full one-shot videos on :**JK Coding Pathshala YouTube channel**

JK Coding Pathshala

https://youtube.com/@jayeshkande9215?feature=shared

Frontend Technologies

Unit -03 WAD(web application Development)

Unit III	FRONT END TECHNOLOGIES	(06 hrs)

Front-End Frameworks: What is web framework? Why Web Framework? Web Framework Types.

MVC: What is MVC, MVC Architecture, MVC in Practical, MVC in Web Frameworks.

TypeScript: Introduction to TypeScript (TS), Variables and Constants, Modules in TS.

AngularVersion 10+: Angular CLI, Angular Architecture, Angular Project Structure, Angular Lifecycle, Angular Modules, Angular Components, Angular Data Binding, Directives and Pipes, Angular Services and Dependency Injections (DI), Angular Routers, Angular Forms.

ReactJS: Introduction to ReactJS, React Components, Inter Components Communication, Components Styling, Routing, Redux- Architecture, Hooks- Basic hooks, useState() hook, useEffect() hook useContext(hook.

May_June _2024

Q1) a) Write a simple application in typescript to demonstrate the use of modules. **[9]**

b) What is pipe? Demonstrate the code for pipes in Angular. **[9]**

OR

Q2) a) Give the simple layout of the Angular application with multiple components. Explain how to create and use components in Angular? **[9]**

b) Explain the basic hooks in React JS. Explain any two hooks in brief. **[9]**

Nov_dec_2022

Q1) a) List and explain different types of structural directives in Angular. **[6]**

b) How would you demonstrate the term web framework? Give the reason for using a web framework. **[6]**

c) What is Angular JS? Explain its features. **[6]**

OR

Q2) a) List and explain the features of any three popular web frameworks. **[6]**

b) Explain MVC architecture with a suitable diagram. **[6]**

c) How would you use the term typescript? Give the advantages and disadvantages of using it. **[6]**

May_June_2023

Q1) a) Explain MVC architecture with a suitable diagram. **[6]**

b) List and explain different types of structural directives in Angular. **[6]**

c) What way would you design simple application in typescript to demonstrate the use of modules? **[6]**

OR

Q2) a) List and explain the features of any three popular web frameworks. **[6]**

b) How would you use the term typescript? Give the advantages and disadvantages of using it? **[6]**

c) Explain any 2 Hooks in React JS with example. **[6]**

Nov_dec_2023

Q1) a) Write a simple application in typescript to demonstrate the use of modules. **[9]**

b) What is pipe? Demonstrate the code for pipes in Angular. **[9]**

OR

Q2) a) Give the simple layout of the Angular application with multiple components. Explain how to create and use components in Angular? **[9]**

b) Explain the basic hooks in React JS. Explain any two hooks in brief. **[9]**

May_June_2022

Q1) a) What is MVC? Explain MVC architecture in detail. **[6]**

b) Explain event binding and property binding in angular with example. **[6]**

c) Explain different types of Hooks in ReactJS. **[6]**

OR

Q2) a) What is TypeScript? List advantages & disadvantages of using it. **[6]**

b) What is Pipe? Explain with example. **[6]**

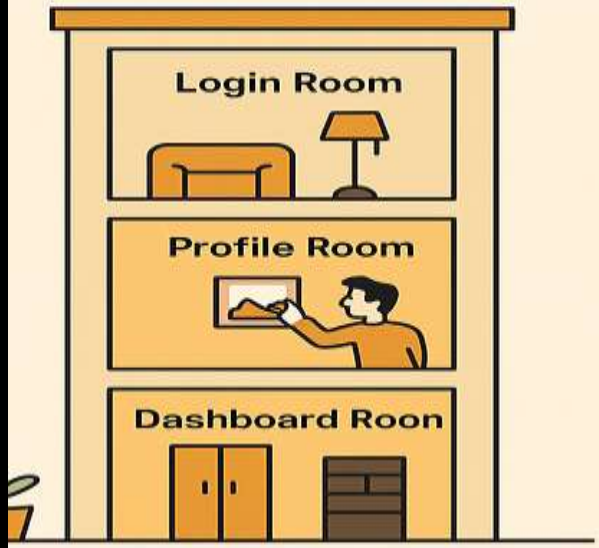
c) Explain Redux - Architecture in detail. **[6]**

1. Front-End Frameworks:

Framework – "Full Ready-Made Structure"

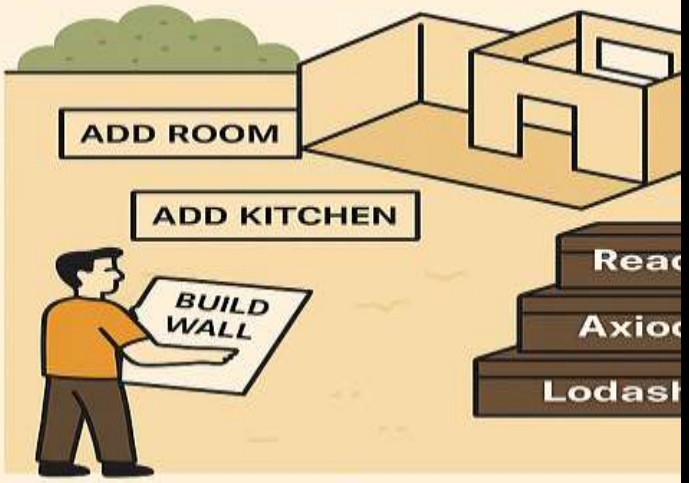
A **framework** provides a full structure to build your application. It controls the flow and calls your code at specific places.

Framework ek a pre-p-built apar



ucture (Routing, Services, DI)
already built

u just add your code in
the allowed places



You control the structure
Call what you need
You are in Control

Why Web Frameworks?

