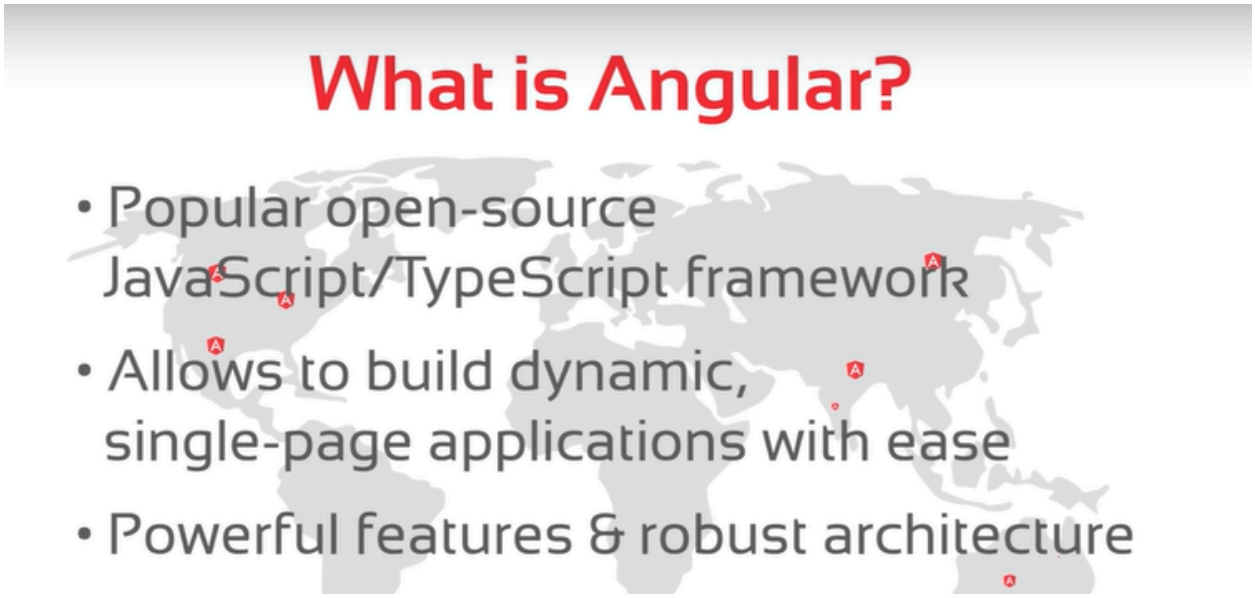



**What does open-source framework mean, and why is Angular said to be an open-source framework?**

An open-source framework is software with publicly available source code that can be used, modified, and distributed by anyone.

Angular as an Open-Source Framework:
1. Source Code Availability: Angular's source code is on GitHub.
2. Licensing: Released under the permissive MIT License.
3. Community Contributions: Developers worldwide contribute to its improvement.
4. Transparent Development: Development is done publicly, allowing anyone to follow and contribute.

### **Benefits**:
- Collaborative Improvement: Enhanced by a global community.
- Rapid Development: New features are developed quickly.
- Customization: Can be tailored to specific project needs.
- Community Resources: Extensive support and resources available.

**What Does Single Page Application (SPA) Mean?**

A Single Page Application (SPA) is a web application that interacts with the user by dynamically rewriting the current page rather than loading entire new pages from a server. This provides a more fluid user experience similar to that of a desktop application. SPAs typically load a single HTML page and dynamically update content as the user interacts with the app.

**How Angular Allows Building Dynamic and Single Page Applications?**

Angular is a powerful framework for building SPAs due to the following features:

1. **Two-Way Data Binding**: Synchronizes data between the model and the view, ensuring that changes in the application state automatically update the user interface and vice versa.

    **### What is Two-Way Data Binding?**

    Two-way data binding synchronizes the data model and the user interface (UI), so changes in one automatically update the other. This ensures the view and the model are always in sync.

    **### How Angular Implements Two-Way Data Binding: {{ }}**

    Angular uses the `ngModel` directive for two-way data binding.

    #### **HTML**:
    <input [(ngModel)]="name" placeholder="Enter your name">
    Hello, {{name}}!

#### **TypeScript**:

```
export class AppComponent {
  name: string = '';
}
```

### Explanation:

- [(ngModel)]: Combines **property binding** (`[]` for model to view) and **event binding** (`()` for view to model) to achieve two-way data binding.
- name: Updates to the input field change the `name` property, and changes to the `name` property update the input field.

**### Benefits:**
- Consistency: Keeps UI and data model consistent.
- Efficiency: Automatically updates UI and data model.

Two-way data binding in Angular makes developing interactive applications easier by keeping the UI and data model in sync.

2. **Component-Based Architecture**: Angular applications are built using components, which are modular, reusable pieces of code that manage their own view and logic.

3. **Client-Side Routing**: Angular's Router module enables navigation between different views or pages within the application without reloading the entire page.

4. **Dependency Injection**: Angular's built-in dependency injection system facilitates the management of service instances, making it easier to develop, test, and maintain applications.

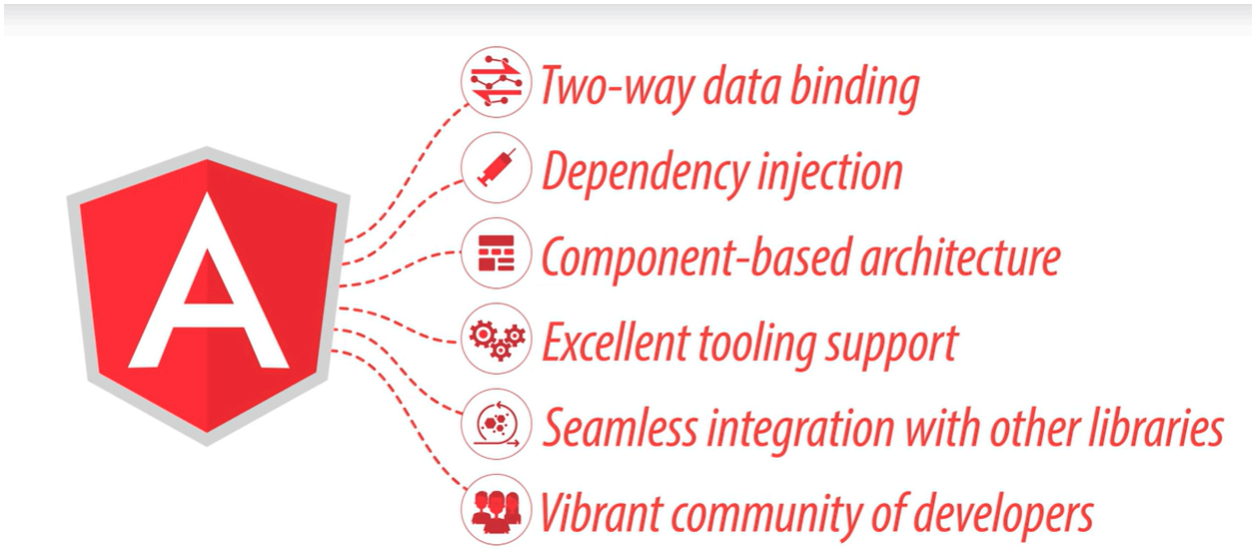
5. **Templates**: Angular uses declarative templates to bind dynamic data to HTML elements, allowing developers to create interactive and dynamic user interfaces efficiently.

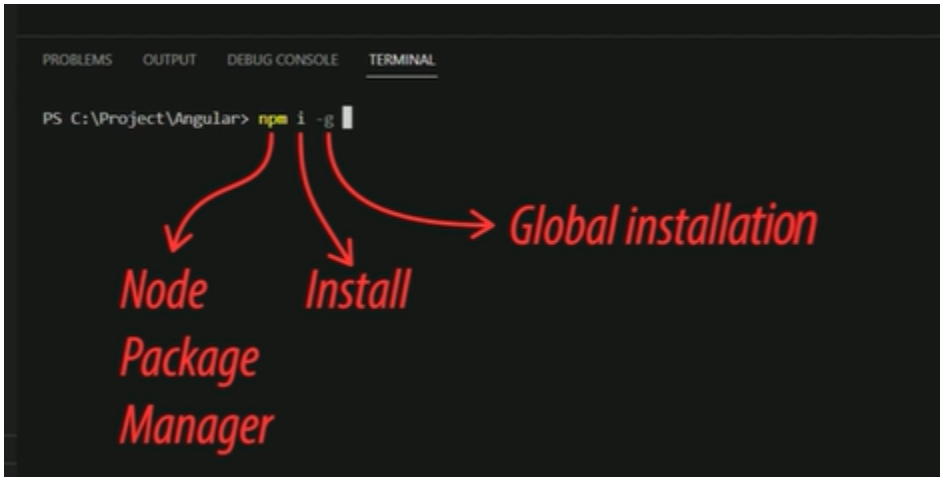
6. **Directives**: Angular directives extend HTML by adding custom behavior to elements. Built-in directives like `ngIf`, `ngFor`, and custom directives help in creating a dynamic user interface.

7. **Services and HTTP Client**: Angular provides a robust way to handle HTTP requests and services, enabling the app to fetch and send data asynchronously, which is crucial for dynamic and responsive SPAs.

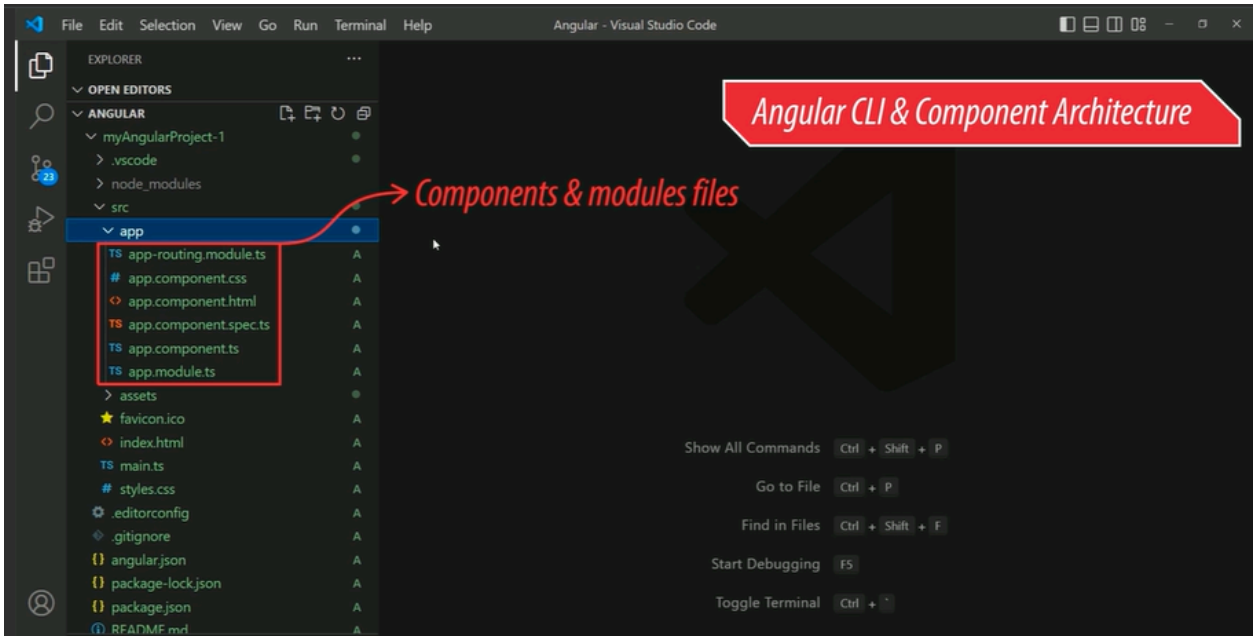
### **Summary**

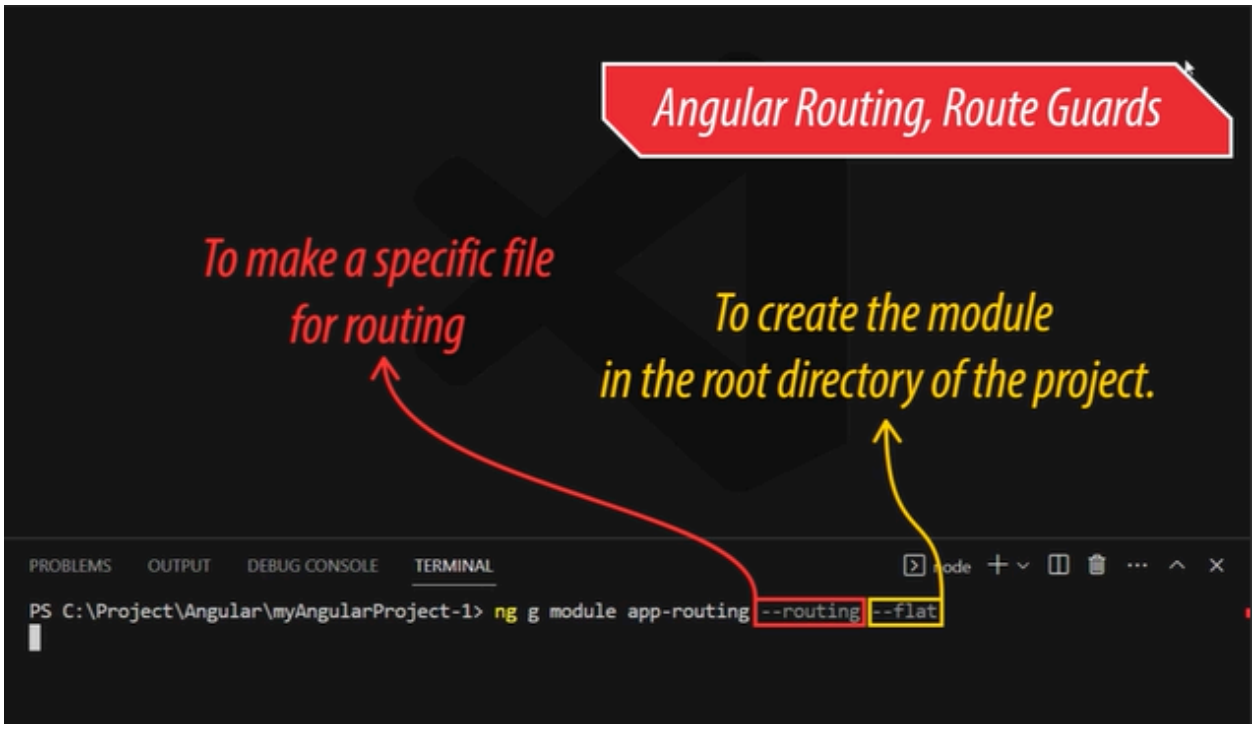
Single Page Applications (SPAs) offer a more dynamic and seamless user experience by updating the current page rather than loading new ones.

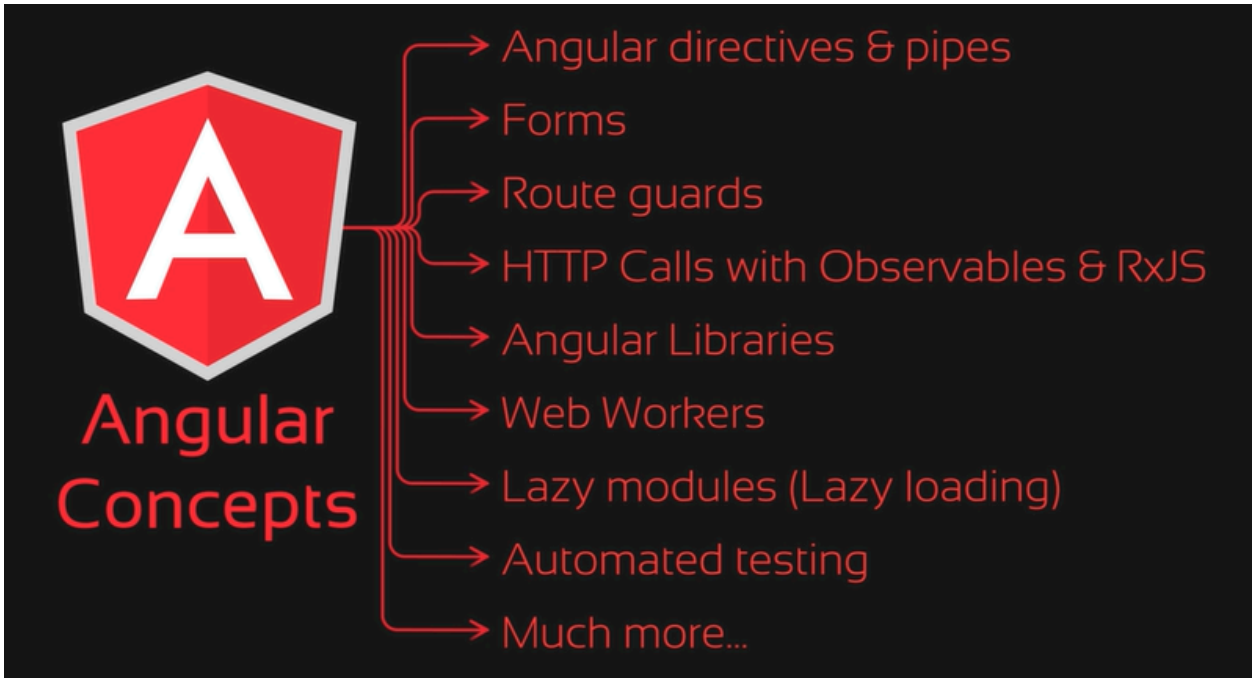

Two-way data binding
Dependency injection
Component-based architecture
Excellent tooling support
Seamless integration with other libraries
Vibrant community of developers


PROBLEMS
OUTPUT
DEBUG CONSOLE
TERMINAL
PS C:\Project\Angular> npm i -g
Global installation
Node
Package
Manager
Install


PROBLEMS
OUTPUT
DEBUG CONSOLE
TERMINAL
PS C:\Project\Angular> npm i -g @angular/cli


File Edit Selection View Go Run Terminal Help
Angular - Visual Studio Code
EXPLORER
OPEN EDITORS
ANGULAR
myAngularProject-1
.vscode
node_modules
src
app
app-routing.module.ts
app.component.css
app.component.html
app.component.spec.ts
app.component.ts
app.module.ts
assets
favicon.ico
index.html
main.ts
styles.css
.editorconfig
.gitignore
angular.json
package-lock.json
package.json
README.md
Angular CLI & Component Architecture
Components & modules files
Show All Commands Ctrl + Shift + P
Go to File Ctrl + P
Find in Files Ctrl + Shift + F
Start Debugging F5
Toggle Terminal Ctrl + `


Angular Routing, Route Guards
To make a specific file for routing
To create the module in the root directory of the project.
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
node
PS C:\Project\Angular\myAngularProject-1> ng g module app-routing --routing --flat


Angular directives & pipes
Forms
Route guards
HTTP Calls with Observables & RxJS
Angular Libraries
Web Workers
Lazy modules (Lazy loading)
Automated testing
Much more...
Angular
Concepts


Header Component
Adding search bar
Category dropdown
Links & cart icons
Components creation
Error 404 page
Styling product card
Creating database
Configuring API Route
Configuring Nodemon
Creating workspace
Data fetching with SQL query
Fetching products in component
Adding keywords in database
Filtering products
Products details component
Shopping cart component
Cart summary & session storage
eStore Project

**Angular CLI** is a command-line interface tool which is used to *initialize, develop, scaffold* and *maintain Angular applications* directly from the command shell.

