

What is Angular?

- Popular open-source JavaScript/TypeScript framework
- Allows to build dynamic, single-page applications with ease
- Powerful features & robust architecture

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">  
<p>Hello, {{name}}!</p>
```

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.



A screenshot of a terminal window with the following text:

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

PS C:\Project\Angular> **npm i -g**

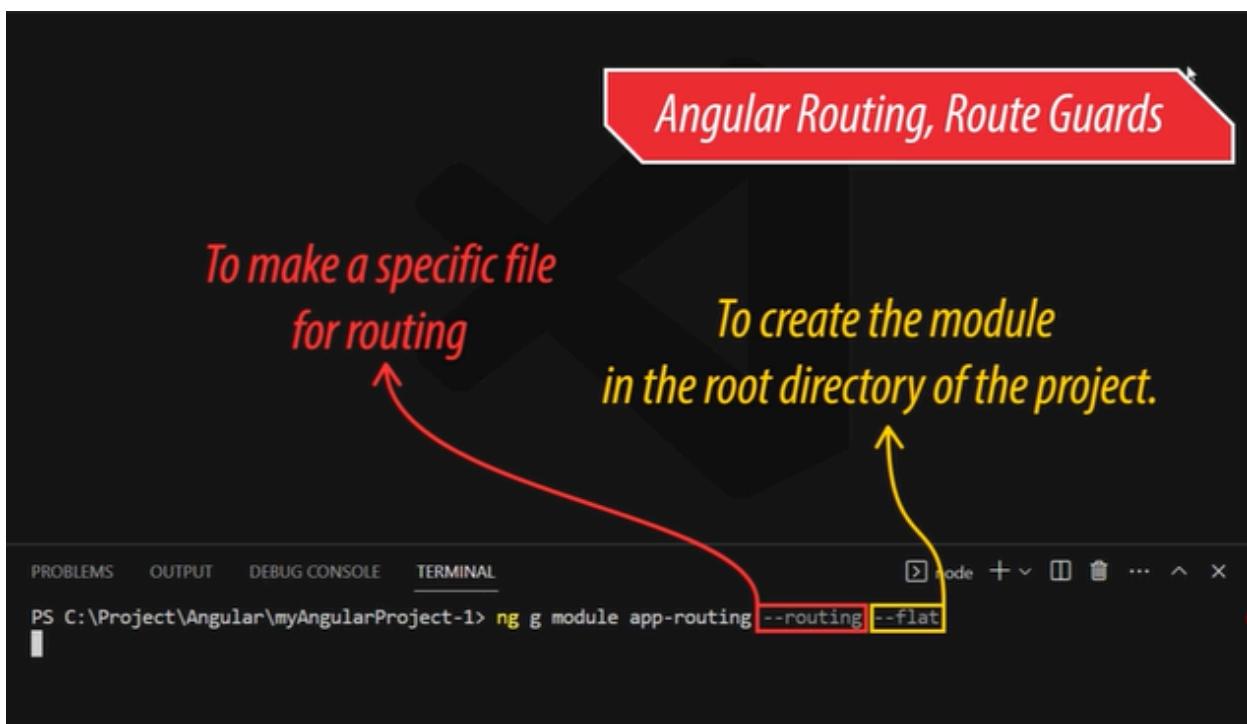
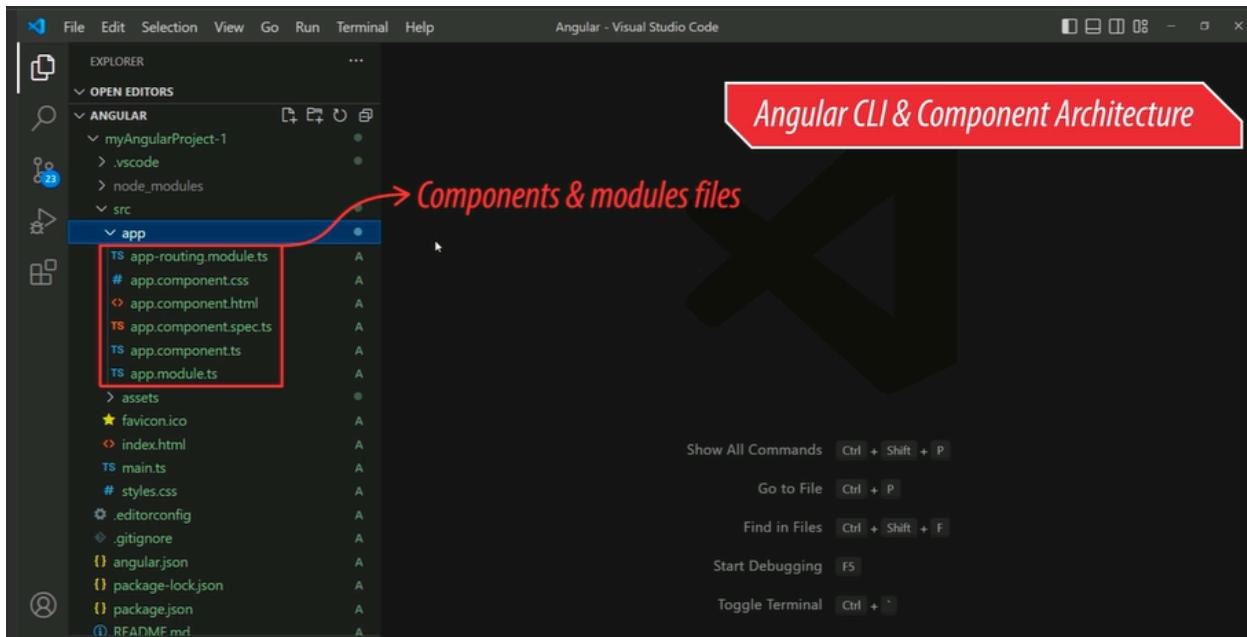
Annotations in red:

- An arrow points from the word "Node" to the "n" in "npm".
- An arrow points from the word "Package" to the "p" in "npm".
- An arrow points from the word "Manager" to the "m" in "npm".
- An arrow points from the word "Install" to the "i" in "npm".
- An arrow points from the "-g" in "npm i -g" to the text "Global installation".

A screenshot of a terminal window with the following text:

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

PS C:\Project\Angular> **npm i -g @angular/cli**





Angular Concepts

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

Header Component

Adding search bar

Category dropdown

Links & cart icons

Components creation

Error 404 page

Styling product card

Creating database

Configuring API Route



eStore Project

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

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

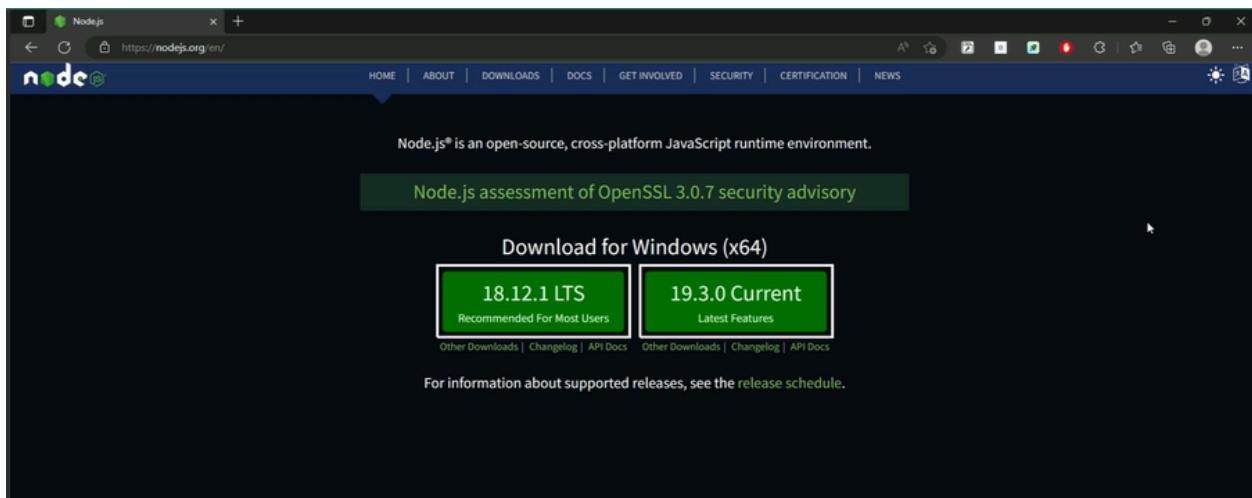
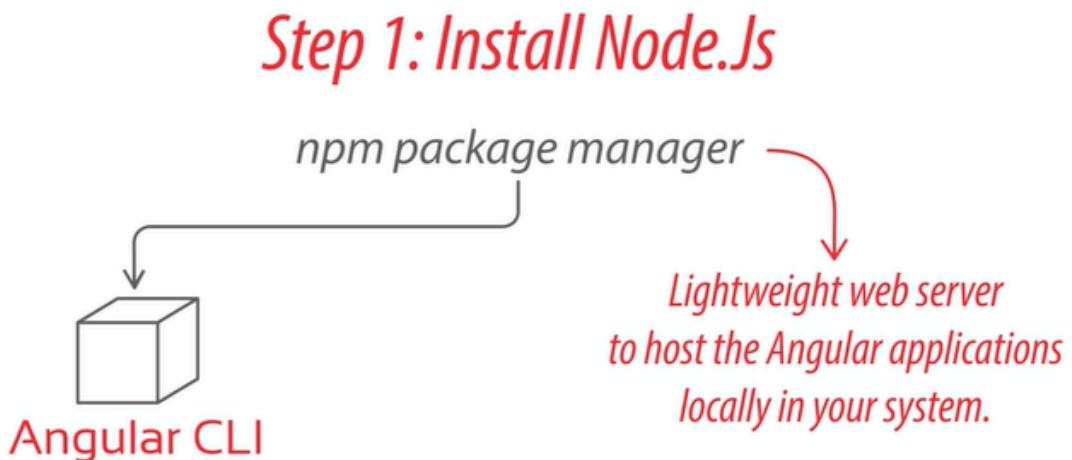
Add
additional features

Test
the Angular application

How to set up the Angular CLI in your project ?

Step 1: Install Node.js

npm package manager
that comes with Node.js, is needed
for various library installations.



Install Nodejs

*To install the Angular CLI
you can execute the command
in the cmd or you can use
the VS code terminal.*

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
PS C:\Project\Angular> node -v
v18.12.1
PS C:\Project\Angular>
```

