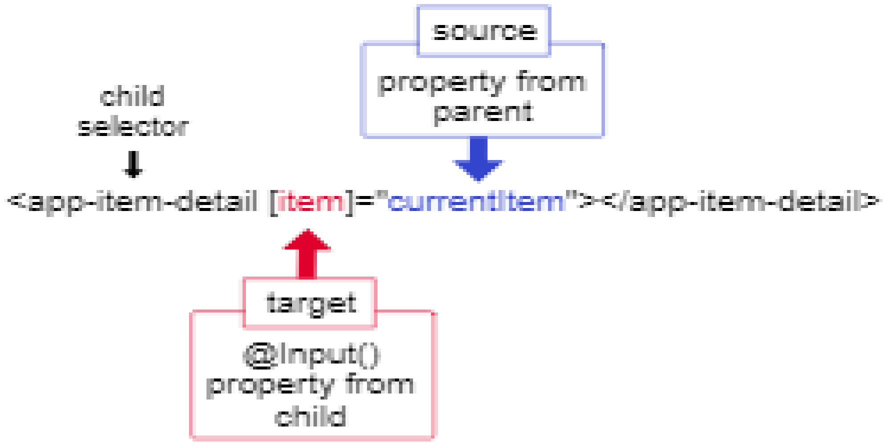
3. In the parent component class, designate a value for `currentItem`:

   src/app/app.component.ts

   ```
   export class AppComponent {
     currentItem = 'Television';
   }
   ```

With `@Input()`, Angular passes the value for `currentItem` to the child so that `item` renders as `Television`.

The following diagram shows this structure:


source
property from parent
child selector
<app-item-detail [item]="currentItem"></app-item-detail>
target
@Input() property from child

2. Sending data to a parent component to child (@output)link

The @`Output`() decorator in a child component or directive lets data flow from the child to the parent.

The child component uses the @`Output`() property to raise an event to notify the parent of the change. To raise an event, an @`Output`() must have the type of `EventEmitter`, which is a class in `@angular/core` that you use to emit custom events.

The following example shows how to set up an @`Output`() in a child component that pushes data from an HTML `<input>` to an array in the parent component.

To use @`Output`(), you must configure the parent and child.

Step1: Configuring the child componentlink

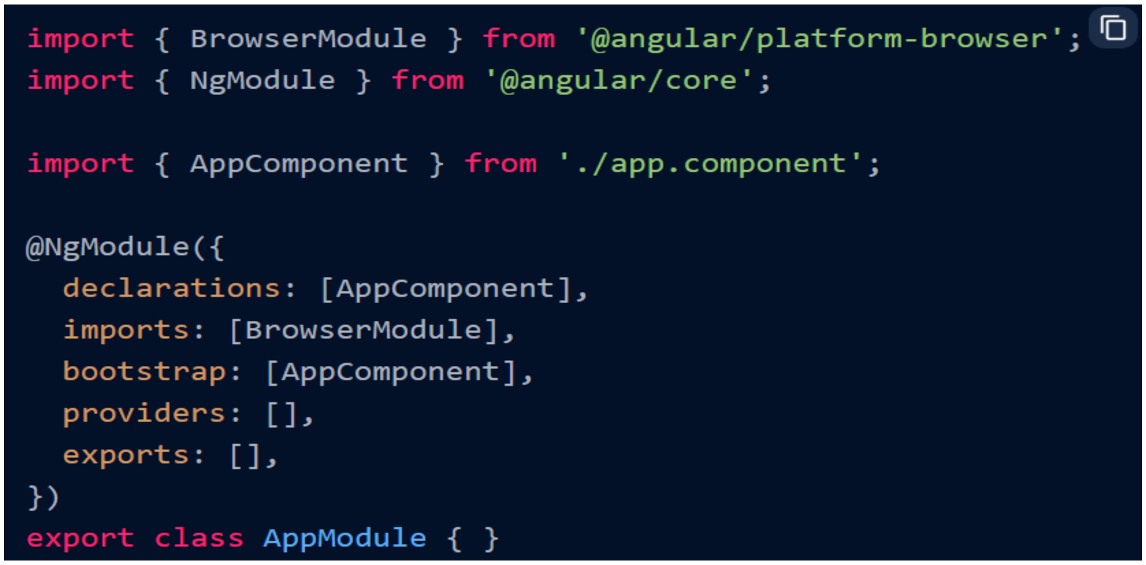
Note: click on the link above for @output decorator.

pass data between sibling components angular

3. Modules

In Angular, a module is a way to organize and bundle components, directives, services, and other features of an application.

Modules are defined using the **`@NgModule`** decorator and have properties like **`declarations`**, **`imports`**, **`exports`**, **`providers`**, and **`bootstrap`**.



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

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

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

- **`declarations`**: Lists the components, directives, and pipes that belong to the module.
- **`imports`**: Imports other modules into the current module, allowing the use of their components, directives, and services.
- **`exports`**: Specifies components, directives, and pipes that can be used by other modules when they import the current module.
- **`providers`**: Configures dependency injection within the module, allowing the provision of services.
- **`bootstrap`**: Specifies the root component(s) used to bootstrap the application.

Modules help with code reusability and proper maintenance.

Frequently-used modules

modules with examples of some of the things they contain:

NGMODULE	IMPORT IT FROM	WHY YOU USE IT
`BrowserModule`	`@angular/platform-browser`	To run your application in a browser.
`CommonModule`	`@angular/common`	To use `NgIf` and `NgFor`.
`FormsModule`	`@angular/forms`	To build template driven forms (includes `NgModel`).
`ReactiveFormsModule`	`@angular/forms`	To build reactive forms.
`RouterModule`	`@angular/router`	To use `RouterLink`, `.forRoot()`, and `.forChild()`.

HttpClientModule	@angular/common/http	To communicate with a server using the HTTP protocol.

4. Binding

In Angular, binding is a way to connect data and properties between the component and the template (view). There are 2 types of bindings available:

1. One-way Data Binding: Interpolation, Property binding, Event binding

1. *Interpolation/Expression* ({{ }}): curly bracket

- Interpolation binding is used to return HTML output from TypeScript code i.e. from the components to the views.
- Here, the template expression is specified within double curly braces.

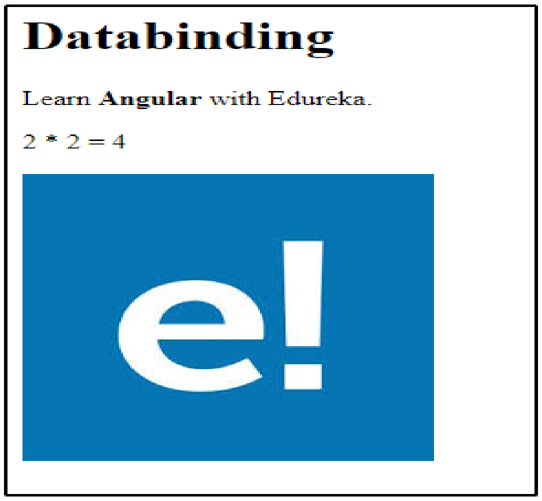
EXAMPLE:

```
<h1>{{title}}</h1>

Learn <b> {{course}}
</b> with Edureka.

2 * 2 = {{2 * 2}}

<div><img src="{{image}}"></div>

```

The Typescript part of this code is as follows:

```
export class AppComponent {
  title = 'Databinding';
  course ='Angular';
  image = 'paste the url here'
}
```

OUTPUT:



Databinding

Learn **Angular** with Edureka.

2 * 2 = 4

2. *Property binding* ([ ]): square braket

- Property binding is a one-way data binding mechanism that allows you to set the properties for HTML elements.
- It involves updating a property value in the component and binding the value to an HTML element in the same view.
- We use property binding for toggle functionality and sharing data between components.
- In general, one can say that the component property value will be set to the element property using Property binding.
- It uses the "[]" syntax for data binding.

Example 1:

```
<button [disabled]="isDisabled">Click me</button>
```

Here, the **`disabled`** property of the **`<button>`** element is bound to the **`isDisabled`** property of the component. The button will be disabled if **`isDisabled`** is **`true`** in the component.

EXAMPLE 2:

```
<h1>Property binding</h1>

<div><img [src]="image"></div>
```

In the above example, the ***src*** property of the image element is bound to a component's *image* property.

Property binding and Interpolation

If you have noticed, you can see that interpolation and property binding can be used interchangeably. Take a look at the example pair given below:

```
<h2>Interpolation</h2>

<div><img src="{{image}}"></div>

<h2>Property binding</h2>

<div><img [src]="image"></div>
```

Please make a note that when you need to set element properties to non-string data values, you must use Property binding and not Interpolation.

3. *Event binding* (`()`): round bracket

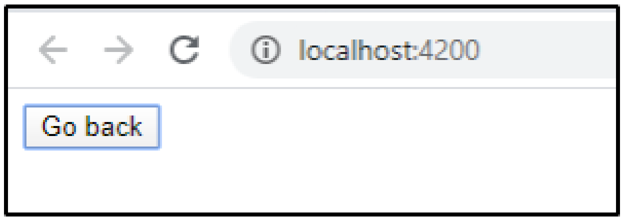
- Event binding allows us to respond to user events such as clicks, keypresses, and mouse movements.
- event binding can be achieved by specifying the target event name within regular brackets on the left of an equal to (=) sign, and the template statement on the right side within quotes (" ").