## Angular CLI

Creating an Angular project

Running
the development server

Deploying
the application

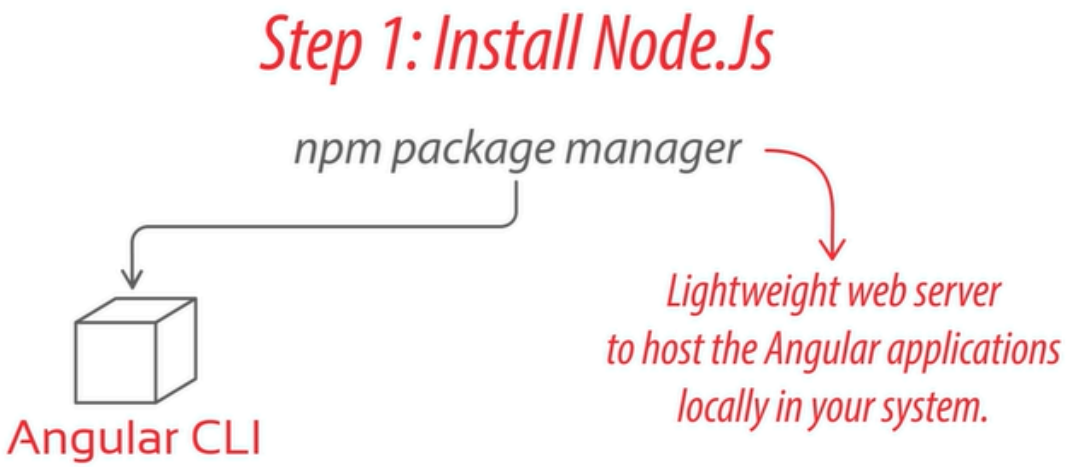
Secure
binding of data

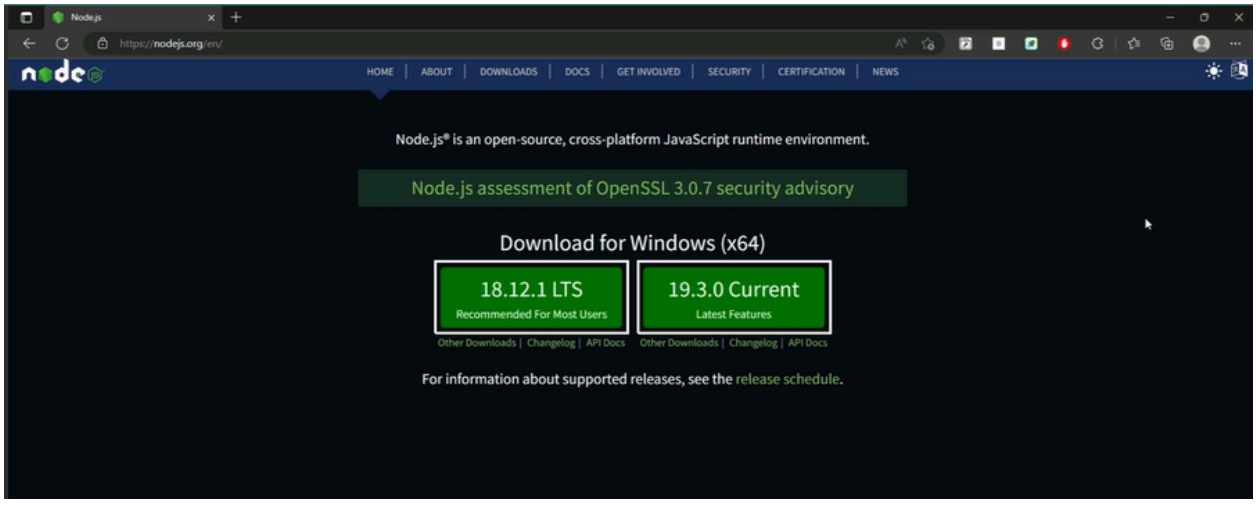
Instant support
& good data security

Add
additional features

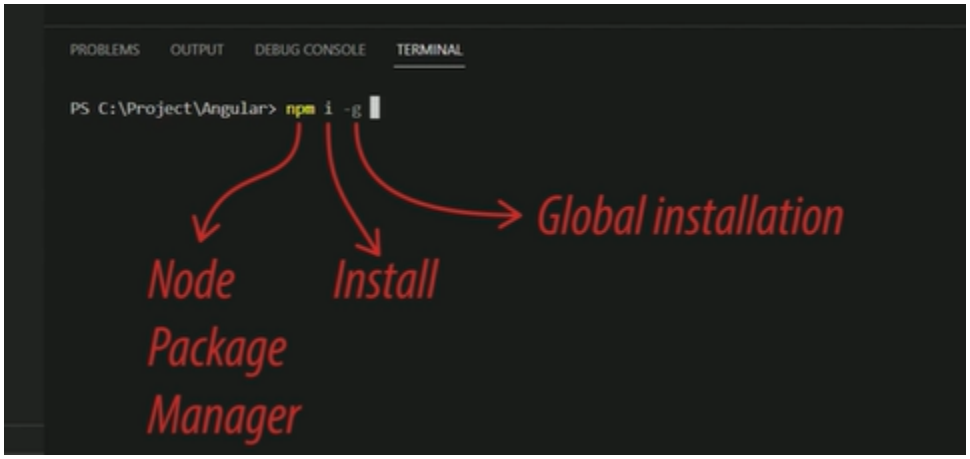
Test
the Angular application

# How to set up the Angular CLI in your project ?


**Install Nodejs**

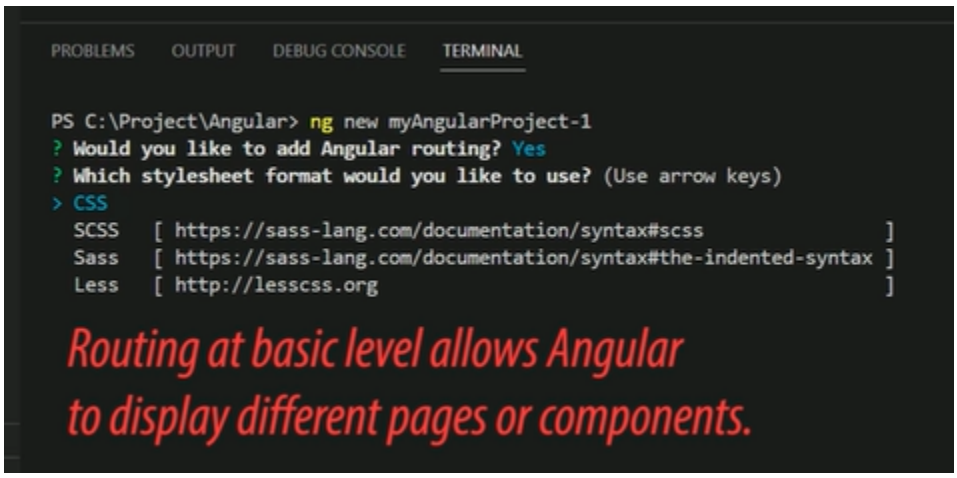



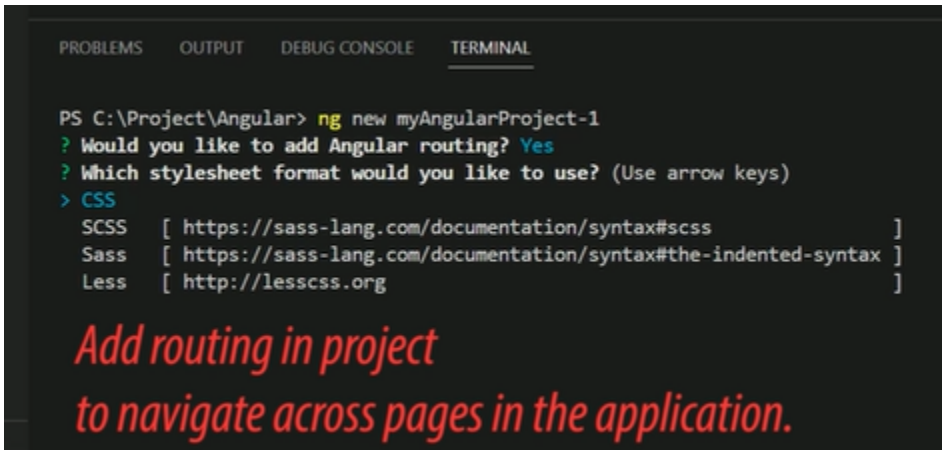

**Why we need the global installation?**

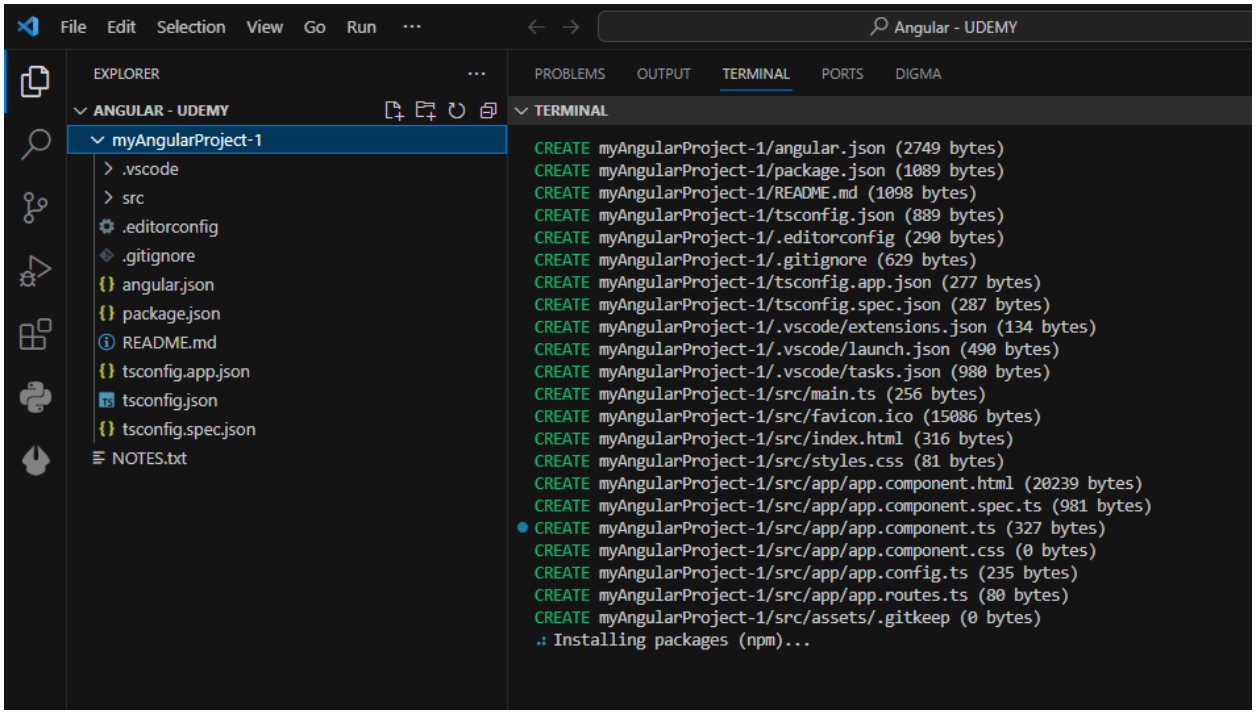
CLI we need not noly for this folder but also for the whole system

**npm i -g @angular/cli**


PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
PS C:\Project\Angular> ng version
Angular CLI: 15.0.4
Node: 18.12.1
Package Manager: npm 9.2.0
OS: win32 x64
Angular:
...
Package Version
@angular-devkit/architect 0.1500.4 (cli-only)
@angular-devkit/core 15.0.4 (cli-only)
@angular-devkit/schematics 15.0.4 (cli-only)
@schematics/angular 15.0.4 (cli-only)
PS C:\Project\Angular>

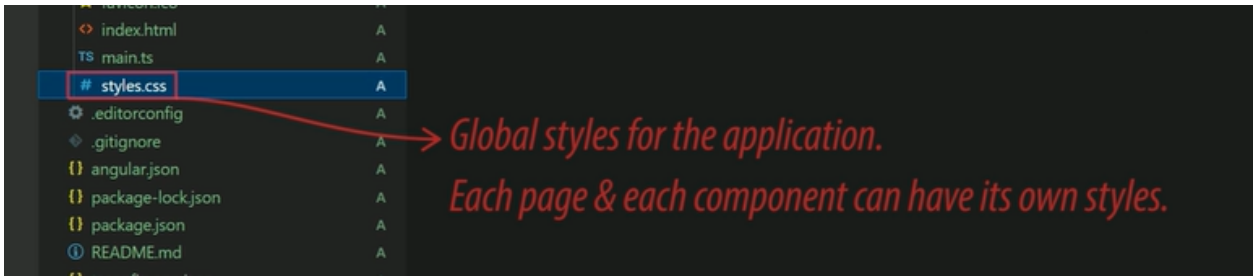

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
PS C:\Project\Angular> ng new myAngularProject-1
? Would you like to add Angular routing? Yes
? Which stylesheet format would you like to use? (Use arrow keys)
> CSS
SCSS [ https://sass-lang.com/documentation/syntax#scss ]
Sass [ https://sass-lang.com/documentation/syntax#the-indented-syntax ]
Less [ http://lesscss.org ]
Routing at basic level allows Angular
to display different pages or components.

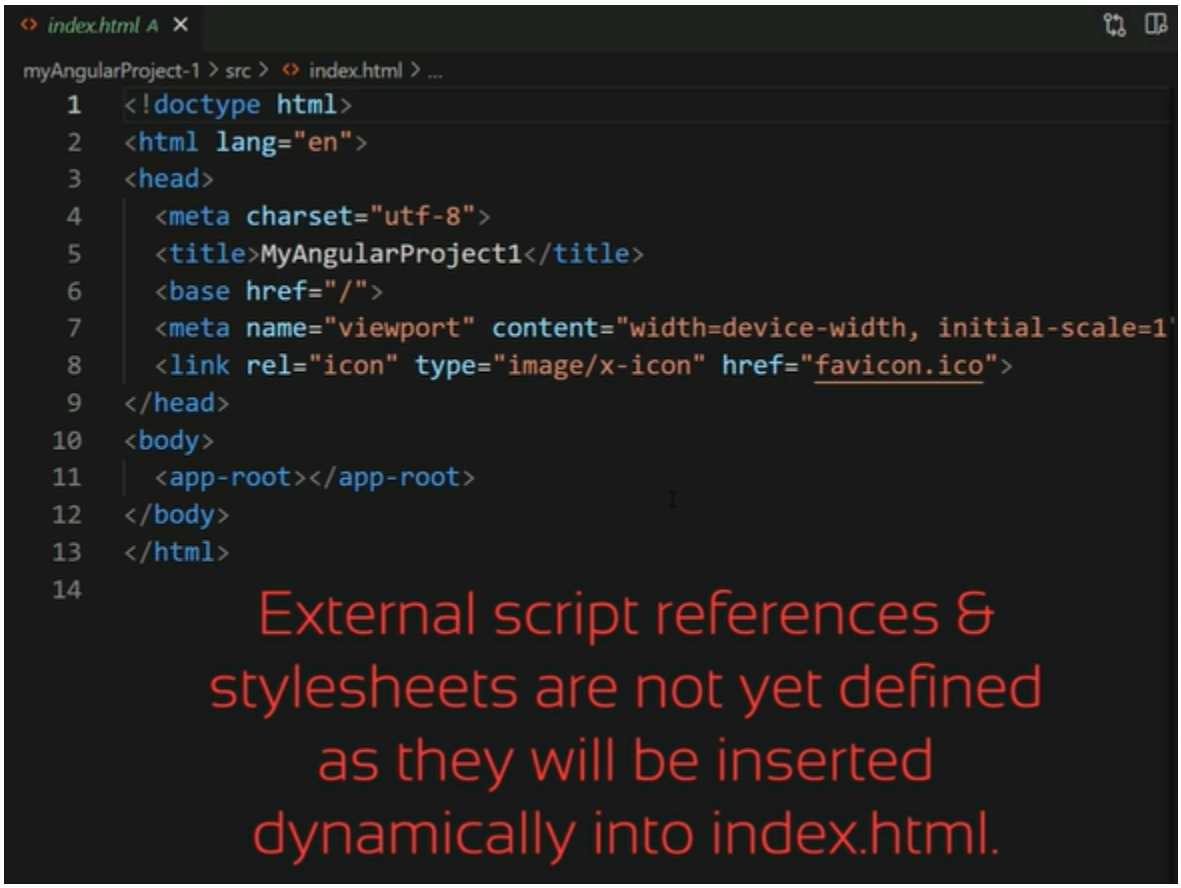

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
PS C:\Project\Angular> ng new myAngularProject-1
? Would you like to add Angular routing? Yes
? Which stylesheet format would you like to use? (Use arrow keys)
> CSS
SCSS [ https://sass-lang.com/documentation/syntax#scss ]
Sass [ https://sass-lang.com/documentation/syntax#the-indented-syntax ]
Less [ http://lesscss.org ]
Add routing in project
to navigate across pages in the application.

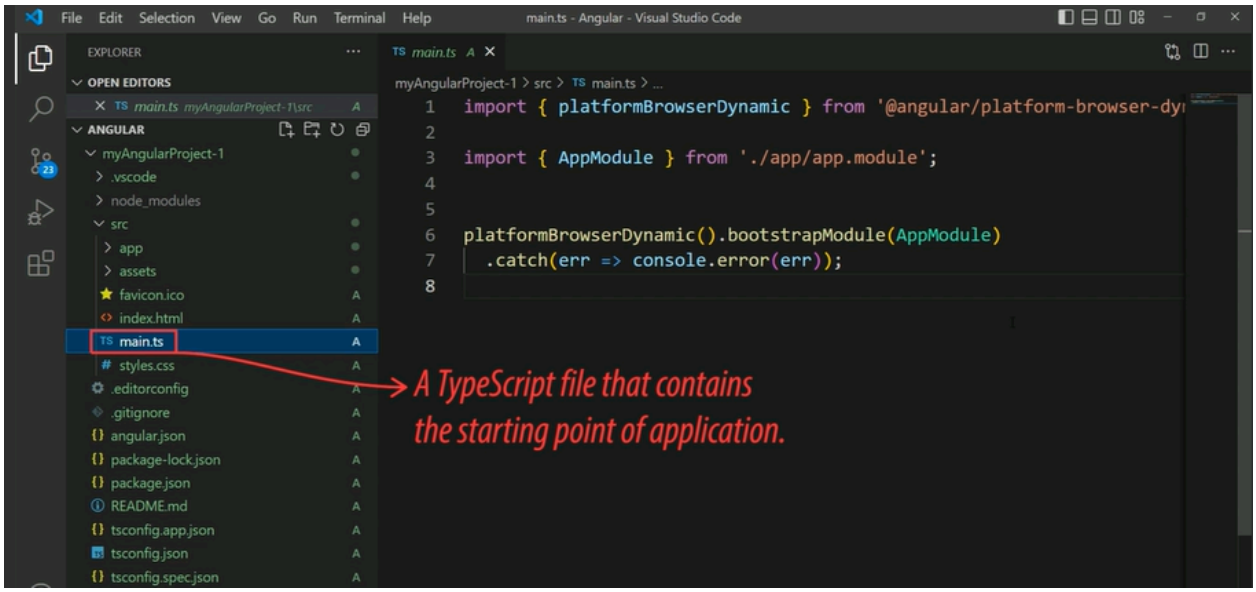



## Structure of Angular Project


myAngularProject-1
.vscode
node_modules
src
app
assets
favicon.ico
index.html
main.ts
styles.css
.editorconfig
.gitignore
angular.json
Default icon displayed in the browser when the Angular application loads.
amazon
Search Amazon
All
Today's Deals
Customer Service
Registry
Gift Cards
Sell


index.html
main.ts
styles.css
.editorconfig
.gitignore
angular.json
package-lock.json
package.json
README.md
Global styles for the application.
Each page & each component can have its own styles.

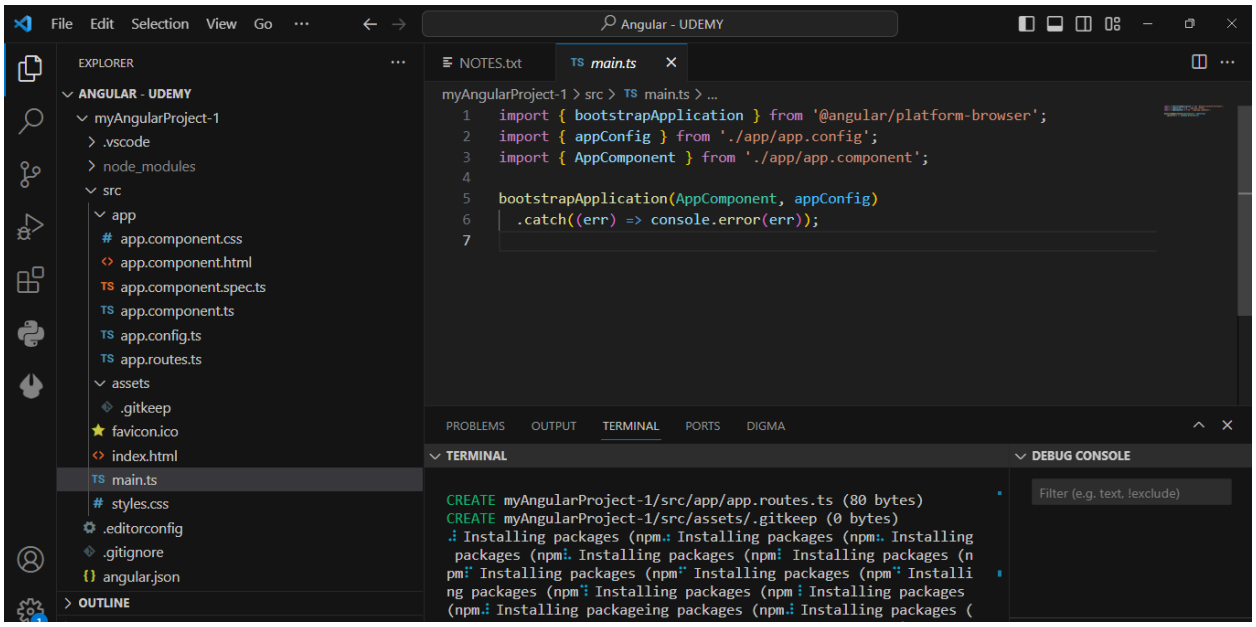

index.html
myAngularProject-1 > src > index.html > ...
<!doctype html>
<html lang="en">
<head>
<meta charset="utf-8">
<title>MyAngularProject1</title>
<base href="/">
<meta name="viewport" content="width=device-width, initial-scale=1"
<link rel="icon" type="image/x-icon" href="favicon.ico">
</head>
<body>
<app-root></app-root>
</body>
</html>
External script references & stylesheets are not yet defined as they will be inserted dynamically into index.html.