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

Global installation

Why we need the global installation?

CLI we need not only 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.

The screenshot shows the VS Code interface with the following details:

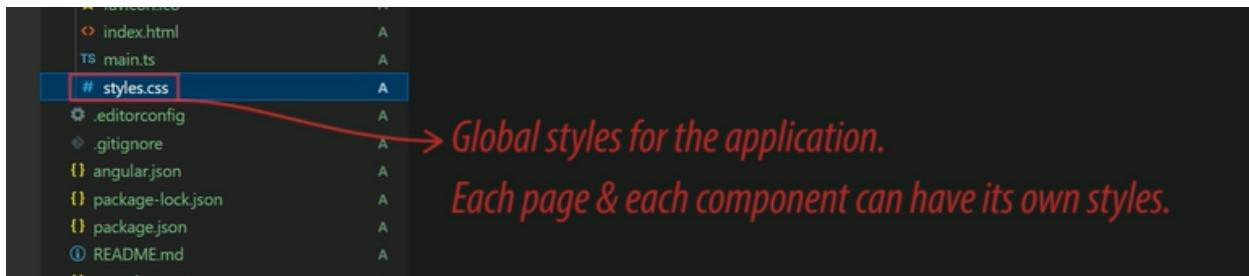
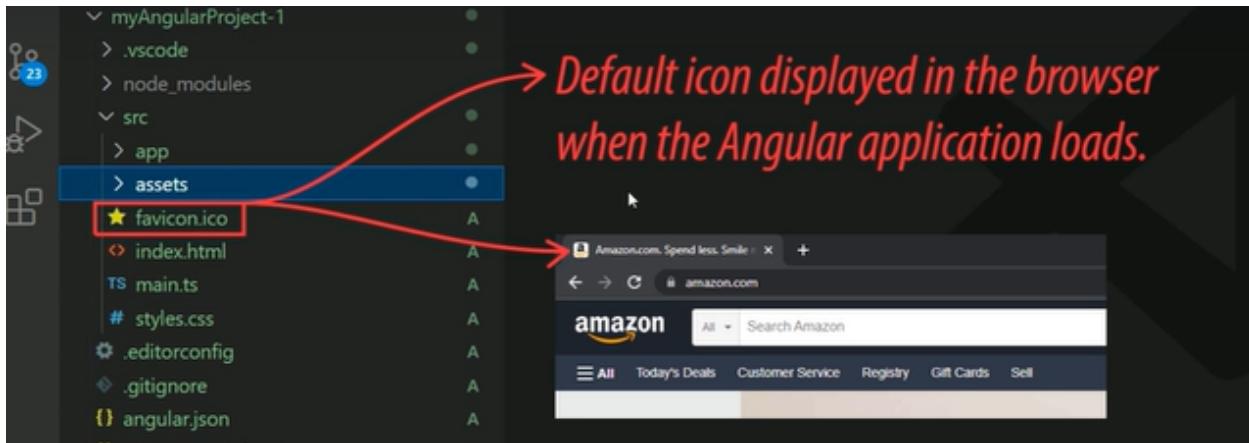
- EXPLORER** pane on the left, expanded to show the project structure of "myAngularProject-1". It includes files like .vscode, src, .editorconfig, .gitignore, angular.json, package.json, README.md, tsconfig.app.json, tsconfig.json, tsconfig.spec.json, and NOTES.txt.
- TERMINAL** pane on the right, titled "Angular - UDEMY". It displays the output of a command, likely "ng new myAngularProject-1", showing the creation of various files and folders. The output includes:

```
CREATE myAngularProject-1/angular.json (2749 bytes)
CREATE myAngularProject-1/package.json (1089 bytes)
CREATE myAngularProject-1/README.md (1098 bytes)
CREATE myAngularProject-1/tsconfig.json (889 bytes)
CREATE myAngularProject-1/.editorconfig (290 bytes)
CREATE myAngularProject-1/.gitignore (629 bytes)
CREATE myAngularProject-1/tsconfig.app.json (277 bytes)
CREATE myAngularProject-1/tsconfig.spec.json (287 bytes)
CREATE myAngularProject-1/.vscode/extensions.json (134 bytes)
CREATE myAngularProject-1/.vscode/launch.json (490 bytes)
CREATE myAngularProject-1/.vscode/tasks.json (980 bytes)
CREATE myAngularProject-1/src/main.ts (256 bytes)
CREATE myAngularProject-1/src/favicon.ico (15086 bytes)
CREATE myAngularProject-1/src/index.html (316 bytes)
CREATE myAngularProject-1/src/styles.css (81 bytes)
CREATE myAngularProject-1/src/app/app.component.html (20239 bytes)
CREATE myAngularProject-1/src/app/app.component.spec.ts (981 bytes)
● CREATE myAngularProject-1/src/app/app.component.ts (327 bytes)
CREATE myAngularProject-1/src/app/app.component.css (0 bytes)
CREATE myAngularProject-1/src/app/app.config.ts (235 bytes)
CREATE myAngularProject-1/src/app/app.routes.ts (88 bytes)
CREATE myAngularProject-1/src/assets/.gitkeep (0 bytes)
.. Installing packages (npm)...
```

Structure of Angular Project

node_modules folder

- *Contains all the third-party libraries on which the Angular application depends upon.*
- *Purely used for the development purpose.*
- *All these libraries are put in a bundle & deployed in the application on compilation.*



```
myAngularProject-1 > src > index.html > ...
1  <!doctype html>
2  <html lang="en">
3  <head>
4  |  <meta charset="utf-8">
5  |  <title>MyAngularProject1</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 </body>
13 </html>
```

External script references & stylesheets are not yet defined as they will be inserted dynamically into index.html.

A screenshot of Visual Studio Code showing the file structure of an Angular project. The Explorer sidebar shows files like .vscode, node_modules, src (containing app, assets, favicon.ico, and index.html), and tsconfig files. The main.ts file is selected in the Explorer and is open in the editor. The code in main.ts is:

```
1 import { platformBrowserDynamic } from '@angular/platform-browser-dynamic';
2
3 import { AppModule } from './app/app.module';
4
5
6 platformBrowserDynamic().bootstrapModule(AppModule)
7   .catch(err => console.error(err));
```

A red arrow points from the text "A TypeScript file that contains the starting point of application." to the main.ts file in the Explorer.

