

Unit 4

Angular

1. **Getting Started with Angular**
2. **Angular App From Scratch**
3. **Components**
4. **Properties, Events & Binding with ngModel**
5. **Fetch Data from a Service**
6. **Submit data to service**
7. **http module**
8. **observables**
9. **Routing**

1. Getting Started with Angular

a.Defination:-

Angular is an open-source web application framework maintained by Google and a community of developers. It is designed to build dynamic and interactive single-page applications (SPAs) efficiently. With Angular, developers can create robust, scalable, and maintainable web applications.

(Or)

Angular is an open-source, JavaScript framework written in TypeScript. Google maintains it, and its primary purpose is to develop single-page applications. As a framework, Angular has clear advantages while also providing a standard structure for developers to work with. It enables users to create large applications in a maintainable manner.

b. History

Angular, initially released in 2010 by Google, has undergone significant transformations over the years. The first version, AngularJS, introduced concepts like two-way data binding and directives. However, as web development evolved, AngularJS faced limitations in terms of performance and flexibility.

In 2016, Angular 2 was released, which was a complete rewrite of AngularJS, focusing on modularity and performance. Since then, Angular has continued to evolve, with regular updates and improvements to meet the demands of modern web development.

c. Why Angular?

JavaScript is the most commonly used client-side scripting language. It is written into HTML documents to enable interactions with web pages in many unique ways. As a relatively easy-to-learn language with pervasive support, it is well-suited to develop modern applications.

But is JavaScript ideal for developing single-page applications that require modularity, testability, and developer productivity? Perhaps not.

These days, we have a variety of frameworks and libraries designed to provide alternative solutions. With respect to front-end web development, Angular addresses many, if not all, of the issues developers face when using JavaScript on its own.

d. Here are some of the features of Angular

1. Custom Components

Angular enables users to build their components that can pack functionality along with rendering logic into reusable pieces.

2. Data Binding

Angular enables users to effortlessly move data from JavaScript code to the view, and react to user events without having to write any code manually.

3. Dependency Injection

Angular enables users to write modular services and inject them wherever they are needed. This improves the testability and reusability of the same services.

4. Testing

Tests are first-class tools, and Angular has been built from the ground up with testability in mind. You will have the ability to test every part of your application—which is highly recommended.

5. Comprehensive

Angular is a full-fledged JavaScript framework and provides out-of-the-box solutions for server communication, routing within your application, and more.

6. Browser Compatibility

Angular works cross-platform and compatible with multiple browsers. An Angular application can typically run on all browsers (Eg: Chrome, Firefox) and operating systems, such as Windows, macOS, and Linux.

7. Two-Way Data Binding: Angular provides seamless synchronization between the model and the view, allowing for easy management of user inputs.

8. Directives: Angular offers a rich set of built-in directives for manipulating the DOM, such as `*ngIf*`, `*ngFor*`, and `*ngSwitch*`.

9. Routing: Angular's powerful routing module enables to build SPAs with multiple views and navigation between them.

10.HTTP Client: Angular includes an HTTP client module for making server requests, simplifying data fetching and manipulation.

e. Advantages of Angular

- **Productivity:** Angular's extensive tooling and ecosystem streamline development tasks, enabling faster project completion.
- **Maintainability:** Angular's modular architecture and clear separation of concerns promote code organization and maintainability.
- **Scalability:** Angular is well-suited for building large-scale applications, thanks to its component-based architecture and robust performance.

- **Community Support:** Being backed by Google and a vast community of developers, Angular enjoys strong community support and continuous improvement.

f. Disadvantages of Angular

- **Learning Curve:** Angular has a steep learning curve, especially for beginners, due to its complex concepts and extensive documentation.
- **Performance Overhead:** Angular's powerful features come with a performance cost, and poorly optimized applications may suffer from performance issues.
- **Size:** Angular applications tend to have larger file sizes compared to other frameworks, which may impact load times, especially on mobile devices.
- **Migration:** Upgrading between major Angular versions can be challenging and time-consuming, requiring significant changes to existing codebases.

g. Angular Prerequisites

There are three main prerequisites.

NodeJS

Angular uses Node.js for a large part of its build environment. As a result, to get started with Angular, you will need to have Node.js installed on your system. You can head to the NodeJS official website to download the software. Install the latest version and confirm them on you command prompt by running the following commands:

Node --version

npm --v



Angular CLI

The Angular team has created a command-line interface (CLI) tool to make it easier to bootstrap and develop your Angular applications. As it significantly helps to make the process of development easier, we highly recommend using it for your initial angular projects at the least.

To install the CLI, in the command prompt, type the following commands

Installation:

npm install -g @angular/cli

Confirmation -

ng--version



Text Editor

You need a text editor to write and run your code. The most popularly used integrated development environment (IDE) is Visual Studio Code (VS Code). It is a powerful source code editor that is available on Windows, macOS, and Linux platforms.



Now, Let's create our first Angular HelloWorld Application.

2. Angular App From Scratch

h. Creating an Angular Application

Step 1: Install Angular CLI: Angular CLI (Command Line Interface) is a powerful tool for scaffolding and managing Angular applications. Install it globally using npm:

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

Step 2: Create a New Angular Project: Use Angular CLI to create a new Angular project. Navigate to the desired directory and run:

```
ng new my-angular-app //creating standalone application
```

(or)

Ng new my-angular-app --standalone false //creating non-standalone application project structure is different adding two more files app.module.ts and app-routing.module.ts

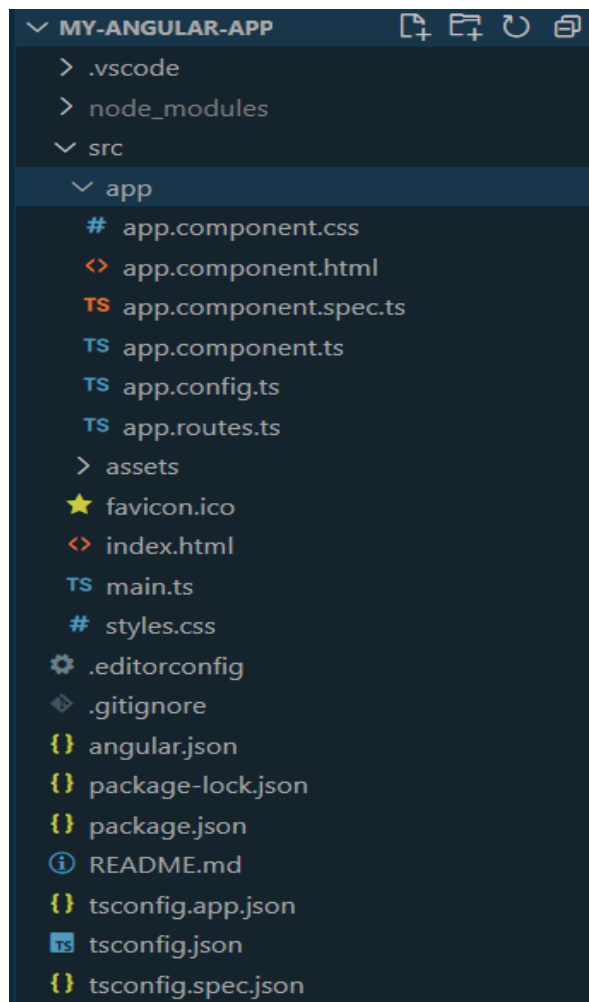
Step 3: Navigate to the Project Directory: Move into the newly created project directory:

```
cd my-angular-app
```

Step 4: Serve the Application: Launch the development server to see your app in action:

```
ng serve
```

Folder Structure:



What is angular

Dependencies:

```
"dependencies": {
  "@angular/animations": "^17.3.0",
  "@angular/common": "^17.3.0",
  "@angular/compiler": "^17.3.0",
  "@angular/core": "^17.3.0",
  "@angular/forms": "^17.3.0",
  "@angular/platform-browser": "^17.3.0",
  "@angular/platform-browser-dynamic": "^17.3.0",
  "@angular/router": "^17.3.0",
  "rxjs": "~7.8.0",
  "tslib": "^2.3.0",
  "zone.js": "~0.14.3"
}
```

Example:

```
<!-- app.component.html -->
```

```
<h1>Hello Angular</h1>
```

```
//app.component.ts
```

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

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

Root HTML - index.html(default code)

```
<!doctype html>
```

```
<html lang="en">
```

```
<head>
```

```
<meta charset="utf-8">
```

```
<title>HelloWorld</title>
```

```
<base href="/">
```

```
<meta name="viewport" content="width=device-width, initial-scale=1">
```

```
<link rel="icon" type="image/x-icon" href="favicon.ico">
```

```
</head>
```

```
<body>
```

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

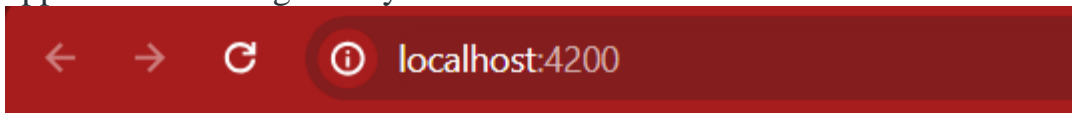
```
</body>
```

```
</html>
```

The only main thing in this file is the **<app-root>** element. This is the marker for loading the application. All the application code, styles, and inline templates are dynamically injected into the index.html file at run time by the **ng serve** command.

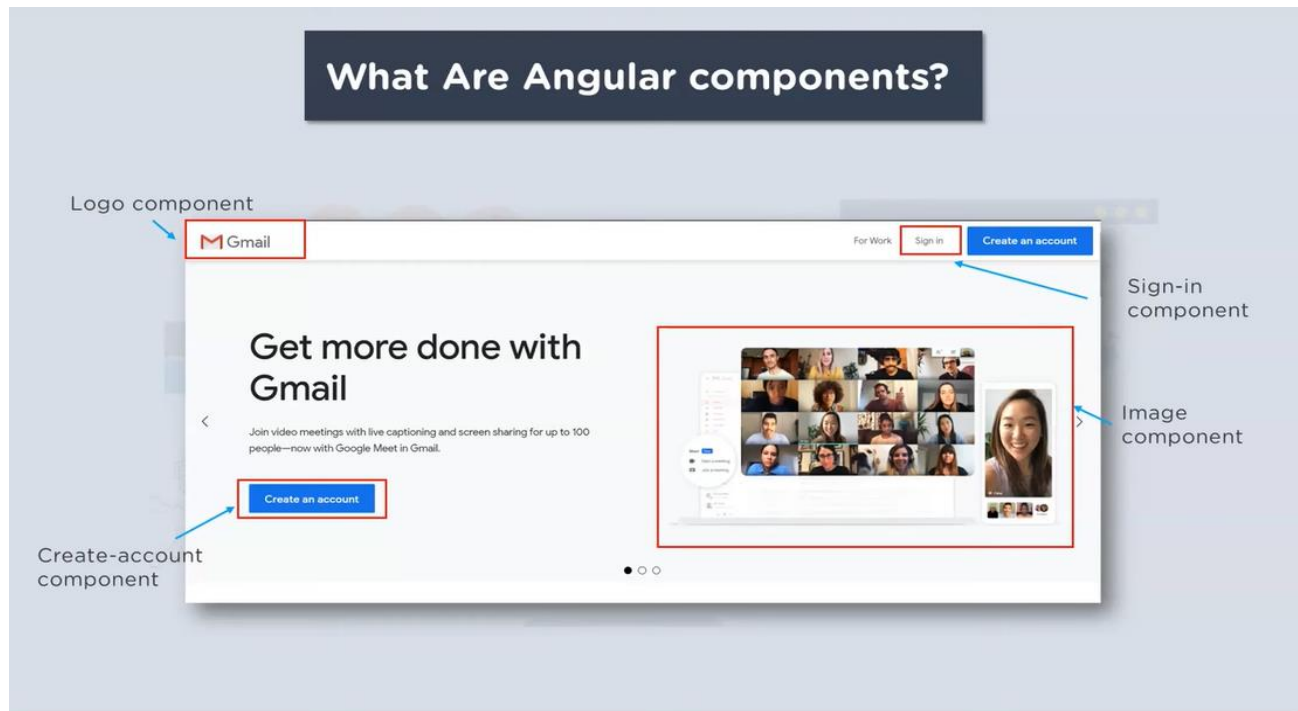
Output:

Upon running **ng serve**, the Angular CLI will compile the application and launch a development server. Open a web browser and navigate to **http://localhost:4200** to view the application running locally.



Hello Angular

2. Component:-



The above image showing Gmail is a Single Page Application each part consider like a component like logo component, sign-in component etc.,

Defination:-

The component is the basic building block of Angular. It has a selector, template, style, and other properties, and it specifies the metadata required to process the component.

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

Importing the component decorator from angular core library

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

Decorating the class with @Component decorator and providing the metadata

```
export class AppComponent {
  title = 'MyAngularApp';
}
```

Creating class to define data and logic for the view

parts of an Angular Component

An Angular component has several parts, such as:

Selector

It is the CSS selector that identifies this component in a template. This corresponds to the HTML tag that is included in the parent component. You can create your own [HTML tag](#). However, the same has to be included in the parent component.

Template

It is an inline-defined template for the view. The template can be used to define some markup. The markup could typically include some headings or paragraphs that are displayed on the UI.

TemplateUrl

It is the URL for the external file containing the template for the view.

Styles

These are inline-defined styles to be applied to the component's view

styleUrls

List of URLs to stylesheets to be applied to the component's view.

Before Creating Angular Component create Angular Project using this command

```
ng new Projectname or na new projectname --standalone false
```

Creating a Component in Angular 8:

To create a component in any angular application, follow the below steps:

- Get to the angular app via your terminal.(ng new project_name)
- Create a component using the following command:

```
ng g c <component_name>
```

OR

```
ng generate component <component_name>
```

- Following files will be created after generating the component:

Note:-write below picture four files in exam important to explaining component

```
Microsoft Windows [Version 10.0.18362.1082]  
(c) 2019 Microsoft Corporation. All rights reserved.
```

```
C:\Users\hp\demo>ng g c gfg  
CREATE src/app/gfg/gfg.component.html (18 bytes)  
CREATE src/app/gfg/gfg.component.spec.ts (607 bytes)  
CREATE src/app/gfg/gfg.component.ts (263 bytes)  
CREATE src/app/gfg/gfg.component.css (0 bytes)  
UPDATE src/app/app.module.ts (363 bytes)  
  
C:\Users\hp\demo>
```

Using a component in Angular 8:

- Go to the component.html file and write the necessary HTML code.

gfg.component.html:

```
<h1>GeeksforGeeks</h1>
```

- Go to the component.css file and write the necessary CSS code.

gfg.component.css:

```
h1{  
  
  color: green;  
  
  font-size: 30px;  
  
}
```

- Write the corresponding code in component.ts file.

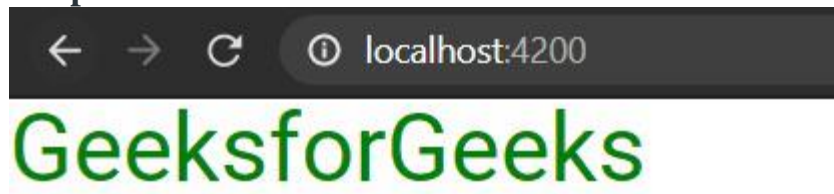
gfg.component.ts:

```
import { Component, OnInit } from '@angular/core';  
  
@Component({  
  
  selector: 'app-gfg',
```

```
templateUrl: './gfg.component.html',  
  
styleUrls: ['./gfg.component.css']  
  
})  
  
export class GfgComponent{  
  
  a ="GeeksforGeeks";  
  
}
```

- Run the Angular app using **ng serve --open**

Output:



4. Properties, Events & Binding with ngModel

Data Binding

Data binding is the core concept of Angular 8 and used to define the communication between a component and the DOM. It is a technique to link your data to your view layer. In simple words, you can say that data binding is a communication between your typescript code of your component and your template which user sees. It makes easy to define interactive applications without worrying about pushing and pulling data.