Angular bootstrapping this AppModule first - **in previous versions**

## In Angular 17

No AppModule - AppComponent with AppConfig




**Bootstrapping in Angular: Then vs. Now**

**Before Angular 17:**

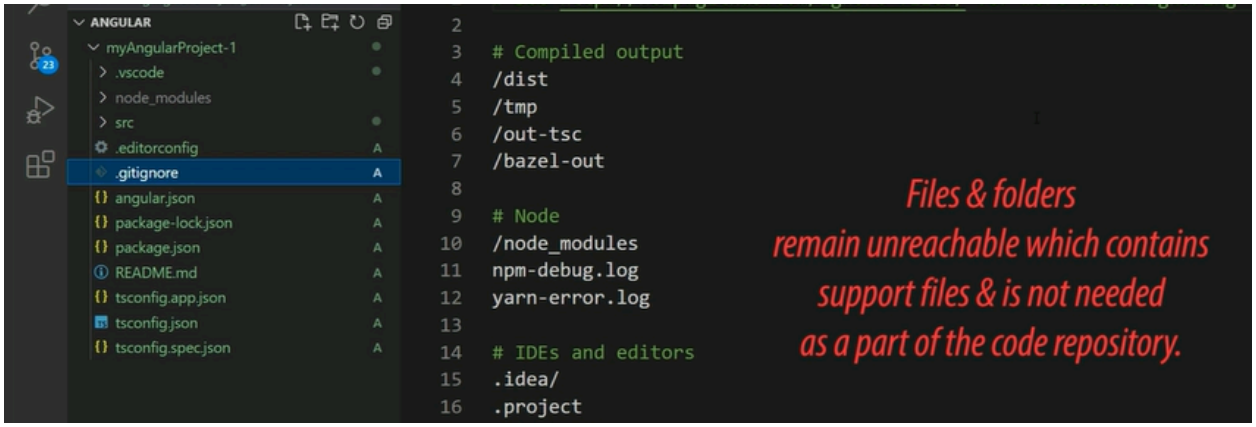
**AppModule: The king, loaded first.**

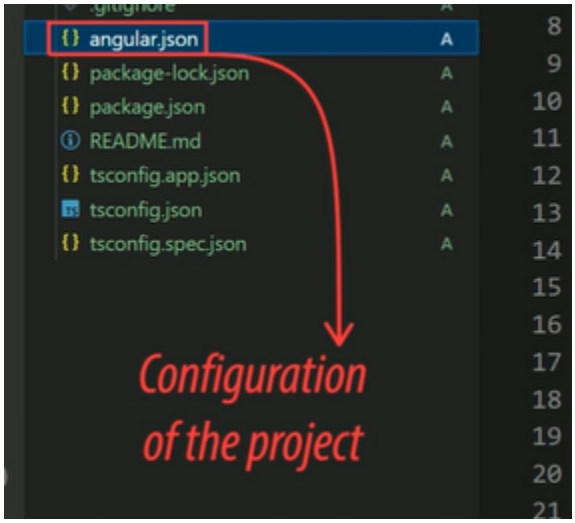
**Bootstrapping:** `platformBrowserDynamic().bootstrapModule(AppModule)`.
**AppComponent: Bootstrapped by AppModule.**

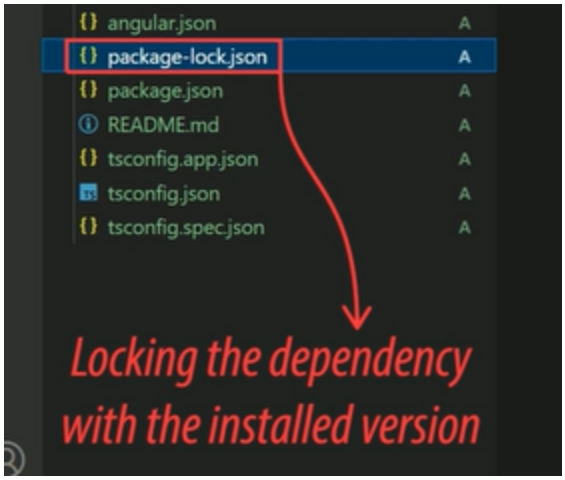
**Angular 17 (Standalone Components):**

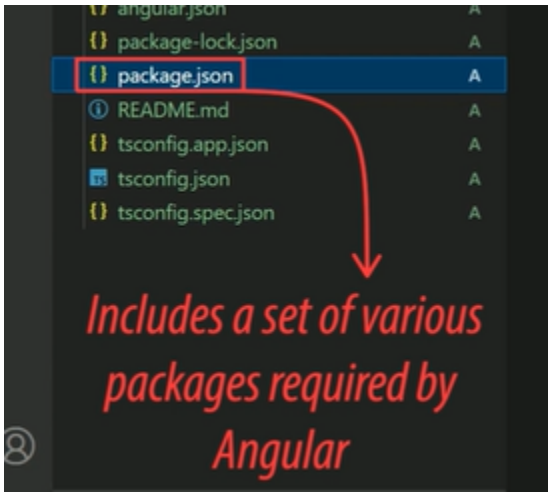
**AppModule: Gone (for standalone components).**
**Bootstrapping:** `bootstrapApplication(AppComponent, appConfig)`.
**AppComponent: Bootstrapped directly.**

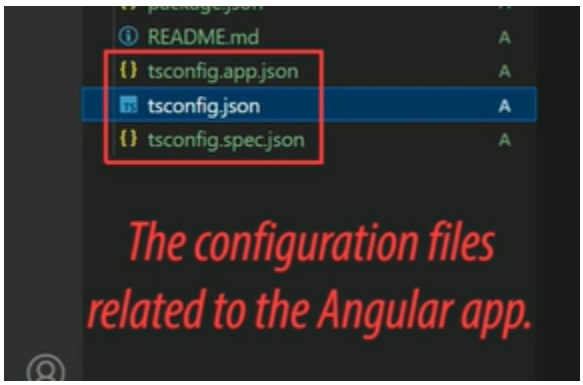
**Key Difference: Standalone components ditch AppModule, use** `bootstrapApplication`.



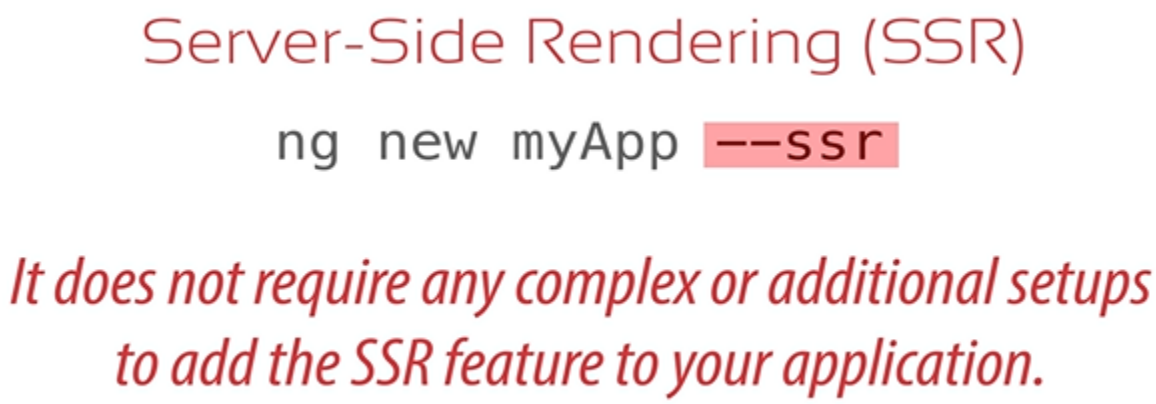




angular.json A
package-lock.json A
package.json A
README.md A
tsconfig.app.json A
tsconfig.json A
tsconfig.spec.json A
Locking the dependency with the installed version


package-lock.json A
package.json A
README.md A
tsconfig.app.json A
tsconfig.json A
tsconfig.spec.json A
Includes a set of various packages required by Angular


README.md A
tsconfig.app.json A
tsconfig.json A
tsconfig.spec.json A
The configuration files related to the Angular app.


**Server-Side Rendering (SSR) in Angular 17**




**What is SSR?**

Server-side rendering (SSR) is a technique where the initial HTML content of a web application is generated on the server instead of the client (browser). This improves:

**SEO (Search Engine Optimization)**: Search engines can easily crawl and index your content.

**Initial Load Time**: Users see a full page immediately, enhancing perceived performance.

**Angular 17 and SSR:**

Angular 17 leverages Angular Universal, an official library, to enable SSR. Here's the process:

Server-Side Rendering:

- The server receives an HTTP request for a URL.
- Angular Universal renders the application on the server, including components, data, and routing.
- The server sends the fully-rendered HTML to the browser.

Client-Side Hydration:

- The browser receives the HTML and starts bootstrapping the Angular application.
- Angular Universal rehydrates the application, attaching event listeners and making it interactive.

Benefits of SSR in Angular 17:

Improved SEO
Faster initial load times
Better user experience, especially for slow connections

Concise Summary:

SSR in Angular 17, powered by Angular Universal, renders pages on the server, providing SEO benefits and faster initial loads.

## Directive Syntax

```
//Before - *ngIf
<div *ngIf="loggedIn; else randomUser">
   The user is logged in
</div>
<ng-template #randomUser>
   The user is not logged in
</ng-template>


//After Angular 17
@if (loggedIn) {
   The user is logged in
} @else {
   The user is not logged in
}
```

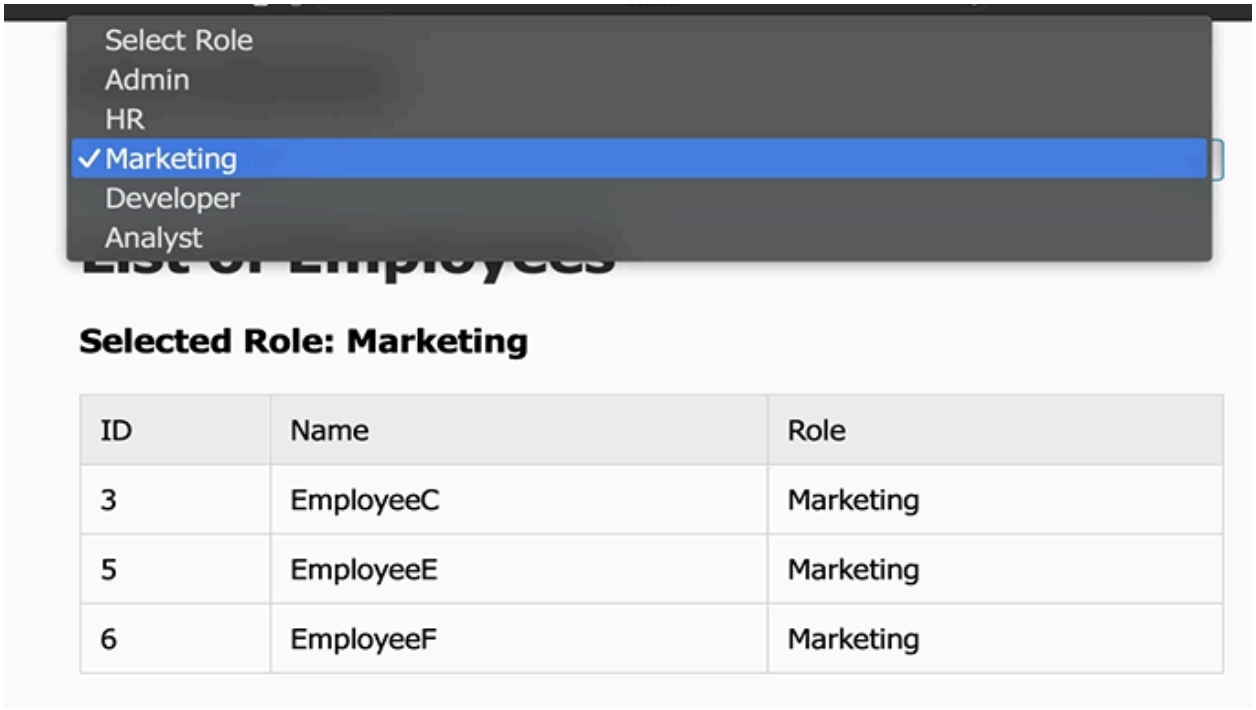
```
//Before - *ngFor
<ng-container *ngFor="let user of users; trackBy: trackById">
   {{ user.name }}
</ng-container>
<ng-container *ngIf="users.length === 0">
   Empty list of users
</ng-container>


//After Angular 17
@for (user of users; track user.id) {
   <p>{{ user.name }}</p>
} @empty {
   <p>Empty list of users</p>
}
```

```
//Before - *ngSwitch
<div [ngSwitch]="accessLevel">
   <admin *ngSwitchCase="admin"/>
   <moderator *ngSwitchCase="moderator"/>
   <user *ngSwitchDefault/>
</div>



//After Angular 17
@switch (accessLevel) {
   @case ('admin') { <admin/> }
   @case ('moderator') { <moderator/> }
   @default { <user/> }
}
```

```
//After Angular 17

@if (loggedIn) {
   The user is logged in
} @else {
   The user is not logged in
}
@for (user of users; track user.id) {
   <p>{{ user.name }}</p>
} @empty {
   <p>Empty list of users</p>
}
@switch (accessLevel) {
   @case ('admin') { <admin/> }
   @case ('moderator') { <moderator/> }
   @default { <user/> }
}
```

- *Better performance since it can be handled more efficiently behind the scenes.*
- *These syntaxes are still in the "Developer Preview" & it may change in future.*

**Use case : content based**




**Deferrable Views (Angular 17): Load Less, Load Later**

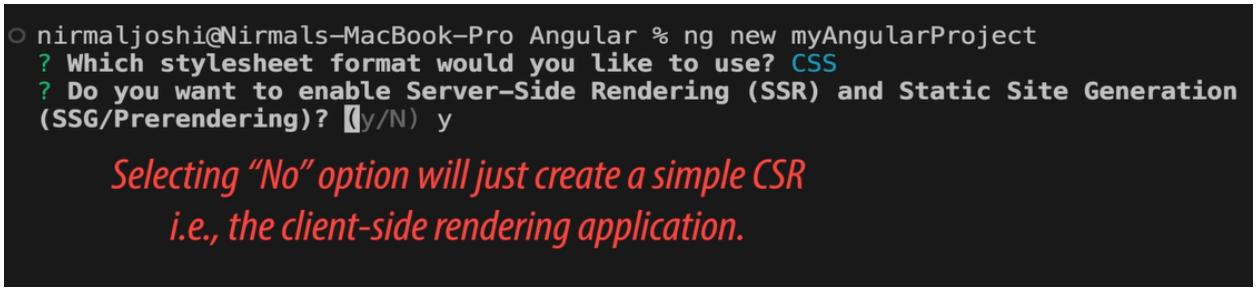
```
//Before - Angular 17
{
   path: 'users',
   loadChildren: () => import('./users/users.module'
   .then(m => m.UsersModule)
}

//After - Angular 17
<button type="button" #loadButton>Click Me</button>
   @defer(on interaction(loadButton)) {
      <app-componentName />
   }@placeholder {
      showed until the file not begin loaded
   }@loading {
      showed during the loading of the file data
   }@error {
      showed if any error happens during loading
   }
```

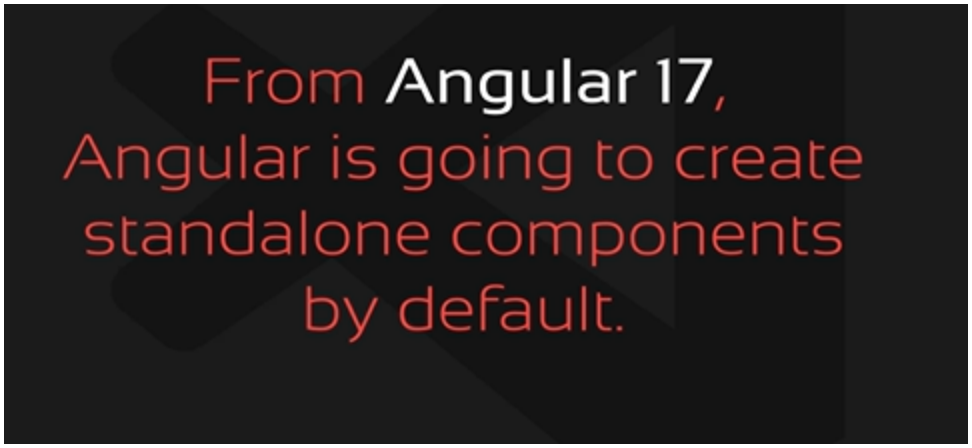
**What:** Delay loading parts of your template until needed.

**Why:** Reduce initial bundle size, improve perceived performance.

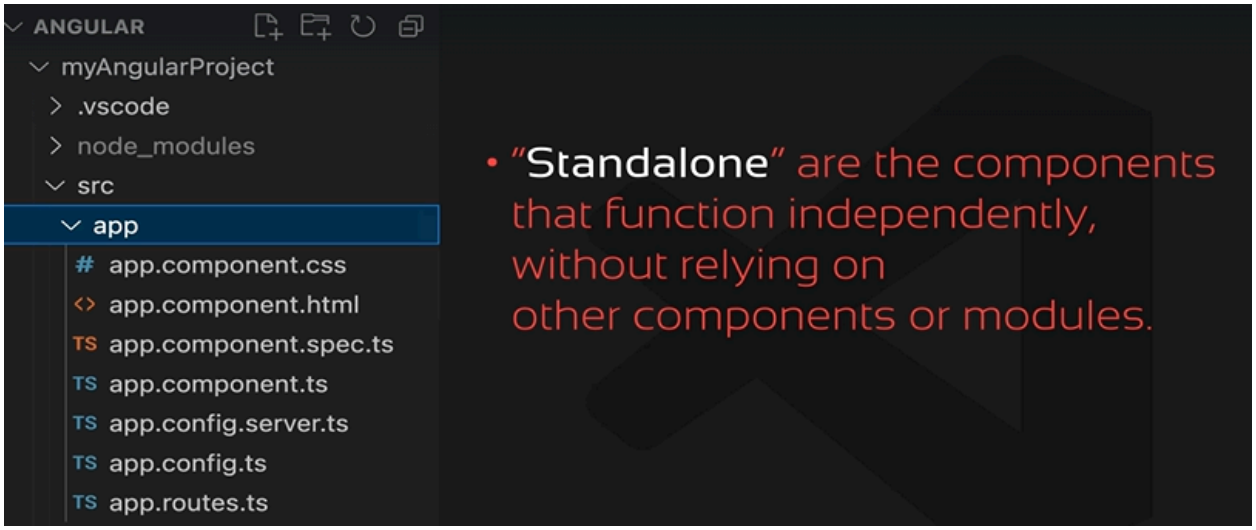
**How:** Wrap sections with `@defer`, define triggers (viewport, user interaction, custom).

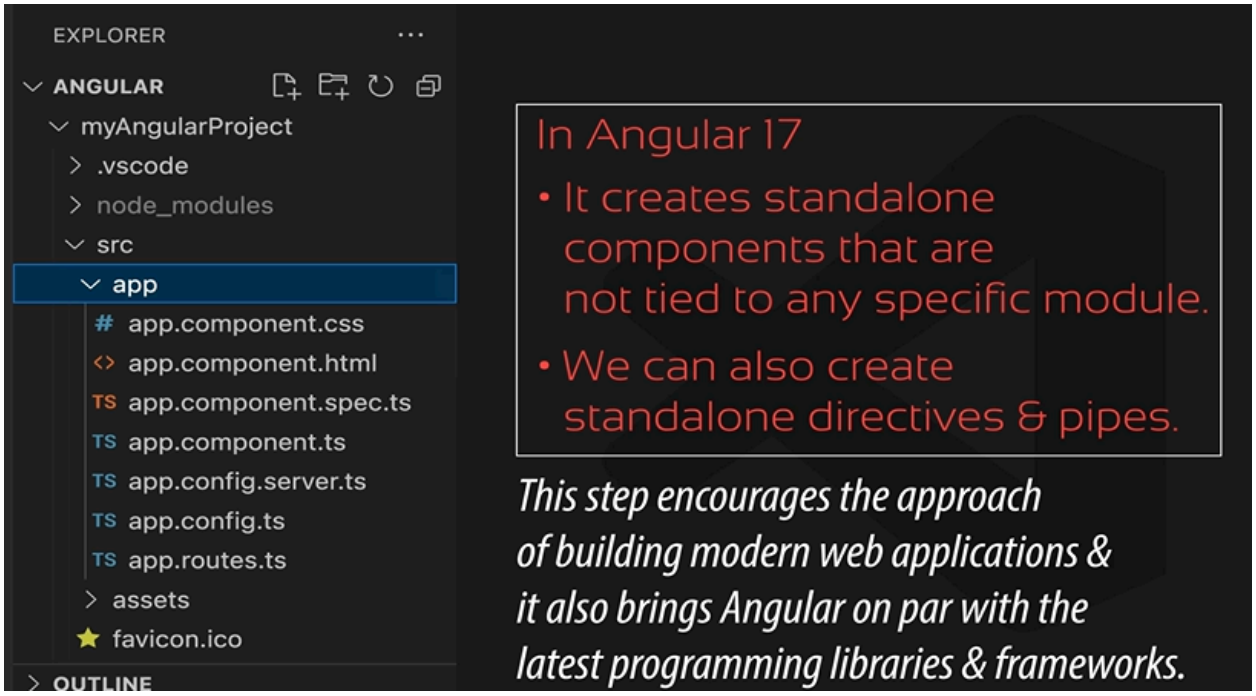
**Benefits:** Faster loads, better user experience.