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

In Angular 17

No AppModule - AppComponent with AppConfig

A screenshot of Visual Studio Code showing the file structure of an Angular project named "Angular - UDEMY". The Explorer sidebar shows files like .vscode, node_modules, src (containing app, assets, favicon.ico, and index.html), and tsconfig files. The main.ts file is selected in the Explorer and is open in the editor. The code in main.ts is:

```
1 import { bootstrapApplication } from '@angular/platform-browser';
2 import { appConfig } from './app/app.config';
3 import { AppComponent } from './app/app.component';
4
5 bootstrapApplication(AppComponent, appConfig)
6   .catch((err) => console.error(err));
```

The terminal below shows the command being run: "CREATE myAngularProject-1/src/app/app.routes.ts (80 bytes)" and "CREATE myAngularProject-1/src/assets/.gitkeep (0 bytes)".

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
myAngularProject-1
  .vscode
  node_modules
  src
  .editorconfig
  .gitignore
    angular.json
    package-lock.json
    package.json
    README.md
    tsconfig.app.json
    tsconfig.json
    tsconfig.spec.json
```

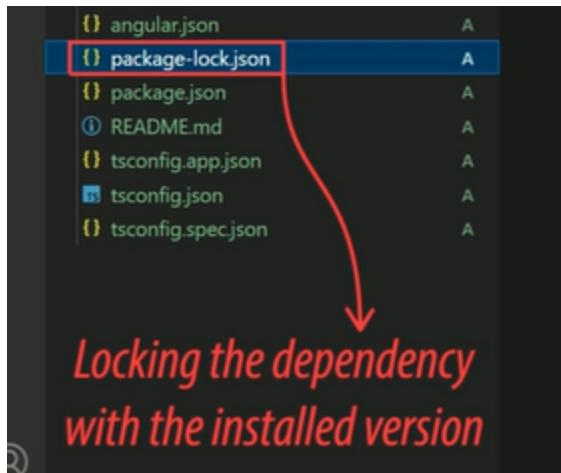
2
3 # Compiled output
4 /dist
5 /tmp
6 /out-tsc
7 /bazel-out
8
9 # Node
10 /node_modules
11 npm-debug.log
12 yarn-error.log
13
14 # IDEs and editors
15 .idea/
16 .project

Files & folders remain unreachable which contains support files & is not needed as a part of the code repository.

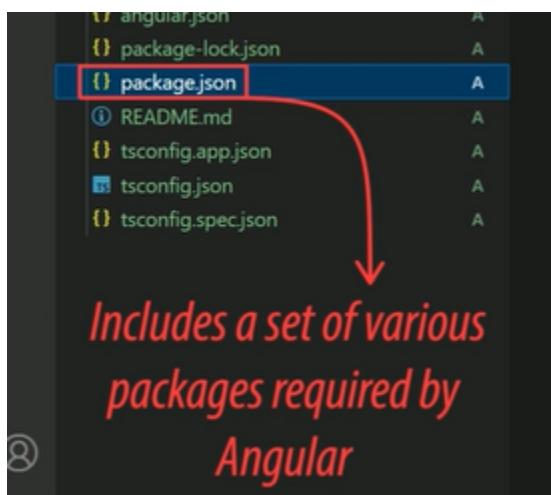
angular.json

Configuration of the project

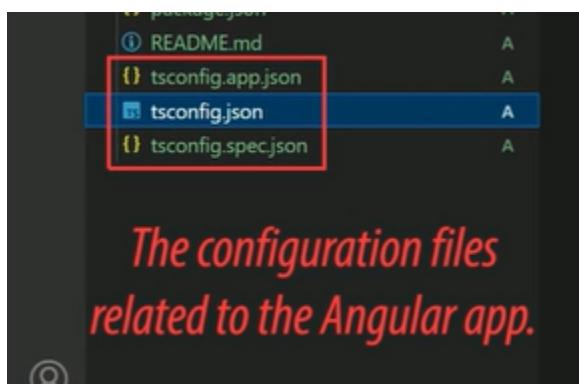
angular.json	A	8
package-lock.json	A	9
package.json	A	10
README.md	A	11
tsconfig.app.json	A	12
tsconfig.json	A	13
tsconfig.spec.json	A	14
		15
		16
		17
		18
		19
		20
		21



*Locking the dependency
with the installed version*



*Includes a set of various
packages required by
Angular*



*The configuration files
related to the Angular app.*

- Easier Server-side Rendering SSR
- New control flow syntax (@if, @for, @switch)
- Deferrable views (@defer)
- More Stable Signals
- Better build performance

Server-Side Rendering (SSR) in Angular 17

Server-Side Rendering (SSR)

```
ng new myApp --ssr
```

It does not require any complex or additional setups to add the SSR feature to your application.

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

Select Role

Admin

HR

✓ Marketing

Developer

Analyst

Selected Role: Marketing

ID	Name	Role
3	EmployeeC	Marketing
5	EmployeeE	Marketing
6	EmployeeF	Marketing

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.

```

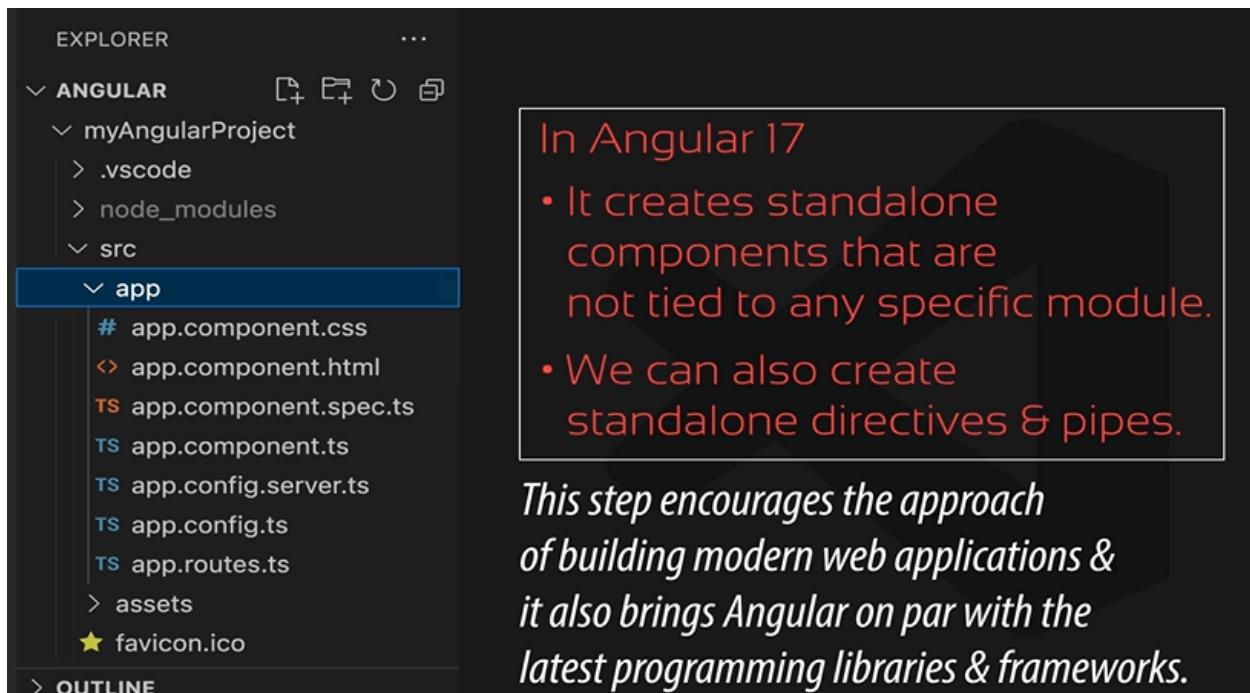
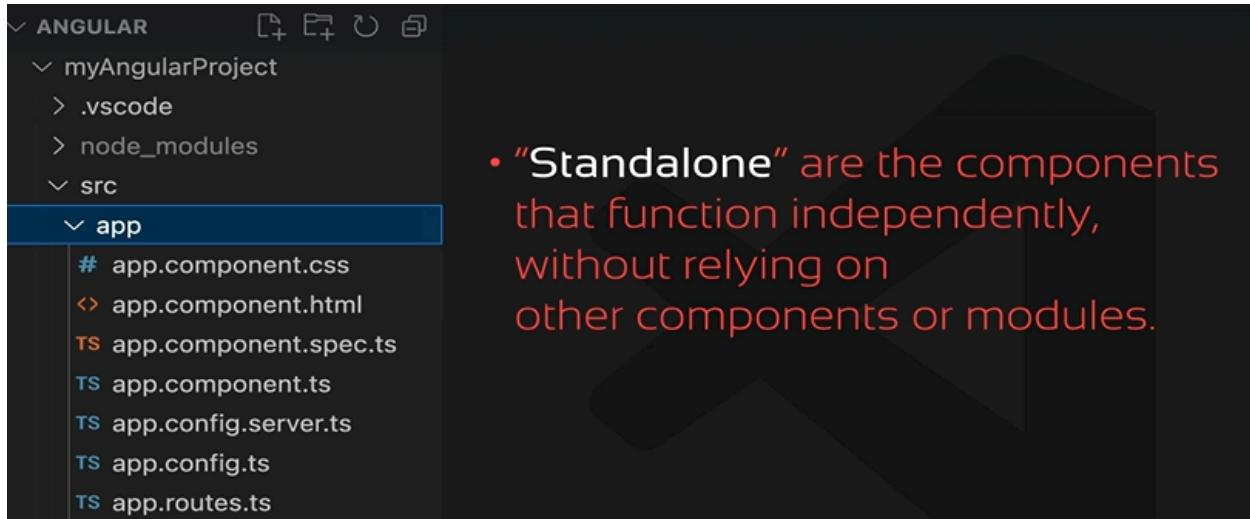
○ nirmaljoshi@Nirmals-MacBook-Pro:Angular % ng new myAngularProject
? Which stylesheet format would you like to use? CSS
? Do you want to enable Server-Side Rendering (SSR) and Static Site Generation
(SSG/Prerendering)? (y/N) y

```

*Selecting “No” option will just create a simple CSR
i.e., the client-side rendering application.*

From Angular 17,
Angular is going to create
standalone components
by default.

Standalone Components



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.*

