**SETUP:**

1. Angular tutorial - Prat 1 -setup + project creation
2. Install Node JS : https://nodejs.org/en/download
3. check if Node & NPM installed or not : npm -v & node -v
4. Install angular cli - npm i @angular/cli
5. Check if it installed : ng version
6. https://angular.dev/tutorials/learn-angular : angular website
7. create new project : ng new angular-18-tutorial
8. GitHub repo : https://github.com/rawatankita/Angular\_18\_tutorial
9. npm install
10. ng serve --o
11. multiple port : ng serve --port 4204

-------------------------------

**Setup questions:**

✔ Which stylesheet format would you like to use? CSS [ https://developer.mozilla.org/docs/Web/CSS ] ✔ Do you want to enable Server-Side Rendering (SSR) and Static Site Generation (SSG/Prerendering)? No ? Do you want to create a 'zoneless' application without zone.js (Developer Preview)?

**Answer**:

Great question! A zoneless Angular application is one that runs without zone.js, which traditionally handles change detection in Angular. This new approach is part of Angular’s Developer Preview and aims to improve performance, debugging, and compatibility.

🚀 Why Go Zoneless?

Better performance: Avoids unnecessary change detection cycles.

Smaller bundle size: Removes zone.js from your build.

Improved debugging: Cleaner stack traces.

Future-ready: Avoids monkey-patching browser APIs, which can break with newer features.

--------------------------------

**new changes in angular 18+**

no app.module.ts file : changed to app.config.ts

zoneless change detection

app.route.ts : constant file

-----------------------------

**Angular 18 tutorial part 2:**

**Project files and structure:**

package.json : (dependencies and packages)

angular.json : (project setup/ architecture file)

style.css

gitignore

tsconfig.json : (typescript configuration)

main.ts ( first file to execute when app start : bootstarap app)

index.html (Single page application one file only index.html load)

app.config.ts

--------------------------------

**package.json**

dependency : ( part of angular build)

dev dependency : (to run angular project locally and not part of your build)

-------------------------------

🧭 1. main.ts — The True Entry Point

Located in src/main.ts, this file is where Angular begins execution.

---------------------------------

**Angular 20 highlights**

You're absolutely right to notice a change in Angular 20's structure. In earlier versions, the angular.json file explicitly listed the path to index.html under the build options. But in Angular 20, the configuration has evolved.

🧾 What's New in Angular 20?

According to the official Angular workspace configuration guide:

The angular.json file still exists and controls build and development settings.

However, the index option may not be explicitly listed unless customized.

Angular CLI assumes default paths unless overridden.

🗂 Default Behavior

If you don’t see index.html in angular.json, it’s because Angular 20 uses convention over configuration:

It assumes your index.html is located at src/index.html.

Unless you change it, there's no need to declare it explicitly.

🛠 How to Customize It (If Needed)

You can still manually add or override the index file path like this:

json

"projects": {

"your-app-name": {

"architect": {

"build": {

"options": {

"index": "src/custom-index.html"

}

}

}

}

}

You're absolutely right to notice that in Angular 20, the standalone: true flag might not appear explicitly in newly generated components—and that's by design.

🧠 What's Happening in Angular 20?

Starting with Angular 19 and continuing into Angular 20:

Standalone components are now the default.

That means when you generate a component using:

ng generate component my-component

You're absolutely on point—Angular 20 has introduced a major shift in naming conventions and project structure, which is why you're not seeing app.component.ts anymore.

**🚨 What's Changed in Angular 20?**

Angular 20 has **dropped the** .component.ts **naming convention** for newly generated files. Instead:

* The main component is now simply named app.ts.
* This change aligns with Angular’s push toward **standalone components** and a cleaner, more modern file structure.

So instead of:

Code

src/app/app.component.ts

You’ll now see:

Code

src/app/app.ts

**🧩 Why This Might Cause Issues**

Some tools (like Visual Studio or older build setups) still expect the old app.component.ts file. This can lead to errors like:

"Could not find file 'app.component.ts'" or "File 'src/app/app.component.ts' is missing from the TypeScript compilation"

These are compatibility issues with tooling that hasn’t yet adapted to Angular 20’s new structure2.

**🛠 How to Fix or Work Around It**

1. **Use Angular 19 if your tooling depends on the old structure**:

bash

npm install -g @angular/cli@19

1. **Re-enable old naming conventions in Angular 20**: Update your angular.json to use legacy schematic settings:

json

"schematics": {

"@schematics/angular:component": {

"style": "css",

"skipTests": false,

"prefix": "app",

"namePattern": "component"

}

}

1. **Manually rename files** if needed and update imports accordingly.

**Components**

1. How to create component?

ng g c component

1. What is component?

In Angular, a **component** is the fundamental building block of the user interface. It encapsulates everything needed to render and manage a piece of the UI—like a button, form, or entire page.

**🧱 Anatomy of an Angular Component**

Each component typically includes:

* **TypeScript class**: Defines the logic and behaviour.
* **HTML template**: Describes what appears on the screen.
* **CSS styles**: Controls the visual appearance.
* **Selector**: A custom HTML tag used to embed the component.