Web frameworks are **pre-built libraries and tools** that simplify and accelerate the development of web applications. Instead of writing everything from scratch, developers use frameworks to:

- Reuse common patterns (like MVC)
- Ensure security
- Improve code structure
- Reduce development time
- Simplify integration with databases, user authentication, etc.

Types of Web Frameworks:

Frontend Frameworks (Client-side): Focused on building the **user interface** and improving interactivity.

Examples: React.js, Angular, Vue.js

- **Backend Frameworks** (Server-side)

- Handle **server logic**, routing, database interaction, and APIs.

- Examples: Express.js, Django, Laravel

Full-Stack Frameworks

Combine frontend and backend in one framework.

Examples: Next.js, Meteor, Ruby on Rails

a) List and explain the features of any three popular web frameworks. **[6]**

1. **React.js** (Frontend Framework)

Features:
•**Component-based**: Reusable UI components.
•**Virtual DOM**: Improves app performance by minimizing direct DOM manipulation.
•**One-way Data Binding**: Ensures unidirectional data flow.
•**Strong Ecosystem**: Tools like React Router and Redux for state management.
•**JSX Syntax**: HTML + JavaScript combined.

2. Angular (Frontend Framework by Google)

•**Features**:

- **Two-way Data Binding**: Automatic synchronization of data between model and view.
- **Component-based architecture**: Encapsulates HTML, CSS, and JavaScript into reusable components.
- **RxJS for reactive programming**: For handling asynchronous operations.
- **Dependency Injection**: For better testing and easier dependency management.
- **TypeScript**: Built with TypeScript for type safety and easier debugging.

3. **Express.js** (Backend Framework for Node.js)

Features:

•**Minimalist & Flexible**: Provides core features without much overhead.
•**Middleware**: Use of middleware functions to handle requests, responses, and errors.
•**Routing**: Define routes for different HTTP methods (GET, POST, etc.).
•**RESTful API**: Built-in support for creating REST APIs.
•**Session management**: Includes support for cookies and sessions.

4. Java (Spring Boot) (Backend Framework for Java)

•**Features**:

- **Spring Boot**: Simplifies Java-based web development by reducing boilerplate code.
- **Microservices Support**: Ideal for building microservices-based applications.
- **Embedded Servers**: Includes embedded web servers (Tomcat, Jetty).
- **Comprehensive Data Access**: Built-in support for data access, including JPA and Hibernate.
- **Security**: Provides built-in security features, like authentication and authorization.

b) How would you demonstrate the term web framework? Give the reason for using a web framework. **[6]**

A **web framework** is a structured platform for developing web applications. It provides tools, libraries, and best practices that help simplify and speed up the development process.

Example using Angular:

Angular is a powerful **TypeScript-based web framework** developed by Google. It is used for building dynamic single-page applications (SPAs).

```
// app.component.ts
import { Component } from '@angular/core';

@Component({
  selector: 'app-root',
  template: `<h1>Welcome to Angular!</h1>`,
})
export class AppComponent { }
```

This simple Angular component displays a message and is part of a structured application with routing, services, and modules.

Reasons for using a web framework like Angular:

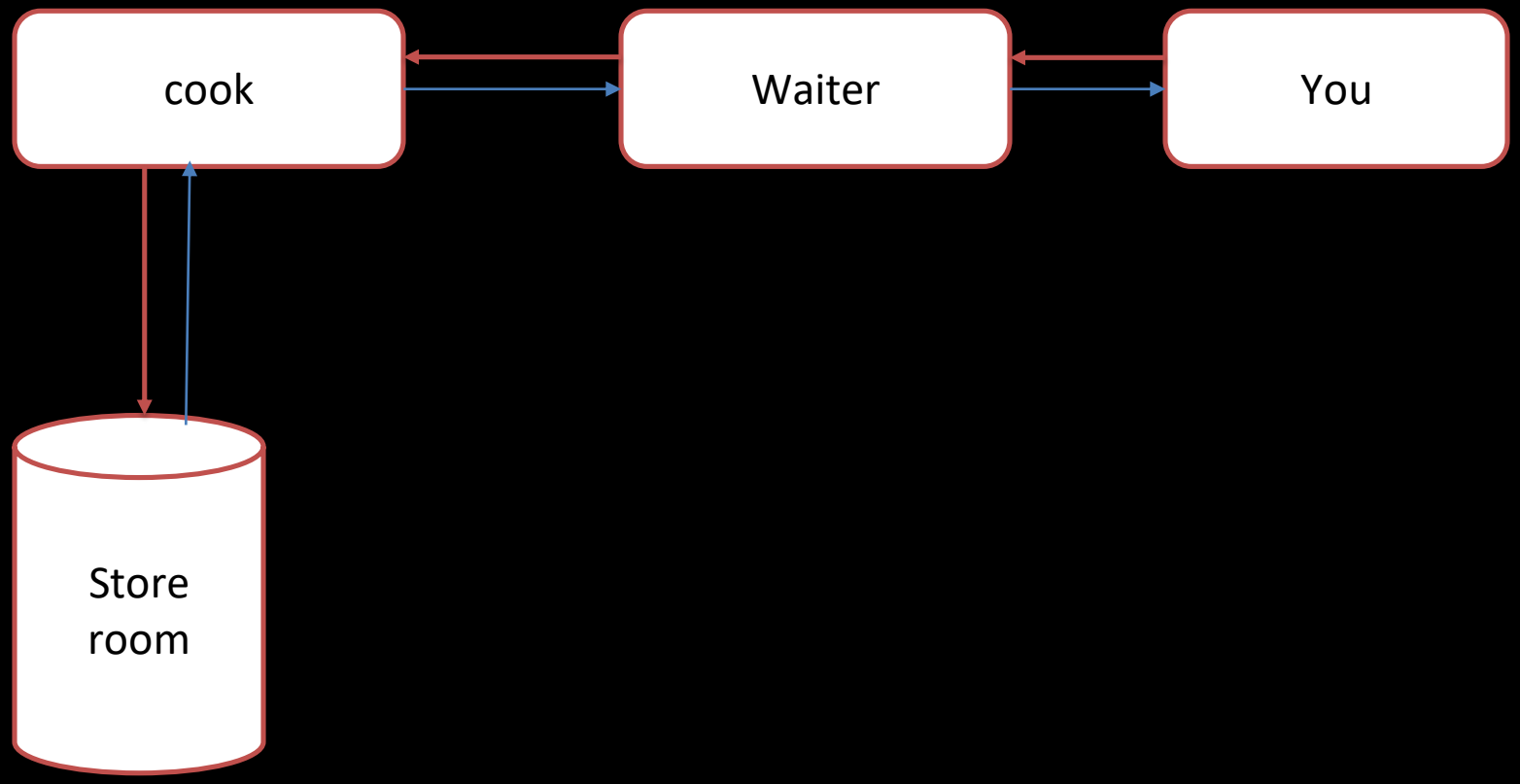
1. **Two-way data binding** – Syncs data between model and view automatically.
2. **Component-based architecture** – Promotes reusability and clean code.
3. **Built-in features** – Includes routing, form validation, HTTP services, etc.
4. **Strong TypeScript support** – Helps catch errors at compile time.
5. **Modular structure** – Easier to manage large applications.
6. **Community and support** – Backed by Google with regular updates and a large ecosystem.