Data binding can be either one-way data binding or two-way data binding.

One-way databinding

One way databinding is a simple one way communication where HTML template is changed when we make changes in TypeScript code.

Or

In one-way databinding, the value of the Model is used in the View (HTML page) but you can't update Model from the View. Angular Interpolation / String Interpolation, Property Binding, and Event Binding are the example of one-way databinding.

Two-way databinding

In two-way databinding, automatic synchronization of data happens between the Model and the View. Here, change is reflected in both components. Whenever you make changes in the Model, it will be reflected in the View and when you make changes in View, it will be reflected in Model.

This happens immediately and automatically, ensures that the HTML template and the TypeScript code are updated at all times.



Angular provides four types of data binding and they are different on the way of data flowing.

- [String Interpolation](#)
- [Property Binding](#)
- [Event Binding](#)
- Class binding
- Style binding
- [Two-way binding](#)

One way Data Binding:-

1. [String Interpolation](#)
2. [Property Binding](#)
3. [Event Binding](#)
4. Class binding
5. Style binding

1.String Interpolation

String Interpolation is a **one-way databinding** technique which is used to output the data from a TypeScript code to HTML template (view). It uses the template expression in **double curly braces** to display the data from the component to the view. String interpolation adds the value of a property from the component.

For example:

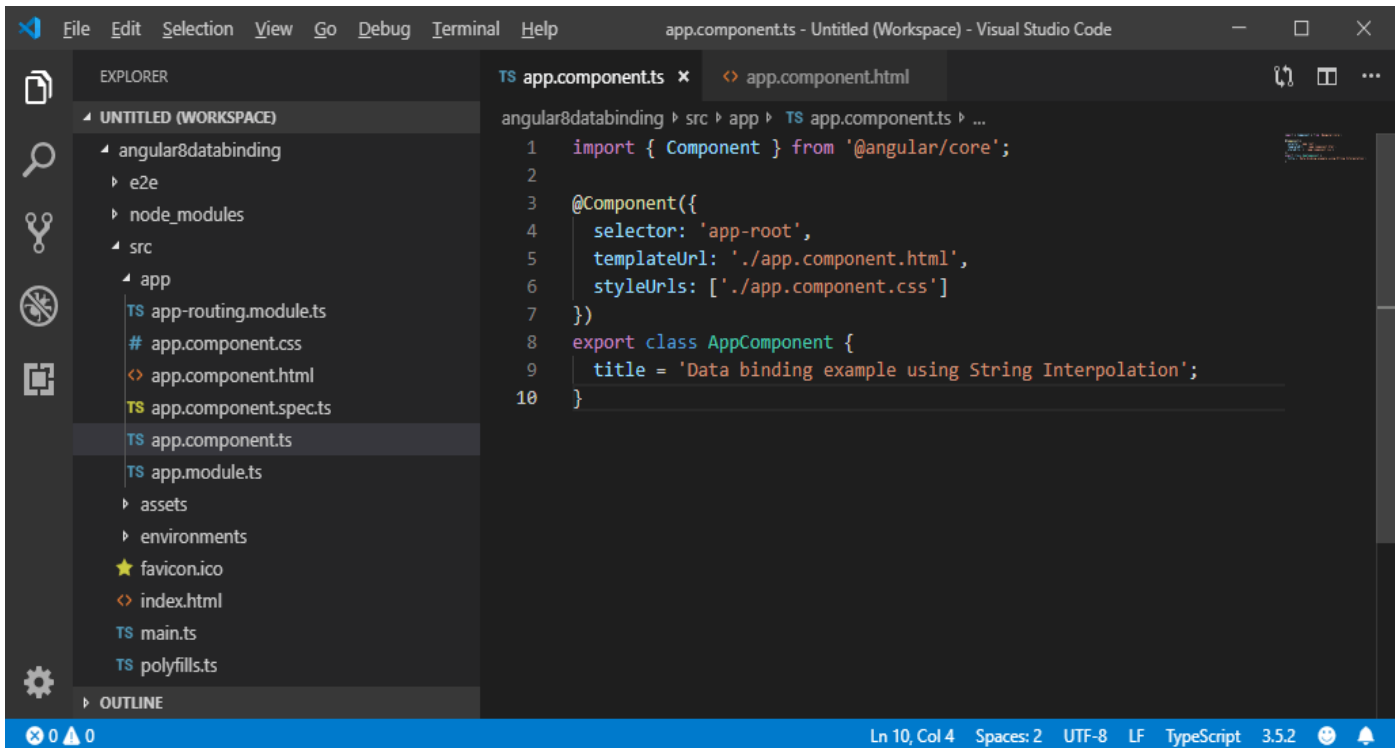
```
{{ data }}
```

We have already created an Angular project using Angular CLI.

Here, we are using the same project for this example.

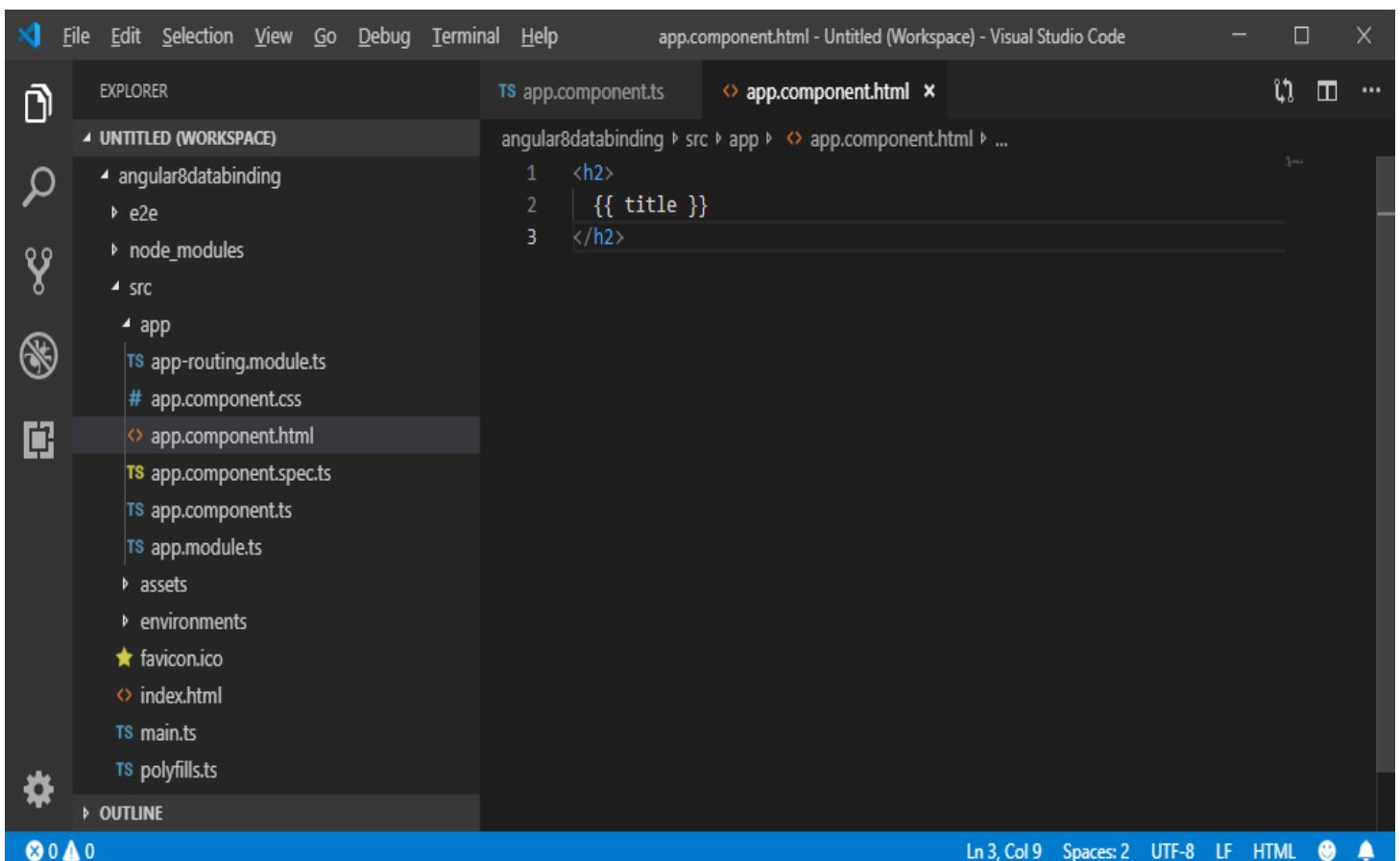
Open **app.component.ts** file and use the following code within the file:

```
1. import { Component } from '@angular/core';
2. @Component({
3.   selector: 'app-root',
4.   templateUrl: './app.component.html',
5.   styleUrls: ['./app.component.css']
6. })
7. export class AppComponent {
8.   title = 'Data binding example using String Interpolation';
9. }
```



Now, open **app.component.html** and use the following code to see string interpolation.

1. `<h2>`
2. `{{ title }}`
3. `</h2>`



Now, open Node.js command prompt and run the **ng serve** command to see the result.

Output:

ADVERTISEMENT



String Interpolation can be used to resolve some other expressions too. Let's see an example.

Example:

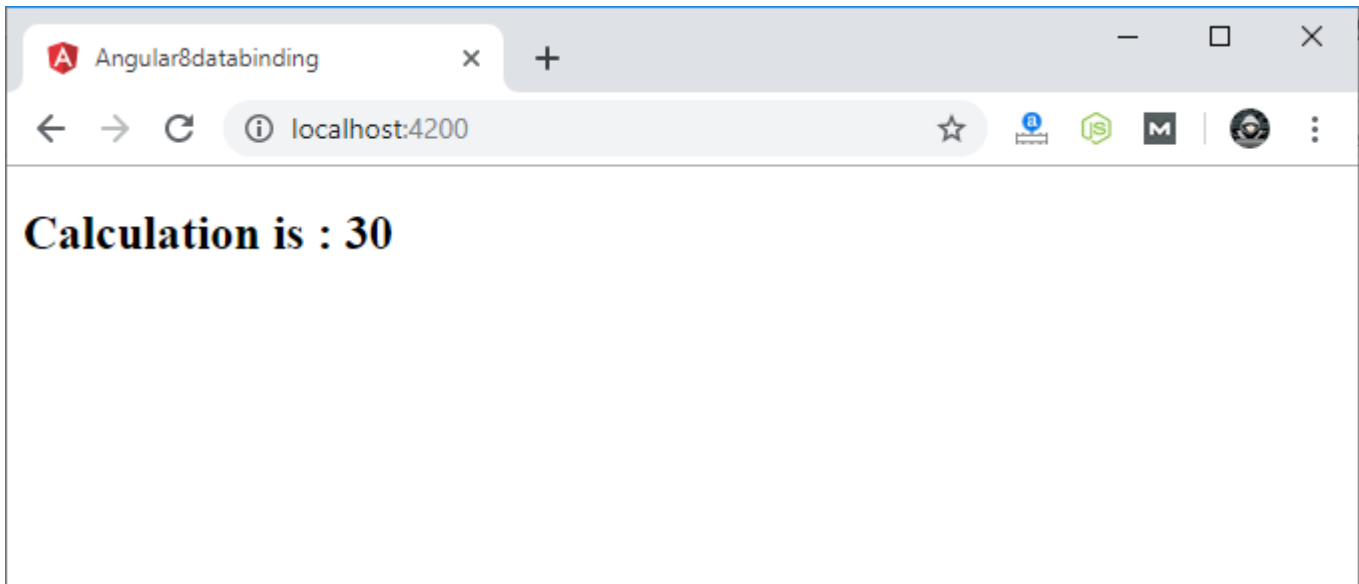
Update the **app.component.ts** file with the following code:

```
1. import { Component } from '@angular/core';
2. @Component({
3.   selector: 'app-root',
4.   templateUrl: './app.component.html',
5.   styleUrls: ['./app.component.css']
6. })
7. export class AppComponent {
8.   title = 'Data binding example using String Interpolation';
9.   numberA: number = 10;
10.  numberB: number = 20;
11. }
```

app.component.html:

```
1. <h2>Calculation is : {{ numberA + numberB }}</h2>
```

Output:



2. Property Binding in Angular 8

Property Binding is also a **one-way data binding** technique. In property binding, we bind a property of a DOM element to a field which is a defined property in our component TypeScript code. Actually Angular internally converts string interpolation into property binding.

For example:

```
<img [src]="imgUrl" />
```

Property binding is preferred over string interpolation because it has shorter and cleaner code. String interpolation should be used when you want to simply display some dynamic data from a component on the view between headings like h1, h2, p etc.

Note: String Interpolation and Property binding both are one-way binding. Means, if field value in the component changes, Angular will automatically update the DOM. But any changes in the DOM will not be reflected back in the component.

Property Binding Example

Open **app.component.ts** file and add the following code:

```
1. import { Component } from '@angular/core';
2. @Component({
3.   selector: 'app-root',
4.   templateUrl: './app.component.html',
5.   styleUrls: ['./app.component.css']
6. })
7. export class AppComponent {
8.   title = "Data binding using Property Binding";
9.   imgUrl="https://static.javatpoint.com/tutorial/angular7/images/angular-7-logo.png";
10. }
```



```
angular8databinding > src > app > TS app.component.ts > ...
1  import { Component } from '@angular/core';
2
3  @Component({
4    selector: 'app-root',
5    templateUrl: './app.component.html',
6    styleUrls: ['./app.component.css']
7  })
8  export class AppComponent {
9    title = "Data binding using Property Binding";
10    imgUrl="https://static.javatpoint.com/tutorial/angular7/images/ang
11  }
```

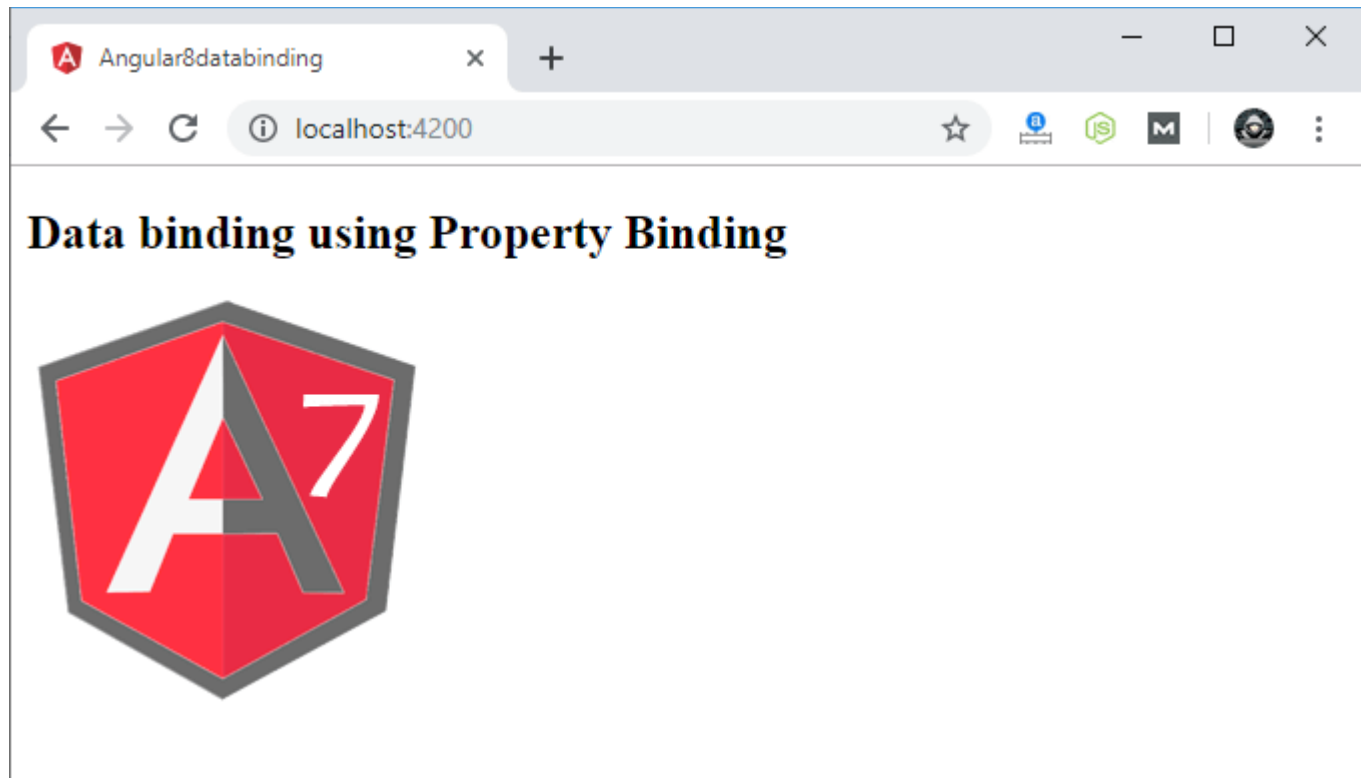
Now, open **app.component.html** and use the following code for property binding:

1. `<h2>{{ title }}</h2> <!-- String Interpolation -->`
2. ` <!-- Property Binding -->`

```
angular8databinding > src > app > < app.component.html > ...
1  <h2>{{ title }}</h2> <!-- String Interpolation -->
2  <img [src]="imgUrl" /> <!-- Property Binding -->
```

Run the ng serve command and open local host to see the result.