```
myAngularProject > src > app > ts app.component.ts > ...  
1 import { Component } from '@angular/core';  
2 import { CommonModule } from '@angular/common';  
3 import { RouterOutlet } from '@angular/router';  
4  
5 @Component({  
6   selector: 'app-root',  
7   standalone: true,  
8   imports: [CommonModule, RouterOutlet],  
9   templateUrl: './app.component.html',  
10  styleUrls: ['./app.component.css'],  
11})  
12 export class AppComponent {  
13   title = 'myAngularProject';  
14}  
15
```

Component A

```
@Component({  
  standalone: true  
  ...  
})
```

Component B

```
@Component({  
  standalone: true  
  ...  
})
```

Ng serve -- open

Two Way Binding

HTML

```
<label for="username">User Name</label>&nbsp;&nbsp;  
  
<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
```

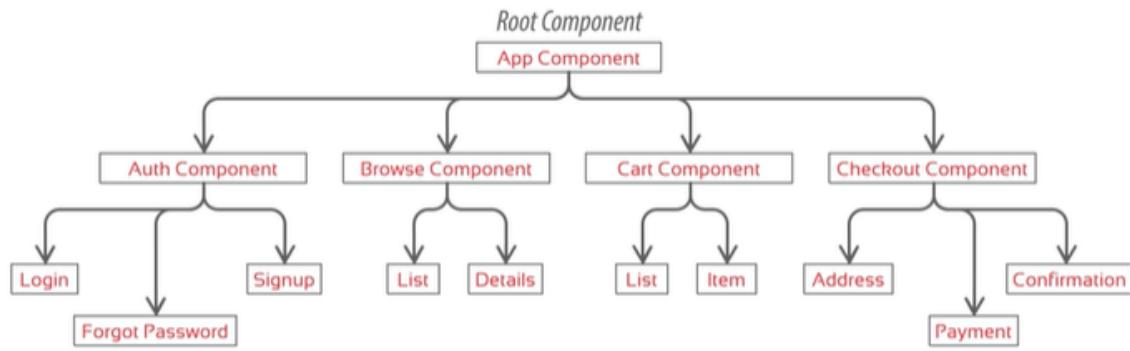
Migrate the angular syntax to angular 17 including the Control flow

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

```

myAngularProject-1 > 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 = 'myAngularProject-1';
10 }
11
  
```

*Imports
"component" decorator
from
"@angular/core" library
inside the
"node_modules" folder*

```

@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent {
  title = 'myAngularProject-1';
}
  
```

*Decorator
A function
which modifies
a class or property definition*

@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

The component name inside the selector property, must be the same name inside the index.html file.

The screenshot shows the file structure of an Angular project named 'myAngularProject'. The 'src' directory contains an 'app' folder which includes files like 'app.component.css', 'app.component.html', 'app.component.spec.ts', 'app.component.ts', 'app.config.server.ts', 'app.config.ts', 'app.routes.ts', and 'index.html'. There are also 'assets' and 'favicon.ico' files. The 'index.html' file is shown with its code:

```
myAngularProject > src > index.html > ...
1  <!doctype html>
2  <html lang="en">
3  <head>
4  | <meta charset="utf-8">
5  | <title>MyAngularProject</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 </body>
13 </html>
14
```

Change name to app-root to app

app.component.ts

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

Index.html

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

It will work.

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

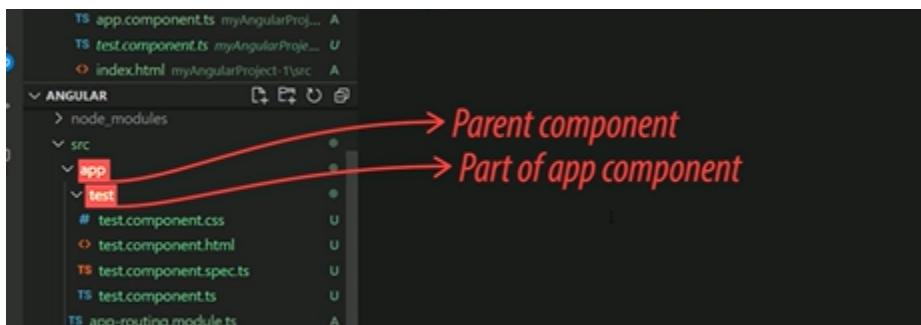
App component Is the root component - entry to the angular application we can change it.

Create new components

Ng g c <component-name>

Ng generate component <component-name>

index.html generally contains all the global components or declarations & libraries incorporated.



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).

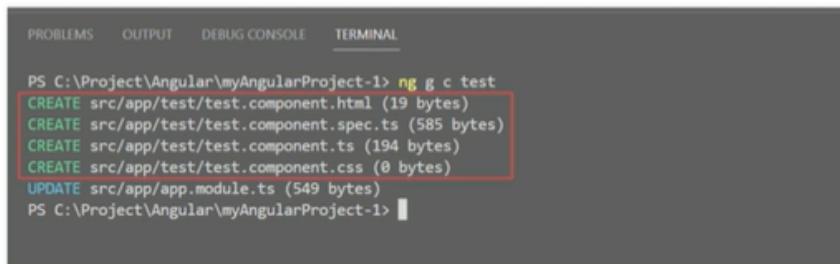
A component based architecture basically involves defining the components, which are the basic building blocks of any Angular application.

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.



```
PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL

PS C:\Project\Angular\myAngularProject-1> ng g c test
CREATE src/app/test/test.component.html (19 bytes)
CREATE src/app/test/test.component.spec.ts (585 bytes)
CREATE src/app/test/test.component.ts (194 bytes)
CREATE src/app/test/test.component.css (0 bytes)
UPDATE src/app/app.module.ts (549 bytes)
PS C:\Project\Angular\myAngularProject-1>
```

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



Displaying Message Dynamically (Interpolation)

Relative Questions : 1

Interpolation

It refers to embedding expressions into marked up text by using the double curly braces {{e.g.}} as delimiters.

The screenshot shows a Visual Studio Code interface with three panes. The left pane shows a browser window at 'localhost:4200' displaying the text 'Angular 14' and 'Hello, Welcome to this page'. The middle pane shows the 'app.component.html' file with two lines of code: '

{{title}}

' and '

{{msg}}

'. The right pane shows the 'app.component.ts' file with the following code:

```
myAngularProject-1 > src > app > app.component.ts > AppComponent > AppComponent
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 = 'Angular 14';
10  msg = 'Hello, Welcome to this page';
11 }
12
```

Red arrows point from the text in the browser to the corresponding interpolation expressions in the HTML template.

Question - Explain the term interpolation

Interpolation means displaying the variables you have in the TypeScript file inside the HTML. You have to use the double curly braces that displays the value dynamically.

This → accessing class members of the component

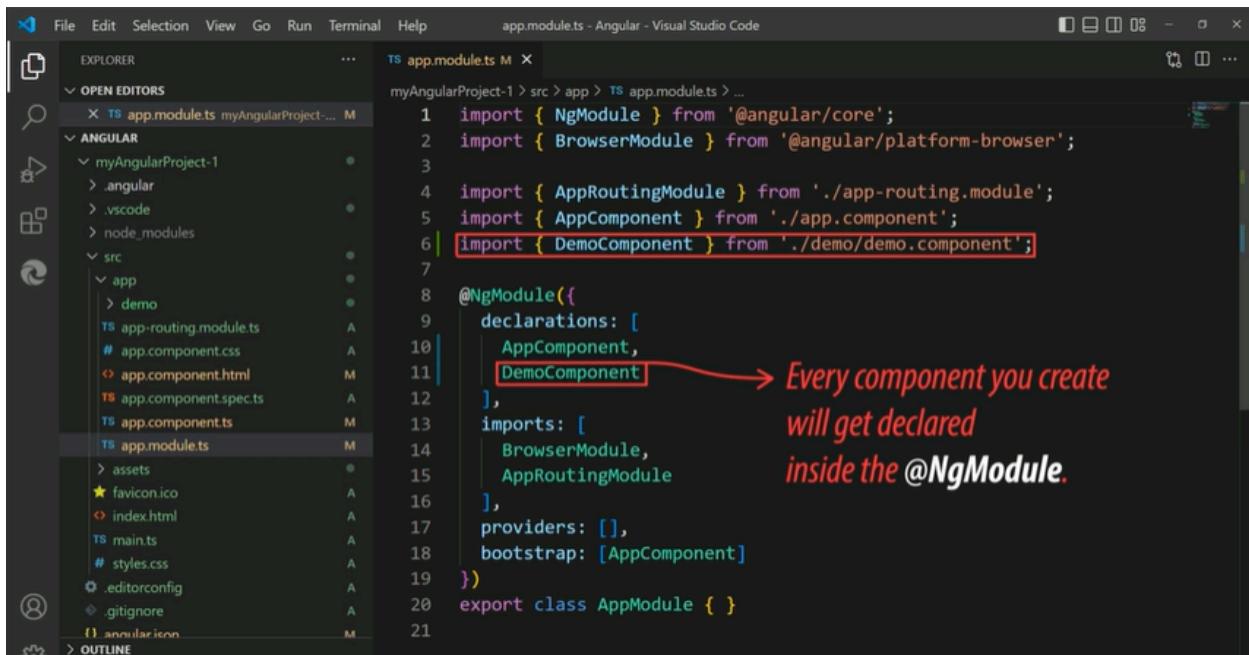
```

ts app.component.ts M X
myAngularProject-1 > src > app > ts app.component.ts > AppComponent > ans
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   a = 10;
10  b = 5;
11  ans = this.a + this.b;
12 }
13
→ Class members of "AppComponent"

```

*"this" keyword is used
to read the values from a property/variable
attached to it.*

Before - angular 17 - app.module.ts



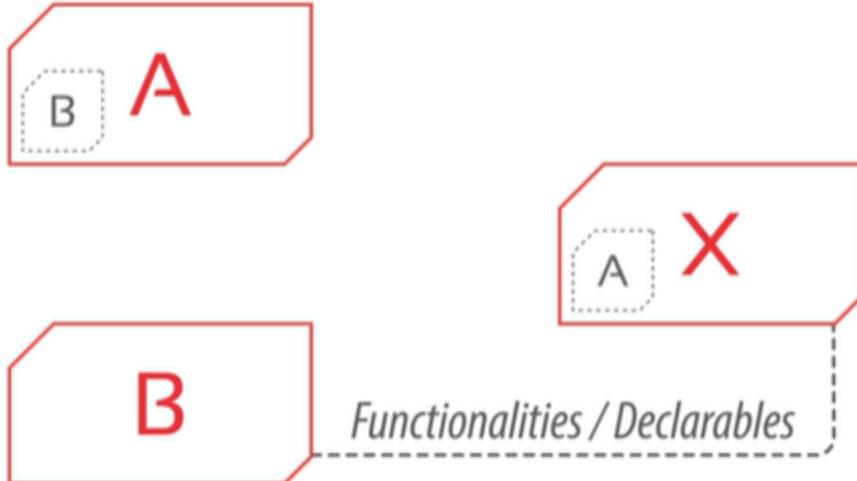
```

File Edit Selection View Go Run Terminal Help app.module.ts - Angular - Visual Studio Code
EXPLORER ... ts app.module.ts M X
myAngularProject-1 > src > app > ts app.module.ts > ...
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 { DemoComponent } from './demo/demo.component';
7
8 @NgModule({
9   declarations: [
10     AppComponent,
11     DemoComponent
12   ],
13   imports: [
14     BrowserModule,
15     AppRoutingModule
16   ],
17   providers: [],
18   bootstrap: [AppComponent]
19 })
20 export class AppModule { }

```

*Every component you create
will get declared
inside the @NgModule.*

Imports in module.ts



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

Any module to be defined inside the @NgModule imports property, has to be explicitly imported first.

Bootstrap - first app to run

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

*→ Root component
By default it will always be
the **AppComponent**.
But displaying different
components is also possible.*

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

Question

1. What is the purpose of the decorator @NgModule?

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)
```

```

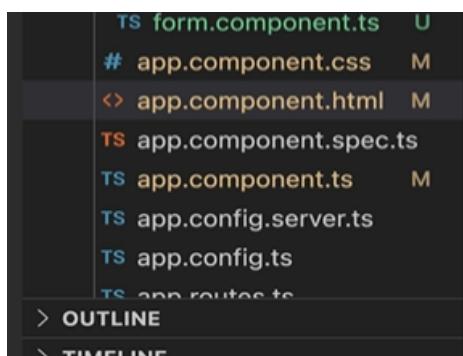
@Component({
  selector: 'app-form',
  standalone: true,
  imports: [FormsModule],
  templateUrl: './form.component.html',
  styleUrls: ['./form.component.css'],
})
export class FormComponent {
  name: string = '';
}

```

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



In previous versions

```

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

```

Bootstrapping in 17 version

```
myAngularProject > src > TS main.ts > ...
1 import { bootstrapApplication } from '@angular/platform-browser';
2 import { appConfig } from './app/app.config';
3 import { AppComponent } from './app/app.component';
4 import { FormComponent } from './app/form/form.component';
5
6 bootstrapApplication(FormComponent, appConfig).catch((err) =>
7   console.error(err)
8 );
9

myAngularProject > src > TS main.server.ts > ...
1 import { bootstrapApplication } from '@angular/platform-browser';
2 import { AppComponent } from './app/app.component';
3 import { config } from './app/app.config.server';
4 import { FormComponent } from './app/form/form.component';
5
6 const bootstrap = () => bootstrapApplication(FormComponent, config);
7
8 export default bootstrap;
```

Same thing in the main.server.ts file

To change the bootstrapping component in angular17 version

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

```
src > TS main.ts > ...
1 import { bootstrapApplication } from '@angular/platform-browser';
2 import { appConfig } from './app/app.config';
3 // import { AppComponent } from './app/app.component';
4 import { TestComponent } from './app/test/test.component';
5
6 // bootstrapApplication(AppComponent, appConfig)
7 //   .catch((err) => console.error(err));
8
9
10 bootstrapApplication(TestComponent, appConfig)
11   .catch((err) => console.error(err));
```

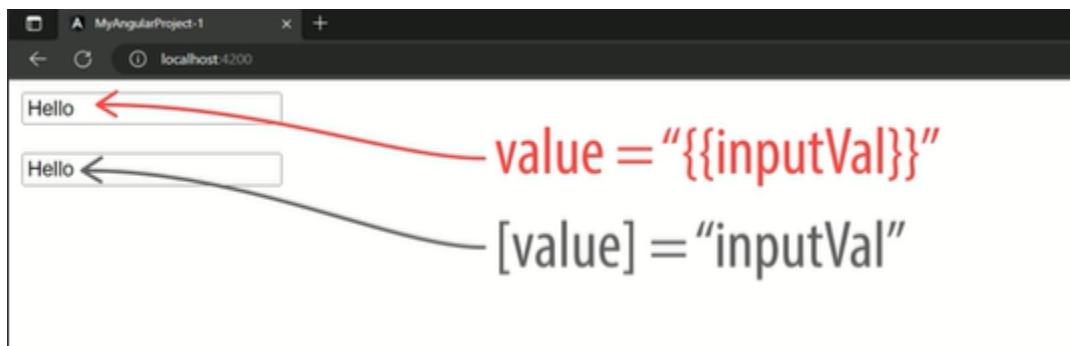
```
app.component.html app.component.ts main.ts index.html

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 app.component.ts

myAngularProject-1 > src > app > app.component.html > input
1 <input type="text" value="{{inputVal}}"/>
2 <br/><br/>
3 <input type="text" [value]="inputVal"/>
```



```
↳ app.component.html M X  TS 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"/>
```