**Standalone Components**

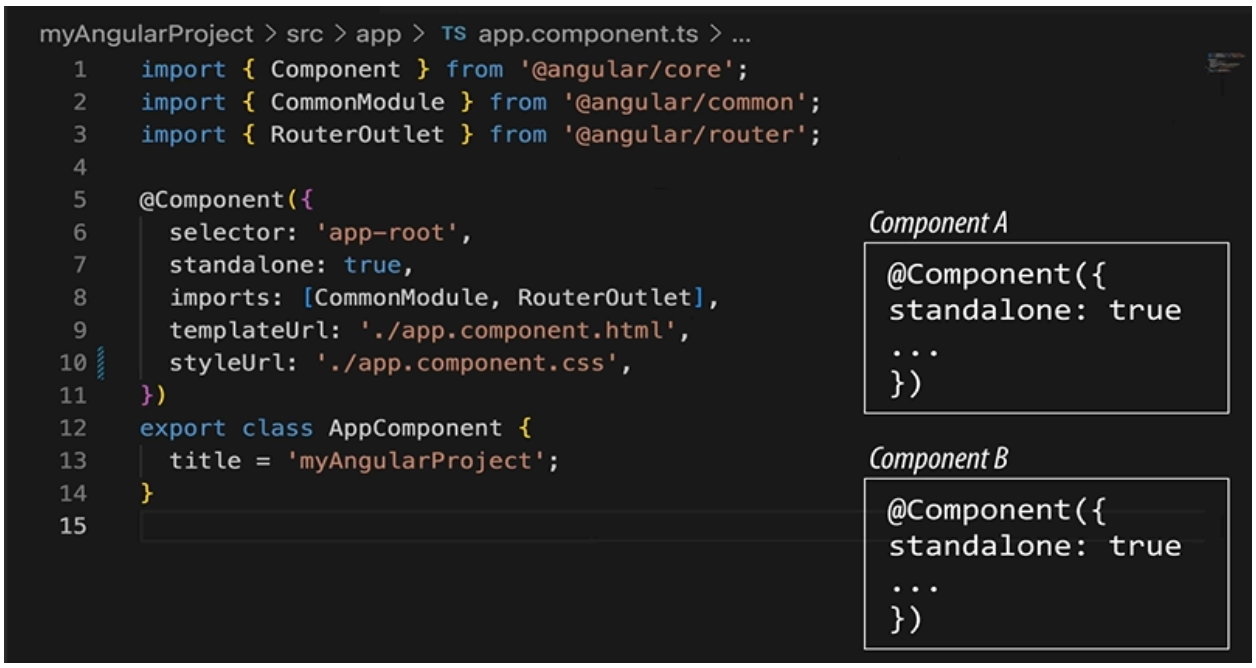

ANGULAR
myAngularProject
.vscode
node_modules
src
app
app.component.css
app.component.html
app.component.spec.ts
app.component.ts
app.config.server.ts
app.config.ts
app.routes.ts
• "Standalone" are the components that function independently, without relying on other components or modules.


EXPLORER
ANGULAR
myAngularProject
.vscode
node_modules
src
app
app.component.css
app.component.html
app.component.spec.ts
app.component.ts
app.config.server.ts
app.config.ts
app.routes.ts
assets
favicon.ico
OUTLINE
In Angular 17
• It creates standalone components that are not tied to any specific module.
• We can also create standalone directives & pipes.
This step encourages the approach of building modern web applications & it also brings Angular on par with the latest programming libraries & frameworks.

*To continue with the older app structure,*

```
ng new myApp --standalone=false OR
ng new myApp --no-standalone
```

*The app structure will have an **app.module .ts** file generated & the components will not be standalone components.*




**Ng serve -- open**

**Two Way Binding**

**HTML**

<label for="username">User Name</label>  

<input type="text" name="username" id="username" [(ngModel)]="username" placeholder="Enter Username">

<hr>

<h1>Hello {{username}}</h1>

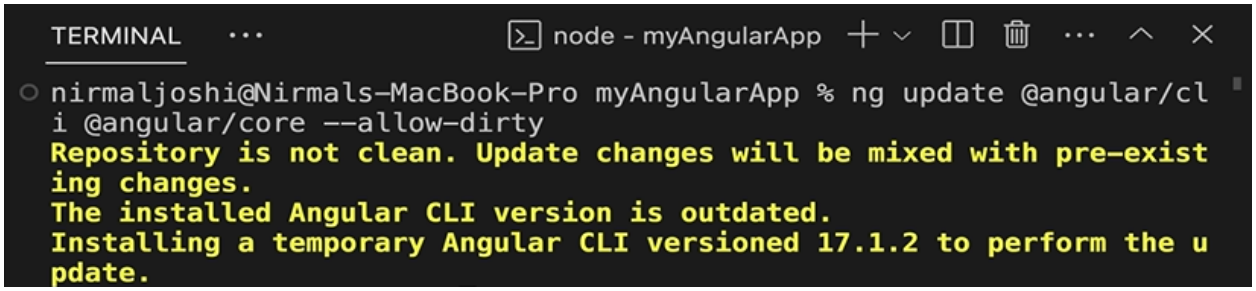
**TS**

Include FormsModule,CommonModule

username:string='';

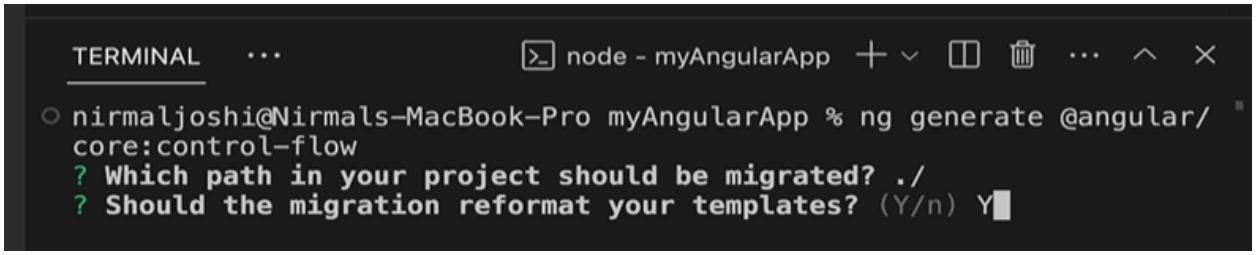
**Migrating to Angular17**

ng update @angular/cli @angular/core –allow-dirty




**Migrate the angular syntax to angular 17 including the Control flow**

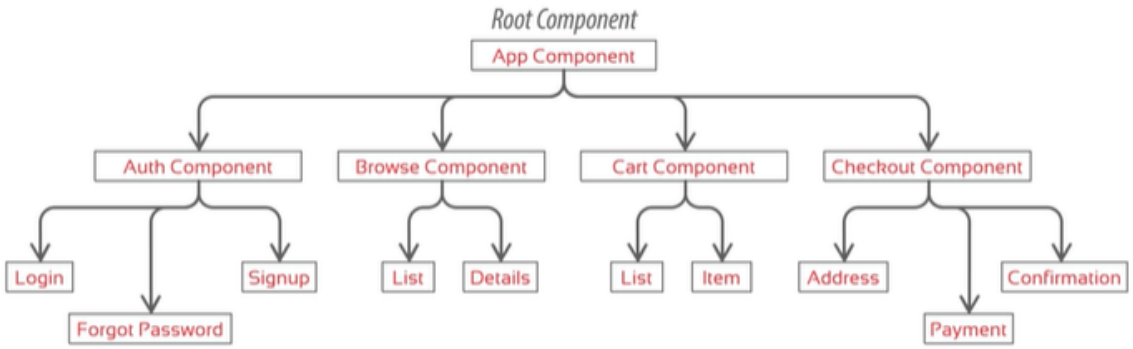
***ngFor,*ngIF → @if @for**




**Component Architecture:**

Component is the basic building block of any angular application

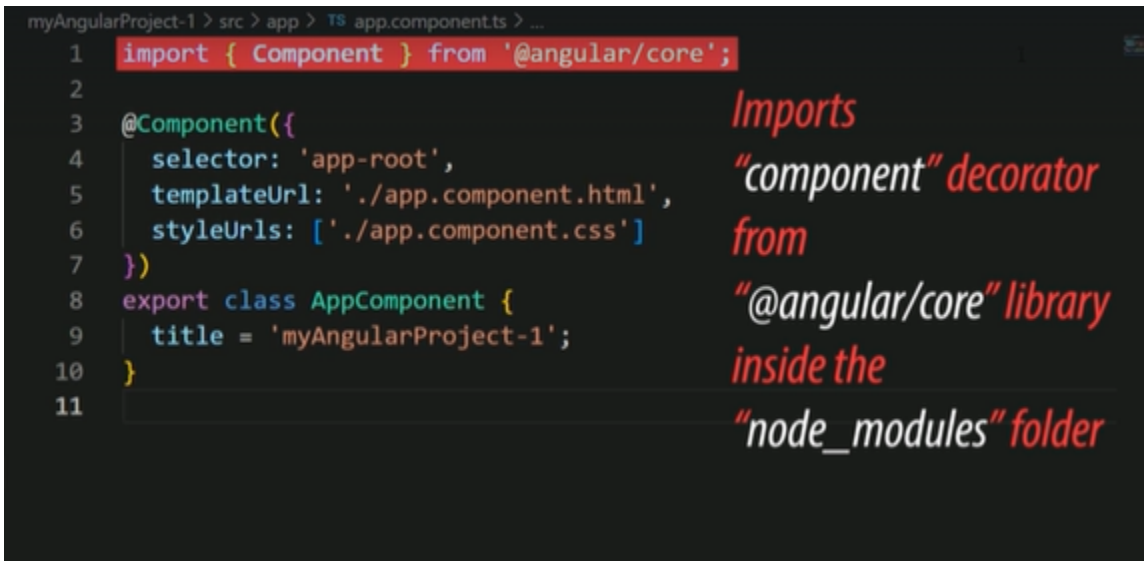
**AppComponent is root component**

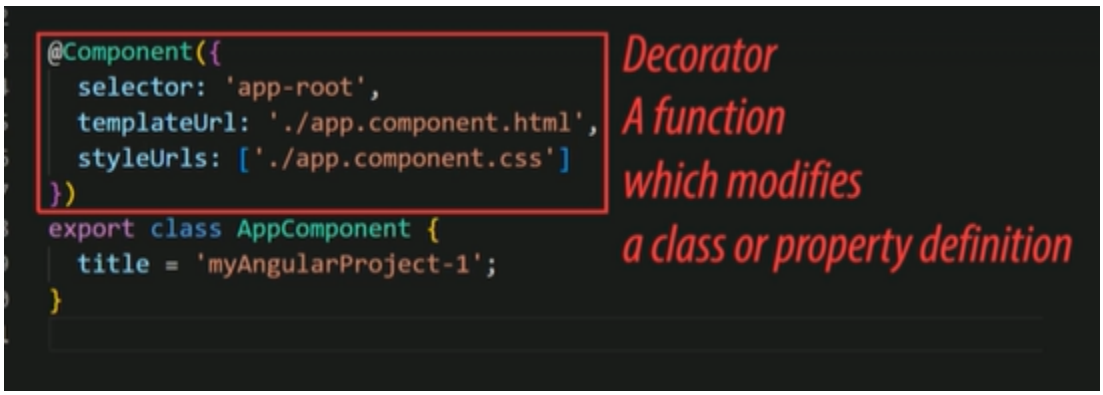



**Ts - Logical expressions & Binding components**

spec.ts - testing purposes

Application we see in local server

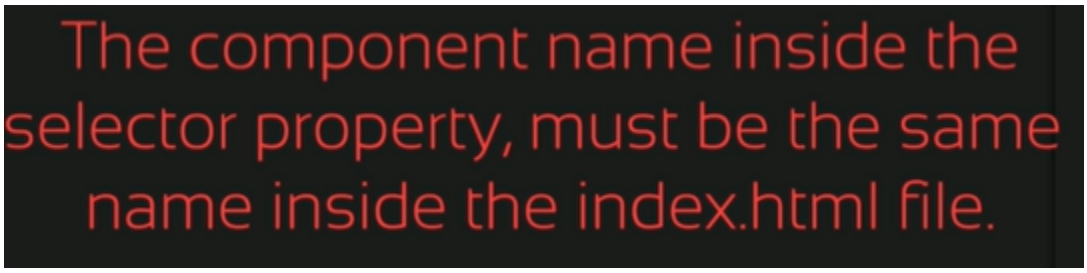


@Component - decorator contains **Metadata**

Selector - component name

If we want to change the content of the selector to some other name, then we need to change the name of the component in the html file





**Change name to app-root to app**

**app.component.ts**

```
@Component({
  selector: 'app',
  standalone: true,
  imports: [RouterOutlet,FormsModule,CommonModule],
  templateUrl: './app.component.html',
  styleUrl: './app.component.css'
})
```

**Index.html**

```
<body>
  <app></app>
</body>
</html>
```

**It will work.**

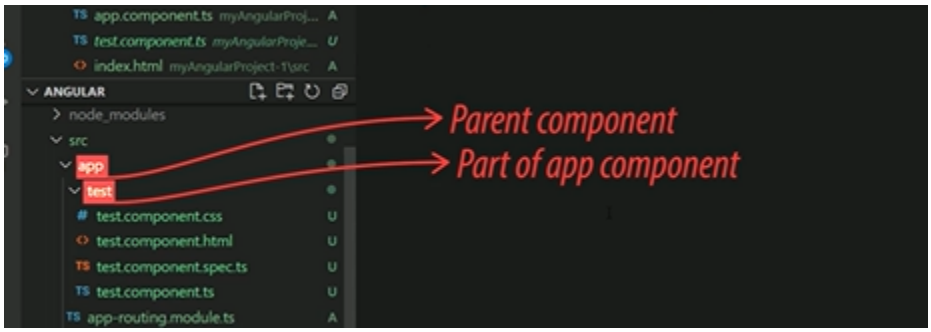
Each time changes in index.html - re run **(ng serve)**

App component Is the root component - entry to the angular application we can change it.

**Create new components**

**Ng g c <component-name>**

**Ng generate component <component-name>**



QUESTION -1 /ASSIGNMENT

# 1. What is a component based architecture in Angular?

Angular's component-based architecture revolves around creating reusable UI building blocks called components. Each component encapsulates:

**Template:** Defines the view (HTML) displayed to the user.

**Logic:** Contains TypeScript code for handling data, user interactions, and component behavior.

**Styles:** Styles the component's appearance (CSS).

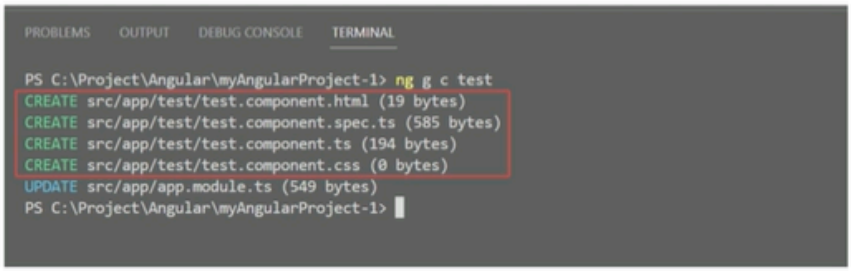
A component based architecture basically involves defining the components, which are the basic building blocks of any Angular application.

Components communicate with other components. A component itself is a user interface which can be reused. You can define / design a component that can be used by you or other developers.

**Question**

## 2. Which files does any basic Angular component have?

Whenever a new component is created apart from the root component, it will have an HTML file, a CSS file, a TypeScript and a testing file.

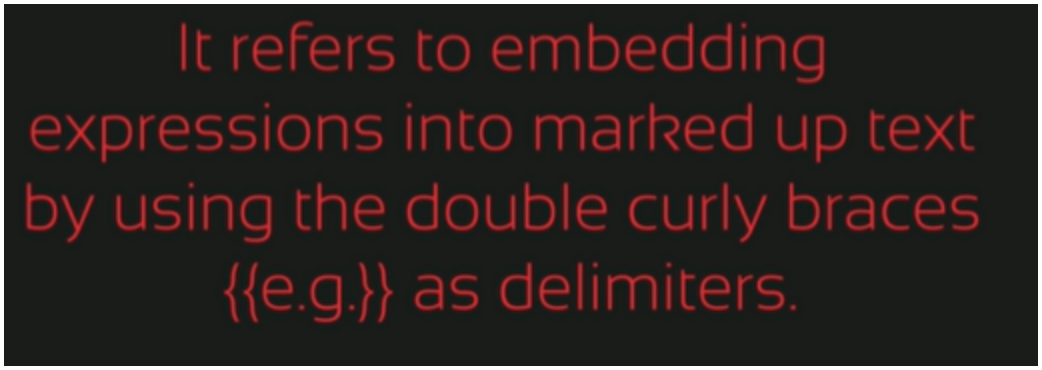


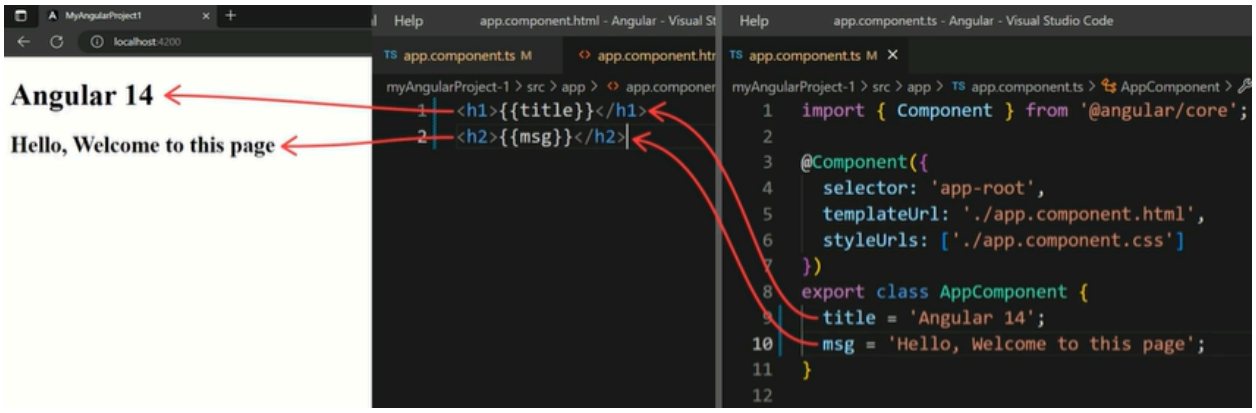

## 3. Differentiate between index.html and component.html.

The index.html is used for defining the initial loading components, global declarations or libraries you want to include like CSS libraries. Whereas the component.html is limited for the component. It actually is the interface of the component only.

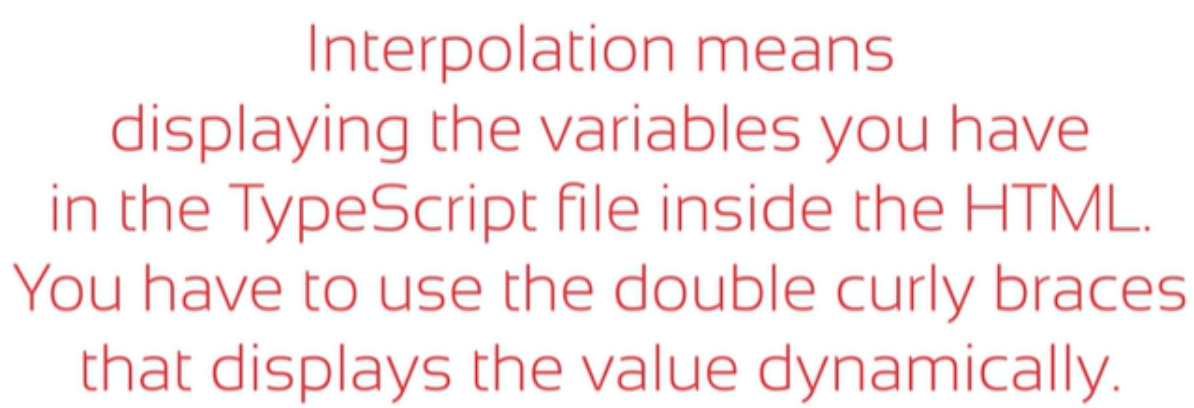
**Displaying messages dynamically - Interpolation**