MVC (Model-View-Controller):

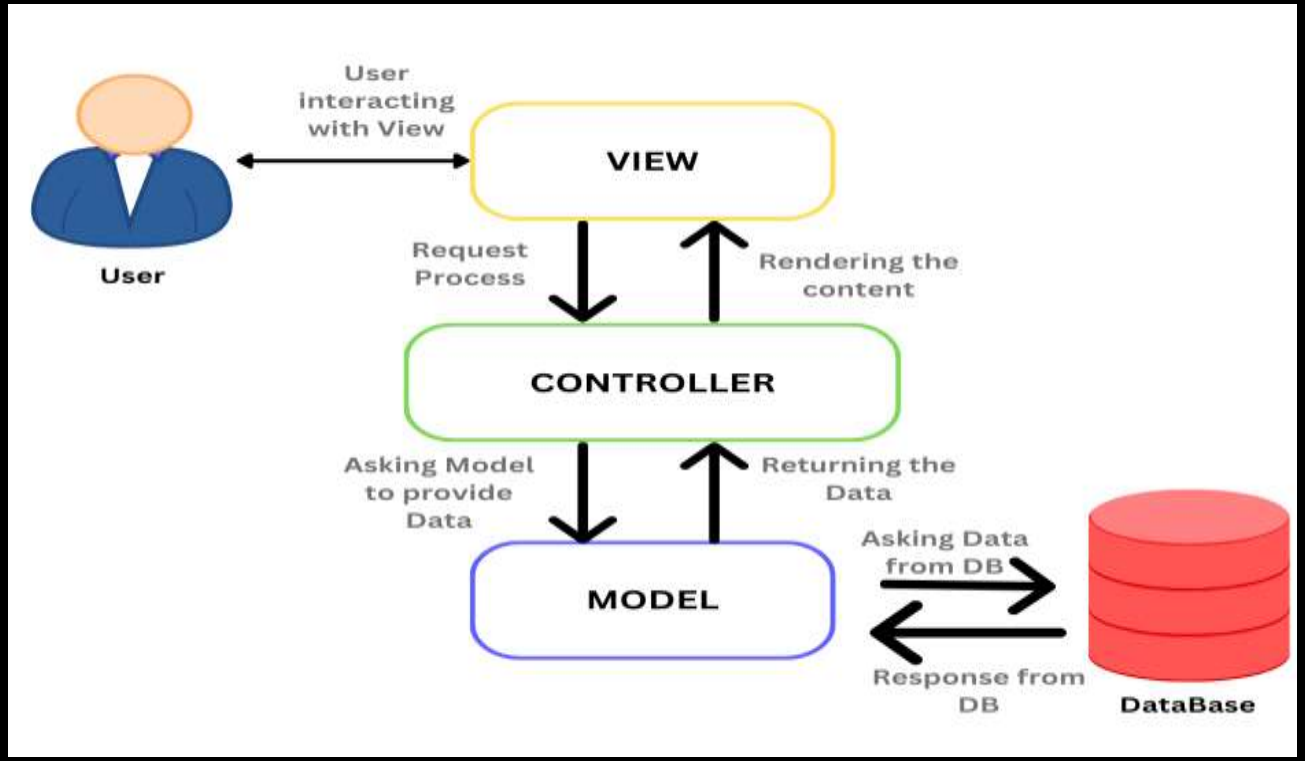
1. What is MVC?

MVC is a **software design pattern** used for developing user interfaces by dividing an application into three interconnected components:

- **Model** – Manages the data and business logic.
- **View** – Handles the display and presentation.
- **Controller** – Acts as an interface between Model and View, handling user input.


cook
Waiter
You
Store
room

2. MVC Architecture:



- **Model**: Interacts with the database, fetches or updates data.

- **View**: Represents UI elements. It reads data from the model to display to the user.

- **Controller**: Receives input, processes it (possibly updating the model), and determines the next view.

Flow:

User → Controller → Model → View → User

MVC in Practical – MERN Stack Example

- **Model (M) — MongoDB with Mongoose**
  This is the data layer. It defines the structure of the data and interacts with the database.

◆ **View (V) — React.js**

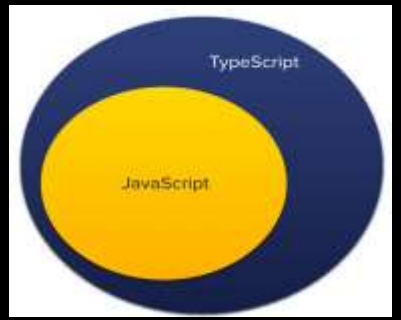
This is the front-end UI that the user interacts with.