You can bind
any property like



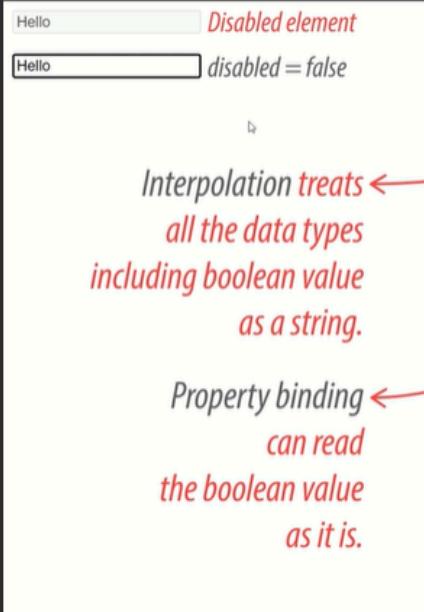
DOM property = "value"

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

```
↳ app.component.html M X  TS 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"/>
```

interpolation ≠ property binding

Main difference between interpolation and property binding



Disabled element

```
app.component.html M x ts app.component.ts M
yAngularProject-1 > src > app > app.component.html > input
1 | <input type="text" value="{{inputVal}}" disabled="{{isDisabled}}"/>
2 | <br/><br/>
3 | <input type="text" [value]="inputVal" [disabled]="isDisabled"/>
```

Interpolation treats all the data types including boolean value as a string.

Property binding can read the boolean value as it is.

Help app.component.ts - Angular - Visual Studio Code

```
app.component.ts M x
yAngularProject-1 > src > app > app.component.ts > AppComponent > isDisabled
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 = 'myAngularProject-1';
10  inputVal = 'Hello';
11  isDisabled = false;
12 }
```



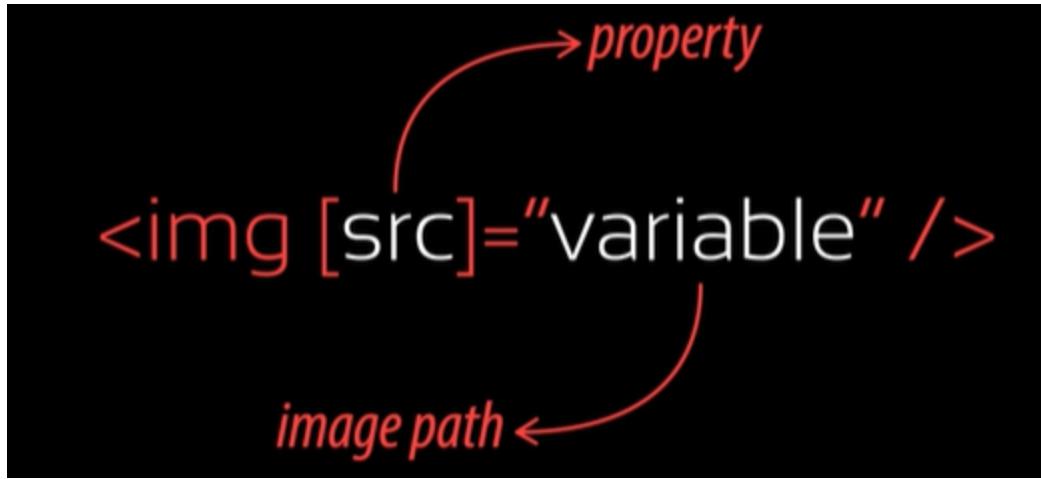
Using interpolation with properties is not recommended.

Help app.component.html - Angular - Visual Studio Code

```
app.component.html M x ts app.component.ts M
yAngularProject-1 > src > app > app.component.html > input
1 | <input type="text" value="{{inputVal}}" disabled="{{isDisabled}}"/>
2 | <br/><br/>
3 | <input type="text" [value]="inputVal" [disabled]="isDisabled"/>
```

Help app.component.ts - Angular - Visual Studio Code

```
app.component.ts M x
yAngularProject-1 > src > app > app.component.ts > AppComponent > isDisabled
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 = 'myAngularProject-1';
10  inputVal = 'Hello';
11  isDisabled = false;
12 }
```



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] = "variable" ←

Property binding is an approach where you set properties for the HTML elements or directives to do things like toggle button features or set paths programmatically or share values among the components.

Property binding with img tag

```
app.component.html M
myAngularProject-1 > src > app > app.component.html >
1 <img [src]="imgSrc" alt="" />

app.component.ts M TS app.component.ts M
myAngularProject-1 > src > app > app.component.ts > AppCompon
1 import { Component } from '@angular/c
2
3 @Component({
4   selector: 'app-root',
5   templateUrl: './app.component.html'
6   styleUrls: ['./app.component.css']
7 })
8 export class AppComponent {
9   title = 'myAngularProject-1';
10  imgSrc = '/assets/Angular.jpg';
11 }
12
```

Event Binding with click event

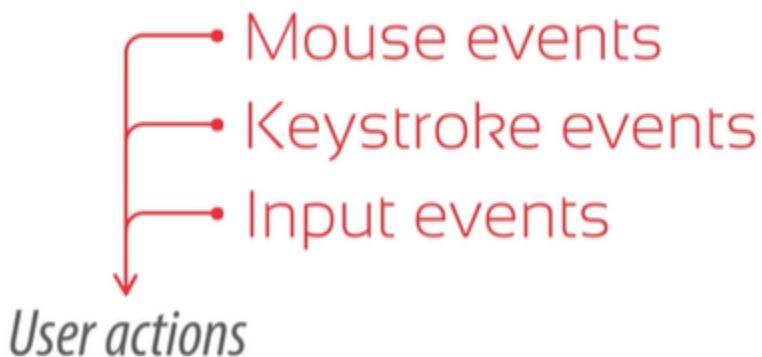
In HTML DOM there are several events which allow JavaScript to register different event handlers on elements in an HTML document.

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



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

```
myAngularProject-1 > src > app > app.component.html > button  
1 | <button type="submit" (click)="">Click</button>
```

JavaScript: onclick event

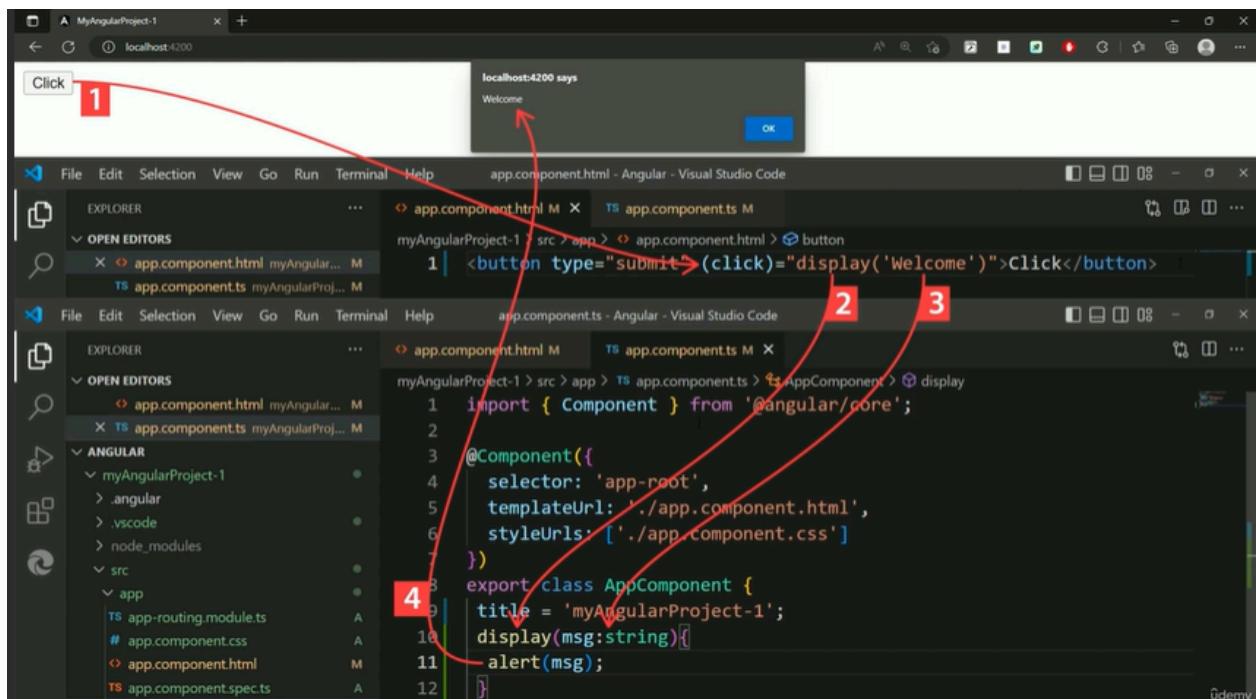
Angular: click event

```
1 | <button type="submit" (click)="display()">Click</button>
```