Output:



Event Binding in Angular 8

In Angular 8, event binding is used to handle the events raised from the DOM like button click, mouse move etc. When the DOM event happens (eg. click, change, keyup), it calls the specified method in the component. In the following example, the cookBacon() method from the component is called when the button is clicked:

For example:

1. `<button (click)="cookBacon()"></button>`

Event Binding Example

Let's take a button in the HTML template and handle the click event of this button. To implement event binding, we will bind click event of a button with a method of the component.

Now, open the **app.component.ts** file and use the following code:

Backward Skip 10sPlay VideoForward Skip 10s

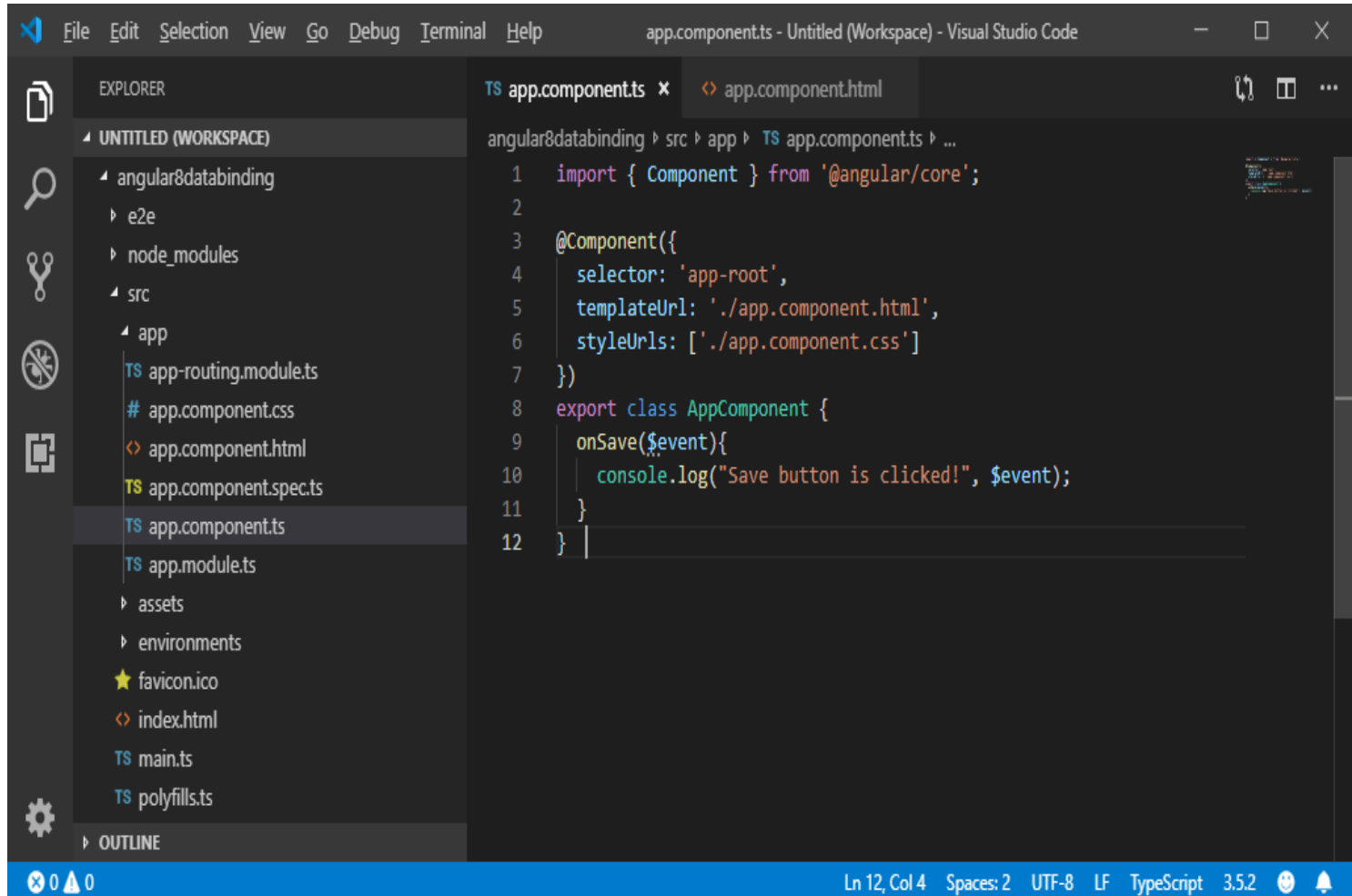
ADVERTISEMENT

1. `import { Component } from '@angular/core';`
2. `@Component({`
3. `selector: 'app-root',`
4. `templateUrl: './app.component.html',`
5. `styleUrls: ['./app.component.css']`

```

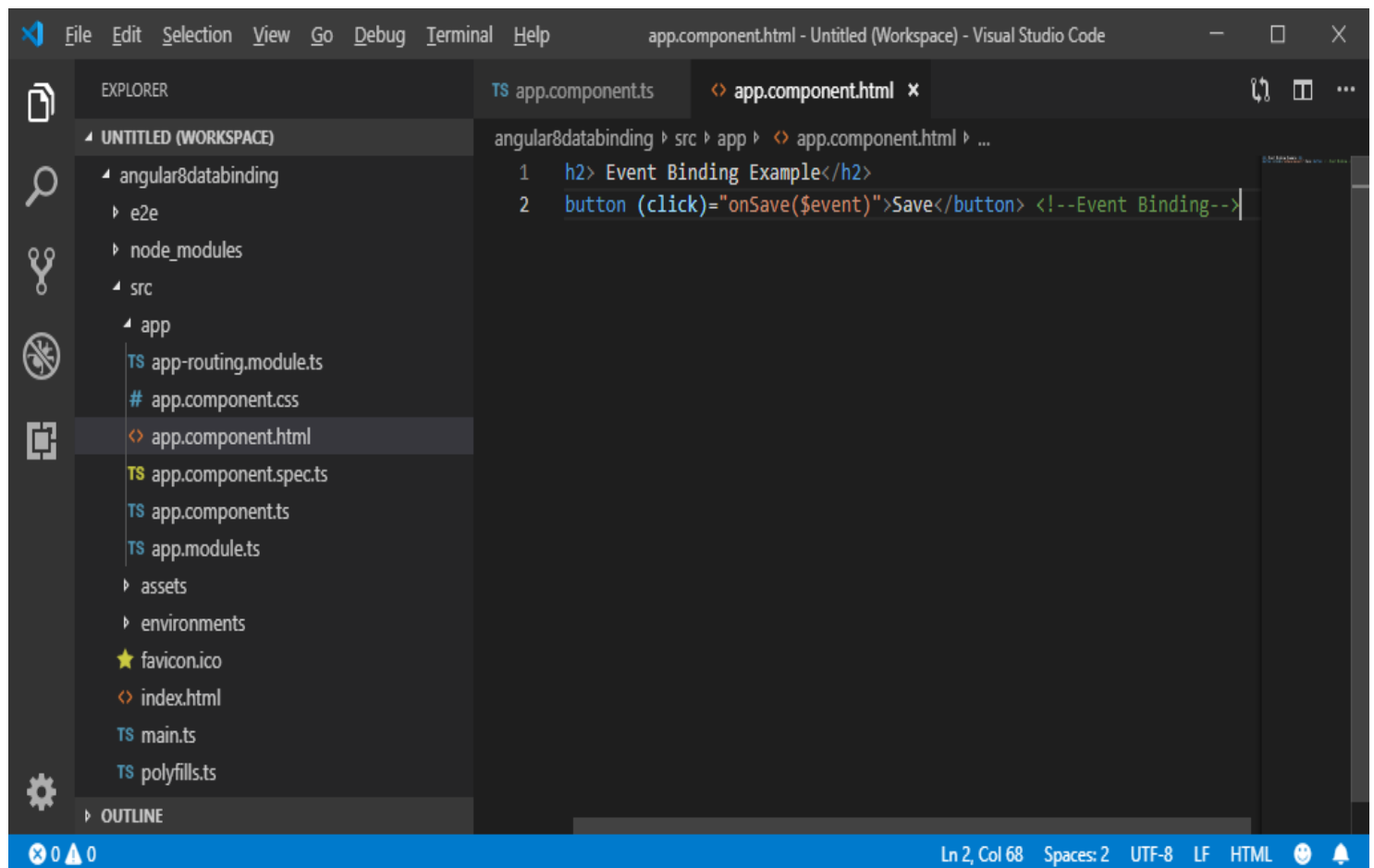
6.  })
7.  export class AppComponent {
8.    onSave($event){
9.      console.log("Save button is clicked!", $event);
10. }
11. }

```



app.component.html:

1. `<h2> Event Binding Example</h2>`
2. `<button (click)="onSave($event)">Save</button> <!--Event Binding-->`

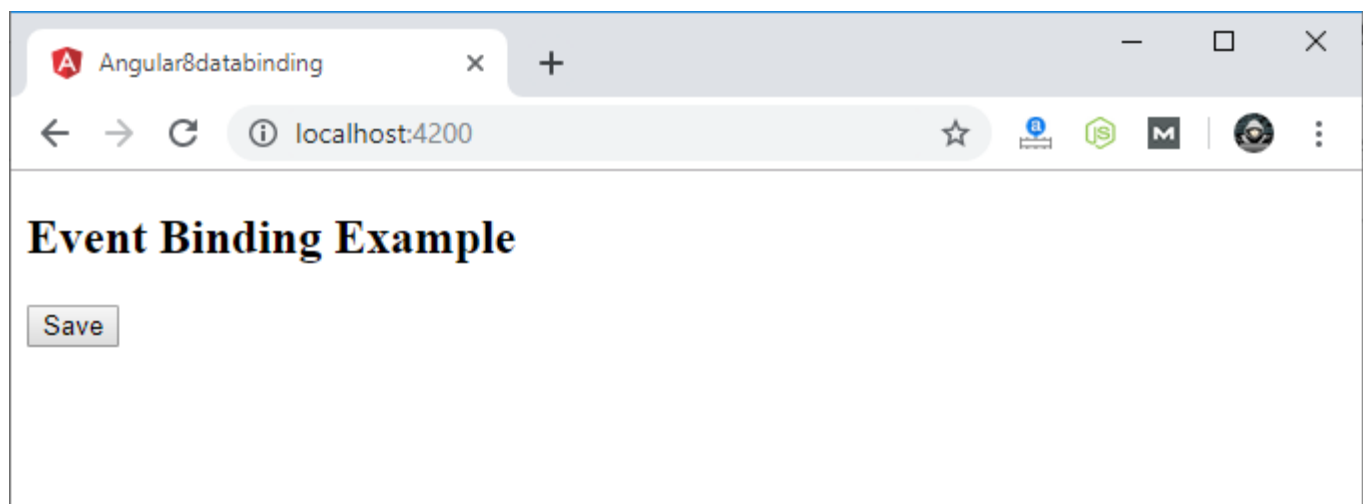


The screenshot shows the Visual Studio Code editor interface. The Explorer sidebar on the left displays the project structure for 'angular8databinding'. The file 'app.component.html' is selected and open in the editor. The code in the editor is as follows:

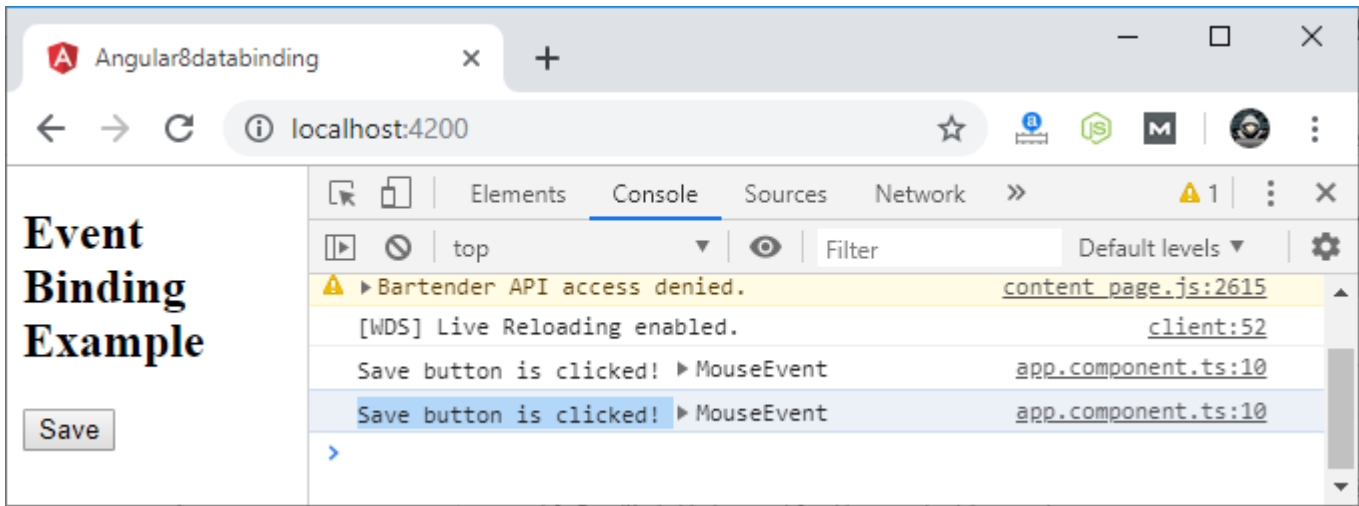
```
1 <h2> Event Binding Example</h2>
2 <button (click)="onSave($event)">Save</button> <!--Event Binding-->
```

The status bar at the bottom indicates the cursor is at line 2, column 68, with 2 spaces, UTF-8 encoding, and LF line endings.