EXAMPLE:

```
<div>
  <button (click)="goBack()">Go back</button>
</div>
```

The '*click*' in the above example is the target events name and '*goBack()'* is the template statement.

OUTPUT:

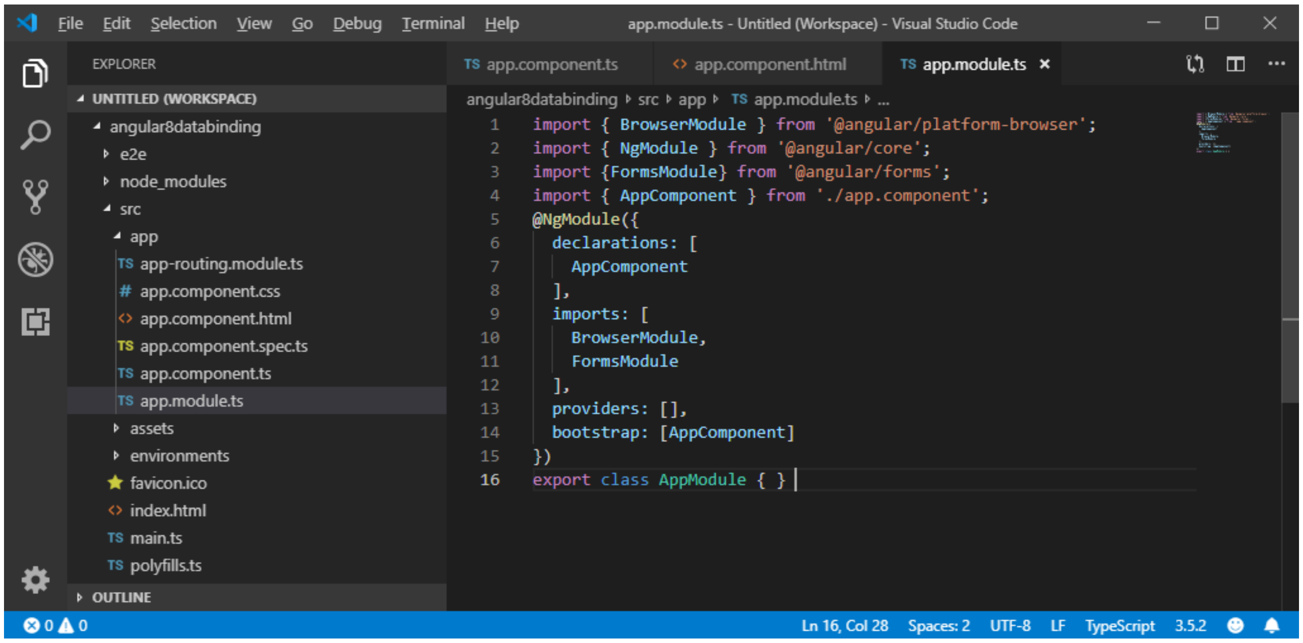


2. Two-way Data Binding (`[()]` or `[(ngModel)]`):

- Angular allows two-way data binding that will allow your application to share data in two directions i.e. from the components to the templates and vice versa.

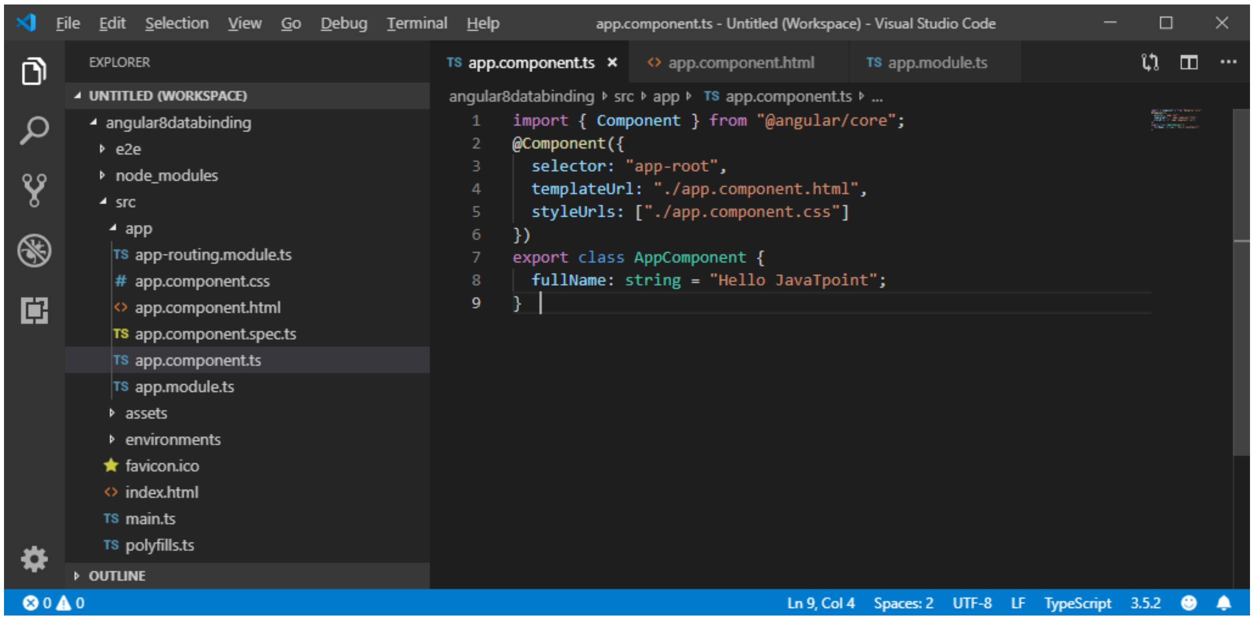
- The syntax of two-way binding is – [( )]. As you can see, it is a combination of the property binding syntax i.e. [ ] and the event binding syntax ( ). According to Angular, this syntax resembles "Banana in a Box".
- **Note:** For two way data binding, we have to enable the ngModel directive. It depends upon FormsModule in angular/forms package, so we have to add FormsModule in imports[] array in the AppModule.

EXAMPLE:



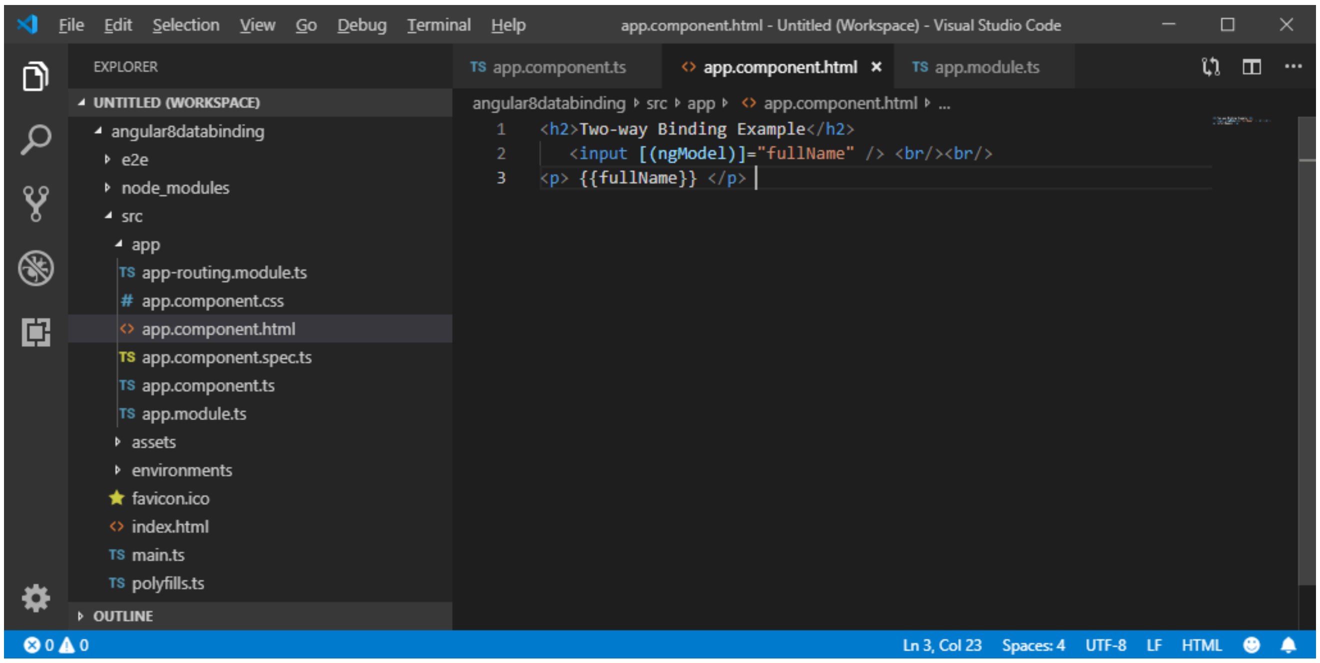
```
import { BrowserModule } from '@angular/platform-browser';
import { NgModule } from '@angular/core';
import {FormsModule} from '@angular/forms';
import { AppComponent } from './app.component';
@NgModule({
  declarations: [
    AppComponent
  ],
  imports: [
    BrowserModule,
    FormsModule
  ],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

app.component.ts file:



```
import { Component } from "@angular/core";
@Component({
  selector: "app-root",
  templateUrl: "./app.component.html",
  styleUrls: ["./app.component.css"]
})
export class AppComponent {
  fullName: string = "Hello JavaTpoint";
}
```

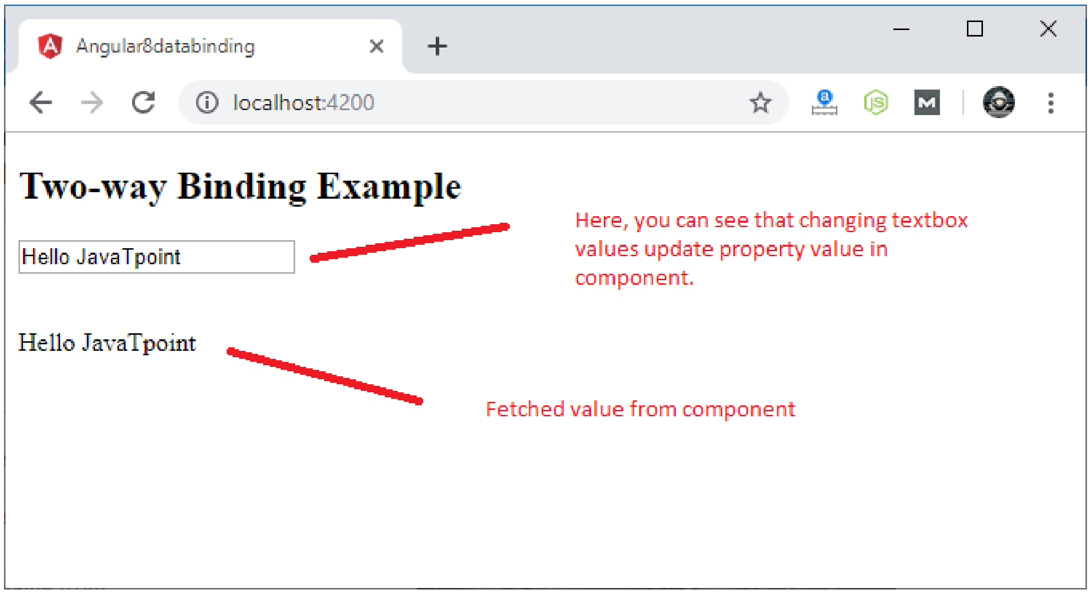
app.component.html file:



```
<h2>Two-way Binding Example</h2>
    <input [(ngModel)]="fullName" /> <br/><br/>
<p> {{fullName}} </p>
```

Now, start your server and open local host browser to see the result.

Output:

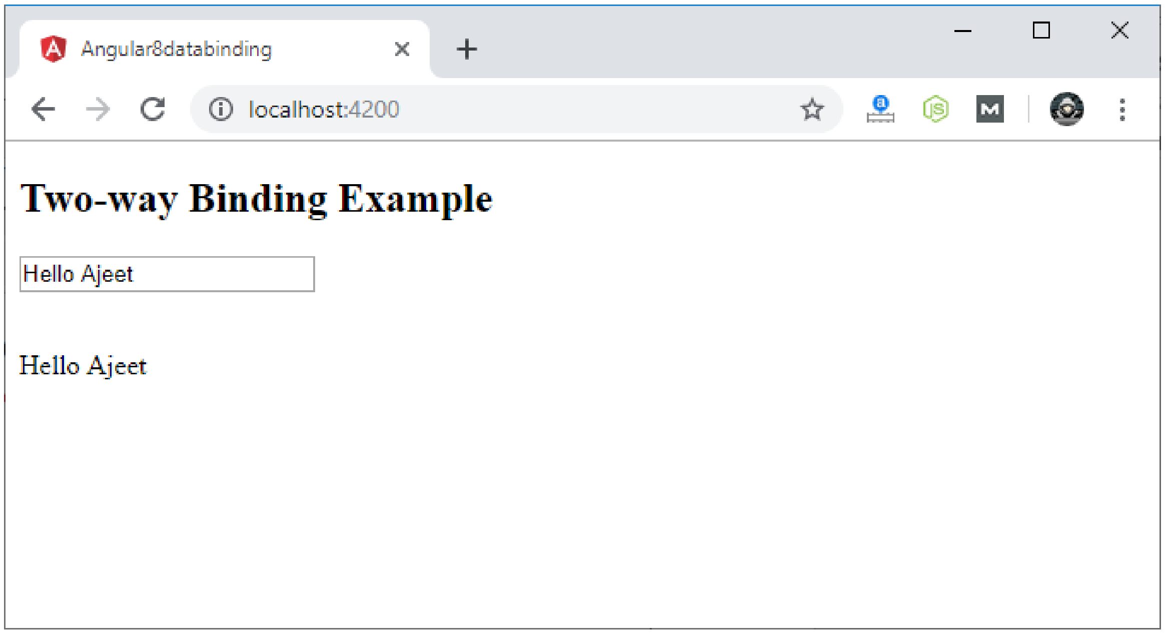

Angular8databinding
localhost:4200
Here, you can see that changing textbox values update property value in component.
Fetched value from component

Two-way Binding Example

```
Hello JavaTpoint

Hello JavaTpoint
```

You can check it by changing textbox value and it will be updated in component as well.

For example:



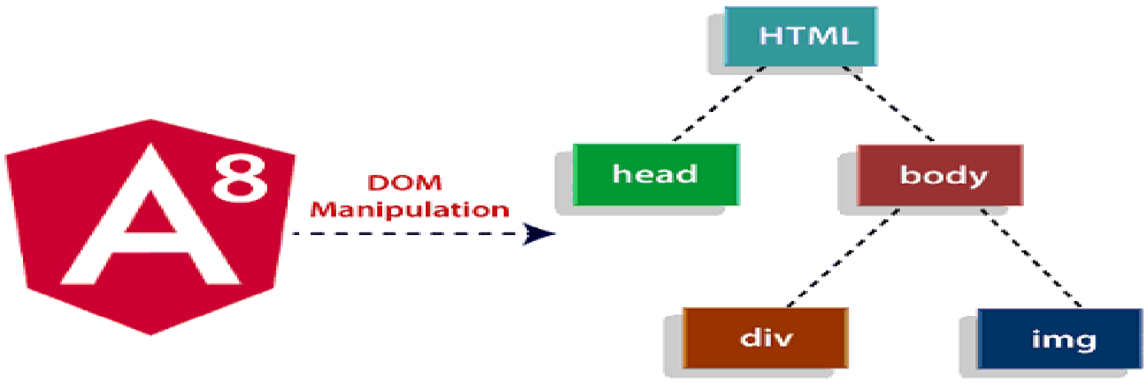
Two-way Binding Example

Hello Ajeet

Hello Ajeet

5. Directives

- Angular directives are used to manipulate the DOM.
- By using Angular directives, we can change the appearance, behavior or a layout of a DOM element.



Angular 8 Directive

Angular directives can be classified in 3 categories based on how they behave:

1. Structural Directives

- Structural directives start with a * sign. These directives are used to manipulate and change the structure of the DOM elements.
- For example, *ngIf directive, *ngSwitch directive, and *ngFor directive.

***ngIf Directive:** The ngIf allows us to Add/Remove DOM Element.
***ngSwitch Directive:** The *ngSwitch allows us to Add/Remove DOM Element. It is similar to switch statement of C#.
***ngFor Directive:** The *ngFor directive is used to repeat a portion of HTML template once per each item from an iterable list (Collection).

Angular ngIf Directive

The ngIf Directives is used to add or remove HTML Elements according to the expression.

ngIf Syntax

```
<p *ngIf="condition">
    condition is true and ngIf is true.
</p>
<p *ngIf="!condition">
    condition is false and ngIf is false.
</p>
```

The *ngIf directive form with an "else" block

```
<div *ngIf="condition; else elseBlock">
Content to render when condition is true.
</div>
<ng-template #elseBlock>
Content to render when condition is false.
</ng-template>
```

The ngIf directive does not hide the DOM element. It removes the entire element along with its subtree from the DOM. It also removes the corresponding state freeing up the resources attached to the element.

The *ngIf directive is most commonly used to conditionally show an inline template. See the following example:

```
@Component({
  selector: 'ng-if-simple',
  template: `
    <button (click)="show = !show">{{show ? 'hide' : 'show'}}</button>
    show = {{show}}
    <br>
    <div *ngIf="show">Text to show</div>
`
})
export class NgIfSimple {
  show: boolean = true;
}
```

Same template example with else block

```
@Component({
  selector: 'ng-if-else',
  template: `
    <button (click)="show = !show">{{show ? 'hide' : 'show'}}</button>
    show = {{show}}
    <br>
    <div *ngIf="show; else elseBlock">Text to show</div>
    <ng-template #elseBlock>Alternate text while primary text is hidden</ng-template>
`
})
export class NgIfElse {
  show: boolean = true;
}
```

Angular *ngFor Directive

The *ngFor directive is used to repeat a portion of HTML template once per each item from an iterable list (Collection).

Syntax of ngFor

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

Example:

App.component.ts file: -

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