◆ **Controller (C) — Express.js Routes/Logic**
This handles user requests, interacts with the model, and returns a response.

TypeScript

- ✅ Introduction to TypeScript
- ✅ Installation & Setup
- ✅ Variables and Constants
- ✅ Data Types
- ✅ Functions
- ✅ Interfaces
- ✅ Classes & Inheritance
- ✅ Modules
- ✅ Advantages & Disadvantages

TypeScript



What is TypeScript?

TypeScript is a **superset of JavaScript** that adds **static type checking** and **modern features** to the language. It helps developers write **cleaner, safer, and more scalable** code.

- Created by Microsoft.
- Compiles to plain JavaScript (.ts → .js).
- Helps catch errors during development.

TypeScript Setup (Quick)

```
npm install -g typescript      # Install globally

tsc --init                     # Create tsconfig.json

tsc                            # Compile TypeScript files
```

🔤 3. Variables & Constants

Syntax:

```
let age: number = 25;

const name: string = "Alice";

var isAdmin: boolean = true;
```

⚠️ let vs const vs var

- let: block scoped (recommended)
- const: block scoped and can't be reassigned
- var: function scoped (avoid using it)

4. Data Types

Type	Example
number	let x: number = 5;
string	let name: string = "Bob";
boolean	let isDone: boolean = true;
any	let value: any = 10;
array	let list: number[] = [1,2];
tuple	let t: [string, number] = ["A", 1];
enum	enum Color {Red, Green, Blue}
null / undefined	let n: null = null;

🔁 5. Functions

Basic Function:

```
// Function with parameter and return type
function multiply(a: number, b: number): number {
  return a * b;
}
console.log("Multiply:", multiply(4, 5));

// Arrow function syntax
const divide = (x: number, y: number): number => {
  return x / y;
};
console.log("Divide:", divide(10, 2));
```

🎁 6. Interfaces

Used to define **shapes of objects**.

```
interface Person {
  name: string;
  age: number;
}

let p: Person = { name: "Alice", age: 30 };
```

7. Classes and Inheritance

```
class Animal {
  name: string;
  constructor(name: string) {
    this.name = name;
  }
  speak(): void {
    console.log(`${this.name} makes a sound.`);
  }
}

class Dog extends Animal {
  constructor(name: string) {
    super(name);
  }
  speak(): void {
    console.log(`${this.name} barks.`);
  }
}

const d = new Dog("Tommy");
d.speak(); // Tommy barks.
```

8. Modules

TS mathUtils.ts × TS main.ts

TS mathUtils.ts > ...

```typescript
export function add(a: number, b: number): number {
  return a + b;
}
```

TS mathUtils.ts TS main.ts ×

TS main.ts

```typescript
import { add } from './mathUtils';
console.log(add(2, 3));
```

a) Write a simple application in typescript to demonstrate the use of modules. [9]

simple calculator app using modules.

Step 1: mathUtils.ts (Module File)

mathUtils.ts × main.ts

mathUtils.ts > ...

```typescript
// Exported named functions
export function add(a: number, b: number): number {
    return a + b;
}

export function subtract(a: number, b: number): number {
    return a - b;
}
```

TS mathUtils.ts TS main.ts ×

TS main.ts > ...

```typescript
// Importing from the module
import { add, subtract } from './mathUtils';

const num1: number = 10;
const num2: number = 5;

console.log("Addition:", add(num1, num2));
console.log("Subtraction:", subtract(num1, num2));
```

TS mathUtils.ts | JS main.js | TS main.ts

JS main.js > ...

```
1  "use strict";
2  Object.defineProperty(exports, "__esModule", { value: true });
3  // Importing from the module
4  var mathUtils_1 = require("./mathUtils");
5  var num1 = 10;
6  var num2 = 5;
7  console.log("Addition:", (0, mathUtils_1.add)(num1, num2));
8  console.log("Subtraction:", (0, mathUtils_1.subtract)(num1, num2));
9
```

```
PS C:\Users\Admin\Desktop\tyscript> tsc mathUtils.ts
PS C:\Users\Admin\Desktop\tyscript> tsc main.ts
PS C:\Users\Admin\Desktop\tyscript> tsc mathUtils.ts
PS C:\Users\Admin\Desktop\tyscript> tsc main.ts
PS C:\Users\Admin\Desktop\tyscript> node main.js
Addition: 15
Subtraction: 5
```

a) What is TypeScript? List advantages & disadvantages of using it. **[6]**

c) How would you use the term typescript? Give the advantages and disadvantages of using it. [6]

Definition / Usage of TypeScript

TypeScript is a **superset of JavaScript** developed by Microsoft that adds **static typing** and **modern programming features** to JavaScript. It is used to build large-scale and maintainable web applications. TypeScript code is **compiled into JavaScript**, which runs in any browser or JavaScript environment.

Advantages of TypeScript:

1. **Static Typing** – Detects errors during development, before runtime.
2. **Improved Code Quality** – Type checking, interfaces, and strong tooling help catch bugs early.
3. **Better IDE Support** – Features like IntelliSense, auto-completion, and refactoring.
4. **Object-Oriented Programming** – Supports classes, inheritance, interfaces, etc.
5. **Large Community & Ecosystem** – Widely used in modern frameworks like Angular.
6. **Easy to Learn for JavaScript Users** – Uses the same syntax with additional features.

Disadvantages of TypeScript

1. **Compilation Required** – Needs to be compiled into JavaScript before execution.
2. **Learning Curve** – New developers may take time to understand types and configuration.
3. **Extra Setup** – Requires tools like tsc, and configuration (e.g., tsconfig.json).
4. **Verbose Code** – More typing and declarations compared to JavaScript.
5. **May Slow Down Small Projects** – Overhead might not be worth it for tiny scripts.
6. **Third-party Library Types** – Sometimes types may be missing or outdated.