*When the button
is clicked*

*This function is
going to get called*

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

Mouse related events

click *dblclick*

mouseover *mouseleave*

mousedown *mouseup*

drag *dragover*

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

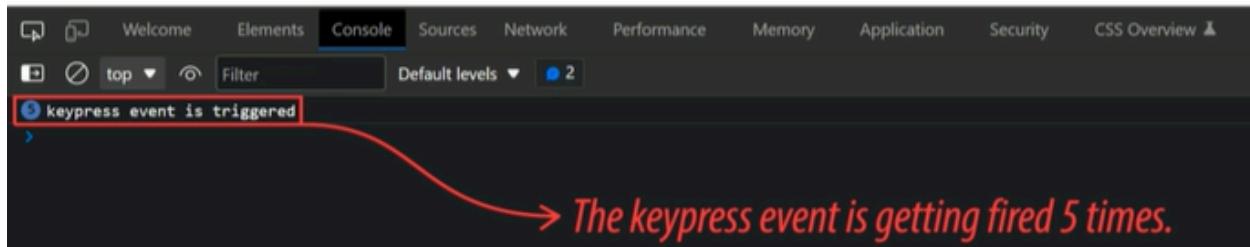
Keyboard related events

<i>keypress</i>	<i>select</i>
<i>keydown</i>	<i>keyup</i>
<i>focus</i>	<i>input</i>
<i>cut</i>	<i>copy</i>
<i>paste</i>	<i>blur</i>



\$event object - Angular

The event object (\$event)
is a reserved keyword
to handle the events in Angular.



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 count : 0

Sevent object

He : He

Angular is running in development mode.

core.js:29869 app.component.ts:65

KeyboardEvent {isTrusted: true, key: 'H', code: 'KeyH', location: 0, ctrlKey: false, ...}

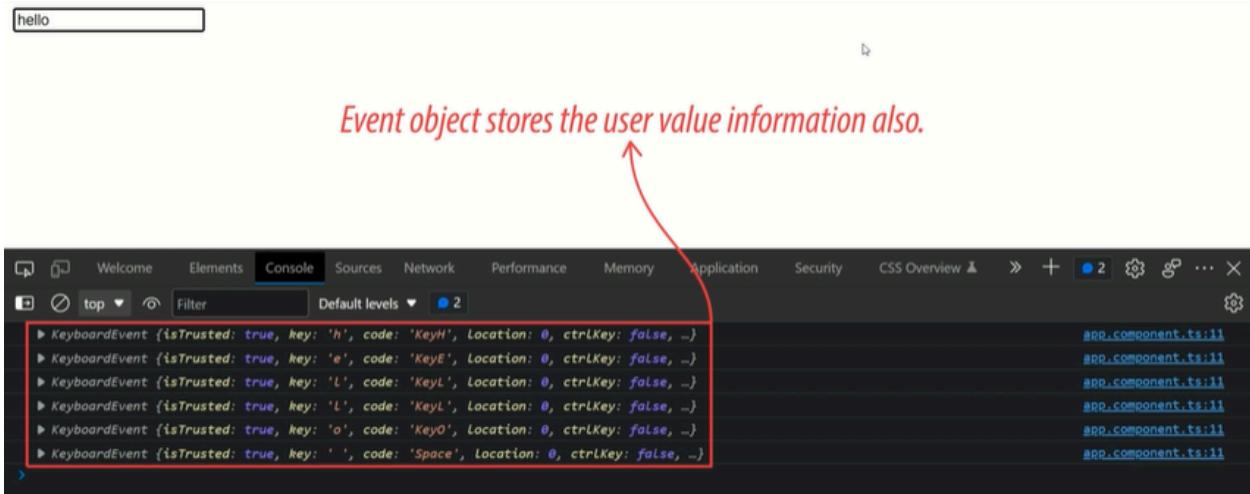
KeyboardEvent {isTrusted: true, key: 'e', code: 'KeyE', location: 0, ctrlKey: false, ...}

Console What's new X

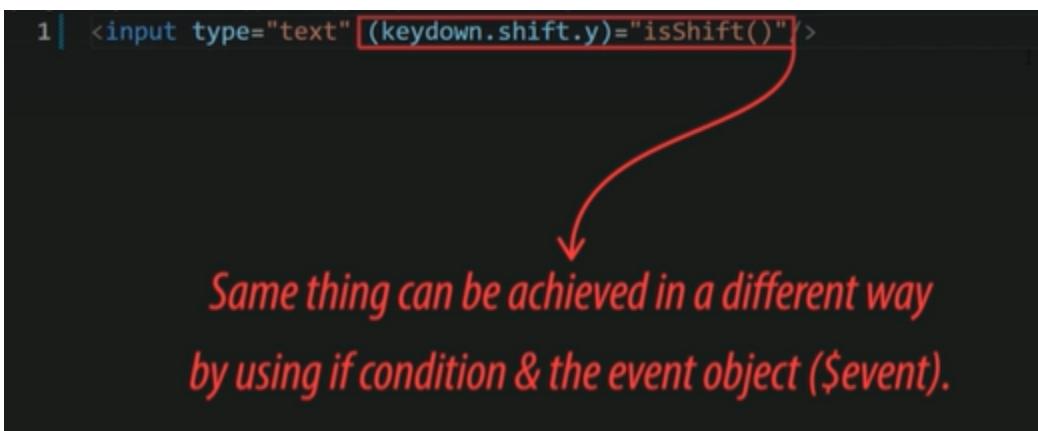
Highlights from the Chrome 124 update

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



```
@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent {
  title = 'myAngularProject-1';
  isShift(event:any){
    if(event.shiftKey && event.key === 'Y'){
      console.log('Shift + Y is pressed',event);
    }
  }
}
```

If the Shift key is pressed with the Key Y.

Trigger the event & display the message.

```
export class AppComponent {
  title = 'myAngularProject-1';
  isShift(event:any){
    if(event.shiftKey && event.key === 'Y'){
      console.log('Shift + Y is pressed',event);
    }
  }
}
```

Other keys with Alt & Ctrl can be checked.

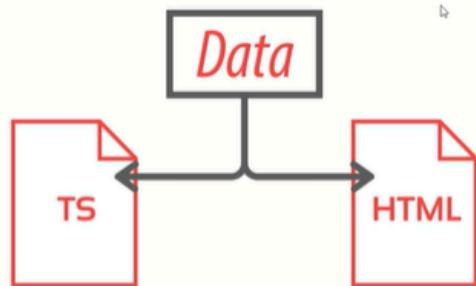
Two Way Binding

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.component.html > input  
1 | <input type="text"/>
```

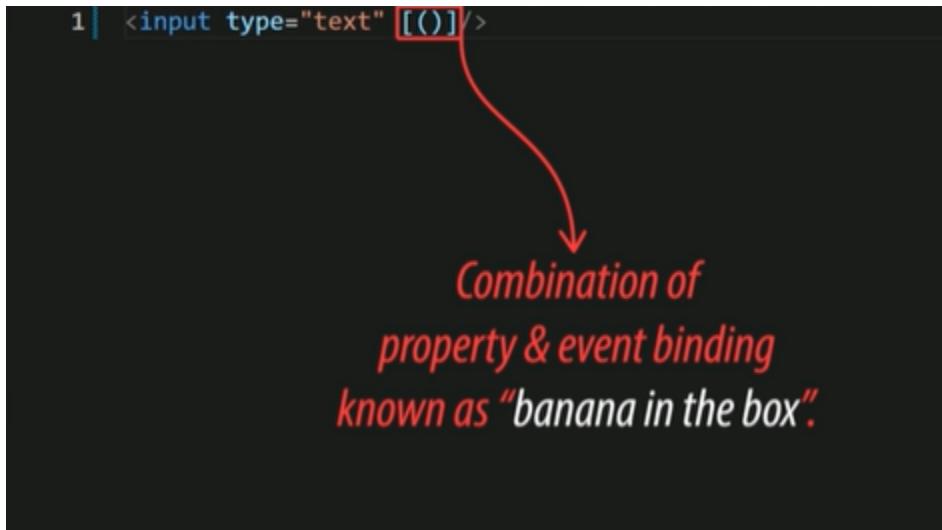
1 Import a module which controls & handles the form related implementations.

Like input, select, textarea or other form controls