Output:



Click on the "Save" button and open console to see result.



Now, you can see that the "Save" button is clicked.

4. Class Binding

Last Updated : 23 Sep, 2020

Class binding in Angular makes it very easy to set the class property of a view element. We can set or remove the CSS class names from an element's class attribute with the help of class binding. We bind a class of a DOM element to a field that is a defined property in our Typescript Code. Its syntax is like that of property binding.

Syntax:

```
<element [class] = "typescript_property">
```

Approach:

- Define a property element in the app.component.ts file.
- In the app.component.html file, set the class of the HTML element by assigning the property value to the app.component.ts file's element.

Example 1: Setting the class element using class binding.

app.component.html

HTML

```
<h1 [class] = "geeky">

  GeeksforGeeks

</h1>

Upper Heading's class is : "{{ g[0].className }}"
```

app.component.ts

Javascript

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

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

export class AppComponent {

  geeky = "GeekClass";

  g = document.getElementsByClassName(this.geeky);

}
```

Output:

GeeksforGeeks

Upper Heading's class is : "GeekClass"

5.Style Binding

Last Updated : 14 Sep, 2020

8.

It is very easy to give the CSS styles to HTML elements using style binding in Angular 8. Style binding is used to set a style of a view element. We can set the inline styles of an HTML element using the style binding in angular. You can also add styles conditionally to an element, hence creating a dynamically styled element.

Syntax:

```
<element [style.style-property] = "'style-value'">
```

Example 1:

app.component.html:

- HTML

```
<h1 [style.color] = "'green'"  
  
    [style.text-align] = "'center'" >  
  
    GeeksforGeeks  
  
</h1>
```

Output:

GeeksforGeeks

b. Two way Data Binding using ngmodel

We have seen that in one-way data binding any change in the template (view) were not be reflected in the component TypeScript code. To resolve this problem, Angular provides two-way data binding. The two-way binding has a feature to update data from component to view and vice-versa.

In two-way databinding, automatic synchronization of data happens between the Model and the View. Here, change is reflected in both components. Whenever you make changes in the Model, it will be reflected in the View and when you make changes in View, it will be reflected in Model.

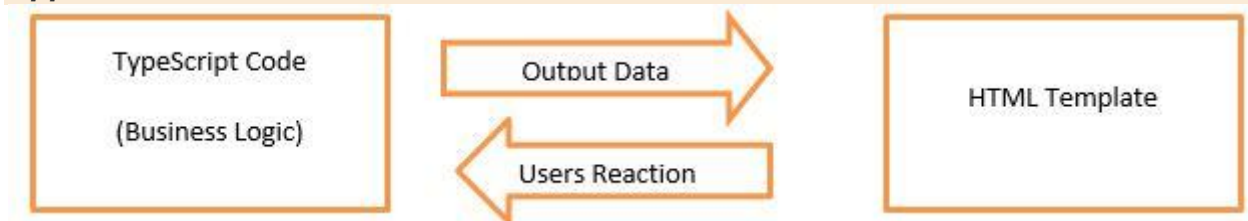
This happens immediately and automatically, ensures that the HTML template and the TypeScript code are updated at all times.

In two way data binding, **property binding and event binding** are combined together.

Syntax:

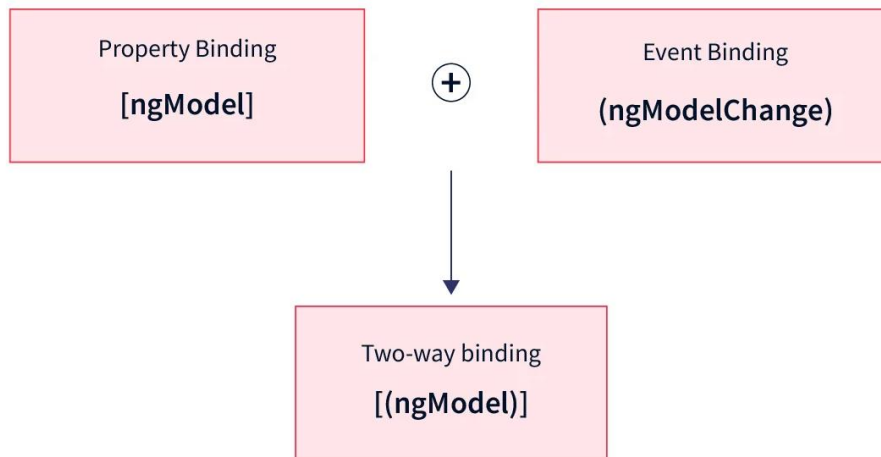
1. [(ngModel)] = "[property of your component]"

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.



Let's take an example to understand it better.

Note:-when you are using ngmodel import FormsModule



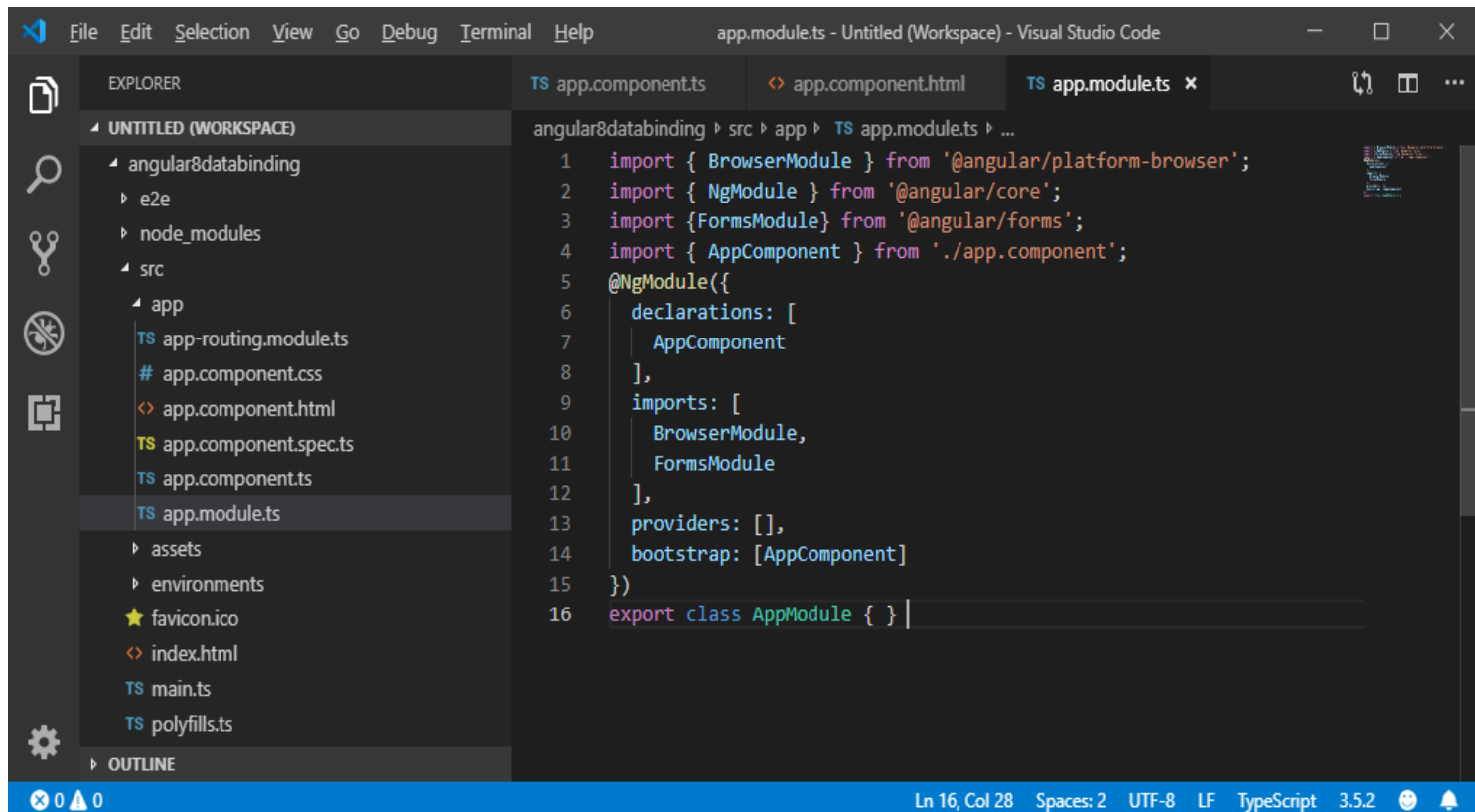
[property binding] + (event binding) = [(property)]

[ngModel] + (ngModelChange) = [(ngModel)]

[text] + (textChange) = [(text)]

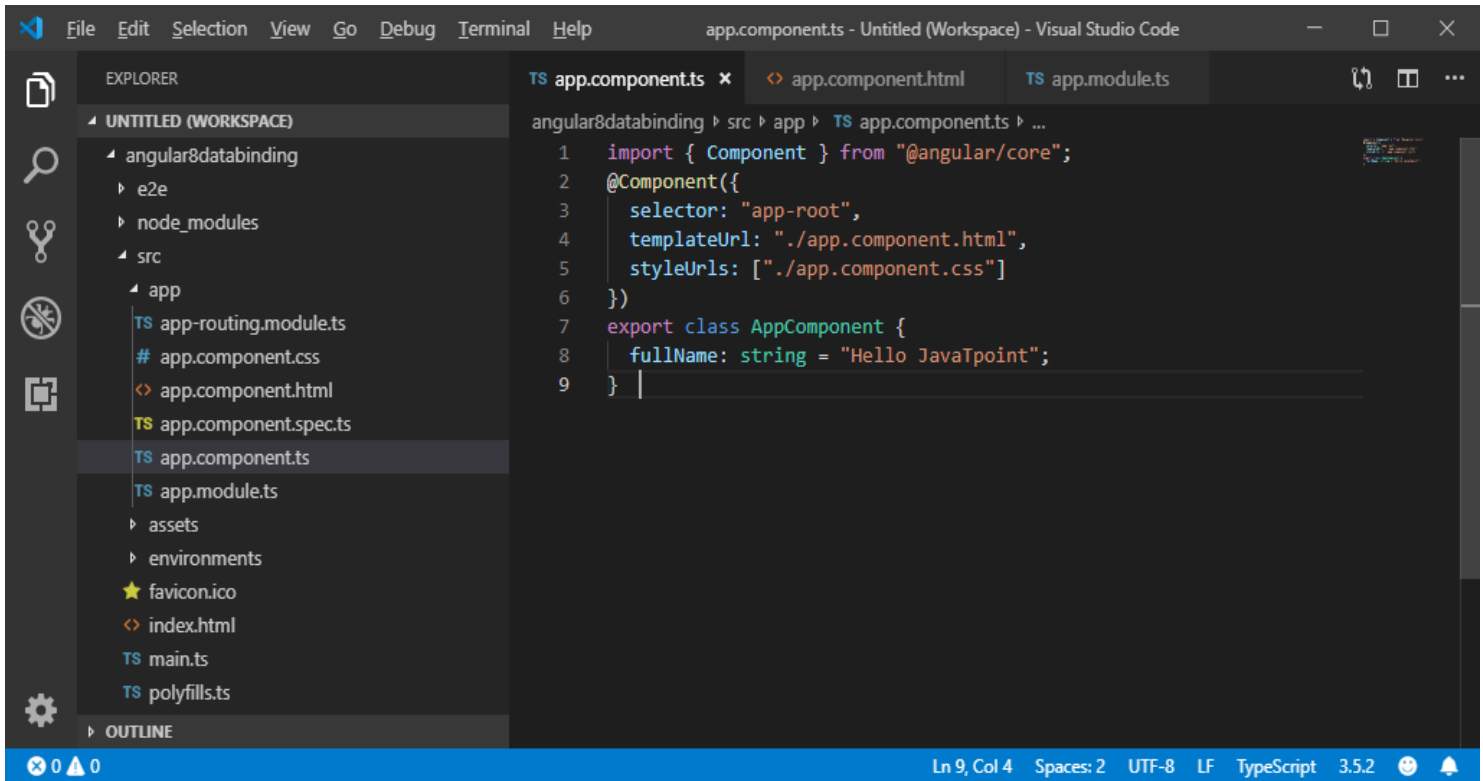
Open your project's **app.module.ts** file and use the following code:

```
1. import { BrowserModule } from '@angular/platform-browser';
2. import { NgModule } from '@angular/core';
3. import {FormsModule} from '@angular/forms';
4. import { AppComponent } from './app.component';
5. @NgModule({
6.   declarations: [
7.     AppComponent
8.   ],
9.   imports: [
10.    BrowserModule,
11.    FormsModule
12.  ],
13.  providers: [],
14.  bootstrap: [AppComponent]
15. })
16. export class AppModule { }
```

app.component.ts file:

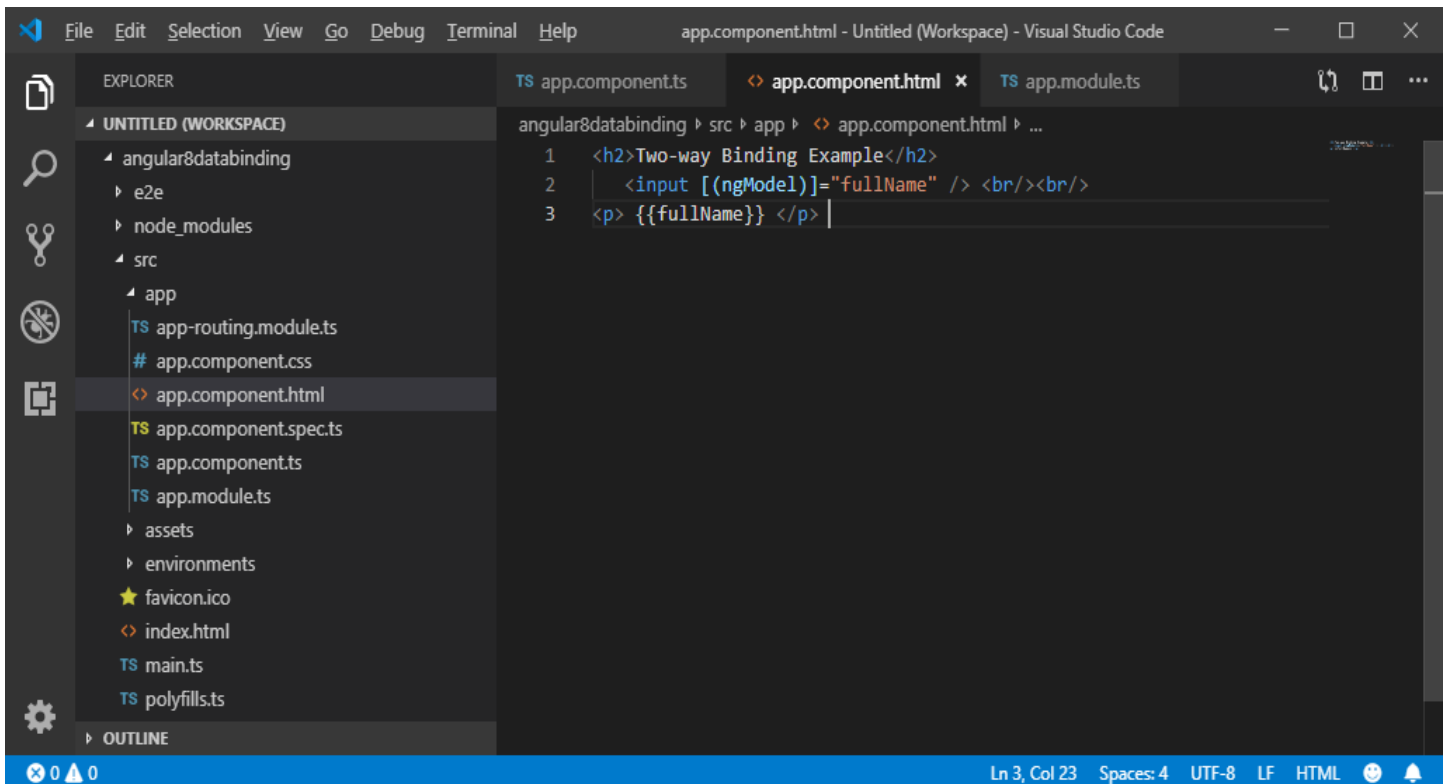
1. import { Component } from "@angular/core";
2. @Component({
3. selector: "app-root",
4. templateUrl: "./app.component.html",
5. styleUrls: ["/app.component.css"]
6. })
7. export class AppComponent {
8. fullName: string = "Hello JavaTpoint";
9. }



app.component.html file:

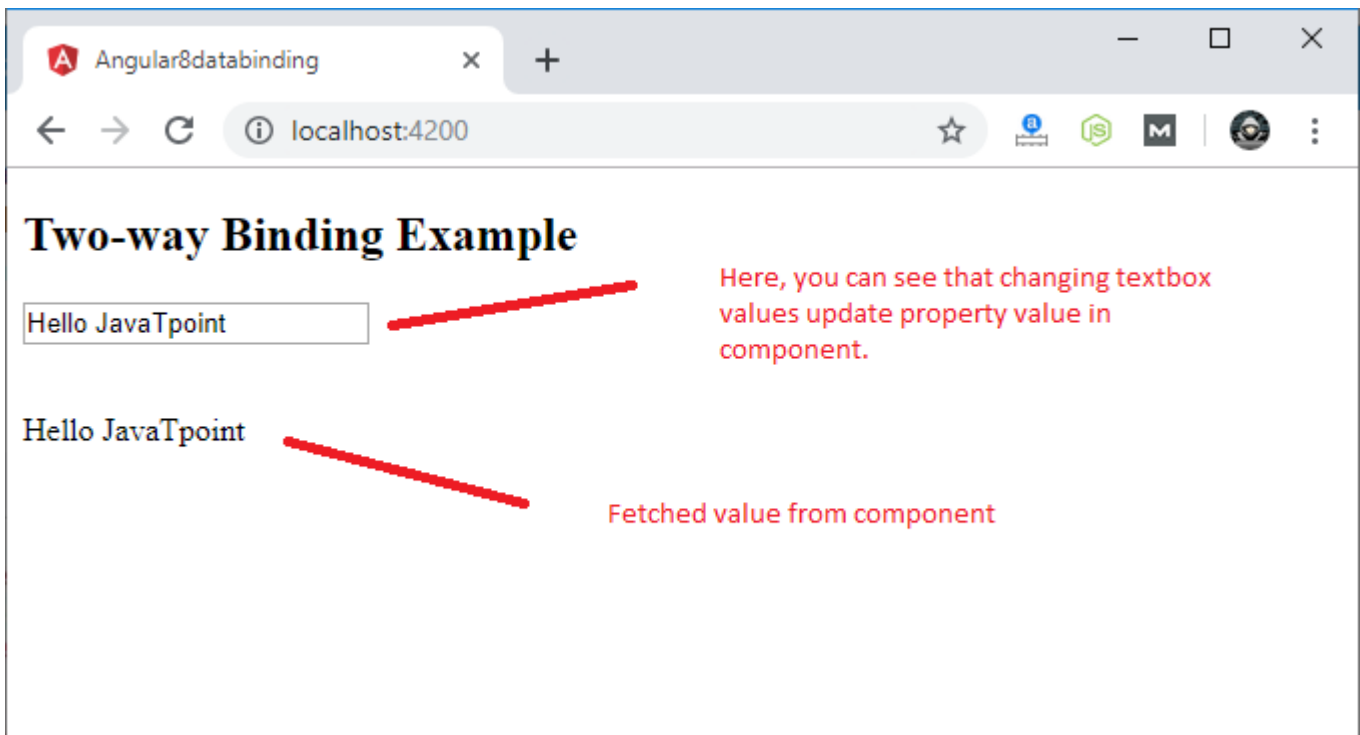
1. `<h2>Two-way Binding Example</h2>`
2. `<input [(ngModel)]="fullName" />

`
3. `<p> {{fullName}} </p>`



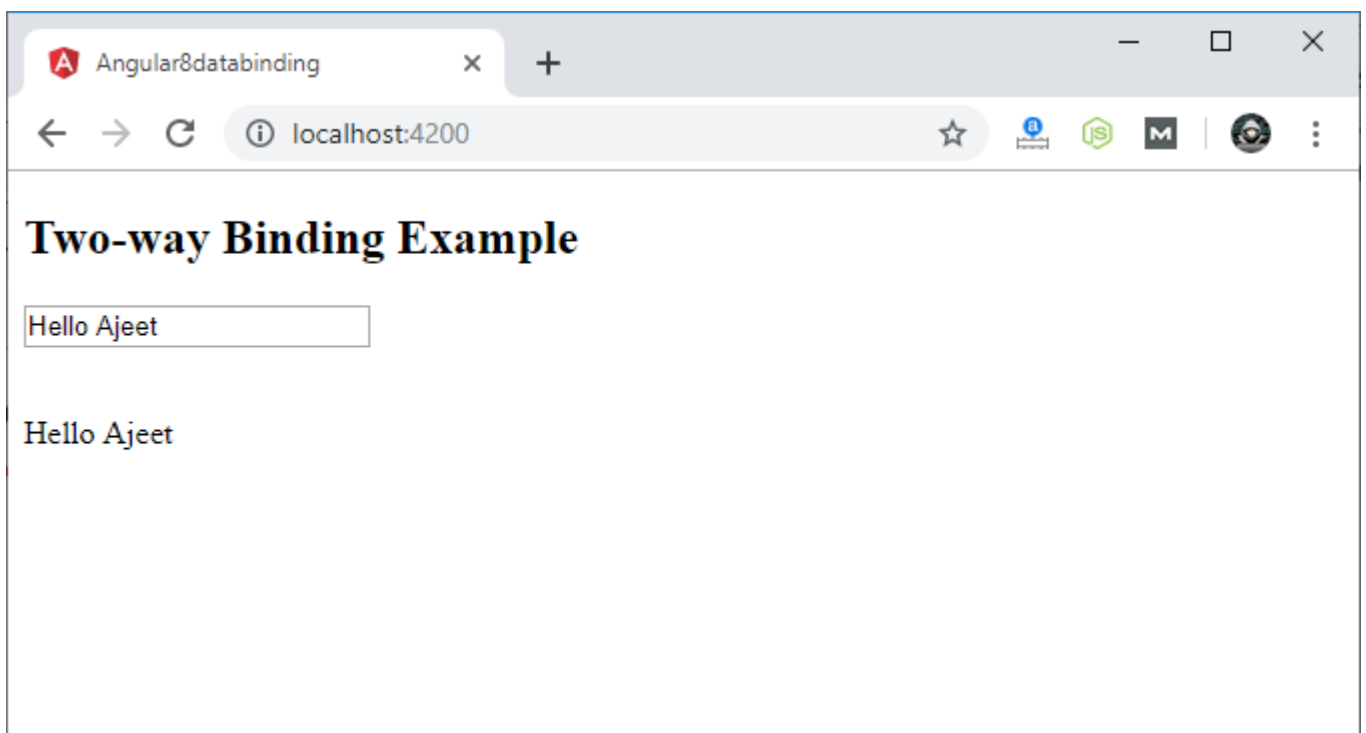
Now, start your server and open local host browser to see the result.

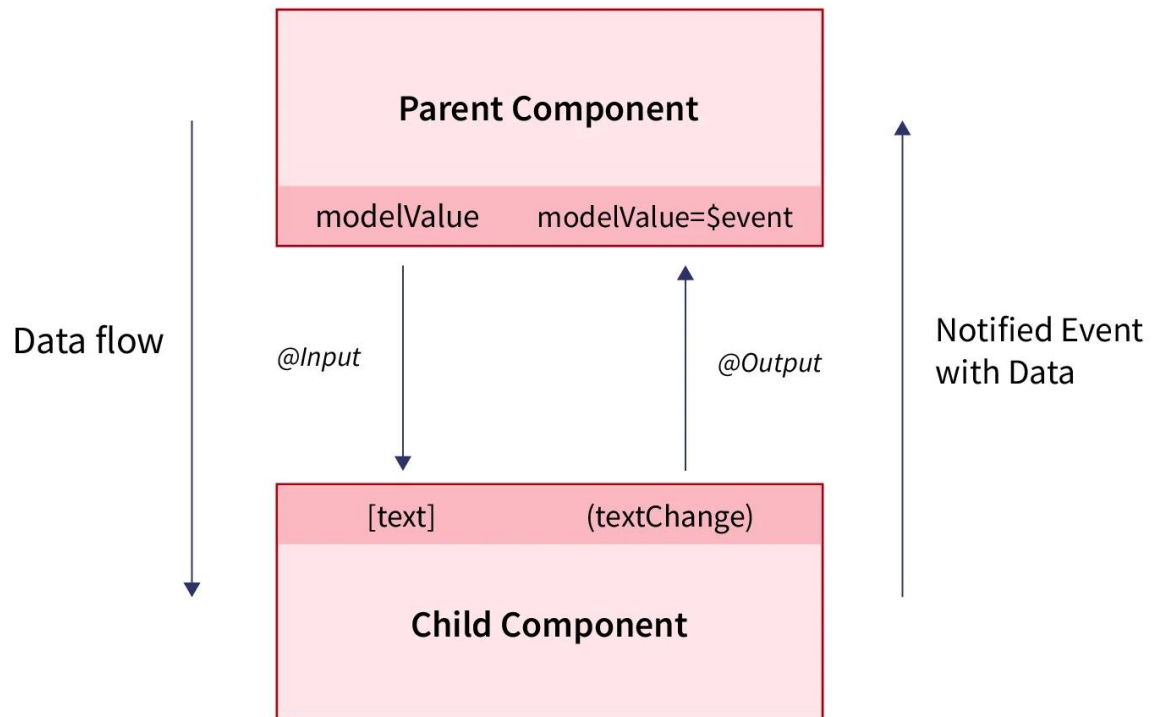
Output:



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

For example:





(OR)

Without using ngmodel

[property binding] + (event binding) = [(property)]

app.component.html

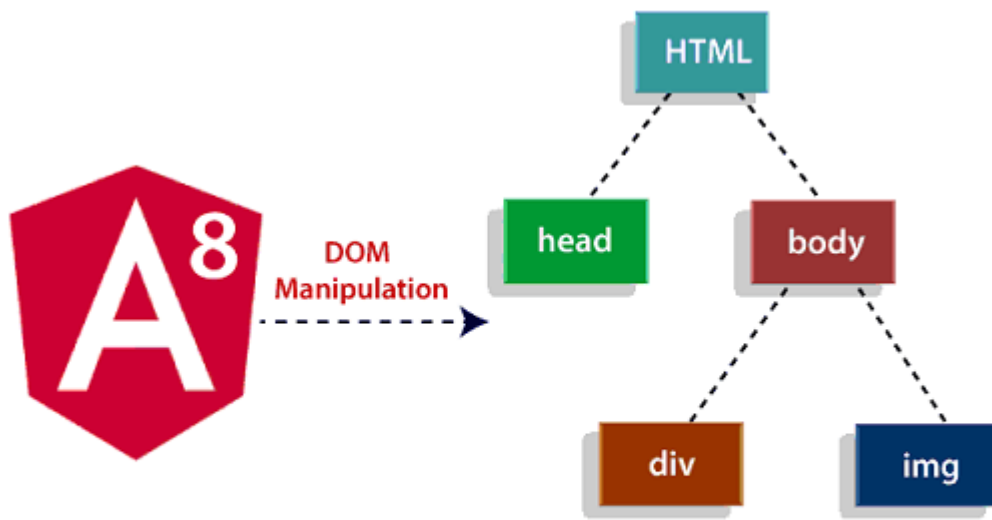
```
<label>User Name</label>
<input type="text" [value]="text" (input)="updateValue
($event)">
<h1>{{text}}</h1>
```

app.component.ts