AngularJS

AngularVersion 10+: Angular CLI, Angular Architecture, Angular Project Structure, Angular Lifecycle, Angular Modules, Angular Components, Angular Data Binding, Directives and Pipes, Angular Services and Dependency Injections (DI), Angular Routers, Angular Forms.

🔰 What is AngularJS?

📌 AngularJS kya hai?

AngularJS ek **JavaScript-based front-end framework** hai, jo Google ne banaya tha **2010 me**. Ye mainly **Single Page Applications (SPAs)** banane ke liye use hota tha.

Iska goal tha:
➡️ HTML ko **dynamic** banana
➡️ JavaScript aur HTML ke integration ko easy banana

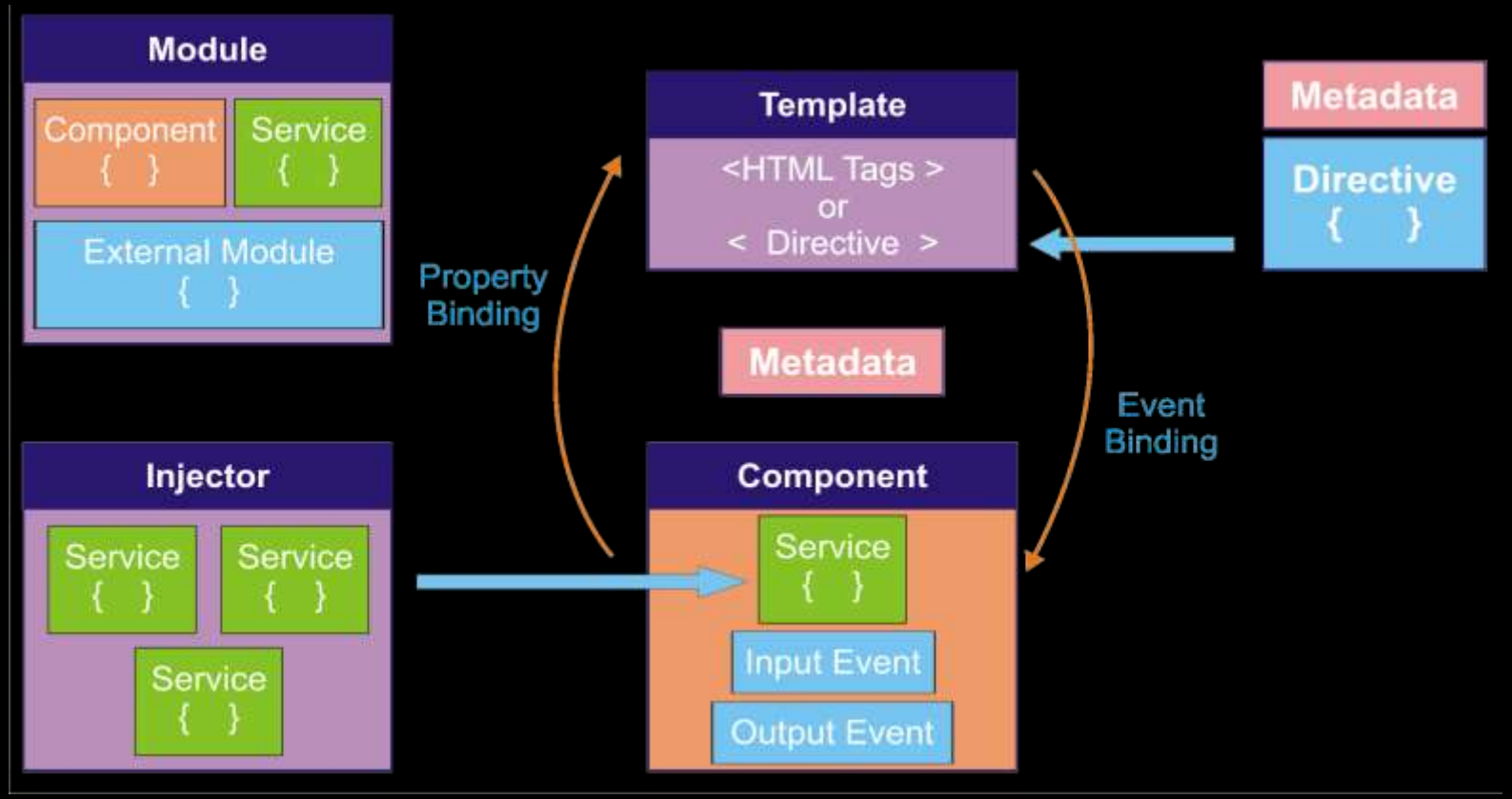
Feature	Explanation
MVC Pattern	AngularJS me Model-View-Controller architecture use hota hai.
Two-Way Data Binding	UI aur model ke data me auto synchronization hota hai.
Directives	Custom HTML tags ya attributes define karne ke liye.
Dependency Injection	Services ko automatically inject karne ka feature.
Templates	Dynamic HTML banane ke liye expressions ({{ }}) use karte hain.
Filters	Data ko format karne ke liye (e.g., date, currency).

1. Angular CLI (Command Line Interface)

Angular CLI ek command-line tool hai jo Angular project banana, build karna, aur serve karne mein madad karta hai.

```
ng new my-app   # New Angular project
ng serve        # Run the project
```

2. Angular Architecture



🌟 Angular Architecture – Full Explanation

Angular ek **component-based framework** hai jise modern web apps banane ke liye use kiya jaata hai. Iska architecture modular, scalable aur maintainable hai.

◈ 1. **Module**

•Angular application ek ya ek se zyada **modules** se bana hota hai.

