**Significance and Role of Components in Angular**

1 **UI Building Blocks**

 Each piece (component) has its **own design and job** — like a header, footer, or button.

 You join these pieces together to create the **full application**.

**2 Encapsulation and Reusability**

* **Encapsulation** → A component keeps **its HTML, CSS, and logic together**, so changes in one don’t mess up others.
* **Reusability** → Once you build a component (e.g., a login form), you can **use it anywhere** in your app without rewriting it.

### ****Component-Based Architecture****

* Your app is made up of **small, independent components** instead of one giant file.
* This makes the app **easier to develop, test, and maintain**.

### 4 ****Component Lifecycle Hooks****

* Components have a **life cycle** — they are **created, updated, and destroyed**.
* Angular gives you **special functions (hooks)** to run code at each stage.  
  Example: Do something when the component is loaded (ngOnInit).

### 5 ****Hierarchical Structure****

* Components can **contain other components** inside them.
* Like a **family tree**:
  + Parent component (App)  
    → Child component (Navbar)  
    → Grandchild component (Menu Item)

## **What Are Angular Components?**

### 1️ ****Everything is a Component****

* In Angular, almost **every visible thing** on the screen is made from a **component**.
* Example: Header, footer, sidebar, button — all are components.

### 2 ****Primary Building Block****

* Components are the **main pieces** you use to build an Angular app — like bricks in a wall.

### 3 ****Reusable, Modular, and Encapsulated****

* **Reusable** → Make once, use many times (e.g., a login form).
* **Modular** → Each component is like a **separate mini-program**.
* **Encapsulated** → HTML, CSS, and TypeScript for a component are kept together in one package.

### 4 ****One Component Per Element****

* In an HTML template, **one component can control one element** — no two components should try to take over the same element.

### 5 ****Component Must Belong to an NgModule****

* You must **declare the component** inside an Angular **module** (usually app.module.ts), otherwise Angular won’t know it exists.

### 6 ****@Component Decorator****

* This is **special code** that tells Angular:
  + What is the component’s HTML file?
  + What is the component’s CSS?
  + What is its name (selector)?

@Component({

selector: 'app-hello',

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

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

})

### 7 I****mplements Lifecycle Hooks****

* Components can have **special functions** that run automatically during their life (e.g., ngOnInit runs when the component is created).

## **Component Structure in Angular**

A component has **3 main parts** — think of it like a **human**:

### 1️ ****Template (HTML) – The Face****

* **What it is:** The **HTML code** that decides **how the component looks** on the screen.
* **Example:**

<h1>Hello, {{name}}</h1>

<button>Click Me</button>

* It’s like the **appearance** of the component.

### 2 ****Class (TypeScript) – The Brain****

* **What it is:** The **TypeScript code** that handles the **logic, data, and actions**.
* **Example:**

export class HelloComponent {

name = 'Angular Student';

}

* It’s like the **thinking part** — decides what the template should show.

### 3 ****Metadata – The ID Card****

* **What it is:** Extra information about the component, provided using the @Component decorator.
* **Includes:**
  + **Selector:** Name to use in HTML (e.g., <app-hello>).
  + **Template URL:** File containing HTML.
  + **Style URLs:** CSS files.
* **Example:**

@Component({

selector: 'app-hello',

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

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

})

**1 @Component Decorator**

* **What it is:** A special **instruction** for Angular, written with **@ symbol**.
* **Why:** It gives Angular extra details about the component.

**2️ What Metadata Includes**

* **Selector** → Name you use in HTML to show this component.
* **Template / Template URL** → HTML design for the component.
* **Style URLs** → CSS for the component.

**3️ Selector**

* Tells Angular **where to place the component**.
* Example:

selector: 'my-app'

If you write <my-app></my-app> in index.html, Angular will show this component there.

### 4 ****Template****

* Tells Angular **how to display** the component.
* Can be:
  + Inline HTML:

template: `<h1>Hello Angular</h1>`

* Or from an HTML file:

templateUrl: './app.component.html'

**1 Two Ways to Write a Template**

* **Inline Template** → Written directly inside the component’s TypeScript file using the template property.
  + Best for **small and simple** HTML.
  + Example:

@Component({

selector: 'app-inline',

template: `<h1>Hello Inline</h1>`

})

* **External Template** → Stored in a **separate HTML file** using templateUrl.
  + Best for **larger and more complex** HTML.
  + Example:

@Component({

selector: 'app-external',

templateUrl: './external.component.html'

})

**2️ Component Class**

* Written in **TypeScript**.
* **Purpose:** Holds the **data** and **logic** of the component.
* You define:
  + **Properties** → Store data.
  + **Methods** → Perform actions.

**3️ Example**

export class ExampleComponent {

title: string = 'Example Component';

count: number = 0;

increaseCount() {

this.count++;

}

}

* **title** → A property (stores a string).
* **count** → Another property (stores a number).
* **increaseCount()** → A method (increases count by 1).

## **Component Lifecycle Hooks**

### 1️ ****What They Are****

* Special **methods** Angular runs **automatically** at different moments in a component’s life.
* Think of them like **checkpoints**:
  + When the component is **created**
  + When something **changes**
  + When the component is **removed**

### 2️ ****Why Use Them****

* To **run code** at the right time, such as:
  + Load data when the component starts.
  + Clean up when the component is destroyed.
  + React when an input value changes.

### 3️ ****Examples****

* **ngOnInit** → Runs once when the component is created. Good for initialization or API calls.
* **ngOnDestroy** → Runs just before the component is removed. Good for cleanup (like unsubscribing from data streams).
* **ngOnChanges** → Runs when component input values change.

### 4️ ****How to Use a Component****

* Each component has a **selector** (like <app-example>).
* To use it in another component’s HTML:

<app-example></app-example>

* Angular will replace that tag with the **component’s content**.

# 1) Create the project

ng new my-first-app

# ? Add routing: No

# ? Stylesheet: CSS

cd my-first-app

# 2) Run the app

ng serve -o

Your browser opens at http://localhost:4200 (Angular’s default page).

Stop server anytime with **Ctrl + C** in the terminal.

# 3) Create your first component

ng generate component hello

# (shorthand: ng g c hello)

Files created:

src/app/hello/hello.component.ts (class + metadata)

src/app/hello/hello.component.html (template)

src/app/hello/hello.component.css (styles)

# 4) Put simple content in the component

**src/app/hello/hello.component.ts**

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

@Component({

selector: 'app-hello',

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

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

})

export class HelloComponent {

name = 'Angular Beginner'; // property (data)

greet() { // method (logic)

return `Welcome, ${this.name}!`;

}

}

**src/app/hello/hello.component.html**

<h2>{{ greet() }}</h2> <!-- interpolation: show method result -->

That’s already enough to show **data** + **logic** + **UI**.

# 5) Show your component on the page

Open **src/app/app.component.html** and replace everything with:

<h1>My First Angular App</h1>

<app-hello></app-hello> <!-- Use the selector from @Component -->

The browser auto-refreshes. You should see:

* Title
* Greeting
* A button that increases the count