```
fullName: string = "Hello JavaTpoint";
```

Angular Directives

The Angular 8 directives are used to manipulate the DOM. By using Angular directives, you can change the appearance, behavior or a layout of a DOM element. It also helps you to extend HTML.



Angular 8 Directive

Angular 8 directives can be classified in 3 categories based on how they behave:

- Component Directives
- Structural Directives
- Attribute Directives

Component Directives: Component directives are used in main class. They contain the detail of how the component should be processed, instantiated and used at runtime.

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).

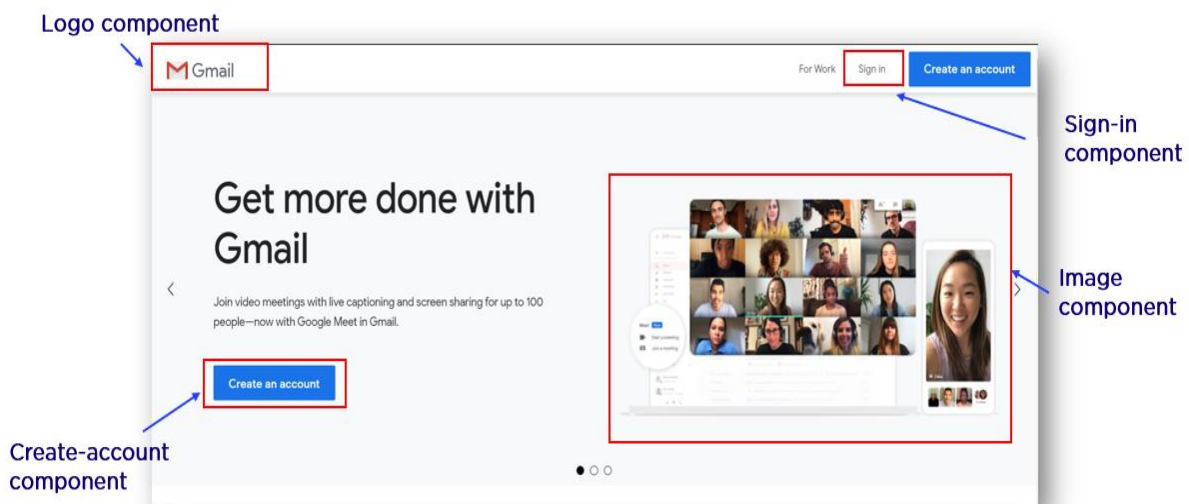
Attribute Directives: Attribute directives are used to change the look and behavior of the DOM elements. For example: ngClass directive, and ngStyle directive etc.

- **ngClass Directive:** The ngClass directive is used to add or remove CSS classes to an HTML element.
- **ngStyle Directive:** The ngStyle directive facilitates you to modify the style of an HTML element using the expression. You can also use ngStyle directive to dynamically change the style of your HTML element.

5. Fetch Data from a Service

What is the Need for Angular Services?

We're sure you are aware of the concept of components in Angular. The user interface of the application is developed by embedding several components into the main component.

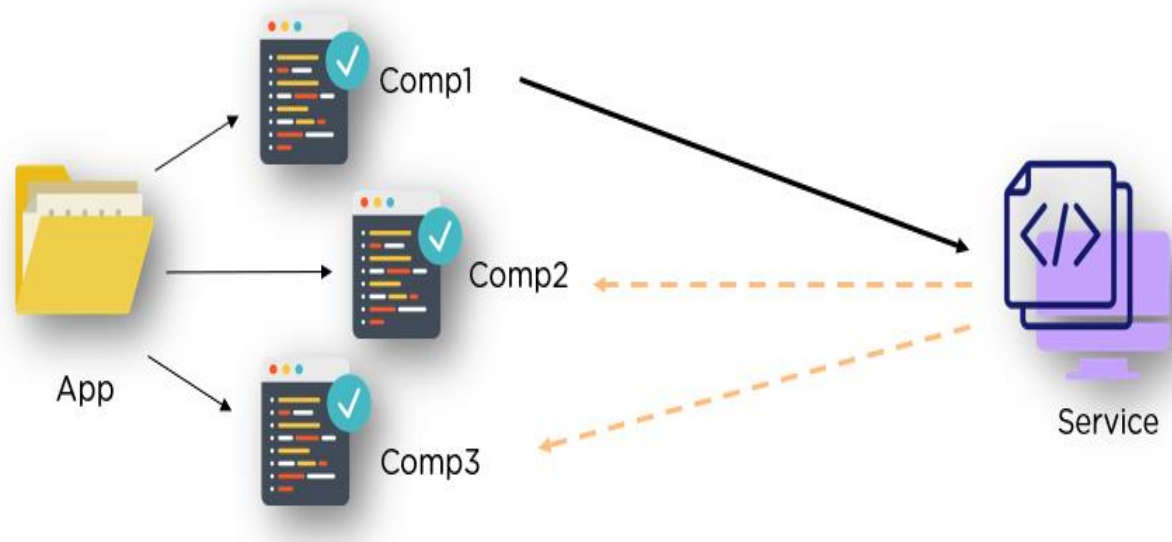


However, these components are generally used only for rendering purposes. They are only used to define what appears on the user interface. Ideally, other tasks, like data and image fetching, network connections, database management, are not performed. Then how are these tasks achieved? And what if more than one component performs similar tasks? Well, Services take care of this. They perform all the operational tasks for the components.

- **Services avoid rewriting of code. A service can be written once and injected into all the components that use that service**
- A service could be a function, variable, or feature that an application needs

What Are Angular Services?

Angular services are objects that get instantiated just once during the lifetime of an application. They contain methods that maintain data throughout the life of an application, i.e., data is available all the time.



The main objective of a service is to organize and share business logic, models, or data and functions with different components of an Angular application. They are usually implemented through dependency injection.

Features of Angular Services

- Services in Angular are simply typescript classes with the `@injectable` decorator. This decorator tells angular that the class is a service and can be injected into components that need that service. They can also inject other services as dependencies.
- As mentioned earlier, these services are used to share a single piece of code across multiple components. These services are used to hold business logic.
- Services are used to interact with the backend. For example, if you wish to make AJAX calls, you can have the methods to those calls in the service and use it as a dependency in files.



A Service is a Class



Decorated with
`@Injectable`



They share the same
piece of code



Hold the business logic



Interact with the
backend



Share data among
components



Services are singleton



Registered on modules
or components

- In angular, the components are singletons, meaning that only one instance of a service that gets created, and the same instance is used by every building block in the application.

- A service can be registered as a part of the module, or as a part of the component. To register it as a part of the component, you'll have to specify it in the providers' array of the module.

Fetch data from service Example:-

Use this command **ng g s service_name**

Creating Angular Project Use below Commands

- npm install -g @angular/cli //creating cli
- ng version
- ng new prog9 --standalone false //creating angular project SPA(Single page application)
//app component is a default component.
- cd prog9
- ng g c header //creating header component
- ng g c home //creating home component
- ng g c profile //creating profile component
- ng serve //running angular project
- Use this command for creating service **ng g s service_name**

Creating body of about, contact, home and header. Header is a navbar this page creating routerLinks about and contact. Home is a default link when header loaded it is displayed. App.module.ts file creating url paths for each page.

Here test.service.ts file is creating for displaying fruits names you can access service data any component here fetching data service to about.

1)about.component.html:-

```
<h1>This is About Component</h1>
<h3>Which Fruit You Like?</h3>

<br>
<div *ngFor="let m of names">
  {{m}}
</div>
```

About.component.ts:-

```
import { Component } from '@angular/core';
import { TestService } from '../test.service';
@Component({
  selector: 'app-about',
  templateUrl: './about.component.html',
  styleUrls: ['./about.component.css']
})
export class AboutComponent {

  constructor(private ts:TestService){
```



```
}  
names=this.ts.names;  
  
}
```

2)contact.component.html:-

<h1>This is Contact Component</h1>

3)header.component.css:-

```
ul li{  
    list-style: none;  
}  
ul li a{  
    text-decoration: none;  
}  
ul{  
    display: flex;  
    justify-content: flex-start;  
    gap: 20px;  
    background-color: aqua;  
    height: 50px;  
}  
a{  
    line-height:50px;  
    color:black;  
    margin:0 20px;  
    font-weight: bold;  
    font-size:30px;  
}
```

4)header.component.html:-


```
<a routerLink="/about" router="active">about</a>
</li>
<li>
  <a routerLink="/contact" routerActiveLink="active">contact</a>
</li>
</ul>
```

5) home.component.html:-

```
<h1>This is Home Component</h1>
```

6) **notfound.component.html:-**

```
<p>notfound works!</p>
```

7) **app.component.html:-**

```
<app-header></app-header>
<router-outlet></router-outlet>
```

8) **app.module.ts:-**

```
import { AppComponent } from './app.component';
import { HeaderComponent } from './header/header.component';
import { AboutComponent } from './about/about.component';
import { ContactComponent } from './contact/contact.component';
import { HomeComponent } from './home/home.component';
import { NotfoundComponent } from './notfound/notfound.component';
import { RouterModule,Routes } from '@angular/router';

const routes:Routes=[
  {
    path:"",component:HomeComponent
  },
  {
```

```

    path:'about',component:AboutComponent
  },
  {
    path:'contact',component:ContactComponent
  },
  {
    path:'**',component:NotFoundComponent
  }
]

```

imports: [

```

    RouterModule.forRoot(routes)
  ],

```

Test.service.ts:-

```

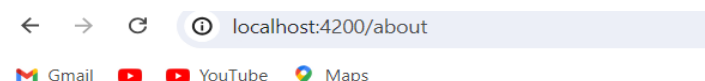
import { Injectable } from '@angular/core';

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

  constructor() { }
  names=['Mango','Banana','Watermelon','Apple'];
}

```

Output:



This is About Component

Which Fruit You Like?

☐ Mango
☐ Banana
☐ Watermelon
☐ Apple

6. Submit data to service:-

npm install bootstrap --save

When Bootstrap is installed open angular.json file and add bootstrap.min.css file reference under "styles":

```
1. "styles": [  
2.   "src/styles.css",  
3.   "node_modules/bootstrap/dist/css/bootstrap.min.css"  
4. ]
```

Now we need to create components and service. Use the following commands to create the same.

ng g c header

ng g c reg

Note

g stands for generate | c stands for Component | s stands for Service

Open app.modules.ts file and add these lines:

```
import { NgModule } from '@angular/core';  
import { BrowserModule } from '@angular/platform-browser';  
  
import { AppRoutingModule } from './app-routing.module';  
import { AppComponent } from './app.component';  
import { RegComponent } from './reg/reg.component';  
import { HeaderComponent } from './header/header.component';  
import { FormsModule, ReactiveFormsModule } from '@angular/forms';  
import { RouterModule, Routes } from '@angular/router';  
  
const routes: Routes = [  
  
  { path: "reg", component: RegComponent }  
];  
  
imports: [  
  BrowserModule,  
  AppRoutingModule,  
  FormsModule,  
  ReactiveFormsModule,  
  RouterModule.forRoot(routes)  
]
```

In app.component.html replace the existing code with the below code:

```
1. <app-header></app-header>  
2.   <router-outlet></router-outlet>
```

Let's start with components now.

Open header.component.html file and replace with the code below.

```
<ul>
  <li>
    <a routerLink="/reg">create</a>
  </li>
</ul>
```

Open header.component.css file and replace with the code below.

```
ul{
  background-color:aqua;
}
```

Now let's create a function in Service.

Open data.service.ts file and replace with the code below.

```
1. public SaveEmployee(empdata) {
2.   console.log("Full Name : " + empdata.regFullName);
3.   console.log("Email Id : " + empdata.regEmail);
4. }
```

Open reg.component.html file and replace with the code below.

```
1. <div class="container" style="margin-top: 150px;">
2.   <form [formGroup]="frmRegister" (ngSubmit)="SaveEmployee(frmRegister.value)">
3.     <div class="panel panel-primary">
4.       <div class="panel-heading">
5.         <h3 class="panel-title">Employee Registration</h3>
6.       </div>
7.       <div class="panel-body">
8.         <div class="form-group">
9.           <label for="fullName">Full Name</label>
10.          <input id="fullName" formControlName="regFullName" type=
"text" class="form-control" required />
11.        </div>
12.        <div class="form-group">
13.          <label for="email">Email</label>
14.          <input id="email" formControlName="regEmail" type="email
" class="form-control" required />
15.        </div>
16.      </div>
17.      <div class="panel-footer">
18.        <button type="submit" class="btn btn-
primary">Save</button>
19.      </div>
20.    </div>
21.  </form>
22. </div>
```

Open reg.component.ts file and replace with the code below.

```
1. import {
2.   Component,
3.   OnInit
4. } from '@angular/core';
```

```

5. import {
6.     FormGroup,
7.     FormBuilder
8. } from '@angular/forms';
9. import {
10.     DataService
11. } from '../data.service';
12. @Component({
13.     selector: 'app-reg',
14.     templateUrl: './reg.component.html',
15.     styleUrls: ['./reg.component.css']
16. })
17. export class RegComponent implements OnInit {
18.     frmRegister: FormGroup;
19.     constructor(private _fb: FormBuilder, private dataservice: DataService)
20.     {}
21.     ngOnInit(): void {
22.         this.frmRegister = this._fb.group({
23.             regFullName: "",
24.             regEmail: ""
25.         });
26.     }
27.     SaveEmployee(value) {
28.         this.dataservice.SaveEmployee(value);
29.     }

```

Now build your application by `ng build`.

Run application by `ng serve`.

Output:-

The screenshot displays the 'Employee Registration' web application. The form has two input fields: 'Full Name' with the value 'Ashok' and 'Email' with the value 'ashok@xyz.com'. A blue 'Save' button is located below the email field. To the right, the Chrome DevTools console is open, showing the following messages:

- Angular is running in the development mode. Call `enableProdMode()` to enable the production mode. (core.js:40848)
- [WDS] Live Reloading enabled. (client:52)
- Full Name : Ashok (data.service.ts:11)
- Email Id : ashok@xyz.com (data.service.ts:12)

The last two log entries are highlighted with an orange box, and orange arrows point from them to the corresponding input fields in the form, indicating that the data was successfully captured and logged.

7. Http Module:-

Defination:-

\$http is an AngularJS service for reading data from remote servers. Implements an HTTP client API for Angular apps that relies on the XMLHttpRequest interface exposed by browsers. Includes testability features, typed request and response objects, request and response interception, observable APIs, and streamlined error handling.

These components are self-sufficient and can be used on their own without being tied to a specific **NgModule**. But, sometimes, when you're working with these standalone components, you might need to fetch data from servers or interact with APIs using HTTP requests.

We need to import the http module to make use of the http service. Let us consider an example to understand how to make use of the http service.

Example1:- Fetching data from API and displayed console

To start using the http service, we need to import the module in **app.module.ts** as shown below –

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

If you see the highlighted code, we have imported the HttpClientModule from @angular/common/http and the same is also added in the imports array.

Let us now use the http client in the **app.component.ts**.

```
import { Component } from '@angular/core';
import { HttpClient } from '@angular/common/http';
@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent {
  constructor(private http: HttpClient) { }
  ngOnInit() {
    this.http.get("http://jsonplaceholder.typicode.com/users").
```

```
subscribe((data) => console.log(data))
}
```

Let us understand the code highlighted above. We need to import http to make use of the service, which is done as follows –

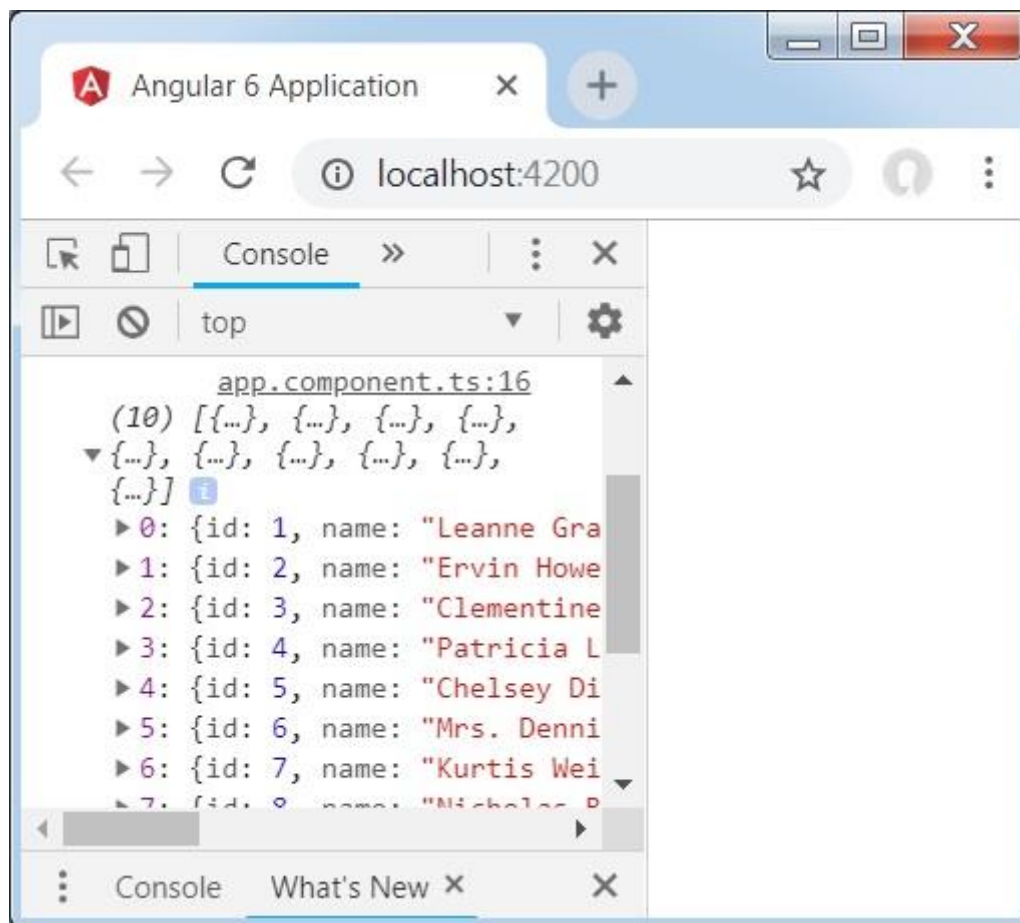
```
import { HttpClient } from '@angular/common/http';
```

In the class **AppComponent**, a constructor is created and the private variable http of type Http. To fetch the data, we need to use the **get API** available with http as follows

```
this.http.get();
```

It takes the url to be fetched as the parameter as shown in the code.

We will use the test url – <https://jsonplaceholder.typicode.com/users> to fetch the json data. The subscribe will log the output in the console as shown in the browser –



If you see, the json objects are displayed in the console. The objects can be displayed in the browser too.

Example2:-

For the objects to be displayed in the browser, update the codes in **app.component.html** and **app.component.ts** as follows –

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



```

@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent {
  constructor(private http: HttpClient) { }
  httpdata;
  ngOnInit() {
    this.http.get("http://jsonplaceholder.typicode.com/users")
      .subscribe((data) => this.displaydata(data));
  }
  displaydata(data) { this.httpdata = data; }
}

```

In **app.component.ts**, using the subscribe method we will call the display data method and pass the data fetched as the parameter to it.

In the display data method, we will store the data in a variable httpdata. The data is displayed in the browser using **for** over this httpdata variable, which is done in the **app.component.html** file.