```
@Component({
selector: 'app-root',
templateUrl: './app.component.html',
})
export class AppComponent {
a=['gfg1', 'gfg2', 'gfg3', 'gfg4']
}
```

App.component.html file: -

```
<ol>
<li *ngFor='let i of a'> {{i}} </li>
</ol>
```

Output:

2. Attribute Directives

https://dotnettutorials.net/lesson/angular-attribute-binding/

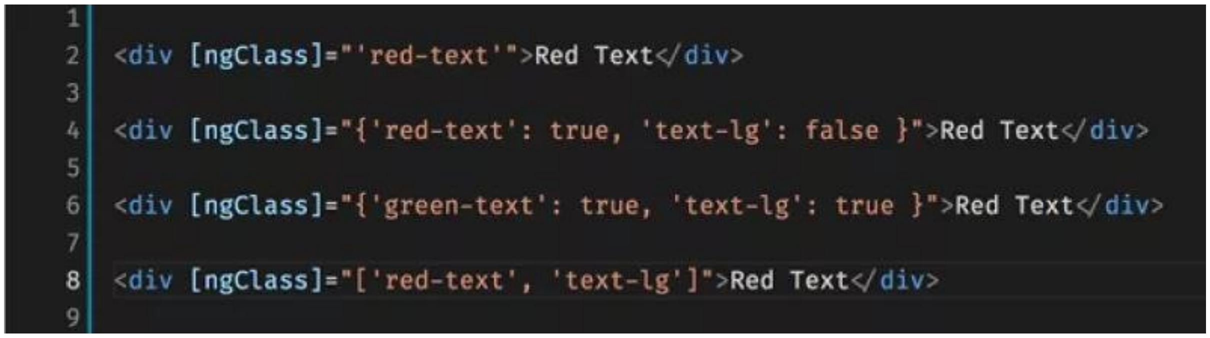
https://www.c-sharpcorner.com/article/attribute-data-binding-in-angular/

- An attribute directive changes the appearance or behavior of a DOM element.
  For example: ngClass directive, ngStyle directive and ngModal directive etc.

1. **NgClass**: It controls the appearance of elements by adding and removing CSS classes dynamically.

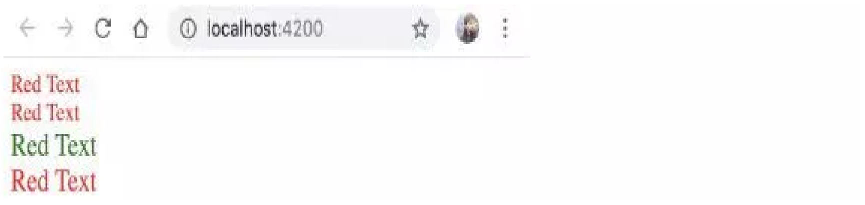
- For our demonstration, let's define a few CSS classes in the app.component.css file which is the CSS file for the AppComponent component.

Next, we will use the NgClass directive to add these classes (or a combination of these classes to HTML elements.



```
<div [ngClass]="'red-text'">Red Text</div>

<div [ngClass]="{'red-text': true, 'text-lg': false }">Red Text</div>

<div [ngClass]="{'green-text': true, 'text-lg': true }">Red Text</div>

<div [ngClass]="['red-text', 'text-lg']">Red Text</div>
```

Output:

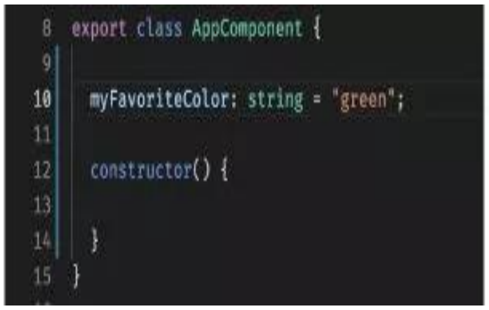


```
Red Text
Red Text
Red Text
Red Text
```

2. **NgStyle**:

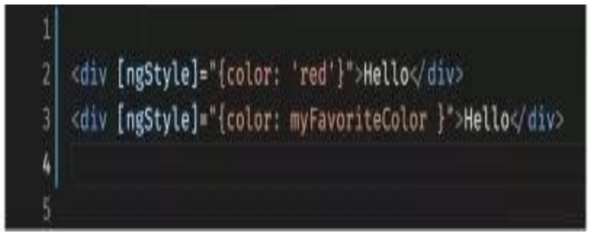
- Based on the component state, dynamic styles can be set by using NgStyle.
- Many inline styles can be set simultaneously by binding to NgStyle.
- It is a very simple directive and works very similar to the style HTML attribute which is used to apply inline CSS styles.

Consider the following code snippet where we have a property called myFavoriteColor defined and it has the value green.



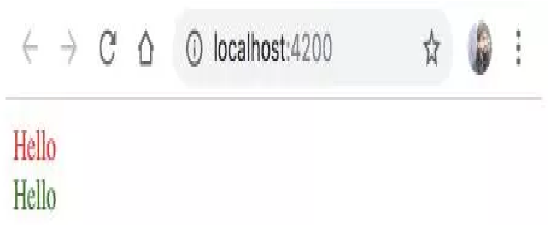
```
export class AppComponent {

  myFavoriteColor: string = "green";

  constructor() {

  }
}
```

In the HTML template, using the NgStyle directive, we can use the property to define the color of the text.



```
<div [ngStyle]="{color: 'red'}">Hello</div>
<div [ngStyle]="{color: myFavoriteColor }">Hello</div>
```

Output:



```
Hello
Hello
```

3. **NgModel**

- ngModal is used to create a top-level form group Instance, and it binds the form to the given form value.

Syntax:

```
<Input [(NgModel)= 'name']>
```

NgModule: Module used by NgModel is:

- **FormsModule**

What is the difference between [(ngModel)] and ngModel in Angular?

The answer is: (ngModel) causes a 1-way data-binding, whereas [(ngModel)] ensures a two-way data binding.

3. Component Directives

https://www.tutorialspoint.com/angular8/angular8_directives.htm

- It forms the main class and is declared by **@Component**.
- It contains the details on component processing, instantiated and usage at run time.
- Example: It contains certain parameters, some of them are shown in this example.

@Component({

selector: 'app-root',

templateUrl: './app.component.html',

styleUrls: ['./app.component.css']

})

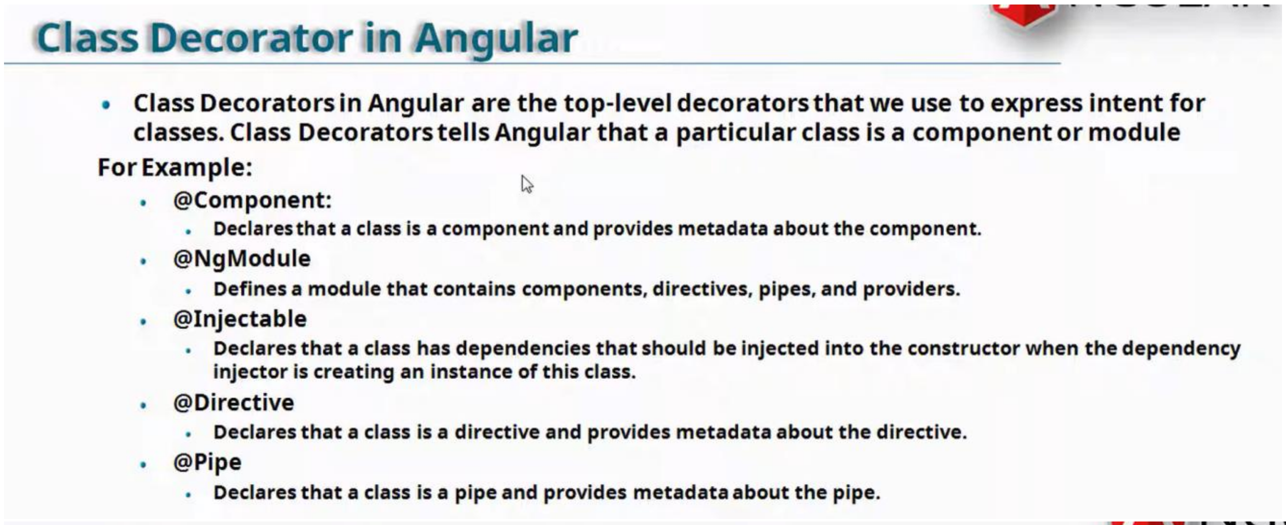
B. Decorator

- https://dotnettutorials.net/lesson/angular-decorators/
- Decorators and Types of Decorator @Input @Output, @Hostbinding @Hostlistener | Angular 10 Tutorials

- Decorators are functions that are invoked with a prefixed @ symbol, and are immediately followed by a class, method or property.
- The whole purpose of Angular decorators is to store metadata about a class, method, or property.

There are four types of decorators in Angular:

1. Class Decorators: **@Component** and **@NgModule**
2. Property Decorators: **@Input** and **@Output**
3. Method Decorators: **@HostListener**
4. Parameter Decorators: **@Inject**

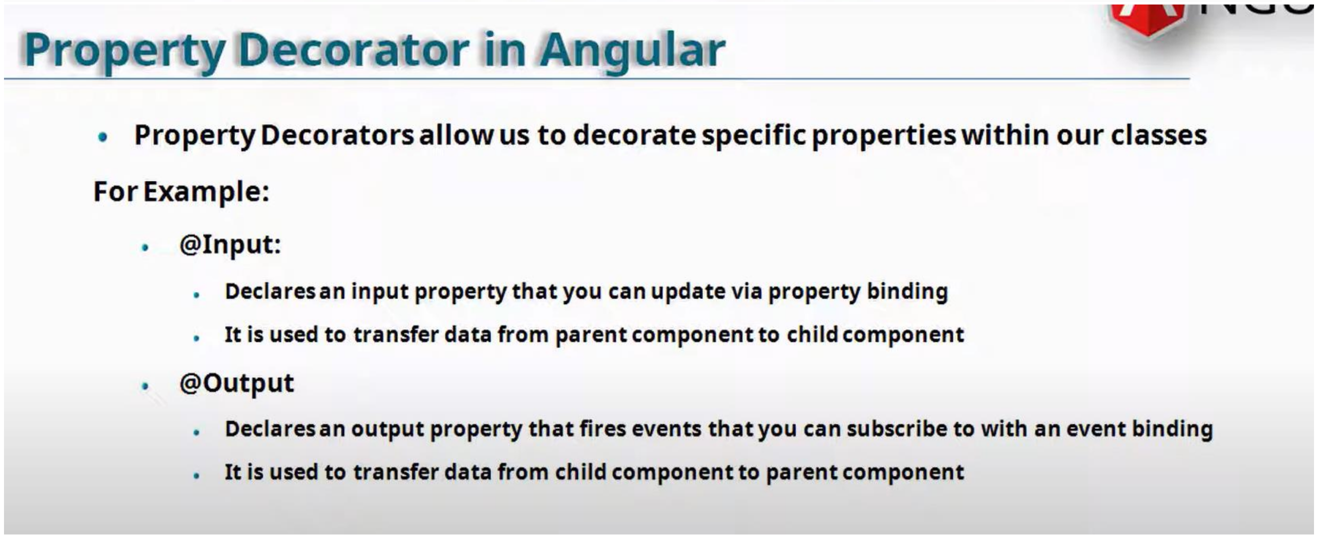


Class Decorator in Angular

- Class Decorators in Angular are the top-level decorators that we use to express intent for classes. Class Decorators tells Angular that a particular class is a component or module

For Example:

- @Component:
  - Declares that a class is a component and provides metadata about the component.
- @NgModule
  - Defines a module that contains components, directives, pipes, and providers.
- @Injectable
  - Declares that a class has dependencies that should be injected into the constructor when the dependency injector is creating an instance of this class.
- @Directive
  - Declares that a class is a directive and provides metadata about the directive.
- @Pipe
  - Declares that a class is a pipe and provides metadata about the pipe.



Property Decorator in Angular

- Property Decorators allow us to decorate specific properties within our classes

For Example:

- @Input:
  - Declares an input property that you can update via property binding
  - It is used to transfer data from parent component to child component
- @Output
  - Declares an output property that fires events that you can subscribe to with an event binding
  - It is used to transfer data from child component to parent component

C. Routing in Angular

https://www.geeksforgeeks.org/routing-in-angular-9-10/

Routing in Angular allows the users to create a single-page application with multiple views and allows navigation between them. Users can switch between these views without losing the application state and properties.

Approach:

- Create an Angular app that to be used.
- Create the navigation links inside the app component and then provide the "*routerLink*" directive to each route and pass the route value to "*routerLink*" directive.

- Then add the routes to the routing.module.ts file and then import the routing.module.ts into the app.module.ts file.

Syntax:

- **HTML:**

```
<li><a routerLink="/about" >About Us</a></li>
<router-outlet></router-outlet>
```

- **TS:**

```
{ path: 'about', component: AboutComponent }
```

Example: We are going to create a simple angular application that uses angular routing. So first, we create an Angular app by running the below command in CLI.

```
ng new learn-routing
```

Then we are creating simple navigation that allows us to navigate between the different components, and we have created some components as well, so users can switch between these components using routing.

- app.component.html

```
<span>
    <ul>
        <li><a routerLink="/" >Home</a></li>
        <li><a routerLink="/products" >Products</a></li>
        <li><a routerLink="/about" >About Us</a></li>
        <li><a routerLink="/contact" >Contact Us</a></li>
    </ul>
</span>
<router-outlet></router-outlet>
```

Here the router-outlet is routing functionality that is used by the router to mark wherein a template, a matched component should be inserted.

Then inside the *app-routing.module.ts* file, we have provided these routes and let the angular know about these routes.

- app-routing.module.ts

```
import { NgModule } from '@angular/core';
import { Routes, RouterModule } from '@angular/router';
import { HomeComponent } from './home.component'
import { ProductComponent } from './product.component'
import { AboutComponent } from './about.component'
import { ContactComponent } from './contact.component'

const routes: Routes = [
  { path: '', component: HomeComponent },
  { path: 'products', component: ProductComponent },
  { path: 'about', component: AboutComponent },
  { path: 'contact', component: ContactComponent, },
];

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

And then simply import "*AppRouting*" module inside the app/module.ts file inside the *@NgModule* imports.

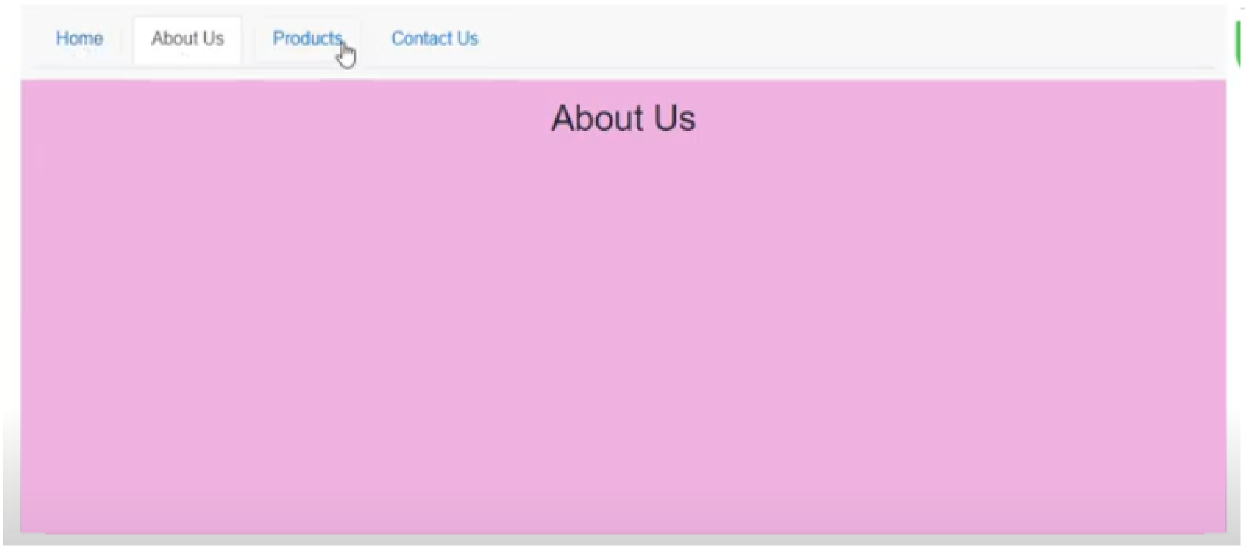
- app.module.ts

```
import { NgModule } from '@angular/core';
import { HomeComponent } from './home.component'
import { ProductComponent } from './product.component'
import { AboutComponent } from './about.component'
import { ContactComponent } from './contact.component'
import { AppRoutingModule } from './app-routing.module';
import { AppComponent } from './app.component';

@NgModule({
  declarations: [
    AppComponent,
    HomeComponent,
    ProductComponent,
    AboutComponent,
    ContactComponent
  ],
  imports: [
    AppRoutingModule,
  ],
  providers: [],
  bootstrap: [AppComponent]
})
```

```
export class AppModule { }
```

So now run this using *"ng serve"* in CLI and open *localhost://4200* in the browser here you see your navigation bar, and you can navigate from one component to another without page reload.

Output:



Q. What is `<router-outlet></router-outlet>` in app.component.html file

- The role of the <router-outlet> tells the router where to display the views.
- Router outlet is used to display the template(view) based on the router navigations.

C. Bootstrap in Angular

Steps for installing bootstrap:

1. $ npm install bootstrap.
2. After we successfully install the below command let's import it in angular.json file.

```
"styles": [
        "node_modules/bootstrap/dist/css/bootstrap.min.css",
        "src/styles.css"
      ],
      "scripts": []
```

Steps for installing bootstrap-icon

1. `npm install bootstrap bootstrap-icons`
2. Change the `angular.json`

```
"styles": [
  "node_modules/bootstrap/scss/bootstrap.scss",
  "node_modules/bootstrap-icons/font/bootstrap-icons.css",
  "src/styles.scss"
],
"scripts": [
  "node_modules/bootstrap/dist/js/bootstrap.bundle.min.js"
]
```

3. `npm install @ng-bootstrap/ng-bootstrap@next`
4. After install, we will import the `NgbModule` module. Change the `app.module.ts` file and add the lines as below:

```
import { NgbModule } from '@ng-bootstrap/ng-bootstrap';

imports: [
  BrowserModule,
  NgbModule,
  AppRoutingModule,
],
```

D. Forms in Angular

Two Types of Forms in Angular: - Template driven and Reactive form

E. Pipes

https://blog.bitsrc.io/pipes-in-angular-4f979af63dd7

- Use pipes to transform strings, currency amounts, dates, and other data for display.
- Pipes are simple functions to use in template expressions to accept an input value and return a transformed value.