•Ek module ke andar:

- •**Components** (UI logic),
- •**Services** (business logic/data),
- •**External Modules** (BrowserModule, FormsModule, etc.) hotey hain.

Purpose: Application ko logical units me divide karna (reusability & separation of concerns).

- Example:

•AppModule – root module.
•UserModule, ProductModule – feature-specific modules.

Module Angular ka building block hota hai, jo components, services, and other modules ko group karta hai.

app.module.ts M

Angular > src > app > 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 { HomeComponent } from './home/home.component';
7  import { RegistrationComponent } from './registration/registration.component';
8  import { LoginComponent } from './login/login.component';
9
10 @NgModule({
11   declarations: [
12     AppComponent,
13     HomeComponent,
14     RegistrationComponent,
15     LoginComponent
16   ],
17   imports: [
18     BrowserModule,
19     AppRoutingModule
20   ],
21   providers: [],
22   bootstrap: [AppComponent]
23 })
24 export class AppModule { }
```

◈ 2. **Component**

•**Heart of Angular App**.

•Ek component ke 3 major parts hote hain:

- **Template** – HTML UI
- **Class** – TypeScript logic
- **Metadata** – selector, style, template info

Purpose: Ek particular view (UI part) ko control karna.

 Example: HeaderComponent, FooterComponent, UserProfileComponent

Component UI ka part hota hai jisme:

- **Template** (HTML)
- **Class** (logic)
- **Metadata** (selector, template etc.)

TS app.component.ts M ×

Angular > src > app > TS app.component.ts > ...

```
1  import { Component } from '@angular/core';
2
3  @Component({
4    selector: 'app-root',//meta data
5    templateUrl: './app.component.html',//template
6    standalone: false,
7    styleUrl: './app.component.css'
8  })
9  export class AppComponent {
10   title = 'JK';//Logic
11 }
```

- **3. Template**

•Ye HTML + Angular syntax (like *ngIf, {{}}, etc.) ka mixture hota hai.

•Template define karta hai ki user ko kya dikhna chahiye.

Purpose: User Interface banana with data binding.

- Example:

```
<h2 *ngIf="isLoggedIn">Welcome User</h2>
```

Template me HTML + Angular directives (like *ngIf, *ngFor) hote hain.

app.component.html M ×

Angular > src > app > app.component.html > h1

```html
<h1>"Welcome to JK Coding Pathshala – Jahaan Coding Seekhna Hai Asaan, Mazedaar aur Career-Building!"</h1>

<p>First time on the website ? do registration</p>

<a routerLink="register">Register</a>
<router-outlet></router-outlet>
```

🔄 4. Data Binding (Property & Event Binding)

✅ Property Binding:

•Template se component ki properties ko bind karna.
•Example: [value]="username“

✅ Event Binding:

•Template se component ki methods ko trigger karna.
•Example: (click)="login()"
Purpose: Component class aur HTML ke beech 2-way communication establish karna.

Data ko TypeScript se HTML me bhejna.

```
<!-- Template -->
<input [value]="username">

// Component
username = 'Jayesh';
```

🔷 5. Event Binding

```
<button (click)="greetUser()">Click Me</button>
```

```
greetUser() {
  alert('Hello!');
}
```

● 5. Directive

•Directives are instructions to DOM.

•3 types:

1. **Component Directive** (normal component)
2. **Structural Directive** – e.g., *ngIf, *ngFor (DOM structure change)
3. **Attribute Directive** – e.g., [ngClass], [style] (change appearance)

Purpose: HTML elements ka behavior ya layout modify karna.

Custom ya built-in behavior jo HTML elements ke behavior ko change karta hai.

```
// Custom directive
import { Directive, ElementRef } from '@angular/core';

@Directive({
  selector: '[appHighlight]'
})
export class HighlightDirective {
  constructor(el: ElementRef) {
    el.nativeElement.style.backgroundColor = 'yellow';
  }
}


<p appHighlight>This text is highlighted</p>
```

6. Service

- Services reusable logic ko encapsulate karte hain.
- Example: API calls, authentication logic, data management, etc.
- Services ko @Injectable decorator ke through define kiya jata hai.

Purpose: Code duplication avoid karna aur logic ko shareable banana.

Service business logic aur data sharing ke liye use hoti hai.
Reusable hoti hai.

```
// message.service.ts
import { Injectable } from '@angular/core';

@Injectable({ providedIn: 'root' })
export class MessageService {
  getMessage() {
    return 'Hello from Service!';
  }
}


// app.component.ts
constructor(private messageService: MessageService) {}

ngOnInit() {
  console.log(this.messageService.getMessage());
}
```

🔌 7. Injector

•Injector ek mechanism hai jo services ko components me **inject (provide)** karta hai.

•Angular ka **Dependency Injection (DI)** system isi injector ke through kaam karta hai.

Purpose: Efficient service sharing aur testability.

8. Metadata

•Angular decorators ke through diya gaya info hota hai.

•Ye batata hai ki class kis purpose ke liye use hogi (component, service, module, etc.)

Example:
•@Component({...})
•@NgModule({...})
•@Injectable({...})

🔁 Summary Flow

- **Module** me components aur services hoti hain.
- **Component** me input-output events aur services inject hote hain.
- **Component** ke HTML templates me **property binding** aur **event binding** hoti hai.
- Templates directives aur metadata use karke behavior define karte hain.
- Services ko **injector** ke through component me provide kiya jata hai.

Symbol	Binding Type	Meaning
[]	Property Binding	Set property from component
()	Event Binding	Call method on user action
[()]	Two-way Binding	Data sync both ways (ngModel)

Angular Project Structure

🧩 **Install:**

1.Node.js → https://nodejs.org/ (LTS version)

2.Angular CLI:

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

Create a new Angular project:

```
ng new project-name --no-standalone
```

Add flags during setup:

- Enable routing → ✓
- Include CSS (default styling) → ✓

Serve (start) the Angular application:

cd project-name ng serve

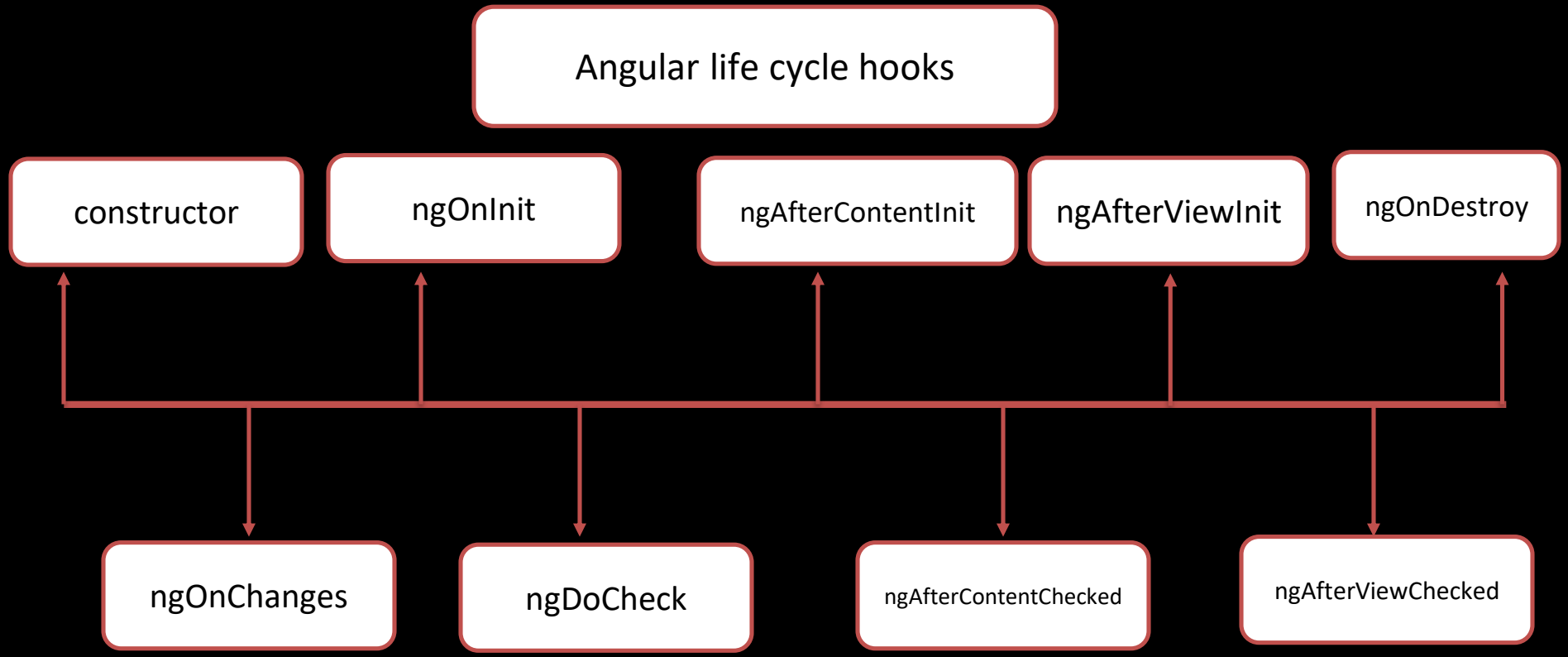
Angular CLI Commands to Generate Components

ng generate component home --no-standalone
ng generate component registration --no-standalone
ng generate component login --no-standalone

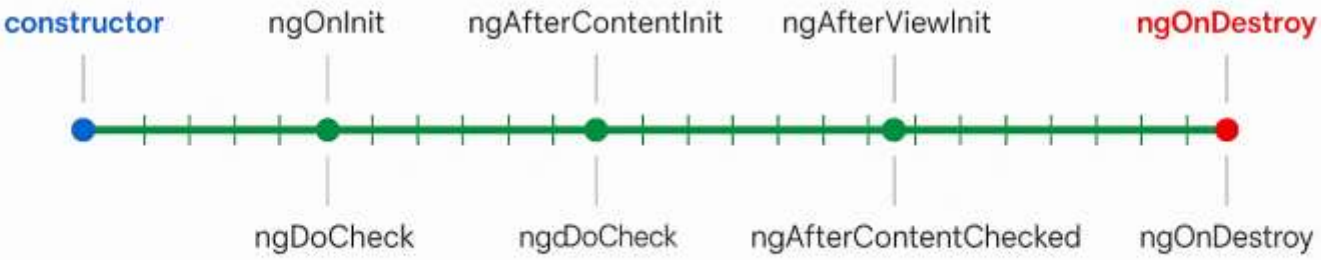
---

ng g c home --no-standalone
ng g c registration --no-standalone
ng g c login --no-standalone
Breakdown:
ng g c → generate component ka short form

--no-standalone → traditional NgModule-based component banane ke liye


Angular life cycle hooks
constructor
ngOnInit
ngAfterContentInit
ngAfterViewInit
ngOnDestroy
ngOnChanges
ngDoCheck
ngAfterContentChecked
ngAfterViewChecked

LIFECYCLE HOOKS



🔄 Angular Lifecycle Hooks Explained:

1.constructor

1. Ye class banne par sabse pehle call hota hai.
2. Yahaan component ka instance create hota hai, but data binding ya DOM access nahi hota.

2.ngOnChanges

1. Jab bhi component ke @Input properties change hote hain, tab ye method trigger hoti hai.
2. Parent component se data receive karne ke baad yeh change track karta hai.

3.ngOnInit

1. Component initialize hone ke baad chalti hai (sirf ek baar).
2. Yahan aap API calls ya initialization logic likh sakte ho.

4.ngDoCheck

1. Angular ke har change detection cycle mein trigger hota hai.
2. Custom change detection ke liye use hota hai.

•**ngAfterContentInit**

•Jab component ke inner content (ng-content) initialize ho jata hai tab call hota hai.
•Ye content projection ke baad run hota hai.

•**ngAfterContentChecked**

•Jab projected content check ho jata hai, tab ye hook trigger hota hai.

•**ngAfterViewInit**

•Jab component ka view (template) aur child views render ho jate hain.
•DOM element ya view child access karne ke liye use hota hai.

•**ngAfterViewChecked**

•Jab view aur child views re-checked hote hain, ye hook call hota hai.

•**ngOnDestroy**
•Jab component destroy hone wala hota hai tab call hota hai.
•Memory clean-up, unsubscribe ya interval clear karne ke liye use hota hai.

Lifecycle Hook	Kab Call Hota Hai?	Use Case Example
ngOnChanges()	@Input property change hone par	Data update from parent
ngOnInit()	Component initialize hone par	API calls, variable set
ngDoCheck()	Har change detection cycle mein	Custom checks
ngAfterContentInit()	Content projection ke baad	ke andar ka kaam
ngAfterContentChecked()	Projected content check hone ke baad	Validate projected data
ngAfterViewInit()	Component view fully load hone ke baad	DOM element access
ngAfterViewChecked()	View check hone ke baad	UI tweaks
ngOnDestroy()	Component destroy hone se pehle	Unsubscribe observables, cleanup timers

•**Constructor (constructor):**
"Sabse pehle humare component ka constructor call hota hai. Yeh tab hota hai jab component ka instance create hota hai.“
Console log: constructor called

•**ngOnChanges:**
"Jab bhi input properties change hoti hain, ngOnChanges call hota hai. Yeh hook tab trigger hota hai jab hum input property ko update karte hain.“

Console log: ngOnChanges called

•**ngOnInit:**
"Component ke initialization ke time pe ngOnInit call hota hai. Yeh ek baar hi trigger hota hai jab component load hota hai."
Console log: ngOnInit called
•**ngDoCheck:**
"Yeh hook custom change detection ke liye hota hai, jab hum apne component mein kuch bhi manually check karte hain.“
•
Console log: ngDoCheck called

- **ngAfterContentInit:**

"Jab content projection complete ho jata hai, tab ngAfterContentInit trigger hota hai."

Console log: ngAfterContentInit called

- **ngAfterContentChecked:**

"Yeh hook har baar content ko check karne ke baad trigger hota hai."

Console log: ngAfterContentChecked called

- **ngAfterViewInit:**

"Jab view (template) initialize hota hai, tab ngAfterViewInit call hota hai."

Console log: ngAfterViewInit called

- **ngAfterViewChecked:**

"View ko check karne ke baad ngAfterViewChecked call hota hai. Yeh har view update ke baad execute hota hai."

Console log: ngAfterViewChecked called

- **ngOnDestroy:**

"Jab component destroy hota hai (ya DOM se remove hota hai), tab ngOnDestroy trigger hota hai."

Console log: ngOnDestroy called

3. Practical Example (with Code Breakdown):
"Ab is example mein dekho ki kaise humne har hook ko use kiya hai. Jab hum name property ko update karenge ya component ko destroy karenge, tab aap console mein dekh sakte ho ki kaunsa hook kab call hota hai."

Before Change Name Button Click:

- constructor called
- ngOnChanges called
- ngOnInit called
- ngDoCheck called
- ngAfterContentInit called
- ngAfterContentChecked called
- ngAfterViewInit called
- ngAfterViewChecked called

- **After Change Name Button Click:**

- ngOnChanges called
- ngDoCheck called
- ngAfterContentChecked called
- ngAfterViewChecked called

- **After Destroy Component Button Click:**

- ngOnDestroy called

lifecycle-demo.component.ts U × | app-routing.module.ts | app.component.html M | lifecycle-demo.component.html U | app.component.ts M | app.modu

lifecycle > src > app > lifecycle-demo > lifecycle-demo.component.ts > ...

(Child Component with Lifecycle Hooks)