**Interpolation**

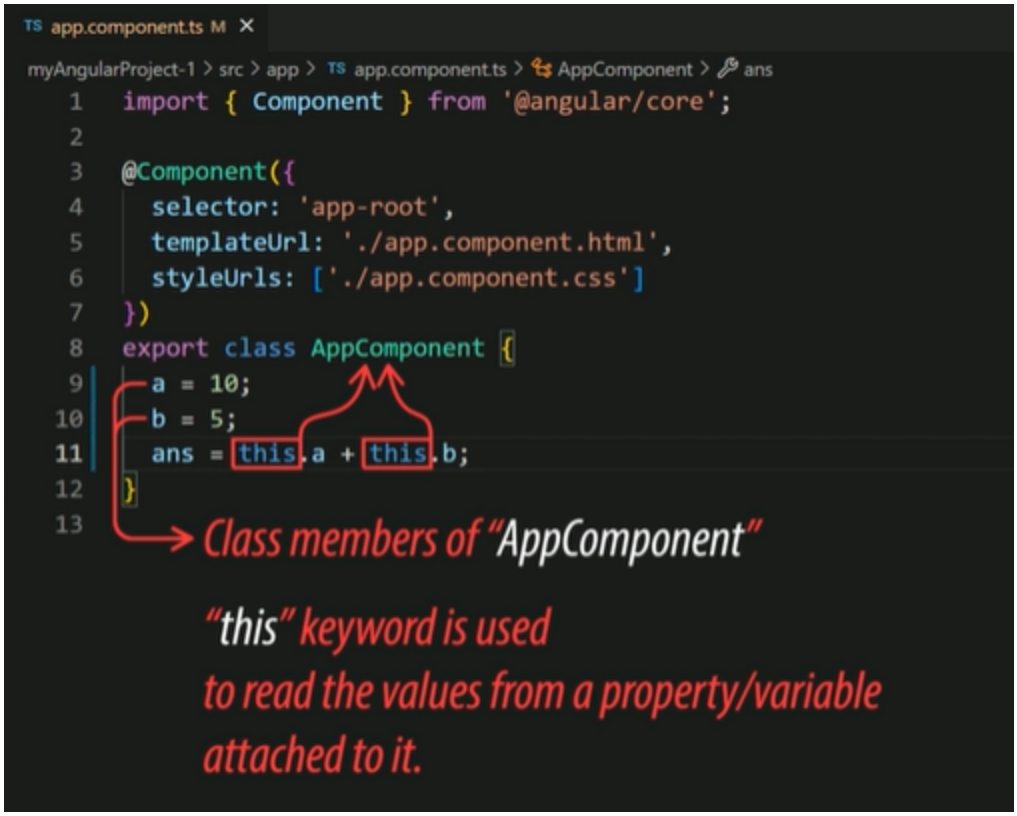


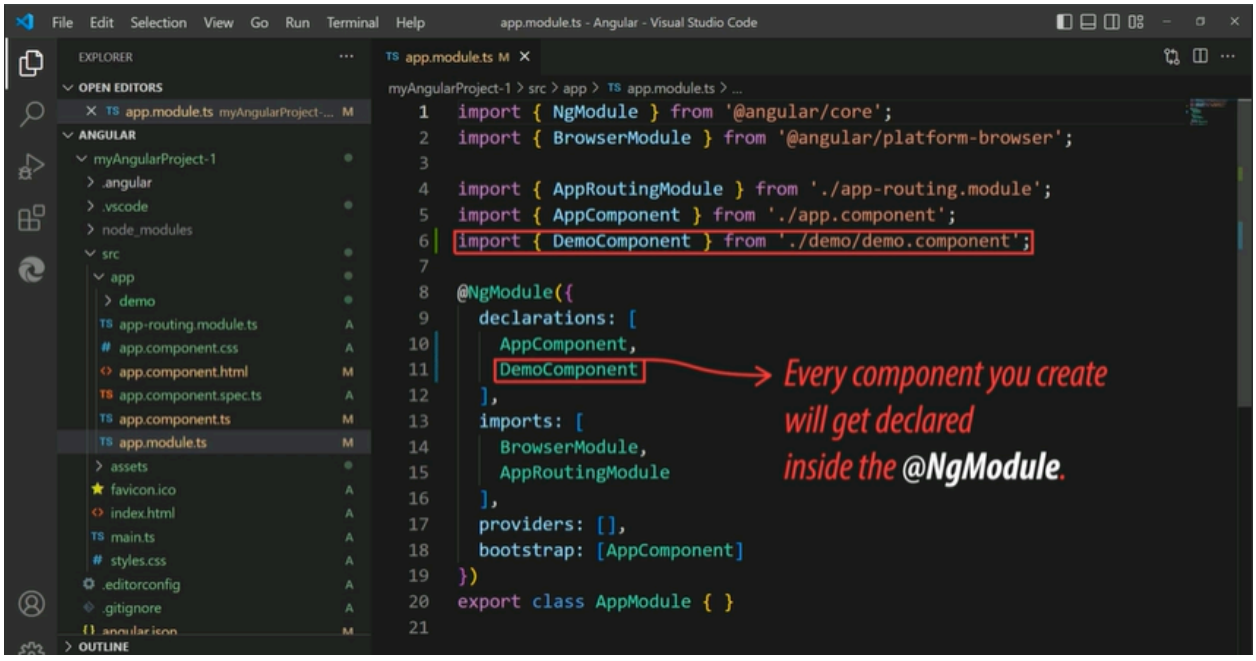

**Question - Explain the term interpolation**



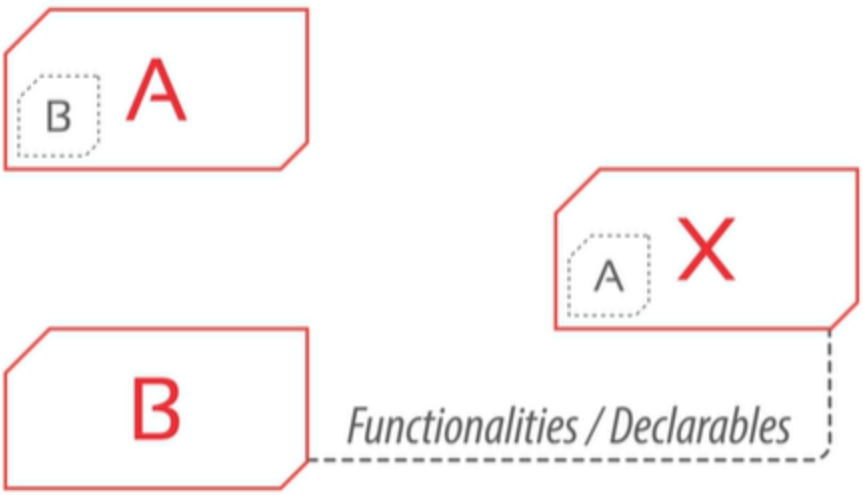

**This → accessing class members of the component**




**Before - angular 17 - app.module.ts**

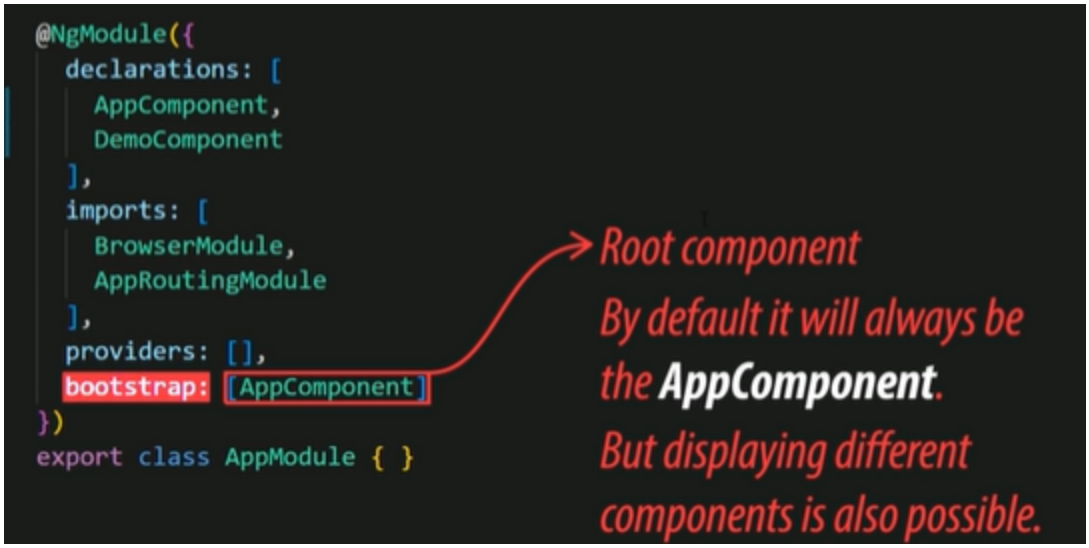



**Imports in module.ts**



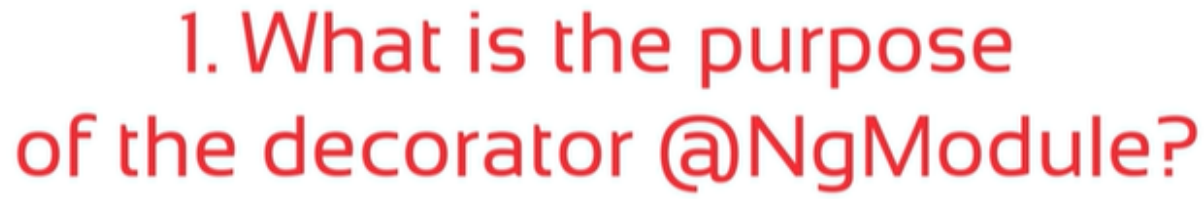


**Bootstrap - first app to run**




**By default it is app component - we can change if we want to any other component**

**Question**




**before angular 17 -** we have app.routing.module.ts,app.module.ts along with the html,css,ts,spec.ts so @NgModule decorator will helps to group all the components,directives.services and organize the angular application.

Also we can change the startup component for the angular App by bootstrapping the required Angular App

The **@NgModule decorator** provides a way to organize the Angular application by grouping components, directives & services in a single file. So the app becomes easy to maintain. Also, we can change the startup component for the Angular app by bootstrapping the required component.


Component without test files - **ng g c <component-name> - - skip - tests**

```
PS D:\OneDrive - TVS Motor Company Ltd\Desktop\WORK\Angular - UDEMY\AngularProject> cd src
PS D:\OneDrive - TVS Motor Company Ltd\Desktop\WORK\Angular - UDEMY\AngularProject\src> cd app
PS D:\OneDrive - TVS Motor Company Ltd\Desktop\WORK\Angular - UDEMY\AngularProject\src\app> ng g c form --skip-tests
  CREATE src/app/form/form.component.html (20 bytes)
  CREATE src/app/form/form.component.ts (238 bytes)
  CREATE src/app/form/form.component.css (0 bytes)
```

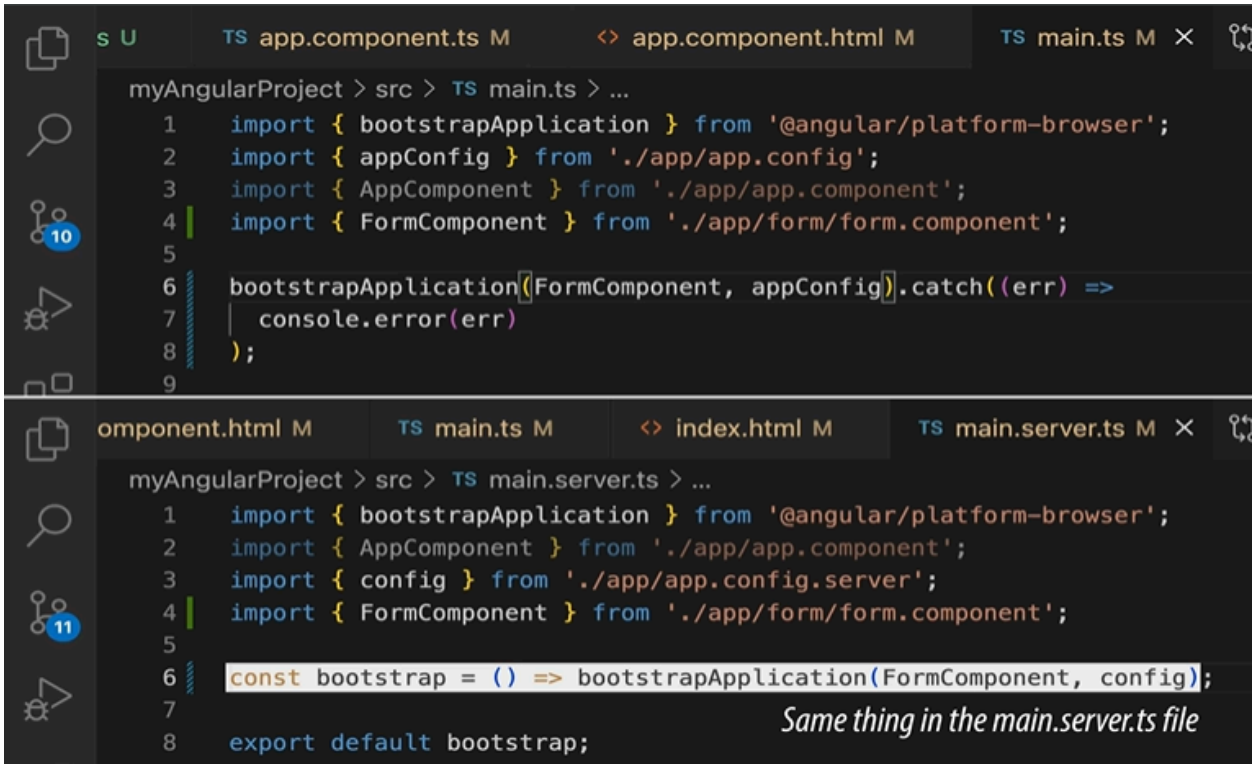
```
@Component({
  selector: 'app-form',
  standalone: true,
  imports: [FormsModule],
  templateUrl: './form.component.html',
  styleUrl: './form.component.css',
})
export class FormComponent {
  name: string = '';
}
```

Any component we create in Angular - from now on they all will be marked as a standalone component.

**We can display one component in another component by importing the component explicitly and including in the imports statement of the component.ts file in which we want to include the component and add component tag in html.**

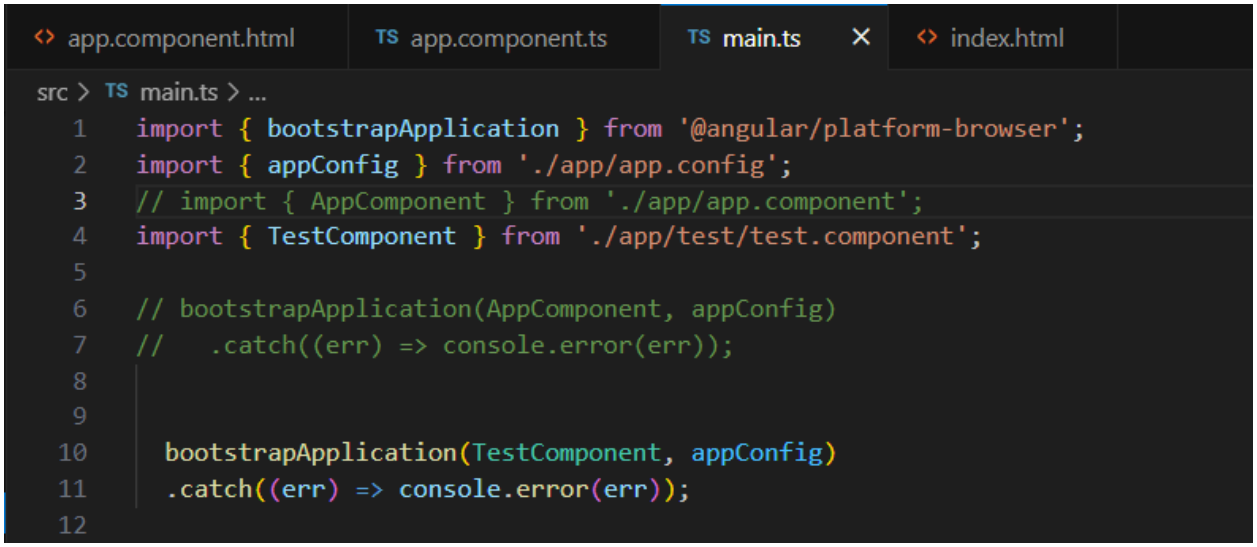
**Bootstrapping in prev versions**


**Bootstrapping in 17 version**




**To change the bootstrapping component in angular17 version**

**Change the component in main.ts and index.html - and load (ng serve)**



app.component.html | app.component.ts | main.ts | index.html

src > index.html > html > body > app-test

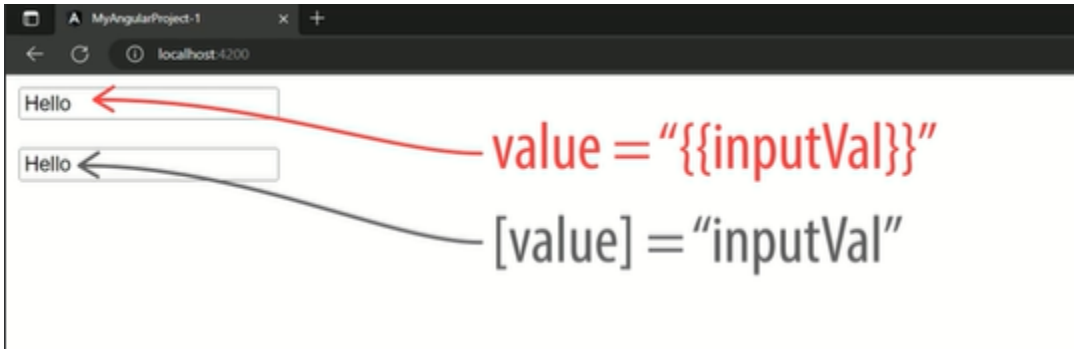
```
4    <meta charset="utf-8">
5    <title>AngularProject</title>
6    <base href="/">
7    <meta name="viewport" content="width=device-width, initial-scale=1">
8    <link rel="icon" type="image/x-icon" href="favicon.ico">
9  </head>
10 <body>
11   <!-- <app-root></app-root> -->
12   <app-test></app-test>
13 </body>
14 </html>
15
```

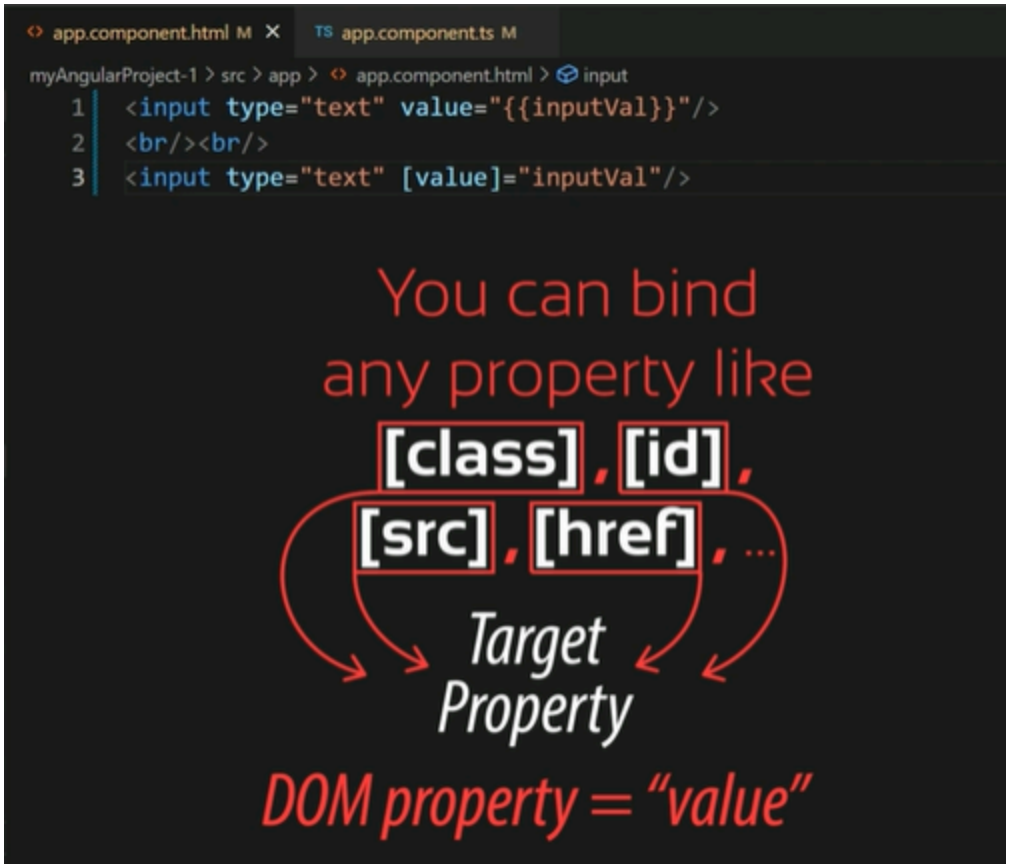
## Interpolation and Property Binding

app.component.html M | app.component.ts M

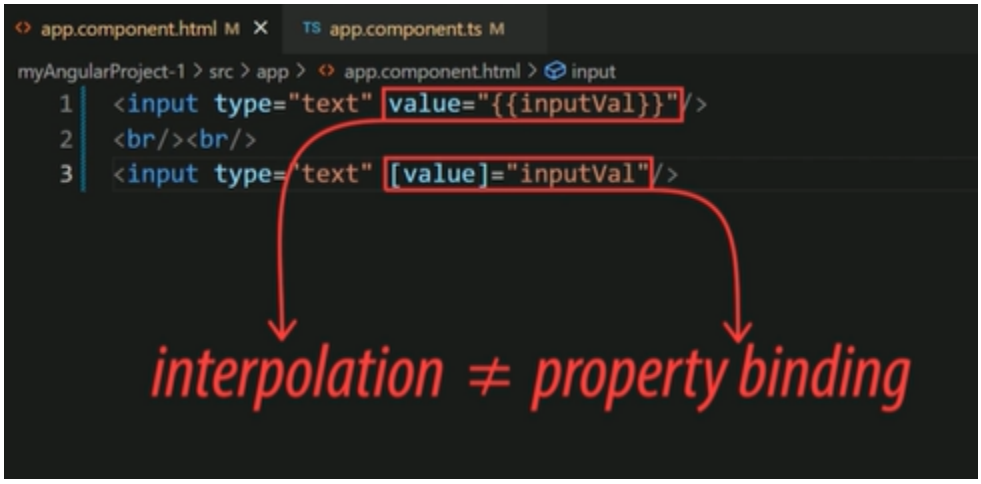
myAngularProject-1 > src > app > app.component.html > input

```
1  <input type="text" value="{{inputVal}}"/>
2  <br/><br/>
3  <input type="text" [value]="inputVal"/>
```