Built-in pipes in Angular

Angular comes with many built-in pipes. Some of them include:

- *`uppercase`* (to convert string in upper case);
- *`lowercase`* (to convert string in upper case);
- *`date`* (to format the date into different types);
- *`json`* (to convert a value or object into JSON formatted string).

Examples of pipes in use:

```
<p>{{ 'Makesh' | uppercase }}</p>
<p>{{ 'Kumar' | lowercase }}</p>
<p>{{ { name: 'makesh' } | json }}</p>
```

Passing arguments to pipes

Pipes like *date, currency* will take arguments for pipe function, to pass argument follow the pipe name (*`currency`*) with a colon (*`:`*) and the parameter value (`‘EUR’`).

For example:

```
<p>{{ 100 | currency: 'INR' }}</p>
<p>{{ currentDate | date: 'dd/MM/yyyy' }}</p>
```

Chaining multiple pipes

To chain multiple pipes together, we just have to use a pipe operator (`|`) between different pipe names, see the below code for example

```
<p>{{ currentDate | date | uppercase}}</p>
```

Types of pipes

Pipes can be classified into:

8. Pure Pipes
9. Impure Pipes

1. Pure Pipes

A pure pipe is only called when Angular detects a change in the value or the parameters passed to a pipe.

2. Impure Pipes

An impure pipe is called for every change detection cycle no matter whether the value or parameter(s) changes.

Creating our own custom pipes

https://www.tutorialspoint.com/angular2/angular2_custom_pipes.htm

https://stackblitz.com/edit/creating-a-custom-pipe-in-angular?file=src%2Fapp%2Ffilter.pipe.ts

To create our own custom pipe, follow the below steps:

The general way to define a custom pipe is as follows.

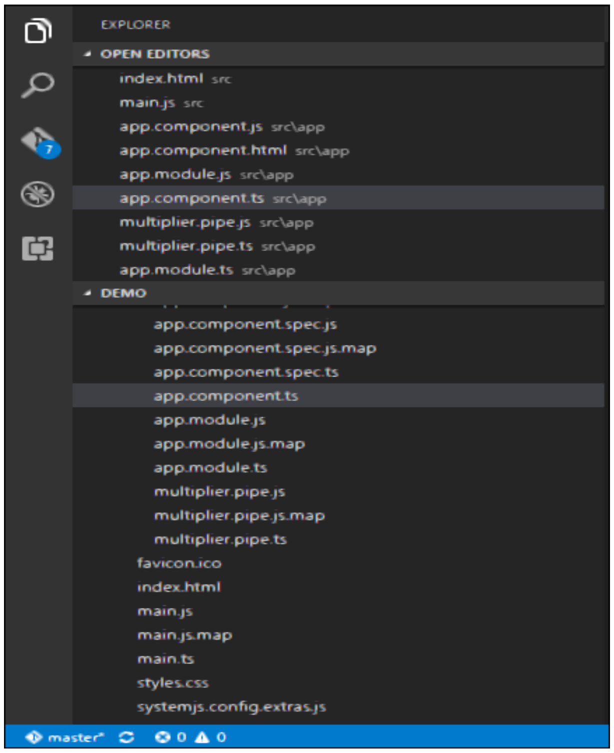
```
import { Pipe, PipeTransform } from '@angular/core';
@Pipe({name: 'Pipename'})

export class Pipeclass implements PipeTransform {
   transform(parameters): returntype { }
}
```

Where,

- **'Pipename'** – This is the name of the pipe.
- **Pipeclass** – This is name of the class assigned to the custom pipe.
- **Transform** – This is the function to work with the pipe.
- **Parameters** – This are the parameters which are passed to the pipe.
- **Returntype** – This is the return type of the pipe.

Let’s create a custom pipe that multiplies 2 numbers. We will then use that pipe in our component class.

Step 1 – First, create a file called multiplier.pipe.ts.



Step 2 – Place the following code in the above created file.

```
import {
   Pipe,
   PipeTransform
} from '@angular/core';

@Pipe ({
   name: 'Multiplier'
})

export class MultiplierPipe implements PipeTransform {
   transform(value: number, multiply: string): number {
      let mul = parseFloat(multiply);
      return mul * value
   }
}
```

Following points need to be noted about the above code.

- We are first importing the Pipe and PipeTransform modules.

- Then, we are creating a Pipe with the name 'Multiplier'.
- Creating a class called MultiplierPipe that implements the PipeTransform module.
- The transform function will then take in the value and multiple parameter and output the multiplication of both numbers.

Step 3 – In the app.component.ts file, place the following code.

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

@Component ({
   selector: 'my-app',
   template: '<p>Multiplier: {{2 | Multiplier: 10}}</p>'
})
export class AppComponent {  }
```

Note – In our template, we use our new custom pipe.

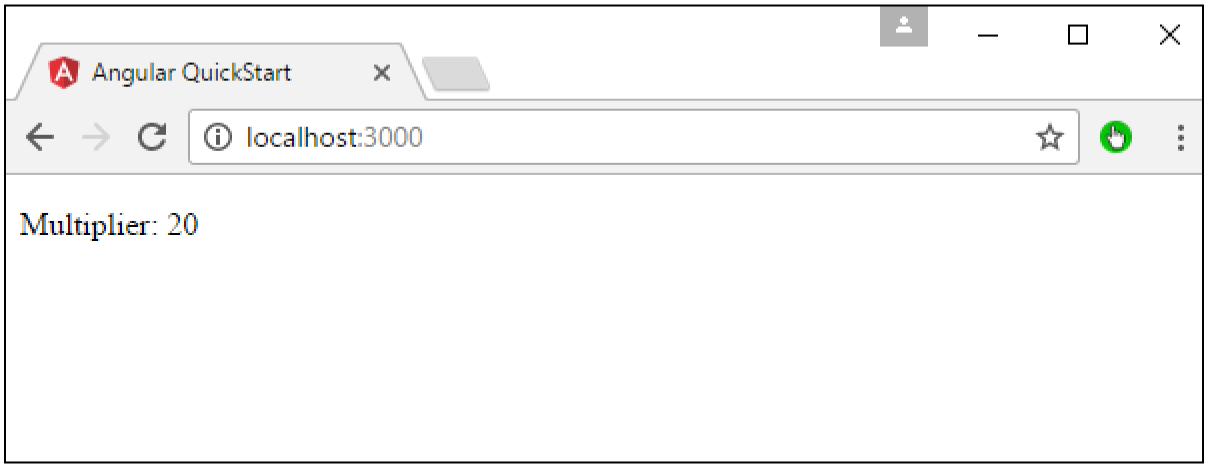
Step 4 – Ensure the following code is placed in the app.module.ts file.

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

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

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

import {
   MultiplierPipe
} from './multiplier.pipe'

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

export class AppModule {}
```

Following things need to be noted about the above code.

- We need to ensure to include our MultiplierPipe module.
- We also need to ensure it is included in the declarations section.

Once you save all the code changes and refresh the browser, you will get the following output.



```
Multiplier: 20
```

F. Component Life Cycle

G. Services and DI in Angular

https://data-flair.training/blogs/angularjs-services/

H. Template Reference Variables in Angular

https://angular.io/guide/template-reference-variables

I. Http API:

Understanding communicating with backend services using HTTP

https://www.telerik.com/blogs/angular-basics-how-to-use-httpclient

https://www.techiediaries.com/angular-11-tutorial-example-rest-crud-http-get-httpclient/

Angular

https://angular.io/start/start-data

https://mockoon.com/tutorials/angular-api-call-and-mocking/

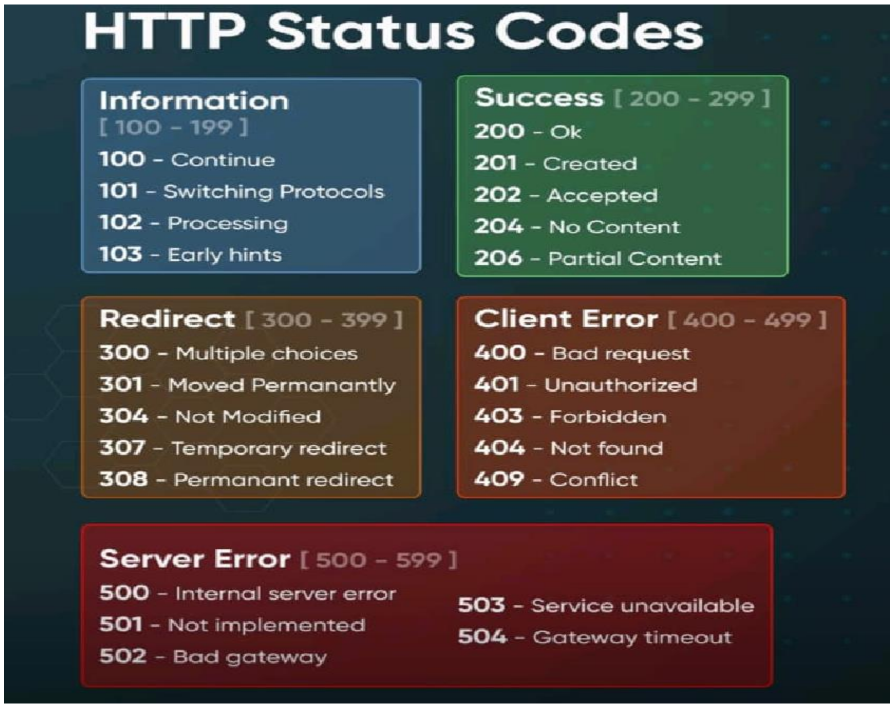
https://mockoon.com/tutorials/getting-started/