```

<ul *ngFor = "let data of httpdata">
  <li>Name : {{ data.name }} Address: {{ data.address.city }}</li>
</ul>

```

The json object is as follows –

```

{
  "id": 1,
  "name": "Leanne Graham",
  "username": "Bret",
  "email": "Sincere@april.biz",

  "address": {
    "street": "Kulas Light",
    "suite": "Apt. 556",
    "city": "Gwenborough",
    "zipcode": "92998-3874",
    "geo": {
      "lat": "-37.3159",
      "lng": "81.1496"
    }
  },

  "phone": "1-770-736-8031 x56442",
  "website": "hildegard.org",
  "company": {
    "name": "Romaguera-Crona",

```

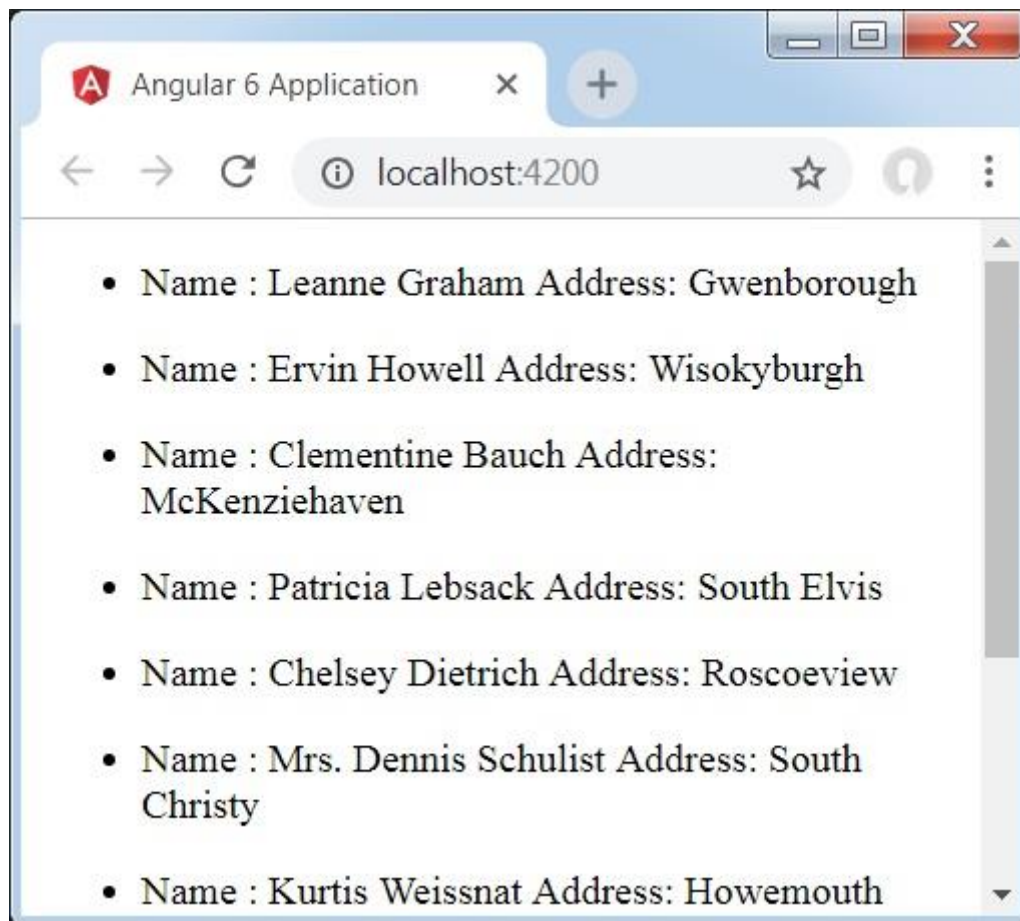
```

    "catchPhrase": "Multi-layered client-server neural-net",
    "bs": "harness real-time e-markets"
  }
}

```

The object has properties such as id, name, username, email, and address that internally has street, city, etc. and other details related to phone, website, and company. Using the **for** loop, we will display the name and the city details in the browser as shown in the **app.component.html** file.

This is how the display is shown in the browser –



Let us now add the search parameter, which will filter based on specific data.

Example 3:-

We need to fetch the data based on the search param passed.

Following are the changes done in **app.component.html** and **app.component.ts** files –

app.component.ts

```

import { Component } from '@angular/core';
import { HttpClient } from '@angular/common/http';
@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})

```

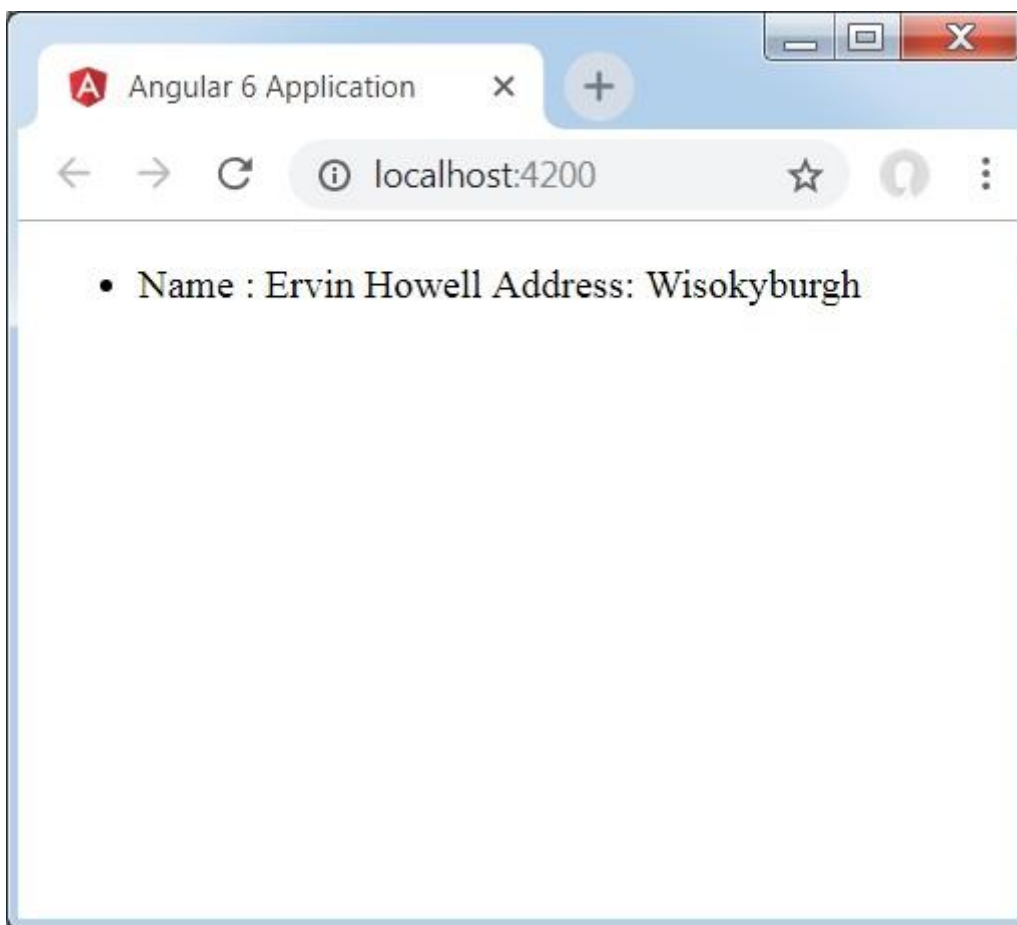
```

export class AppComponent {
  constructor(private http: HttpClient) { }
  httpdata;
  name;
  searchparam = 2;
  ngOnInit() {
    this.http.get("http://jsonplaceholder.typicode.com/users?id="+this.searchparam)
      .subscribe((data) => this.displaydata(data));
  }
  displaydata(data) { this.httpdata = data; }
}

```

For the **get api**, we will add the search param `id = this.searchparam`. The searchparam is equal to 2. We need the details of **id = 2** from the json file.

This is how the browser is displayed –



We have consoled the data in the browser, which is received from the http. The same is displayed in the browser console. The name from the json with **id = 2** is displayed in the browser.

Example4:-

Creating Angular Project Use below Commands

- `npm install -g @angular/cli //creating cli`
- `ng version`
- `ng new prog9 //creating angular project SPA(Single page application) //app component is a default component.`

- cd prog9
- ng g c header //creating header component
- ng g c home //creating home component
- ng g c profile //creating profile component
- ng serve //running angular project

Step1: header.component.css

```
ul li{  
  list-style: none;  
}  
ul li a{  
  text-decoration: none;  
}  
ul{  
  background-color: aqua;  
  height: 50px;  
}  
a{  
  line-height:50px ;  
  font-weight: bold;  
  font-size:20px;  
}
```

Step 2:- Create navbar in header.component.html

```
<ul>  
  <li>  
    <a routerLink="/profile">Profile</a>  
  </li>  
</ul>
```

Step 3:- Create navbar in home.component.html

```
<h1>Welcome to Home Page</h1>
```

Step 4:-Configure route links in **app.module.ts**

```
import { ProfileComponent } from './profile/profile.component';
import { HeaderComponent } from './header/header.component';
import { HomeComponent } from './home/home.component';
import { RouterModule,Routes } from '@angular/router';
import { HttpClientModule } from '@angular/common/http';

const routes:Routes=[
  {
    path:"",component:HomeComponent
  },
  {
    path:'profile',component:ProfileComponent
  },
]

imports: [RouterModule.forRoot(routes),
  HttpClientModule]
```

Step 5:-Use header selector in the **app.component.html** along with <router-outlet>

```
<app-header></app-header>
<router-outlet></router-outlet>
```

Step6:Profile.component.css

```
img {
  border-radius: 50%;
}
```

Step7:Profile.component.html

```
<h1>Welcome to profile page</h1>

<br>

<button (click)="getData()">Get Profile</button>

<br>

<div *ngIf="data">

  <img [src]="ImagePath" alt="Profile">

<table>

  <tr><th>ID</th>

  <td>{{ data.id }}</td></tr>

  <tr><th>Name</th>

  <td>{{ data.name }}</td></tr>

  <tr><th>Email</th>

  <td>{{ data.email }}</td></tr>

  <tr><th>Phone</th>

  <td>{{ data.phone }}</td></tr>

</table>

</div>
```

Step8:Profile.component.ts

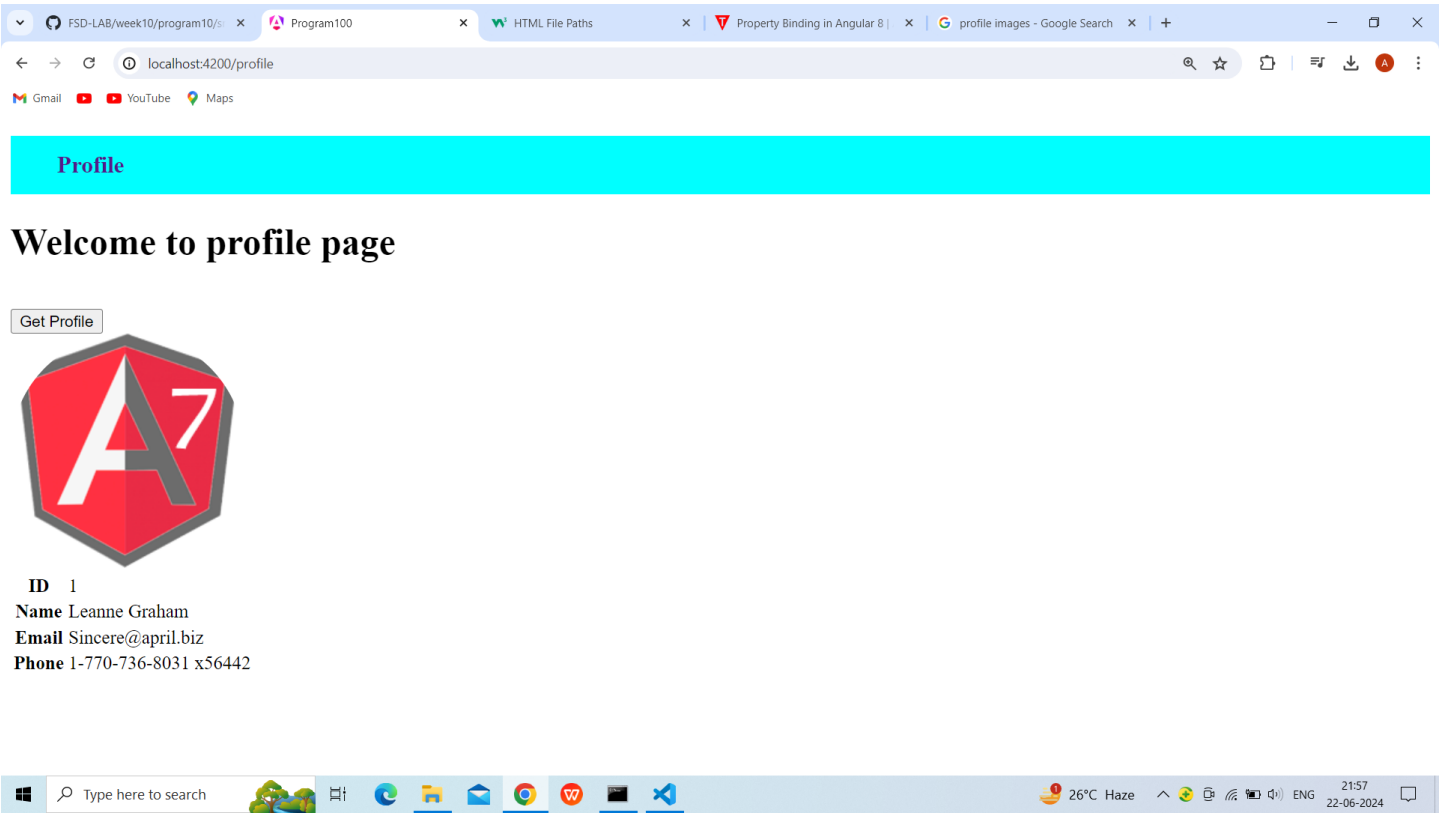
```
import { Component } from '@angular/core';
import { HttpClient } from '@angular/common/http';

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

ImagePath:any;

```
constructor(private http:HttpClient){  
  
  this.ImagePath = 'https://static.javatpoint.com/tutorial/angular7/images/angular-7-logo.png';  
  
}  
  
data:any;  
  
getData(){  
  
  this.http.get('https://jsonplaceholder.typicode.com/users/1')  
  
  .subscribe((data)=>{  
  
this.data=data;  
  
  })  
  
}  
  
}
```

Output:



8.Observables:-

Defination:-

Observables provide support for data sharing between publishers and subscribers in an angular application. It is referred to as a better technique for event handling, asynchronous programming, and handling multiple values as compared to techniques like promises.