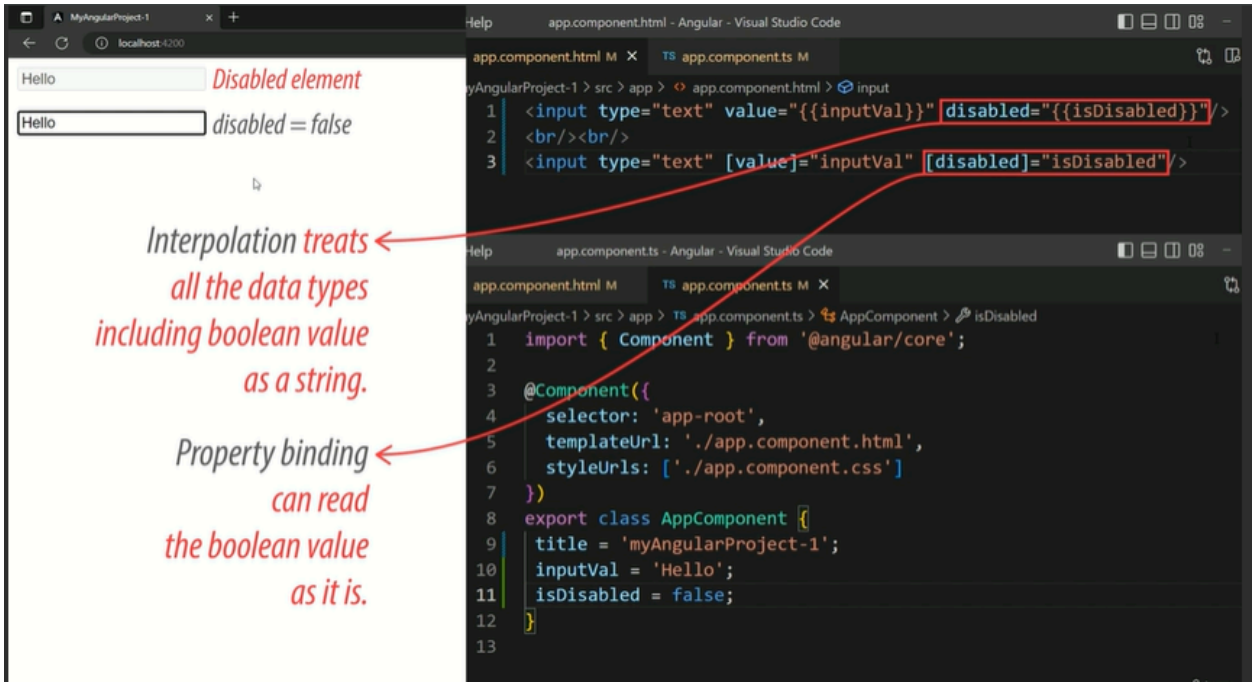


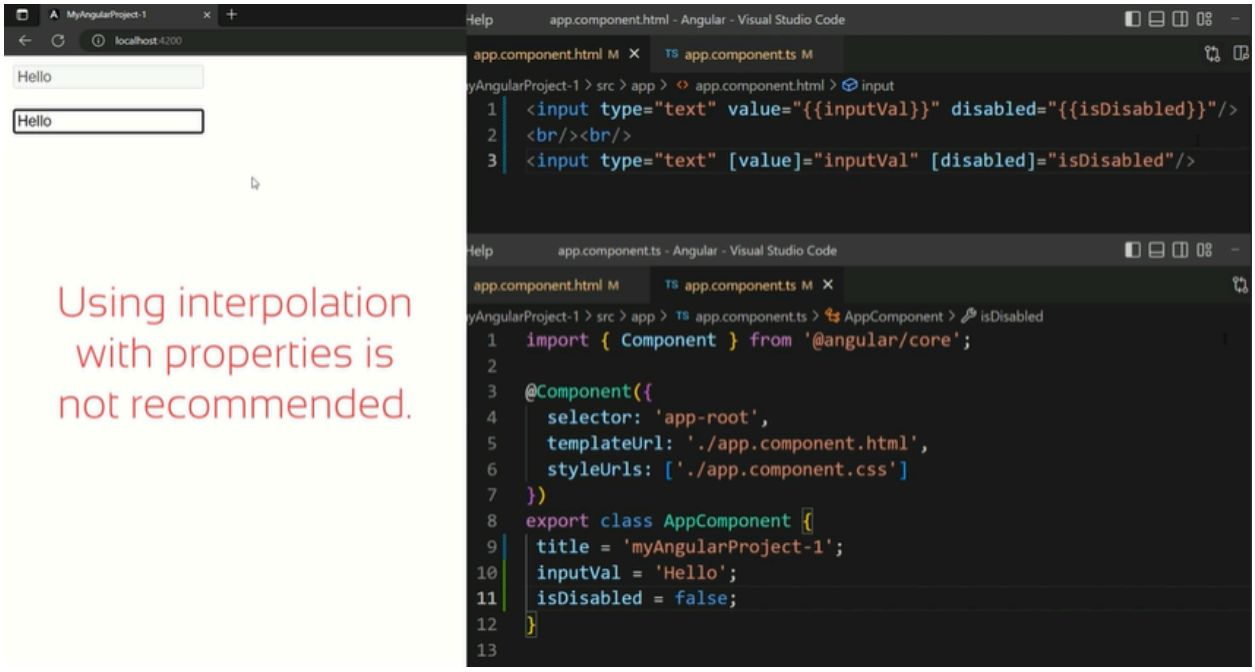


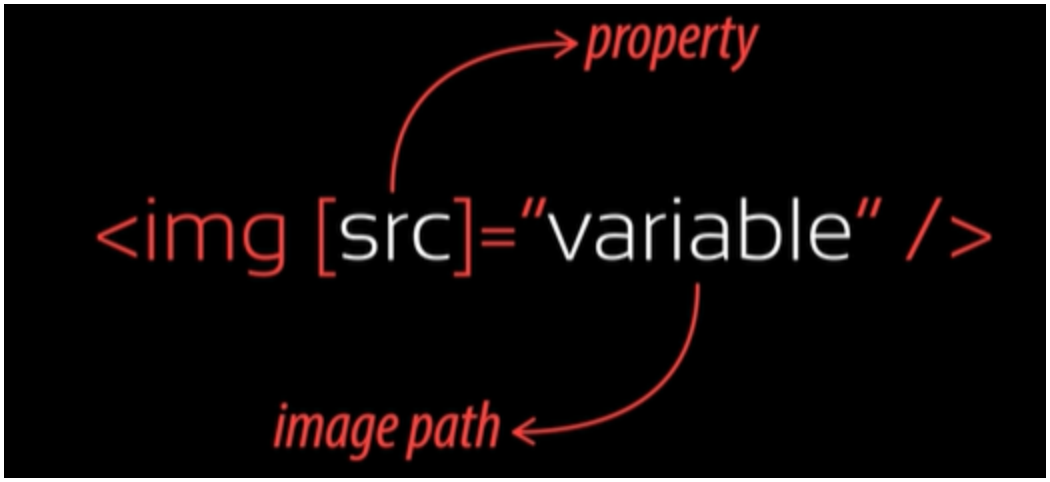

**Target property (DOM Property) will be given dynamically from the ts file**




**Main difference between interpolation and property binding**


MyAngularProject-1
localhost:4200
Hello
Disabled element
Hello
disabled = false
Interpolation treats all the data types including boolean value as a string.
Property binding can read the boolean value as it is.
app.component.html - Angular - Visual Studio Code
app.component.html M
app.component.ts M
<input type="text" value="{{inputVal}}" disabled="{{isDisabled}}"/>
<br/><br/>
<input type="text" [value]="inputVal" [disabled]="isDisabled"/>
app.component.ts - Angular - Visual Studio Code
import { Component } from '@angular/core';
@Component({
selector: 'app-root',
templateUrl: './app.component.html',
styleUrls: ['./app.component.css']
})
export class AppComponent {
title = 'myAngularProject-1';
inputVal = 'Hello';
isDisabled = false;
}

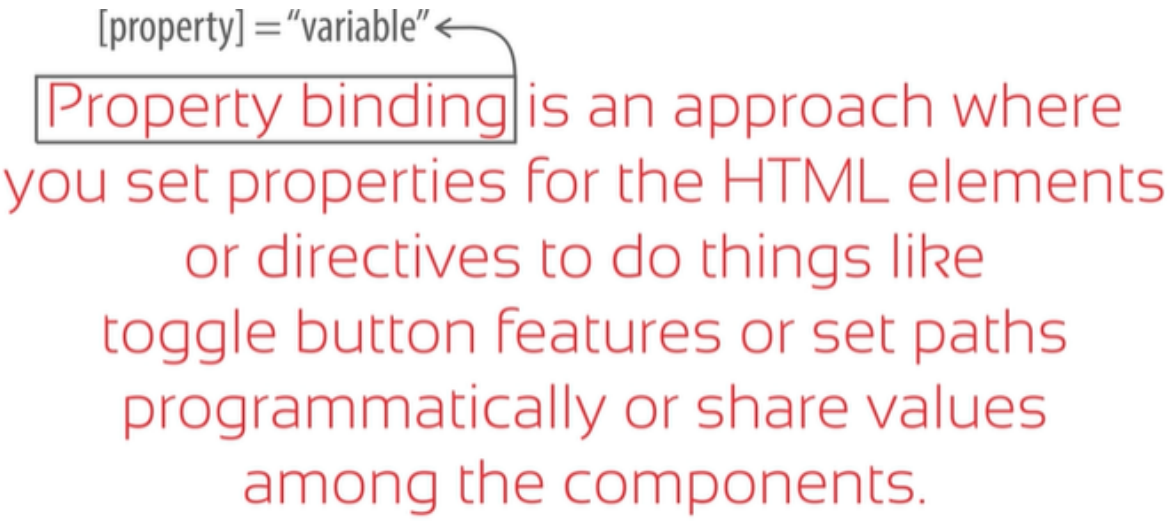

MyAngularProject-1
localhost:4200
Hello
Hello
Using interpolation with properties is not recommended.
app.component.html - Angular - Visual Studio Code
app.component.html M
app.component.ts M
<input type="text" value="{{inputVal}}" disabled="{{isDisabled}}"/>
<br/><br/>
<input type="text" [value]="inputVal" [disabled]="isDisabled"/>
app.component.ts - Angular - Visual Studio Code
import { Component } from '@angular/core';
@Component({
selector: 'app-root',
templateUrl: './app.component.html',
styleUrls: ['./app.component.css']
})
export class AppComponent {
title = 'myAngularProject-1';
inputVal = 'Hello';
isDisabled = false;
}



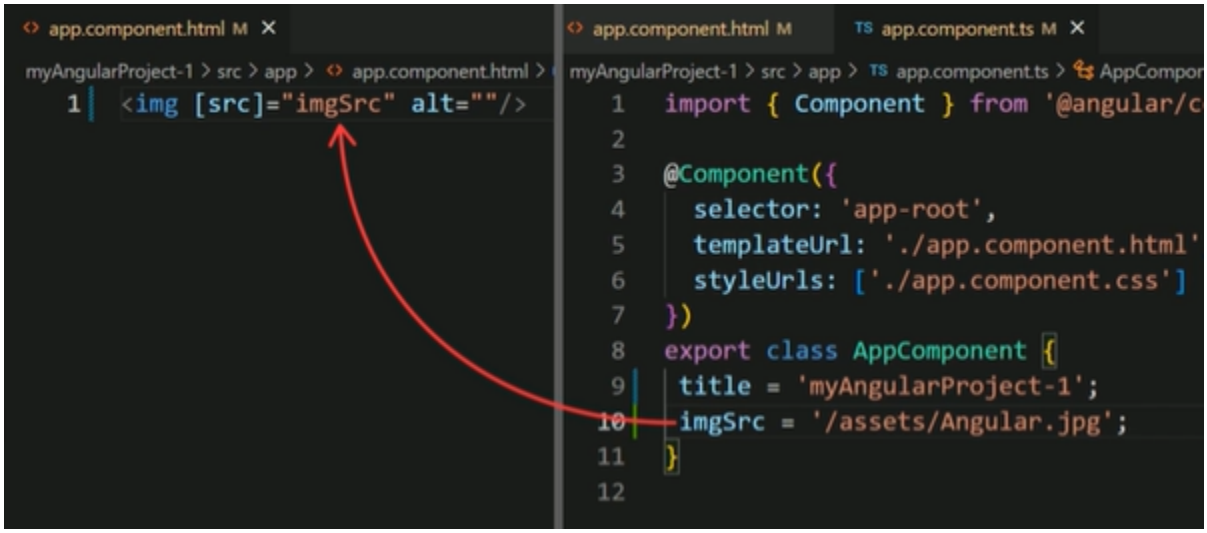

# 1. What is property binding & explain the difference between interpolation Vs property binding?

property = {{variable}}

Interpolation treats all the data types as string. So whenever you want to concat strings or bind string values, interpolation is better option whereas property binding is used when you want to set element's property to a non-string data value.




**Property binding with img tag**




**Event Binding with click event**

Events are basically an action
performed by the user
on a particular element.

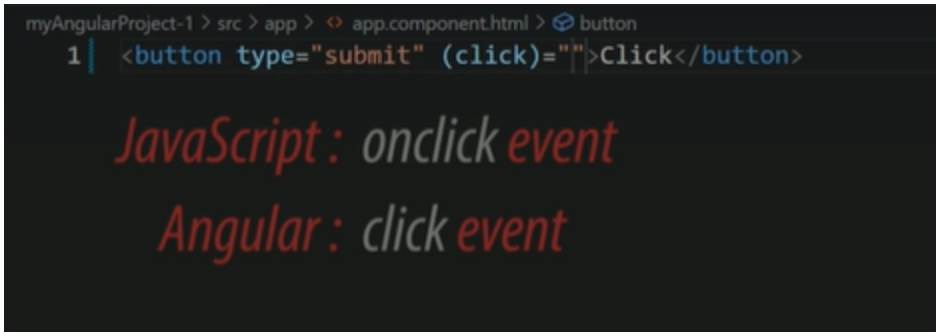

A **click event** is triggered
when the user **clicks** on the element.

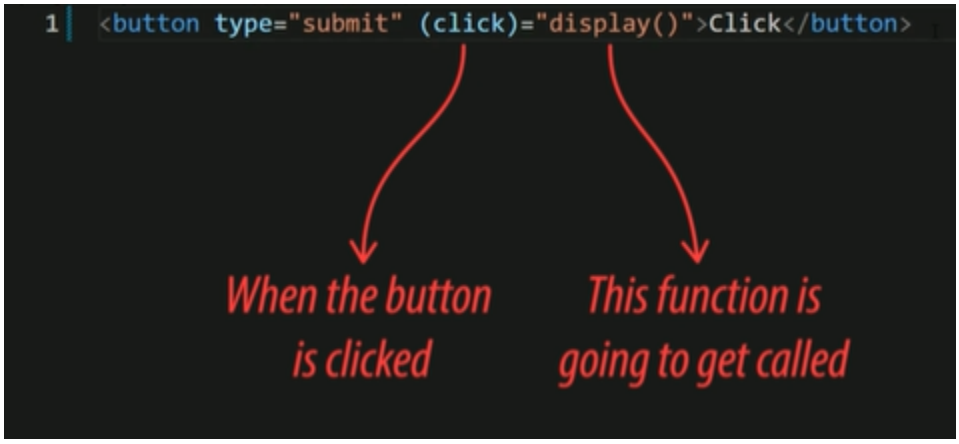
Events in HTML DOM

- Mouse events
- Keystroke events
- Input events

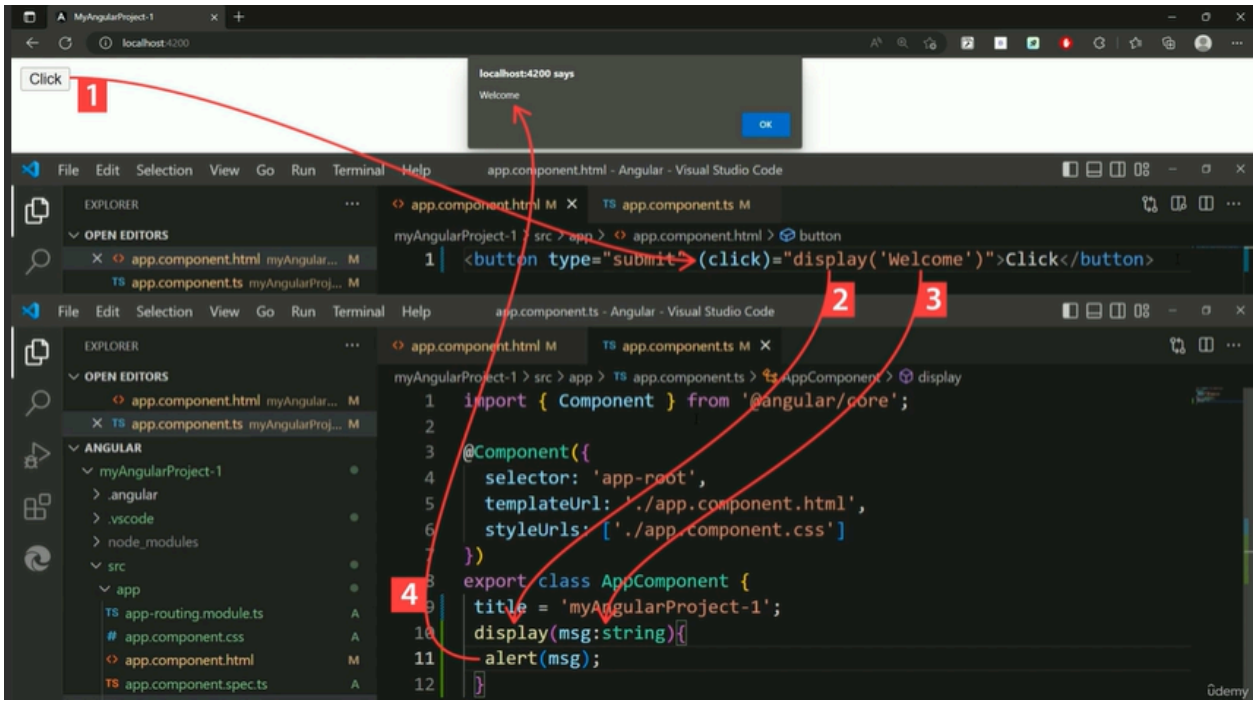
*User actions*

Events are generally used in
combination with functions
and the function will not be executed
before the event occurs.




## Event Binding - General Event Binding is One way Binding




## Question

# 1. Explain the data flow in event binding.

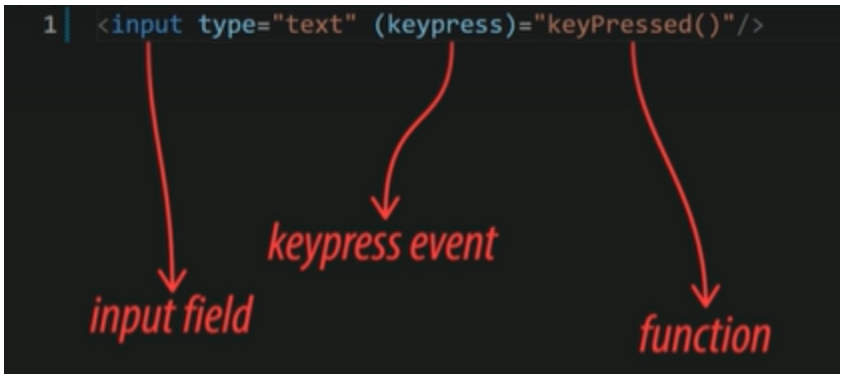
In event binding the information or the data flows from the DOM to the component. When the click event is triggered, the bound method from the component gets called and the data gets passed respectively.

# 2. Explain the difference between event binding Vs property binding.

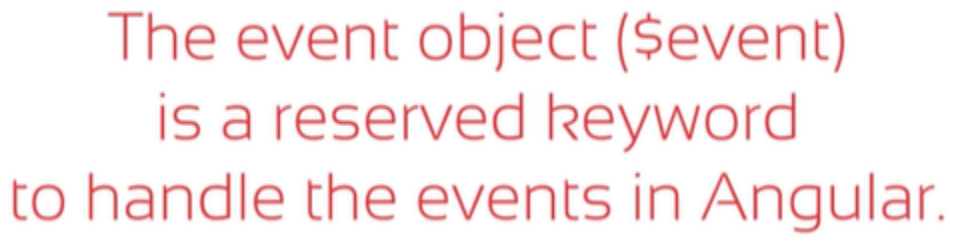
In event binding, the data is sent
from the DOM to the component -
that is why it is called as
one way binding.
Whereas in property binding,
we send the data from the component
to the DOM or view.

**Binding Other Mouse Related Events**

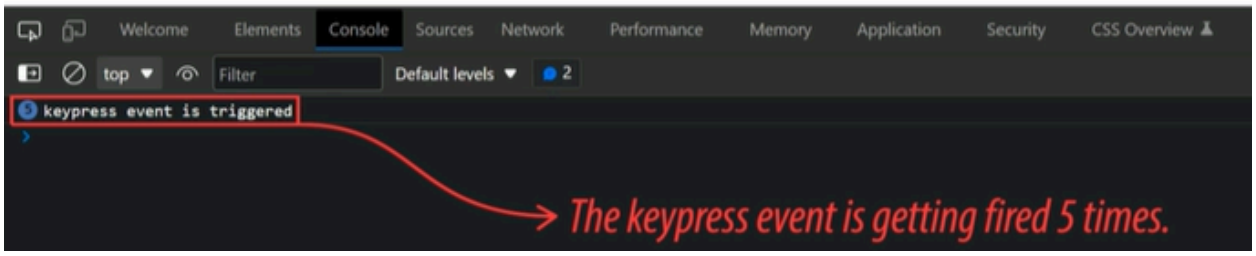

The **keyup** event can be useful in application areas
like *form validation* or *search functionalities*.