```
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 { }
```

1
2

Syntax - two way binding



```
1 | <input type="text" [(ngModel)]/>
```

ngModel is not part of
Angular core library,
but *is a part of*
FormsModule library.

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.



Assignment (Two-way Binding) - Get input field value on button click

Get input field value on button click

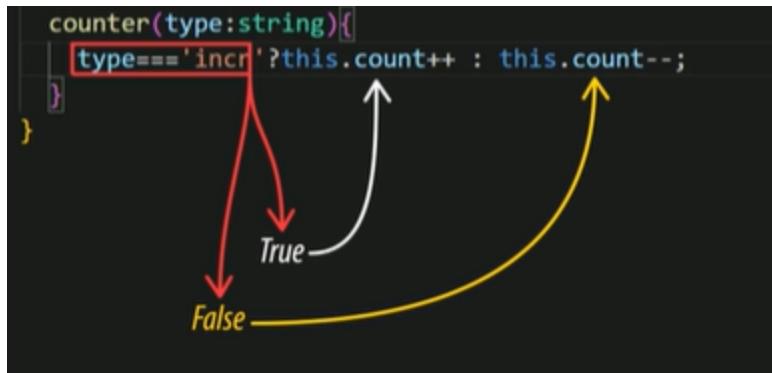
```
myAngularProject-1 > src > app > app.component.html > input
1 | <input type="text" [(ngModel)]="inputVal"/>
2 | <br/><br/>
3 | <button type="submit" (click)="show()">Show</button>

export class AppComponent {           component.ts
  title = 'myAngularProject-1';
  inputVal:string = '';
  dynamicVal:string = '';

  show(){
    this.dynamicVal = this.inputVal;
  }
}
```

```
export class AppComponent {
  title = 'myAngularProject-1';
  count = 0;
  counter(type:string){
    type==='incr'?this.count++ : this.count--;
  }
}
```

→ Strict equality operator



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

6 Binding other mouse related events

7 Binding keyboard events

8 \$event object

8.1 Assignment - Checking Shift key status.

9 Two-way Binding (ngModel)

9.1 What is two-way binding & why do we use it?

9.2 Explain ngModel

9.3 Assignment - (Two-way Binding) Get input field value on button click

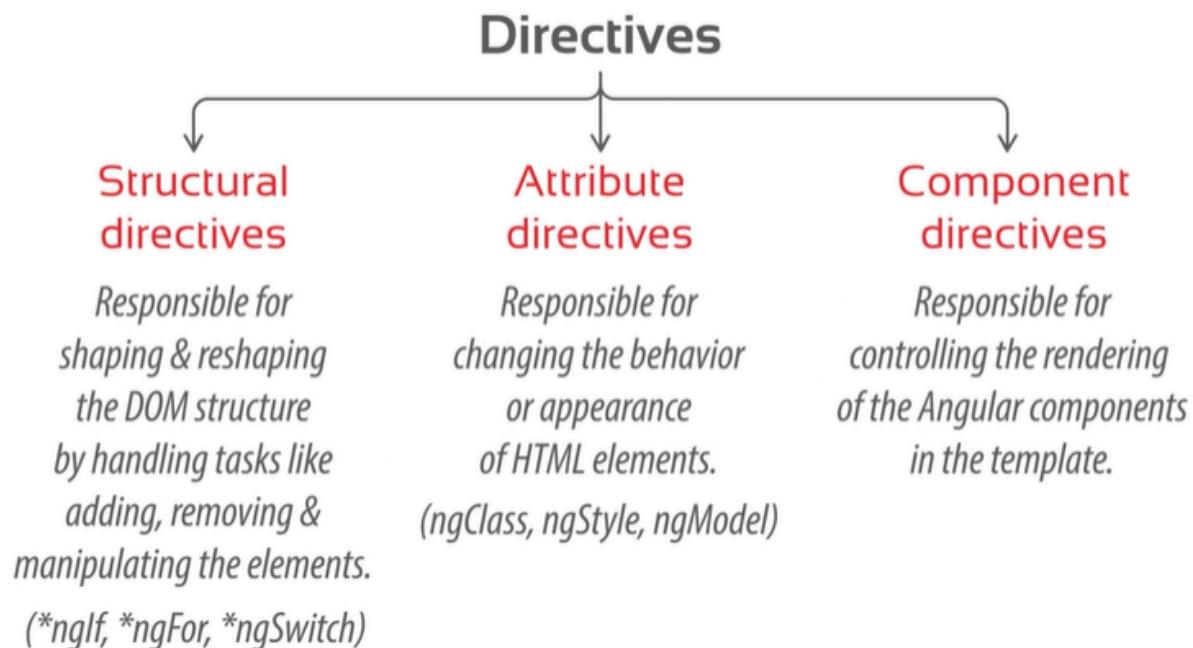
10 Creating Counter Example

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:

Directives

- One of the most important and core features in Angular.
- A set of classes, used for adding some additional behavior to the elements / the components of the Angular application.
- E.g. Adding/Removing an element from the DOM.



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:

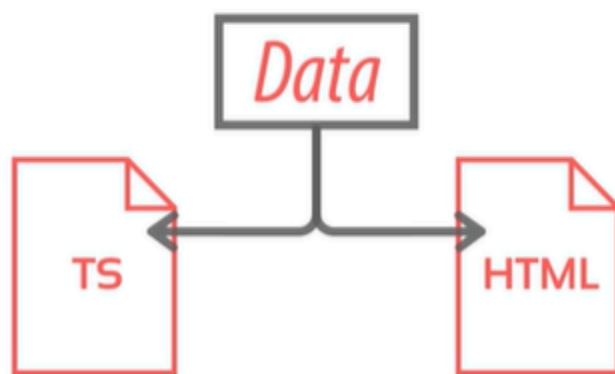
- Components: Directives with templates.
- Attribute Directives: Modify element behavior/appearance.
- Structural Directives: Alter the DOM layout.

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

Directive for two way binding

There are various built-in directives available in Angular.

[(ngModel)]



Structural directives

3 most commonly used structural directives are

Prefixed with the "" sign*

*ngIf
*ngFor
*ngSwitch

Defining the structural directives in HTML elements

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

```
1 | <div *ngIf="display">  
2 |   Structural Directive Example  
3 | </div>
```

- This directive is used for adding / removing the element from the DOM.
- It is very similar to the "if-else" condition.

```
1 | <div *ngIf="display">  
2 |   Structural Directive Example  
3 | </div>
```

- This directive is used for adding / removing the element from the DOM.
- It is very similar to the "if-else" condition.

The div is absent.

*It is not hidden,
but is removed from the DOM.*

```
myAngularProject-1 > src > app > app.component.html > h1  
1 | <h1 *ngIf="num > 0">{{num}} is greater than 0</h1>  
2 | <h1 *ngIf="num == 0">Number is 0</h1>  
3 | <h1 *ngIf="num < 0">N is less than 0</h1>
```

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

The screenshot shows a code editor with two tabs: `app.component.ts` and `app.component.html`. The `app.component.html` tab contains the following code:

```
1 <div *ngFor="let value of fullStackDev">
2   <ul>
3     <li>{{value.id}}</li>
4     <li>{{value.name}}</li>
5   </ul>
6 </div>
```

Annotations are overlaid on the code:

- A red box highlights the `value` variable in the `let value` part of the `*ngFor` directive.
- A red arrow points from the word *Variable name* to the `value` variable.
- A red arrow points from the *Perform an iteration on every item* text to the closing brace of the `for` loop.
- The `fullStackDev` variable is also highlighted with a red box.

Variable name

*Perform an iteration
on every item*

```
ts app.component.ts M X
myAngularProject-1 > src > app > ts app.component.ts > AppComponent
  ○ | styleUrls: ['./app.component.css']
  7 | })
  8 export class AppComponent {
  9   title = 'myAngularProject-1';
10   fullStackDev = [
11     {
12       id : 1,
13       name : 'Angular'
14     },
15     {
16       id : 2,
17       name : 'React'
18     },
19     {
20       id : 3,
21       name : 'Node.js'
22     },
23     {
24       id : 4,
25       name : 'MongoDB'
26     }
27 ]
```

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.*