A special feature of Observables is that it can only be accessed by a consumer who subscribes to it i.e A function for publishing values is defined, but it is not executed by the subscribed consumer (it can be any component) only via which the customer can receive notifications till the function runs or till they subscribed.

An observable can deliver multiple values of any type. The API for receiving values is the same in any condition and the setup and the logic are both handled by the observable. Rest thing is only about subscribing and unsubscribing the information required.

Observers: To handle receiving observable messages, we need an observable interface which consists of callback methods with respect to the messages by observables.

Usage

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.

To make use of the observable, all you need to do is to begin by creating notifications using subscribe() method, and this is done by passing observer as discussed previously. The notifications are generally Javascript objects that handle all the received notifications. Also, the unsubscribe() method comes along with subscribing () method so that you can stop receiving notifications at any point in time.

Types of Notifications and Description

1. **next:** It is called after the execution starts for zero times or more than that. It is a mandatory notification for catching each value delivered.
2. **error:** It is a handler for each error message. An error stops execution of the observable instance.
3. **complete:** It is a handles in which the completion of observable execution is notified.

Before using Observables do import Observables from rxjs library by writing the following code.

```
import {Observables} from 'rxjs'
```

Error Handling:

Observables produce asynchronous values and thus try/catch do not catch any errors because it may lead to stop the code irrespective of other tasks running at that instance of time. Instead, we handle errors by specifying an error callback on the observer. When an error is produced, it causes the observable to clean up subscriptions and stop producing values for that subscription. An observable can either produce values (calling the next callback), or it can complete, calling either the complete or error callback.

The syntax for error callback

```
observable.subscribe({  
  next(val) { console.log('Next: ' + val)},  
  error(err) { console.log('Error: ' + err)}  
});
```


Example:-

app.component.html:-

```
<button (click)="test()">get</button>
<button (click)="lose()">lose</button>
```

app.component.ts:-

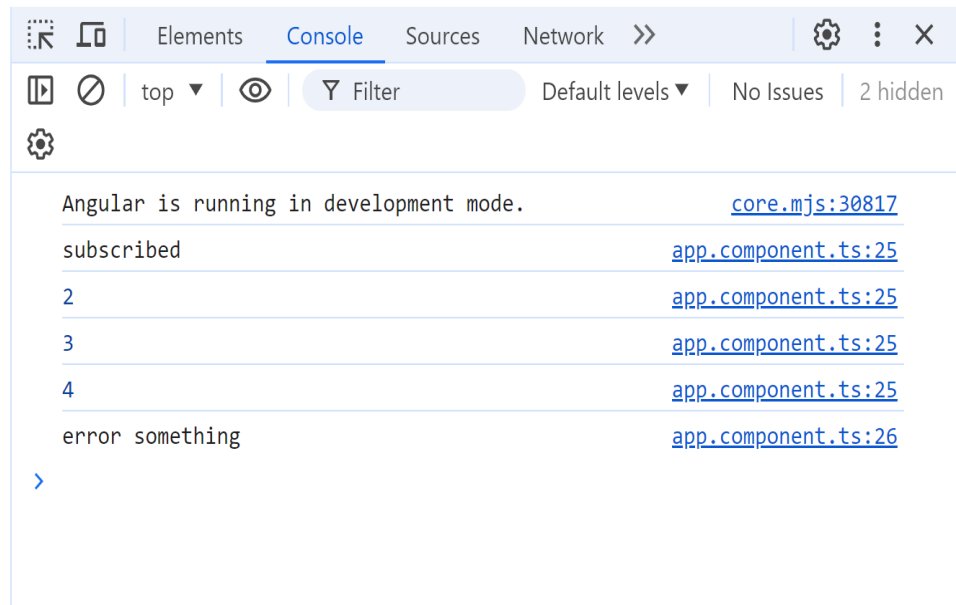
```
import { Component } from '@angular/core';
import { Observable } from 'rxjs';
@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent {

  myobs=new Observable(
    (listener)=>{
      listener.next("subscribed");
      listener.next(2);
      setTimeout(()=>listener.next(3),1000);
      setTimeout(()=>listener.next(4),1000);
      setTimeout(()=>listener.error("error something"),1000);
      setTimeout(()=>listener.next(6),1000);
      //setTimeout(()=>listener.complete(),1000);
    }
  )
  aaa:any;
  test(){
    this.aaa=this.myobs.subscribe(
      (data)=>{ console.log(data)},
      err=>{ console.log(err)},
      ()=>{ console.log("completed")}
    )
  }
  lose(){
    this.aaa.unsubscribe();
  }

}
```

Output:-

get lose



More Information:-

Observables in Angular?

We use Observable to perform asynchronous operations and handle asynchronous data. Another way of handling asynchronous is using promises. We can handle asynchronous operations using either Promises or Observables.

What are asynchronous operations and asynchronous data?

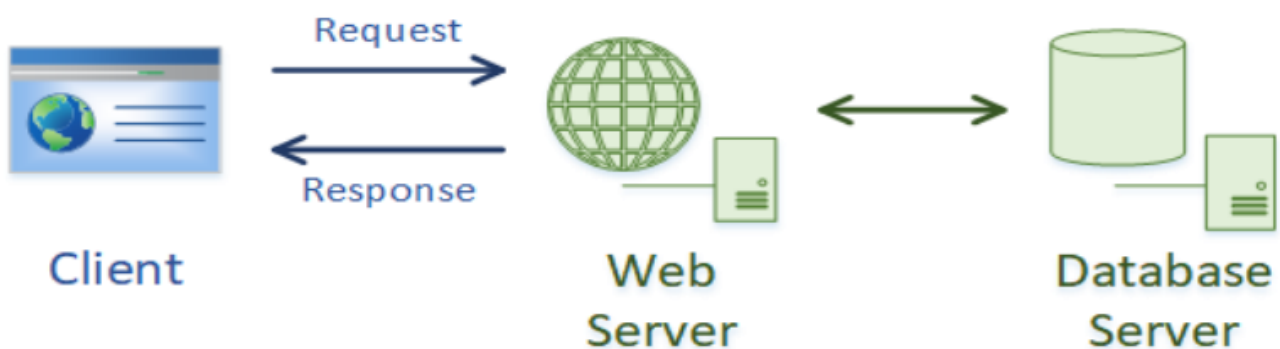
We already know that JavaScript is a single-threaded language. That means the code is executed line by line and once the execution of one code is complete then only the next code of the program will be executed. When we make a request to the HTTP server that will take more time. So the next statement after the HTTP request has to wait for the execution. It will only get executed when the HTTP request completes. We can say that the synchronized code is blocked in nature.

This is the way the asynchronous programs came into the picture. Asynchronous code executing in the background without blocking the execution of the code in the main thread. Asynchronous code is non-blocking. That means we can make HTTP requests asynchronously.

Using an asynchronous program we can perform long network requests without blocking the main thread. There are 2 ways in which we can do that.

- ☐ Using Observables
- ☐ Using Promises

What is the difference between Promises and Observables?



Let's say we are requesting a list of users from the server. From the browser, we are sending a request to the server and the server will get the data from the database. Let's say the data which we are requesting is huge. In that case, the server will get some time to get the data from the database.

Once the data is ready the data will send from the server to the client-side. Here server gathered all the data and when the data is ready that will send back to the client-side. This is how gets the Promise work. It promises to provide data over a period of time. Promise provides us the data once the complete data is ready. The data can be the actual data that we requested or it can also be an error. If there is no internet connection. In that case, also promises to return some data. That data will be the error message or an error object.

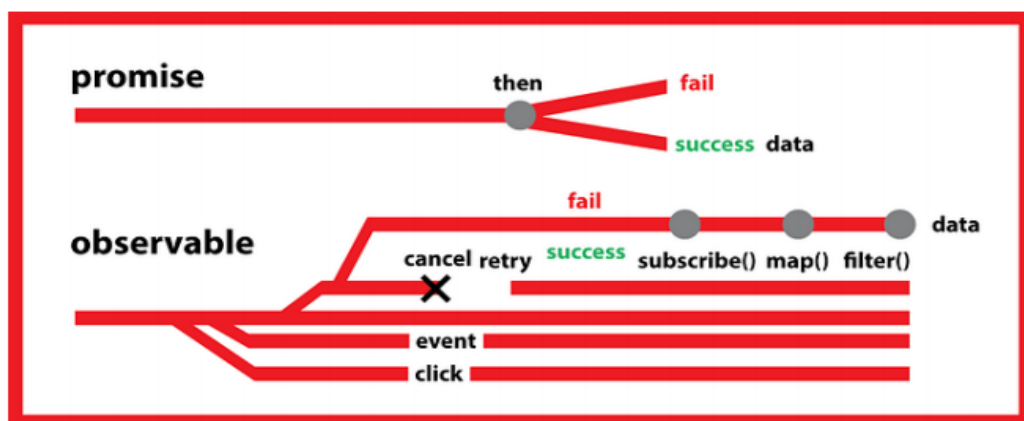
Observables are not waiting for the complete data to be available. An Observable streams the data. When the data is available partially it will send to the client.

Promises

1. Helps you run functions asynchronously, and use their return values (or exceptions), but only once when executed.
2. Not lazy.
3. Not cancellable (there are Promise libraries out there that support cancellation, but ES6 Promise doesn't so far). The two possible decisions are Reject and Resolve.
4. Cannot be retried (Promises should have access to the original function that returned the promise to have a retry capability, which is a bad practice)
5. Provided by JavaScript language.

Observables

1. Helps you run functions asynchronously, and use their return values in a continuous sequence (multiple times) when executed.
2. By default, it is lazy as it emits values when time progresses.
3. Has a lot of operators which simplifies the coding effort.
4. One operator retry can be used to retry whenever needed, also if we need to retry the observable based on some conditions retryWhen can be used.
5. Not a native feature of Angular or JavaScript. Provide by another JavaScript library which is called Rxjs.



An Observable is a function that converts the ordinary stream of data into an Observable stream of data. You can think of Observable as a wrapper around the Ordinary stream of data.

9. Routing

The **ngRoute** module helps your application to become a Single Page Application.

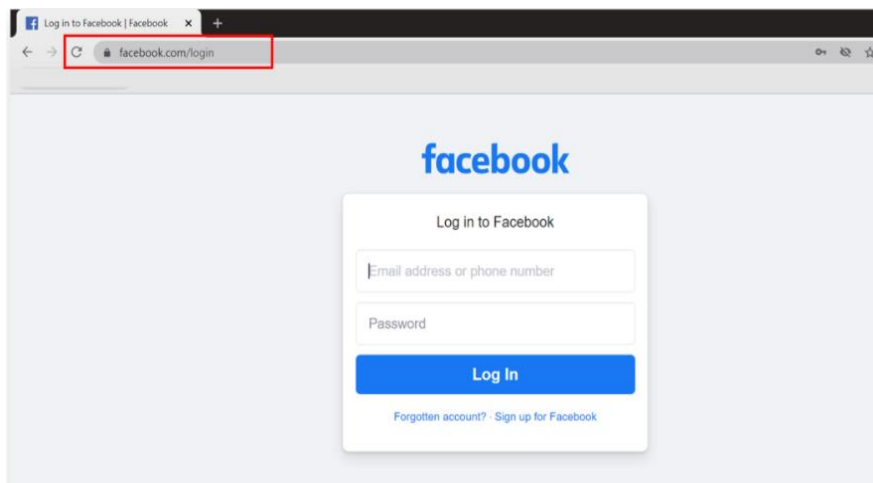
What is Routing in AngularJS?

If you want to navigate to different pages in your application, but you also want the application to be a SPA (Single Page Application), with no page reloading, you can use the **ngRoute** module.

The **ngRoute** module *routes* your application to different pages without reloading the entire application.

In Angular, routing plays a vital role since it is essentially used to create Single Page Applications. These applications are loaded just once, and new content is added dynamically. Applications like Google, Facebook, Twitter, and Gmail are a few examples of SPA. The best advantage of SPA is that they provide an excellent user experience and you don't have to wait for pages to load, and by extension, this makes the SPA fast and gives a desktop-like feel.

It generally specifies navigation with a forward slash followed by the path defining the new content.



Example:-

Note:-Install Node Js Software and Visual Studio.

Creating Angular Project Use below Commands

- `npm install -g @angular/cli //creating cli`
- `ng version`
- `ng new prog9 --standalone false //creating angular project SPA(Single page application)`
`//app component is a default component.`
- `cd prog9`
- `ng g c header //creating header component`
- `ng g c home //creating home component`
- `ng g c profile //creating profile component`
- `ng serve //running angular project`

Creating body of about, contact, home and header. Header is a navbar this page creating routerLinks about and contact. Home is a default link when header loaded it is displayed. App.module.ts file creating url paths for each page.

1)about.component.html:-

```
<h1>This is About Component</h1>
```

2)contact.component.html:-

```
<h1>This is Contact Component</h1>
```

3)header.component.css:-

```
ul li{  
    list-style: none;  
}  
ul li a{  
    text-decoration: none;  
}  
ul{  
    display: flex;  
    justify-content: flex-start;  
    gap: 20px;  
    background-color: aqua;  
    height: 50px;  
}  
a{  
    line-height:50px;  
    color:black;  
    margin:0 20px;  
    font-weight: bold;  
    font-size:30px;  
}
```

4)header.component.html:-

```
<ul>
  <li>
    <a routerLink="/about" router="active">about</a>
  </li>
  <li>
    <a routerLink="/contact" routerActiveLink="active">contact</a>
  </li>
</ul>
```

5) home.component.html:-

```
<h1>This is Home Component</h1>
```

6) **notfound.component.html:-**

```
<p>notfound works!</p>
```

7) **app.component.html:-**

```
<app-header></app-header>
<router-outlet></router-outlet>
```

8) **app.module.ts:-**

```
import { AppComponent } from './app.component';
import { HeaderComponent } from './header/header.component';
import { AboutComponent } from './about/about.component';
import { ContactComponent } from './contact/contact.component';
import { HomeComponent } from './home/home.component';
import { NotfoundComponent } from './notfound/notfound.component';
import { RouterModule,Routes } from '@angular/router';
```

```
const routes:Routes=[
  {
    path:"",component:HomeComponent
```

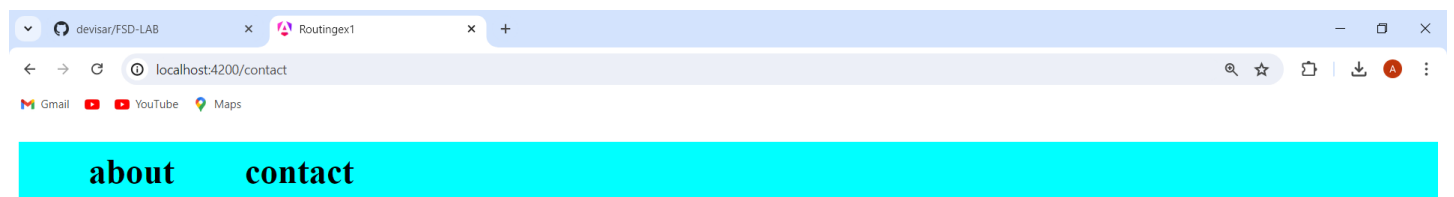
```
},  
{  
  path:'about',component:AboutComponent  
},  
{  
  path:'contact',component:ContactComponent  
},  
{  
  path:'**',component:NotFoundComponent  
}  
]
```

imports: [

RouterModule.forRoot(routes)

],

Output:



This is Contact Component