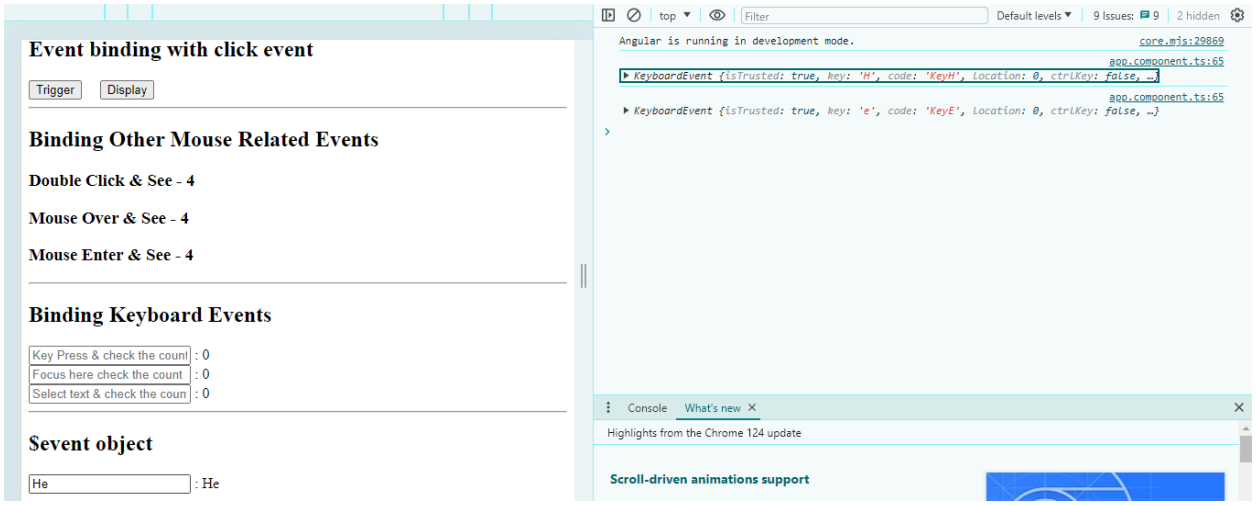




**$event object - Angular**

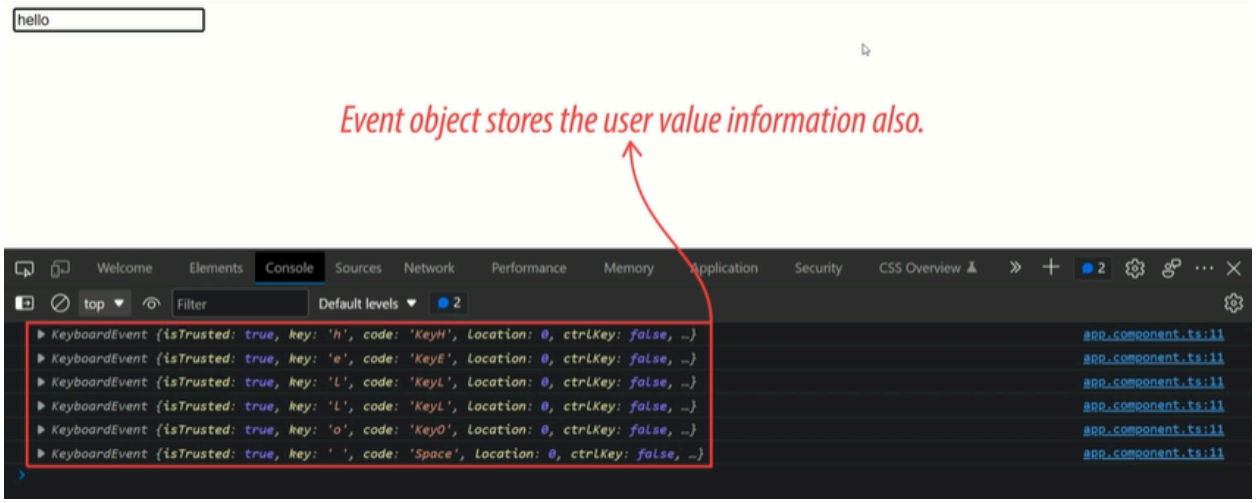



hello
How to display entered values?

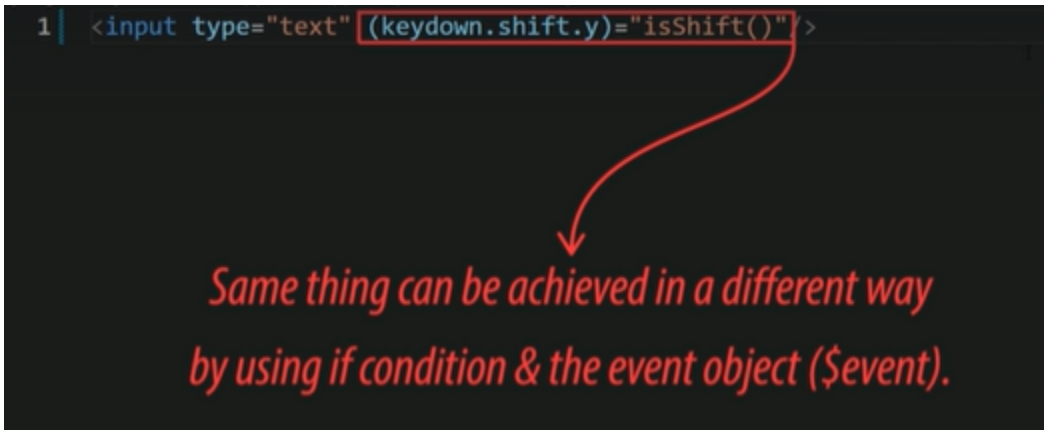

Welcome
Elements
Console
Sources
Network
Performance
Memory
Application
Security
CSS Overview
top
Filter
Default levels
2
keypress event is triggered
The keypress event is getting fired 5 times.

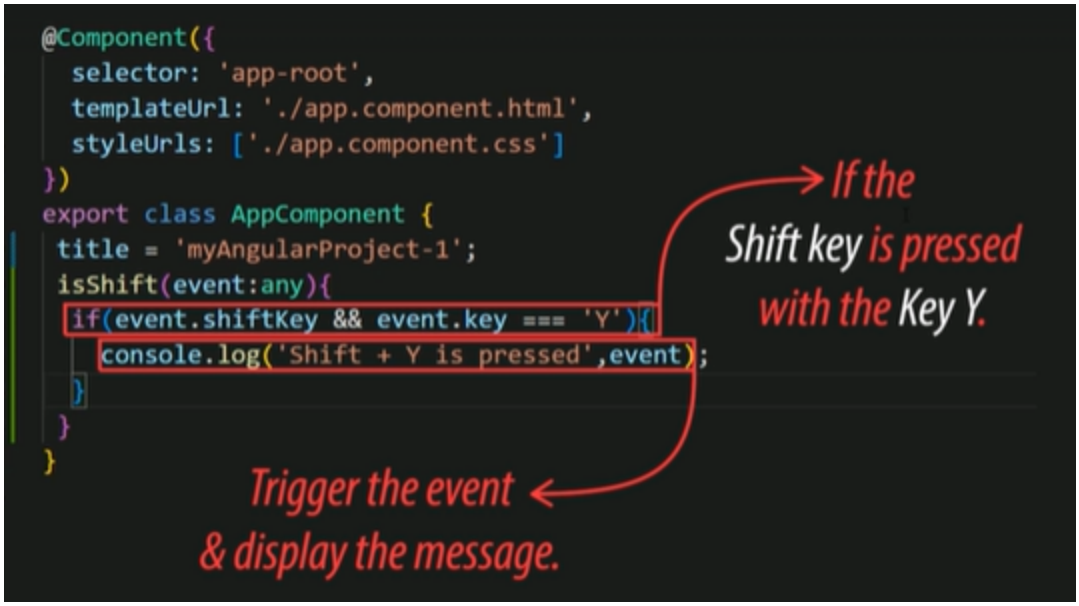

Event binding with click event
Trigger
Display
Binding Other Mouse Related Events
Double Click & See - 4
Mouse Over & See - 4
Mouse Enter & See - 4
Binding Keyboard Events
Key Press & check the count : 0
Focus here check the count : 0
Select text & check the coun : 0
Sevent object
He : He
top
Filter
Default levels
9 Issues: 9
2 hidden
Angular is running in development mode.
core.mjs:29869
app.component.ts:65
KeyboardEvent {isTrusted: true, key: 'H', code: 'KeyH', location: 0, ctrlKey: false, …}
app.component.ts:65
KeyboardEvent {isTrusted: true, key: 'e', code: 'KeyE', location: 0, ctrlKey: false, …}
Console
What's new
Highlights from the Chrome 124 update
Scroll-driven animations support

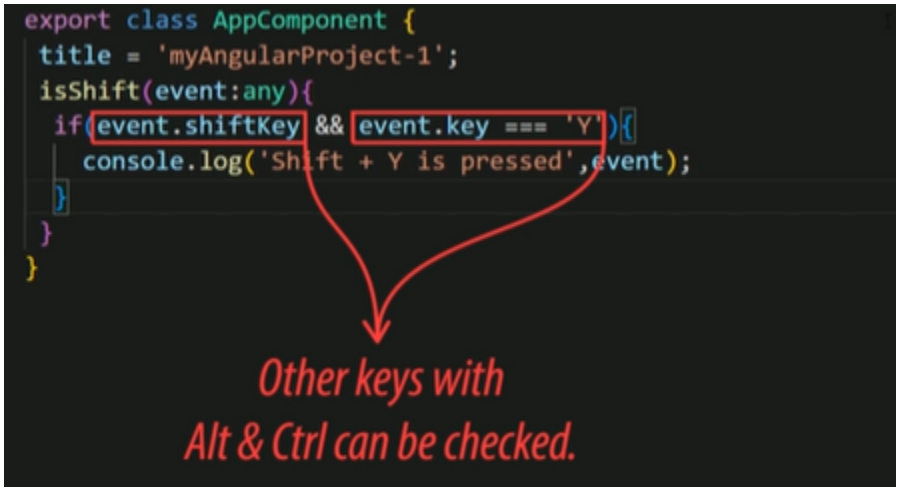
Passing the event object ($event) inside the template statement makes a lot of details for the event available to implement validations or other advanced functionalities.




**Event Binding Assignment**





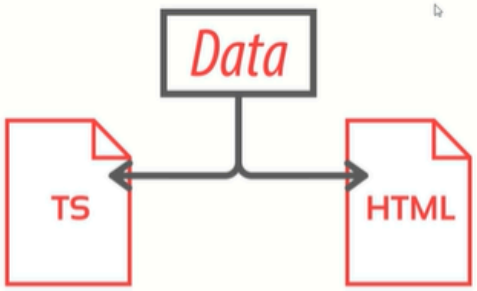

**Two Way Binding**

The two-way binding feature
makes Angular stand out
from other front-end frameworks.

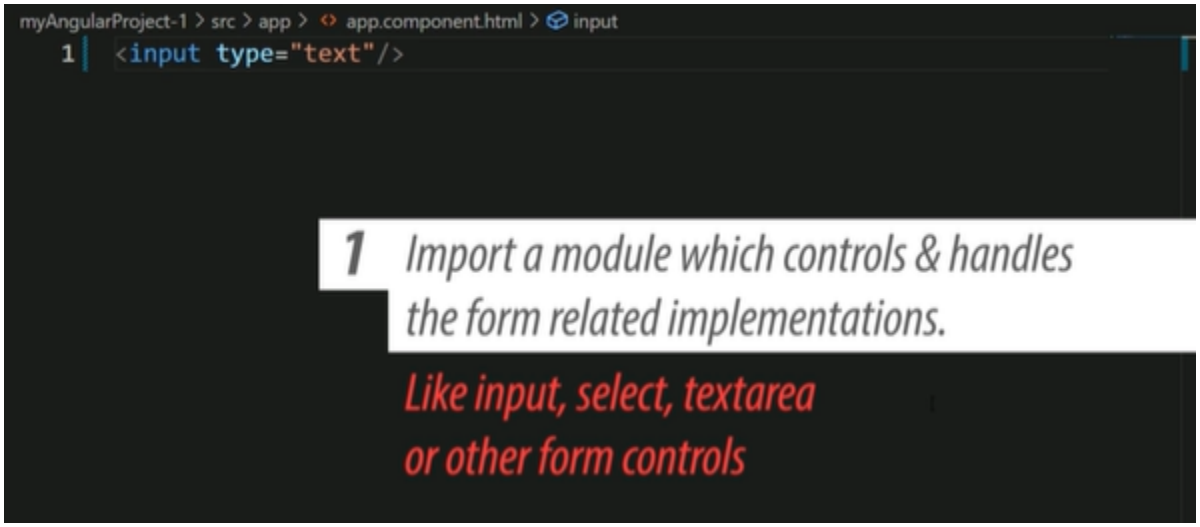
Static two-way binding

Two-Way Binding/ngModel

**Dynamic Two-Way Binding/ngModel**




**Two-way binding** means *to share/display the data/information* to the end user side, which can be changed by the end user with the help of the user interface.



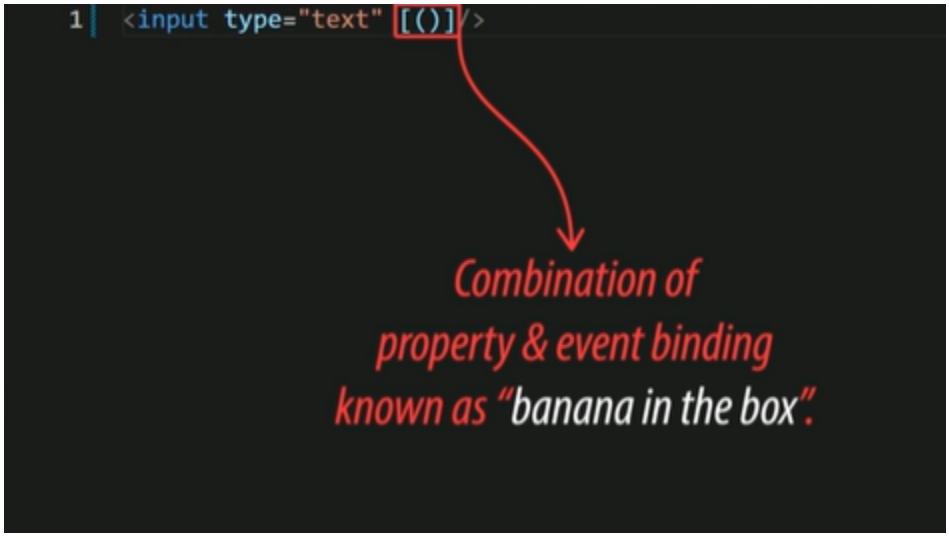
myAngularProject-1 > src > app > app.module.ts > AppModule

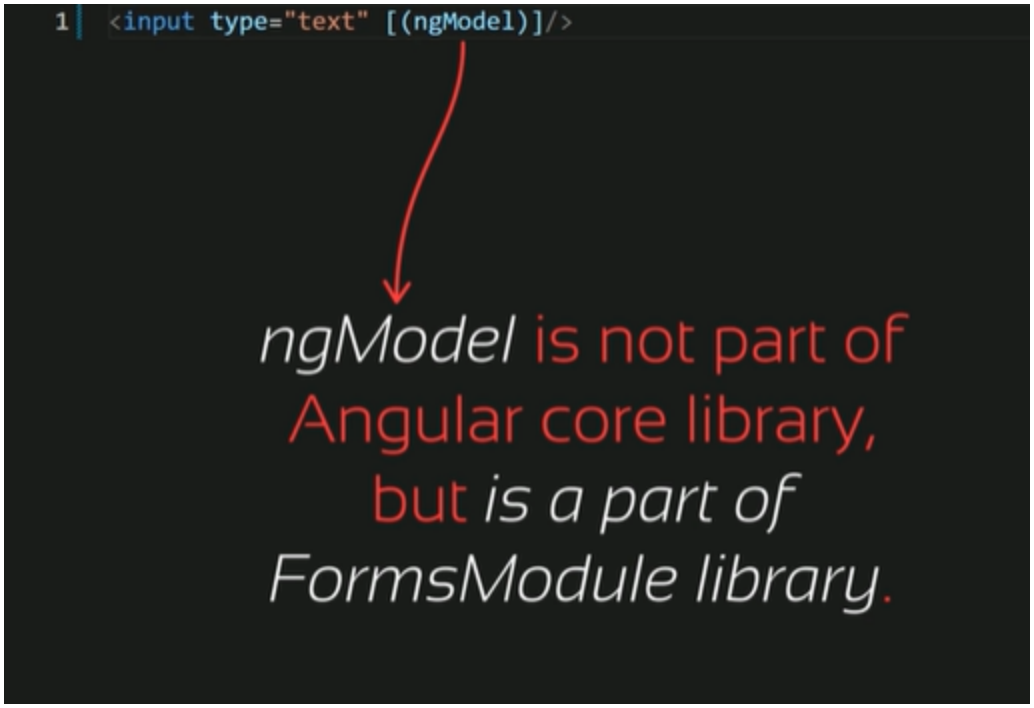
```
1  import { NgModule } from '@angular/core';
2  import { BrowserModule } from '@angular/platform-browser';
3
4  import { AppRoutingModule } from './app-routing.module';
5  import { AppComponent } from './app.component';
6  import { FormsModule } from '@angular/forms';
7
8  @NgModule({
9    declarations: [
10     AppComponent,
11   ],
12   imports: [
13     BrowserModule,
14     AppRoutingModule,
15     FormsModule,
16   ],
17   providers: [],
18   bootstrap: [AppComponent]
19 })
20 export class AppModule { }
21
```

1

2

**Syntax - two way binding**





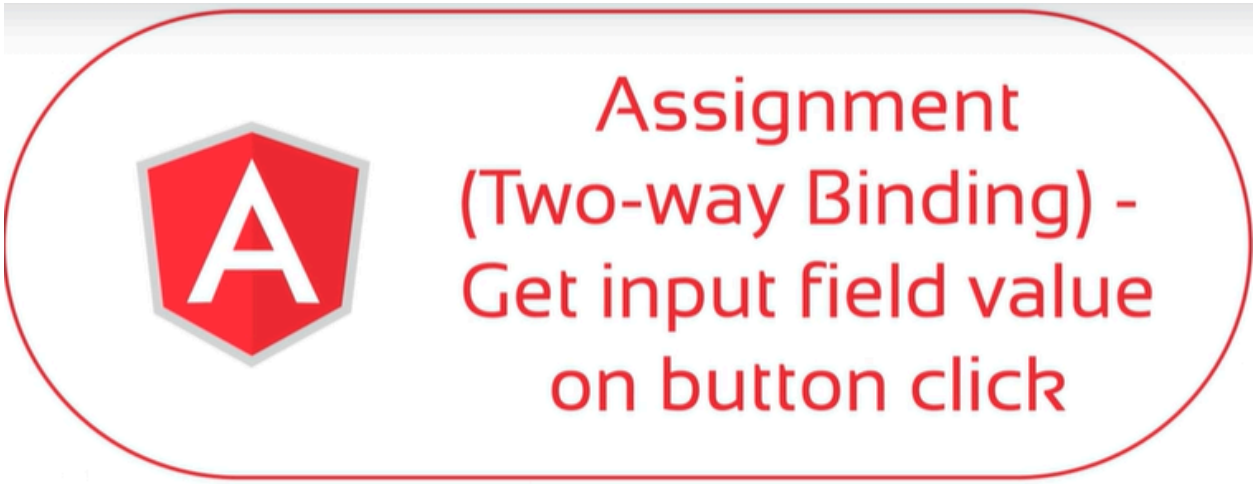

**Question**

Two-way binding simply means
to share/display the data or
information to the end user side,
which can be changed
by the end user
using the user interface.

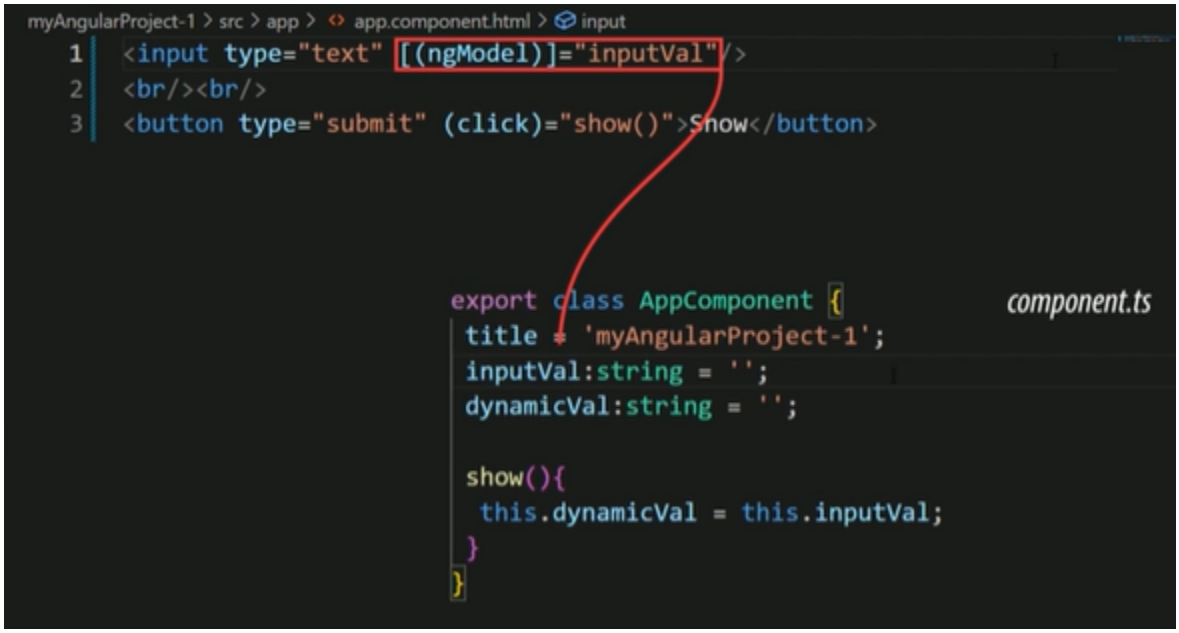
Two-way binding is used to improve
user experience by allowing/adding
more responsive & dynamic views for
making the application more interactive -
as changes made in the view
i.e., the virtual DOM will be immediately
reflected in the model & vice-versa.