How to Make API Calls in Angular Applications

https://www.upgrad.com/blog/api-calls-in-angular/

Need to learn

1.Introduction: Why API Calls are Important in Angular Applications
2.Setting Up the Angular Environment for API Calls
3.Understanding HTTP in Angular: The HttpClient Module
4.Making Simple GET Requests: Retrieving Data from APIs
5.Handling Errors and Exceptions in API Calls
6.Making POST Requests: Sending Data to APIs
7.Uploading Files with API Calls: Using the FormData Object



HTTP Status Codes

Information [100 - 199]
100 - Continue
101 - Switching Protocols
102 - Processing
103 - Early hints

Success [200 - 299]
200 - Ok
201 - Created
202 - Accepted
204 - No Content
206 - Partial Content

Redirect [300 - 399]
300 - Multiple choices
301 - Moved Permanantly
304 - Not Modified
307 - Temporary redirect
308 - Permanant redirect

Client Error [400 - 499]
400 - Bad request
401 - Unauthorized
403 - Forbidden
404 - Not found
409 - Conflict

Server Error [500 - 599]
500 - Internal server error
501 - Not implemented
502 - Bad gateway
503 - Service unavailable
504 - Gateway timeout

Q. How to create the mock/dummy API in angular?

To create dummy api's we can use any of the below :

- **firebase** https://firebase.google.com/
- **mockapi** https://mockapi.io/projects
- **jsonplaceholder** https://jsonplaceholder.typicode.com/

Angular Basics: How To Use HttpClient in Angular

Step 1: I have created the application with the help of angular-cli command: `ng new app-name`.

Step 2: Import or configure the `HttpClientModule` into the `app.module.ts` file as shown below:

```
import { NgModule} from  '@angular/core';
import { BrowserModule } from  '@angular/platform-browser';
import { AppRoutingModule } from  './app-routing.module';
import { AppComponent } from  './app.component';
import { HttpClientModule } from  '@angular/common/http';

@NgModule({
declarations: [AppComponent],
imports: [
BrowserModule,
AppRoutingModule,
HttpClientModule  //imported the module
],
providers: [],
bootstrap: [AppComponent]
})
export  class  AppModule { }
```

Step 3: You can directly use the `HttpClient` in your component, but its best to access it via the service.

We are creating a new service with the help of angular-cli command `ng generate service service-name`. You will see code in that service as below:

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

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

constructor() { }

}
```

Step 4: Inject the `HttpClient` in the service created in the previous step. Then you can see code as below:

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

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

constructor(private http: HttpClient) { }

}
```

Step 5: In this step we are going to get/fetch the data from the server with the help of HTTP GET request. For that, we are adding one method in the service name as `getPosts`—that method we are calling in the component.

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

@Injectable({
providedIn:  'root'
})

export class HttpService {

private url = 'https://my-json-server.typicode.com/JSGund/XHR-Fetch-
Request-JavaScript/posts';

constructor(private http: HttpClient) { }
getPosts() {
      return this.http.get(this.url);
}
}
```

In the above code we have added the method `getPosts` and placed HTTP GET request, and passed the one parameter with the request that is nothing but the `End-point-url`.

Step 6 (Note step-6 is not required, read this step for only understanding):
Let's understand the **HTTP GET** request and its request and response objects. The HTTP GET request has around 15 different types of methods to use.

```
get<T>(url: string, options?: { headers?: [HttpHeaders];
context?: [HttpContext];
observe?: "body";
params?:  [HttpParams];
reportProgress?: boolean;
responseType?:  "json";
withCredentials?: boolean;
}):  Observable<T>
```

Parameters

- `url` – It is the service/API endpoint URL of type `string`.
- `options` – It is used to configure the HTTP request. It is optional and of type `object`, and its default value is `undefined`.

```
options:
{
      headers?: [HttpHeaders],
      observe?: 'body' | 'events' | 'response',
      params?:  [HttpParams],
      reportProgress?:  boolean,
      responseType?: 'arraybuffer'|'blob'|'json'|'text',
      withCredentials?:  boolean,
}
```

Below two are important `options` properties:

- `observe`: How much of the response to return.
- `responseType`: The return data format.

Returns
HTTP GET returns an observable of the `HttpResponse`.

Step 7 (reading data from the server): In this step we are going to use the `getPosts` method in the component. For that, first we need to inject the created service into our component and access the method as shown below:

```
import { Component } from '@angular/core';
import { HttpService } from './http.service';

@Component({
selector: 'app-root',
templateUrl: './app.component.html',
styleUrls: ['./app.component.css']
})

export  class  AppComponent {
title = 'Article by Jeetendra';
posts : any;
constructor(private httpService: HttpService) { }

ngOnInit() {
      this.httpService.getPosts().subscribe(
      (response) => { this.posts = response; },
      (error) => { console.log(error); });
}
}
```

TypeScript
In the above code we have injected the service first in constructor then the important thing is we call the `getPosts` method and `subscribe` to it. Whenever we get the response from this `subscribe` method, it will be a list of object containing `id`, `title`, `path`, as shown below:

Response

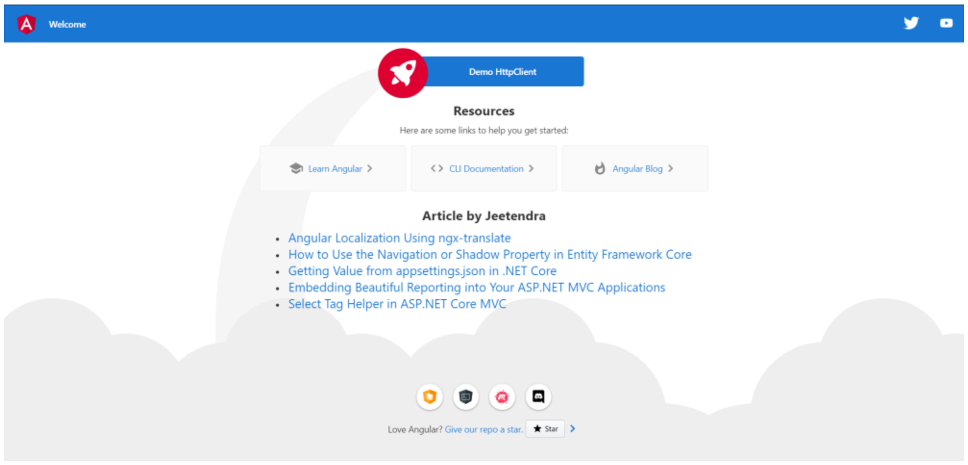
```
[
{id: 1, title: "Angular Localization Using ngx-translate", path:
"https://www.telerik.com/blogs/angular-localization-using-ngx-translate"},

{id: 2, title: "How to Use the Navigation or Shadow Property in Entity Framework
Core", path: "https://www.telerik.com/blogs/how-to-use-the-navigation-or-shadow-
property-in-entity-framework-core"},

{id: 3, title: "Getting Value from appsettings.json in .NET Core", path:
"https://www.telerik.com/blogs/how-to-get-values-from-appsettings-json-in-net-core"},

{id: 4, title: "Embedding Beautiful Reporting into Your ASP.NET MVC Applications",
path: "https://www.telerik.com/blogs/embedding-beautiful-reporting-asp-net-mvc-
applications"},

{id: 5, title: "Select Tag Helper in ASP.NET Core MVC", path:
"https://www.telerik.com/blogs/select-tag-helper-asp-net-core-mvc"}
]
```

We have declared the property as `posts` and assigned the response we get in the `subscribe`, then iterated that in HTML with the help of the `*ngFor` directive as the code below shows:

```
<div>
<li *ngFor="let post of posts">
     <a  href="{{post.path}}">
         <span  style="font-size: 20px;text-align:
center;">{{post.title}}
          </span>
    </a>
</li>
</div>
```

Step 8: Finally, we have implemented our first HTTP request, that `GET`. Run the Angular application with help of angular-cli command `ng serve`

Angular

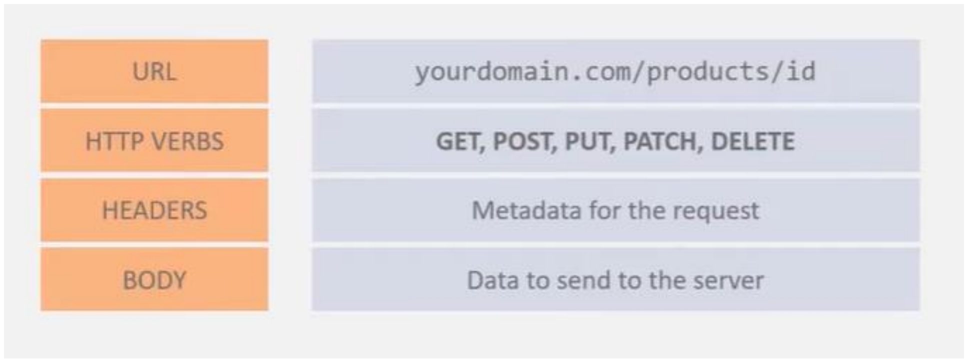


Resources

Here are some links to help you get started:

Article by Jeetendra

- Angular Localization Using ngx-translate
- How to Use the Navigation or Shadow Property in Entity Framework Core
- Getting Value from appsettings.json in .NET Core
- Embedding Beautiful Reporting into Your ASP.NET MVC Applications
- Select Tag Helper in ASP.NET Core MVC



URL	yourdomain.com/products/id
HTTP VERBS	GET, POST, PUT, PATCH, DELETE
HEADERS	Metadata for the request
BODY	Data to send to the server

Delete data from the server

Introduction to HTTP requests in Angular

https://www.pluralsight.com/guides/posting-deleting-putting-data-angular

Introduction to HTTP requests in Angular | Angular HTTP | Angular 13+

Setting up Firebase

Setting up Firebase | Angular HTTP | Angular 13+

What is Observable

What is Observable | Observables | Angular 12+

- The basic usage of Observable in Angular is to create an instance to define a **subscriber function**. Whenever a consumer wants to execute the function the **subscribe ()** method is called. This function defines how to obtain messages and values to be published.
- Observer is nothing but the subscriber which is waiting for the data.

Sending a Post Request

Sending a Post Request | Angular HTTP | Angular 13+

Fetching Data with Get requets

Fetching Data with Get requets | Angular HTTP | Angular 13+

Deleting Data with HTTP Delete Request | Angular HTTP | Angular 12+

Deleting Data with HTTP Delete Request | Angular HTTP | Angular 12+

Update data with HTTP Put Request | Angular HTTP | Angular 13+

Update data with HTTP Put Request | Angular HTTP | Angular 13+

Setting Headers & Query Params in HTTP Requests | Angular HTTP | Angular 13+

Setting Headers & Query Params in HTTP Requests | Angular HTTP | Angular 13+

……

Angular HTTP CRUD Methods Part 1 - GET and POST

Angular HTTP CRUD Methods Part 1 - GET and POST

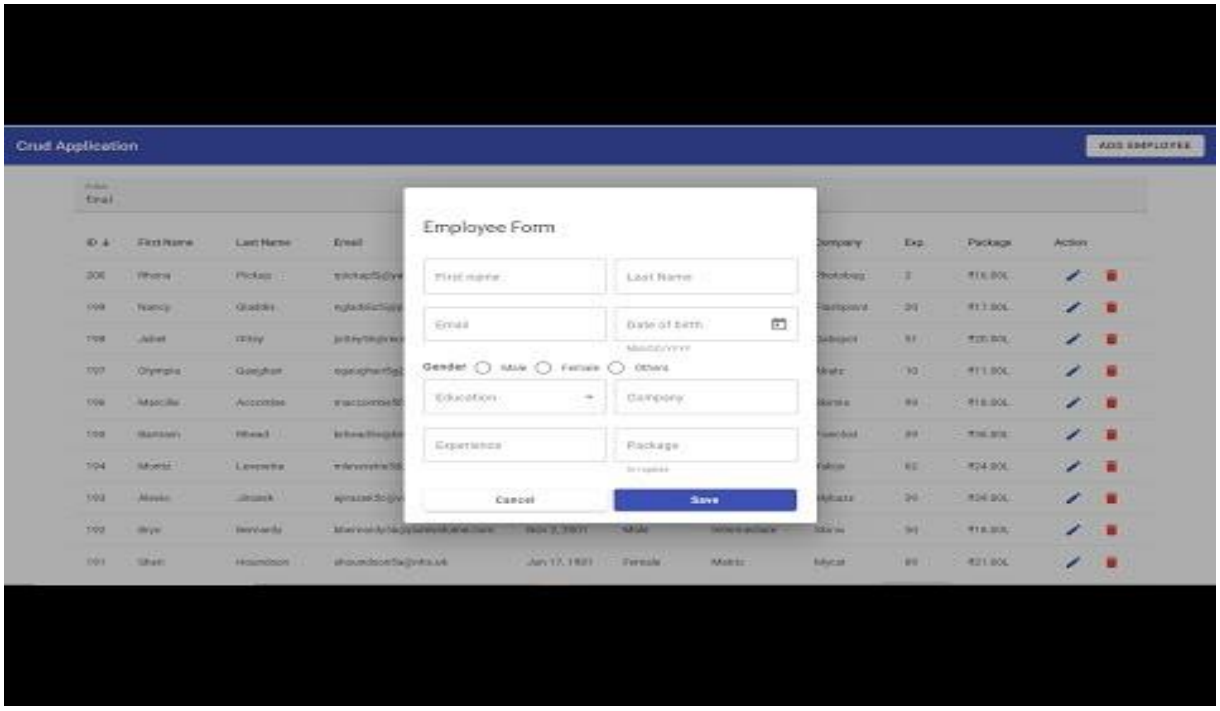
Angular HTTP CRUD methods Part 2 - PUT and DELETE

Angular HTTP CRUD methods Part 2 - PUT and DELETE

Angular 15 CRUD app using material UI | JSON-server | step by step

Angular 15 CRUD app using material UI | JSON-server | step by step

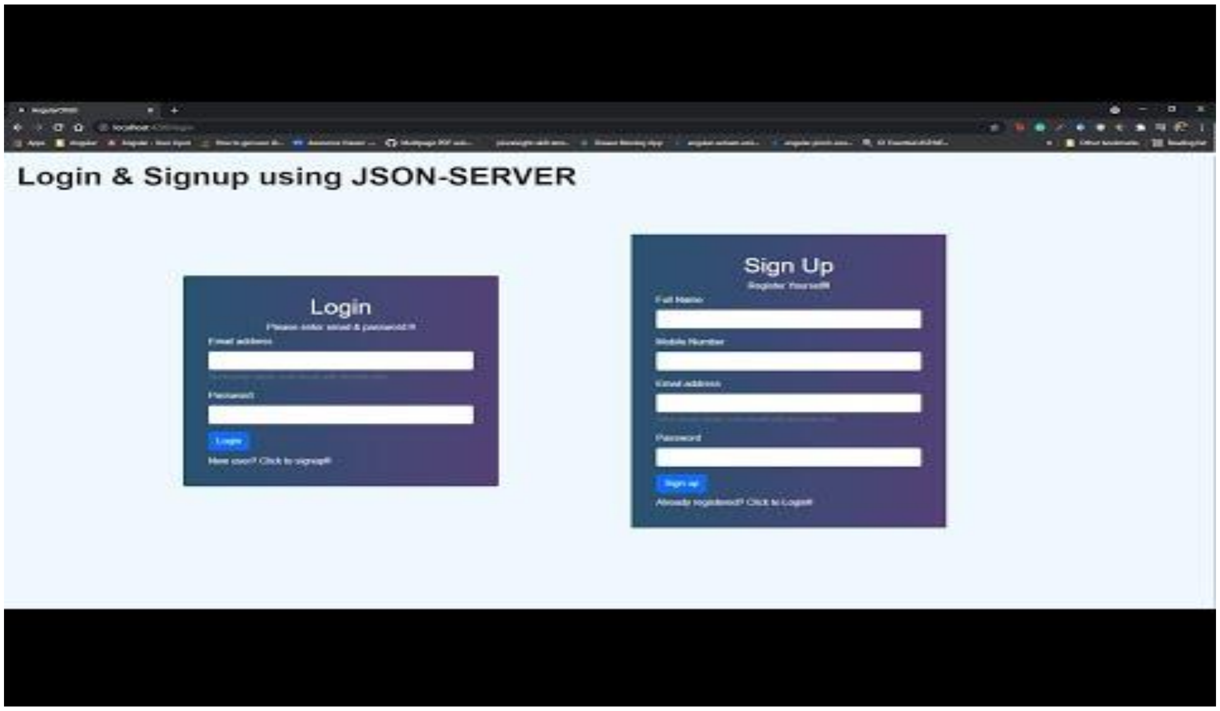
Angular



Angular Login and Signup using JSON-Server | Angular Routing | Angular Reactive Form|

[Angular Login and Signup using JSON-Server | Angular Routing | Angular Reactive Form|]

Angular



Angular HTTP API | FULL COURSE

[Angular HTTP API | FULL COURSE]()

Angular

Difference between ng-template and ng-container

DOM Manipulation

K. Services and DI in Angular

Multiple ways to Print array of objects in Angular with example?

https://www.cloudhadoop.com/angular-print-array-objects