1. What is standalone component?

These are default type of component which are not part of any other component until we import them. In angular 18 we have “standalone: true” option while creating component but from angular 20 onward standalone option is now by default and no need to write it explicitly in imports: [] array @component decorator.

1. What are Component Decorators?

@Component decorator in Angular is a special TypeScript annotation that tells Angular: **“This class is a component.”** It provides metadata that Angular uses to create, render, and manage the component at runtime.

**Data Binding**

1. Types of Data Bindings?

One way binding, Two-way binding, Signals

**On way Binding:**

* + From .ts to html
  + Interpolation
  + Property Binding
  + From html to .ts Event binding

**Two-way Binding:**

* + Using [(ngModel)]

**Signals (Angular 17 onwards)**

⚡ What Is a Signal?

A signal is a reactive primitive introduced in Angular 16+ that holds a value and notifies the UI when that value changes. Think of it like a smarter, leaner version of BehaviorSubject or NgRx, but built right into Angular.

✅ Why Use Signals?

1. Automatic UI Updates

When you change a signal’s value, Angular automatically updates the DOM—no need for manual change detection or async pipes.

ts

count = signal(0);

this.count.set(this.count() + 1); // UI updates instantly

2. Simpler State Management

Signals can replace complex RxJS setups for local component state. No subscriptions, no memory leaks.

3. Better Performance

Signals are fine-tuned for Angular’s rendering engine. They reduce overhead by tracking dependencies precisely—only re-rendering what’s needed.

4. Composable Logic

You can derive new signals from existing ones:

ts

fullName = computed(() => `${this.firstName()} ${this.lastName()}`);

5. Cleaner Templates

No need for async pipe or ngIf gymnastics. Just call the signal like a function:

html

<p>{{ fullName() }}</p> (called as a method)

🧠 When Should You Use Signals?

* For local component state (like form inputs, counters, toggles)
* When you want fine-grained reactivity without external libraries
* To simplify derived values and reactive expressions
* In Angular standalone components or modern apps using the latest features

**ngmodel and signal difference**

**🧠 ngModel: Classic Two-Way Binding**

* **What it is**: A directive from FormsModule that enables two-way binding between form inputs and component properties.
* **Syntax**: [(ngModel)]="value"
* **Use case**: Ideal for template-driven forms and simple input binding.
* **How it works**: Updates the component property when the user types, and vice versa.

ts

export class MyComponent {

name: string = 'Angular';

}

html

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

<p>{{ name }}</p>

**⚡ signal: Modern Reactive Primitive**

* **What it is**: A reactive value introduced in Angular 16+ that automatically triggers UI updates when changed.
* **Syntax**: name = signal('Angular')
* **Use case**: Great for reactive state management, computed values, and fine-grained reactivity.
* **How it works**: You access the value with name() and update it with name.set('New Value').

ts

export class MyComponent {

name = signal('Angular');

}

html

<input [value]="name()" (input)="name.set($event.target.value)" />

<p>{{ name() }}</p>

**🔄 Can You Use ngModel with Signals?**

Yes! You can bind ngModel to a **writable signal**, but you need to handle the getter/setter manually:

html

<input [ngModel]="name()" (ngModelChange)="name.set($event)" />

**Directives [change behaviours of DOM element]**

* + - 1. Structural directive [ Change structure of DOM]

\*ngIf - >Add / remove element from the DOM

\*ngFor -> dynamic DOM structure update

\*ngSwitch -> Switch on basis of conditions

Angular 18+ updates : need to import commonModule separately to use directives

1. Attribute Directives [ Change style of DOM element]

ngStyle

ngClass

1. Component Directives
2. Custom Directives

Angular has **three main types of directives**, each serving a distinct purpose in how you build and control your application's behavior and layout:

**🧱 1. Component Directives**

* **Definition**: These are directives with a template. In fact, every Angular component is a directive.
* **Purpose**: They define UI blocks and encapsulate logic, styles, and templates.
* **Example**:

ts

@Component({

selector: 'app-user',

templateUrl: './user.component.html'

})

export class UserComponent {}

**🎨 2. Attribute Directives**

* **Definition**: These modify the appearance or behavior of an element.
* **Purpose**: Used to dynamically change styles, classes, or properties.
* **Common Built-in Examples**:
  + ngClass: Adds/removes CSS classes.
  + ngStyle: Applies inline styles.
  + ngModel: Enables two-way data binding.
* **Example**:

html

<div [ngClass]="{ 'highlight': isActive }"></div>

**🏗️ 3. Structural Directives**

* **Definition**: These change the DOM layout by adding or removing elements.
* **Purpose**: Control rendering based on conditions or loops.
* **Common Built-in Examples**:
  + \*ngIf: Conditionally includes a template.
  + \*ngFor: Loops over a collection.
  + \*ngSwitch: Switch-case rendering.
* **Example**:

html

<div \*ngIf="isLoggedIn">Welcome back!</div>

**🛠️ Bonus: Custom Directives**

You can create your own **attribute or structural directives** to encapsulate reusable logic.

ts

@Directive({

selector: '[appHighlight]'

})

export class HighlightDirective {

constructor(private el: ElementRef) {

el.nativeElement.style.backgroundColor = 'yellow';

}

}

For a deeper dive into Angular directives, check out the official Angular guide on directives.

Want to try building a custom directive together?