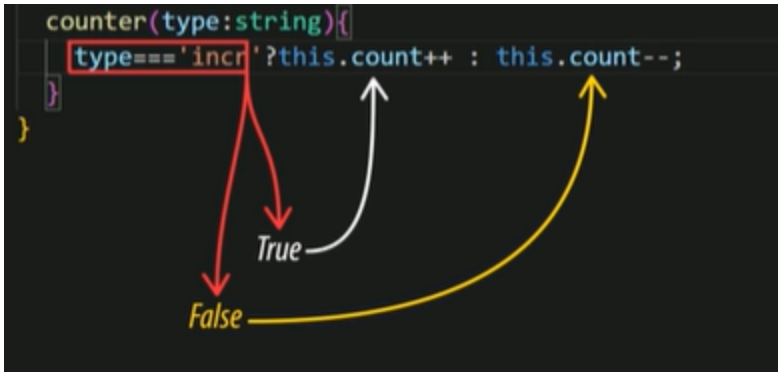
## 2. Explain the use of ngModel.

The ngModel is a directive
provided by the FormsModule
that allows two-way binding by binding
the values of HTML control elements
like input, select, textarea
or any other custom controls.
It stores the value inside the variable
which can be used whenever required.




**Get input field value on button click**





counter(type:string){
type==='incr'?this.count++ : this.count--;
}
}
True
False

# Questions & Assignments List

**1. First Component with Architecture**

1.1 What is a component based architecture in Angular?

1.2 What files does any basic Angular component have?

1.3 Differentiate between index.html and component.html?

**2. Displaying Message Dynamically (Interpolation)**

2.1 Assignment - Displaying sum of two numbers

**3. Understanding @NgModule**

3.1 What is the purpose of @NgModule?

**4. Property Binding**

4.1 What is property binding & explain the difference between interpolation Vs property binding ?

4.2 Property binding with image tag

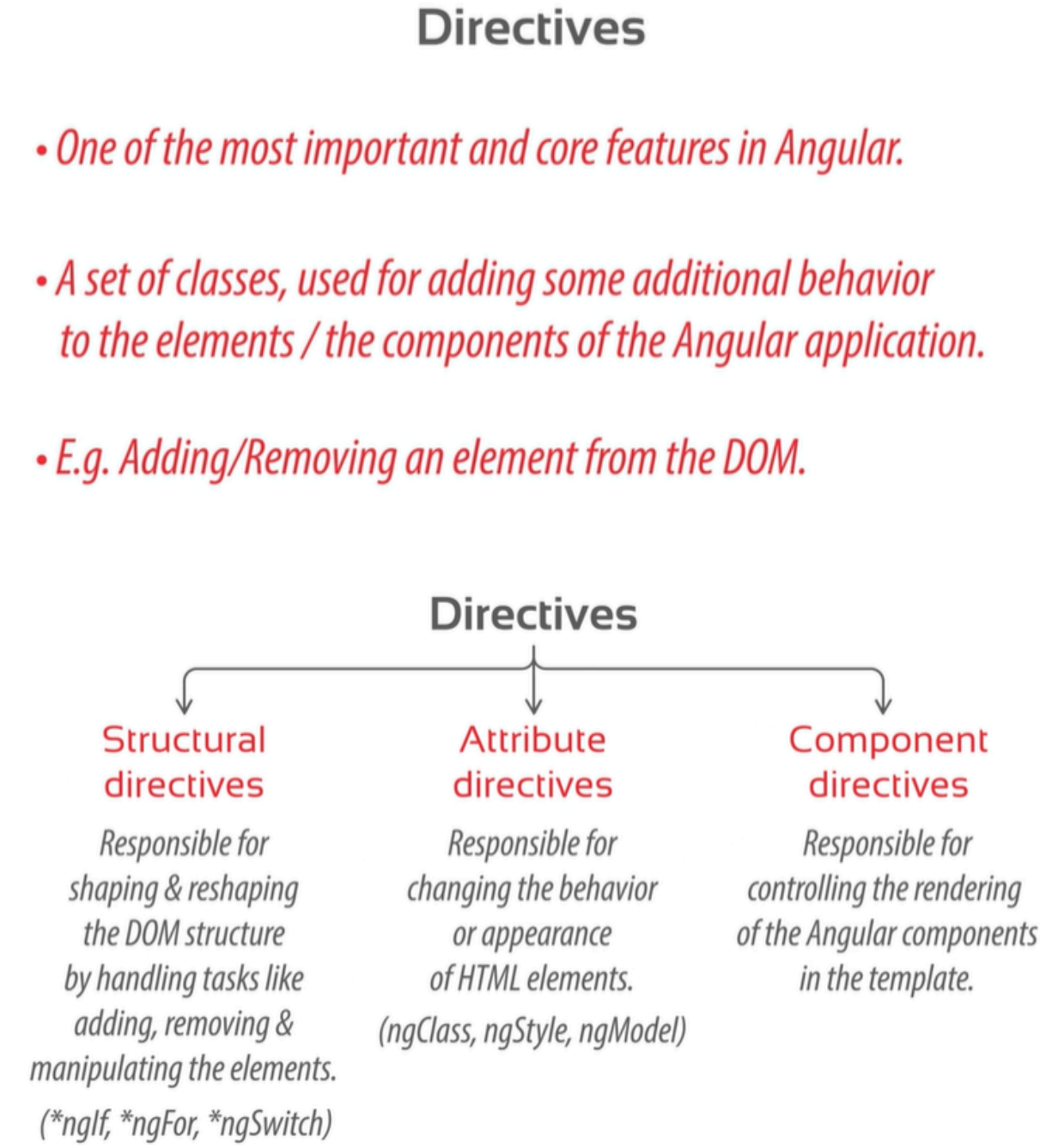
**5 Event Binding with click event**

5.1 Explain the data flow in event binding.

5.2 Difference event binding Vs property binding

## What is a Directive in Angular?

A directive in Angular is a class that adds behavior to elements in your Angular applications. There are three main types of directives:




Component Directives:

- Directives with a template.

**Example**

@Component({

```
  selector: 'app-example',
  template: '<p>Example Component</p>',
})
export class ExampleComponent {}
```

Attribute Directives:

- Change the appearance or behavior of an element.

**Example**

```
@Directive({
  selector: '[appHighlight]'
})
export class HighlightDirective {
  constructor(el: ElementRef, renderer: Renderer2) {
    renderer.setStyle(el.nativeElement, 'backgroundColor', 'yellow');
  }
}
```

**html**

```
<p appHighlight>Highlighted Text</p>
```

**Structural Directives:**

- Change the DOM layout by adding/removing elements.
- **Example:** **`ngIf`**, **`ngFor`.**

```
<div *ngIf="condition">Content</div>
```

```
@Directive({
  selector: '[appUnless]'
})
export class UnlessDirective {
  @Input() set appUnless(condition: boolean) {
    condition ? this.vcRef.clear() : this.vcRef.createEmbeddedView(this.templateRef);
  }
  constructor(private templateRef: TemplateRef<any>, private vcRef: ViewContainerRef)
{}
}
```

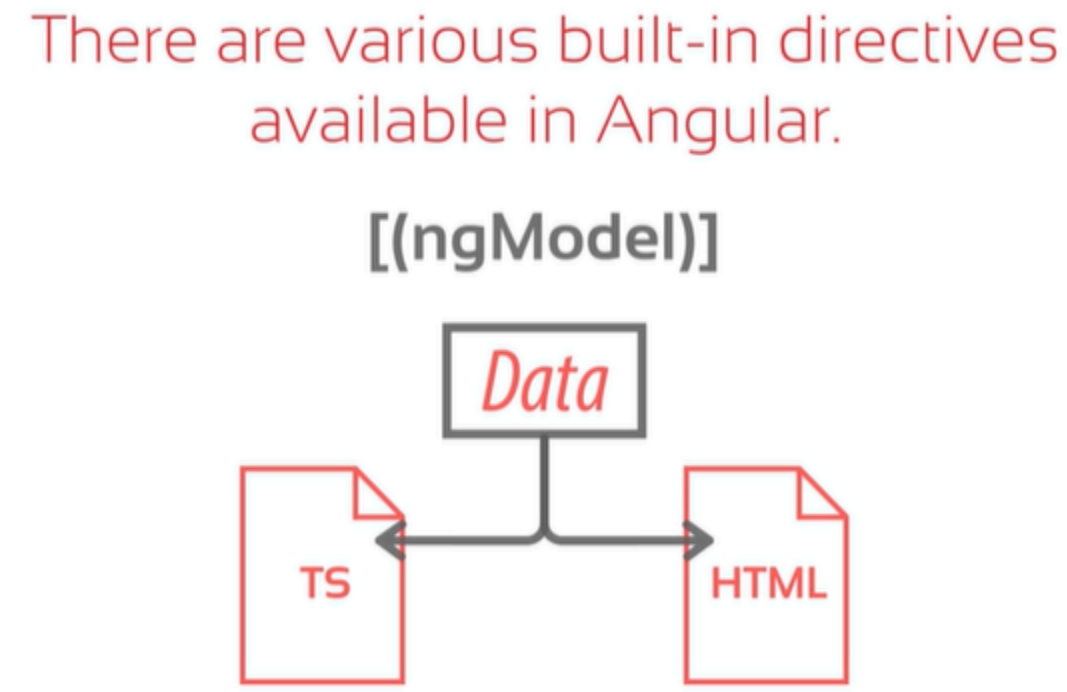
```
<div *appUnless="condition">This shows if condition is false</div>
```

## Summary:

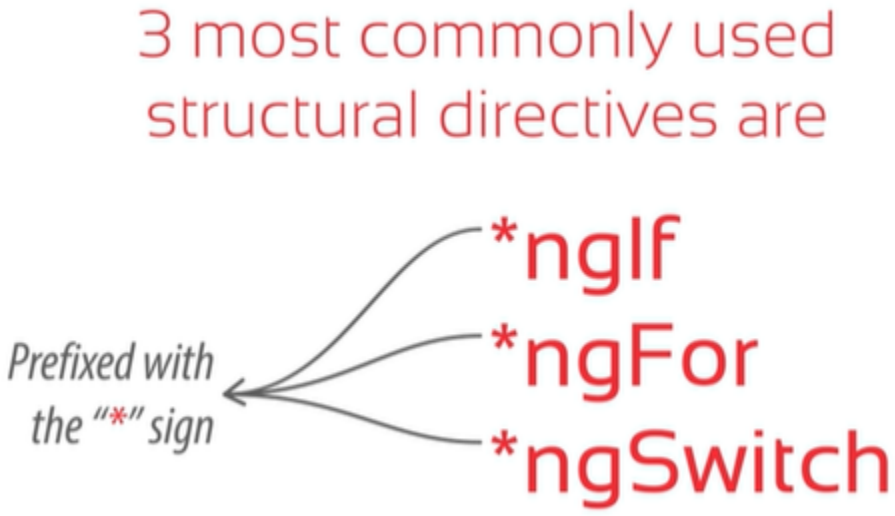
- Components: Directives with templates.
- Attribute Directives: Modify element behavior/appearance.
- Structural Directives: Alter the DOM layout.

Directives enhance HTML by attaching custom behavior to elements, making Angular applications more dynamic and powerful.

**Directive for two way binding**

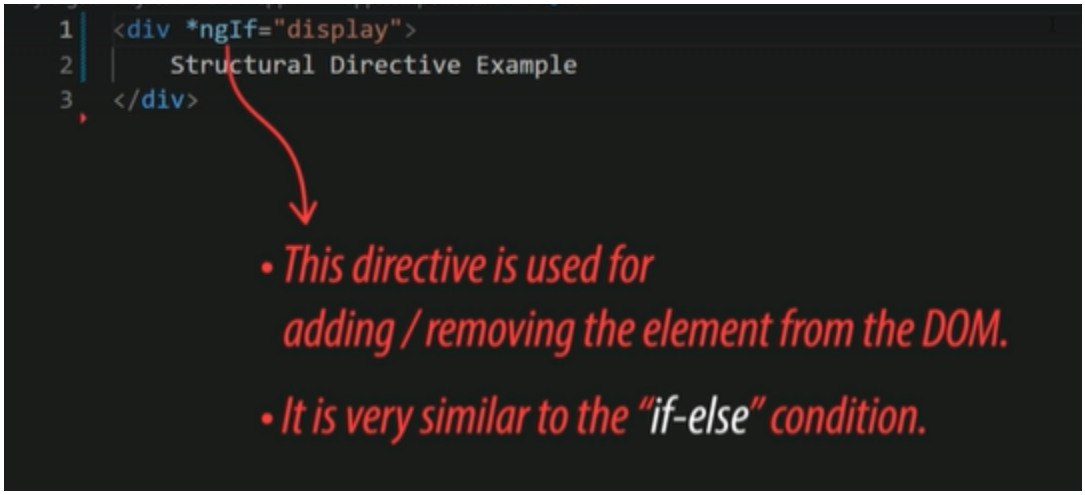


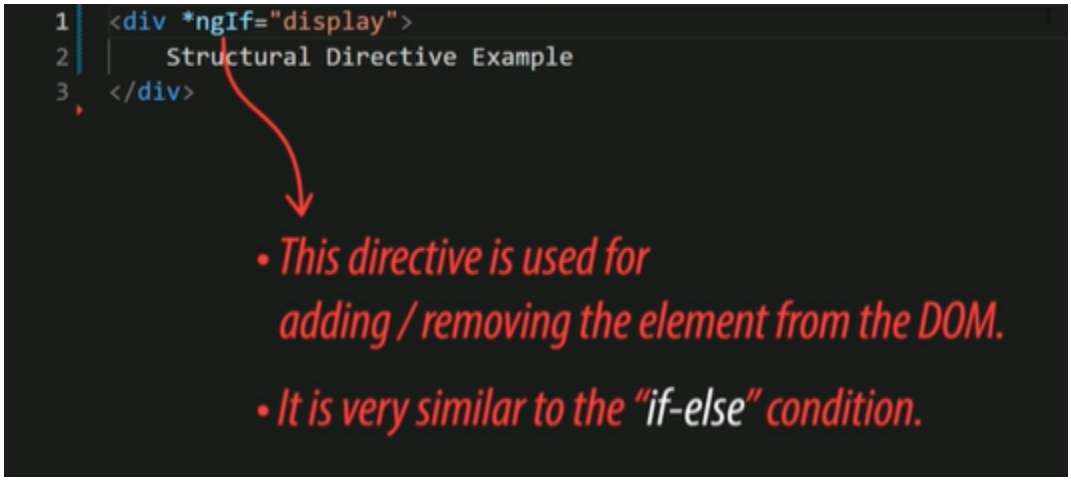

**Structural directives**

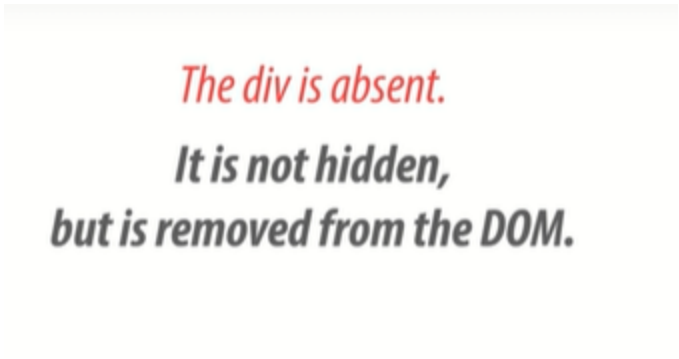


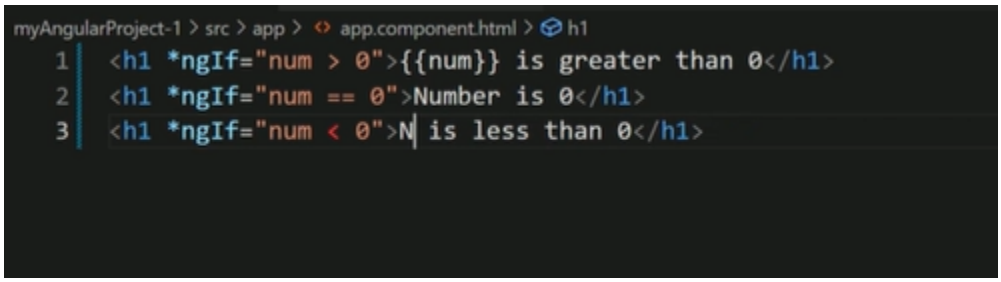

**Defining the structural directives in HTML elements**

Structural directives are defined the same way as we define attributes inside the element.






**QUESTION**

# 1. What is the purpose of *ngIf directive & how does it work?

The *ngIf directive
is a structural directive which is used
to conditionally render the elements
in HTML.

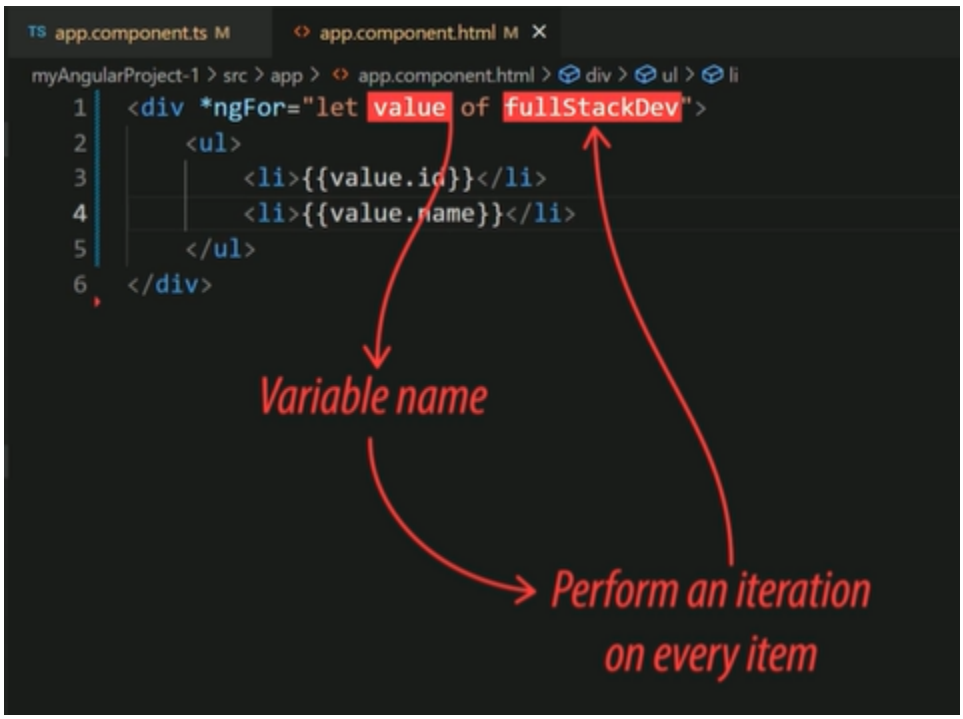
Just like an 'if-else' condition,
it takes the condition value;
if that condition meets the requirement;
the element acts based on that
or else the condition sets to false.

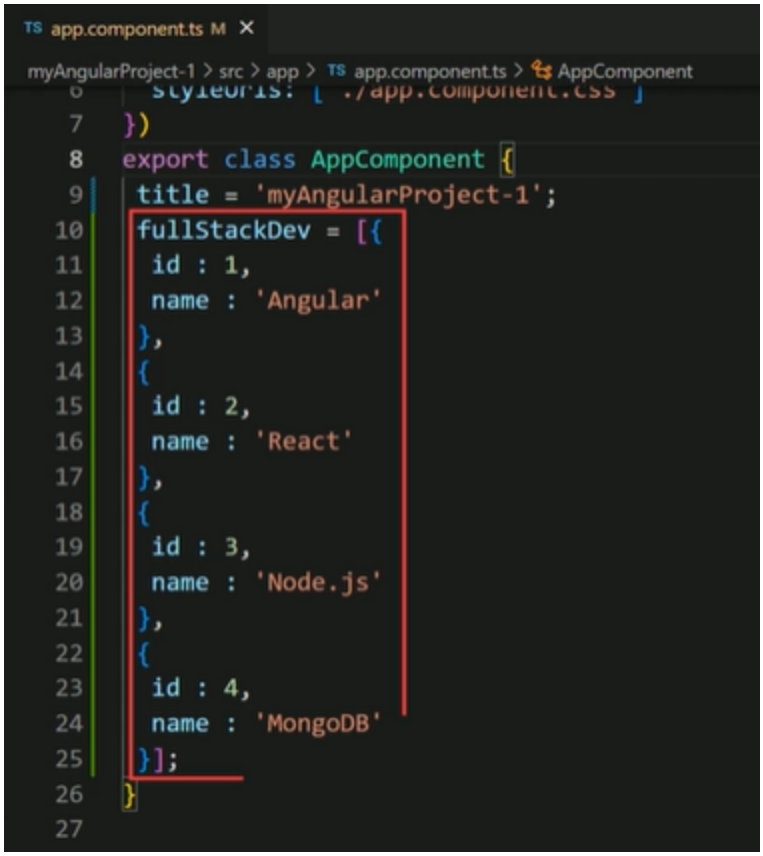

## @if & @else

*This directive helps us to implement most of the common tasks such as hide & show elements on the page or rendering of the components by applying some conditional rendering logic etc.*

- The ***ngIf** is a separate directive written itself in Angular using the **@Directive**.
- The **@if** is part of the template engine itself, like the interpolation syntax **{{variable}}**.

**For**






Using *ngFor directive
makes displaying of the items
way easier compared to
the traditional *"for"* or other loops
which you use for DOM traversal.

```
<ng-container *ngFor="let user of users;
trackBy: trackByIndex">
    {{ user.name }}
</ng-container>
```

*It helps Angular to identify which items in the collection have changed & need to be updated.*

*Angular 17 introduces new built-in control flow statements, enhancing code readability & maintainability.*

```
@for (user of users; track user.id) {
    <p>{{ user.name }}</p>
} @empty {
    <p>Empty list of users</p>
}
```

- *It can be directly specified inside the template.*
- *It offers a looping mechanism.*
- *More concise & executed quickly compared to the *ngFor directive.*