```
// Angular core se lifecycle hooks aur decorators import kar rahe hain
import { Component, Input, OnInit, OnChanges, DoCheck, AfterContentInit,
         AfterContentChecked, AfterViewInit, AfterViewChecked, OnDestroy } from '@angular/core';

// Component decorator define karta hai ki yeh component kaise behave karega
@Component({
  selector: 'app-lifecycle-demo', // is selector se hum HTML mein <app-lifecycle-demo> use karenge
  standalone: false,              // agar yeh component kisi module ka part hai toh standalone false
  hota hai
  templateUrl: './lifecycle-demo.component.html', // HTML template ka path
  styleUrl: './lifecycle-demo.component.css'      // CSS styling ka path
})
// Class start ho rahi hai aur saare lifecycle interfaces implement kiye gaye hain
export class LifecycleDemoComponent implements
  OnInit,                 // component initialization ke baad call hota hai
  OnChanges,              // jab bhi @Input property change ho
  DoCheck,                // jab Angular ka custom change detection chale
  AfterContentInit,       // jab ng-content load ho jaye
  AfterContentChecked,    // jab ng-content ka change detection complete ho
  AfterViewInit,          // jab component ka view fully render ho jaye
  AfterViewChecked,       // jab view ka change detection complete ho
  OnDestroy {             // jab component destroy hone wala ho
```

```
// Input decorator se yeh property parent component se value receive karegi
@Input() name: string = '';

// Constructor sabse pehle call hota hai — yahan aap dependencies inject karte ho
constructor() {
  console.log('constructor called');
}

// Jab bhi input property (name) change hoti hai toh ye method call hoti hai
ngOnChanges() {
  console.log('ngOnChanges called');
}

// Component initialization ke baad ye method call hoti hai — mostly setup ke liye use hoti hai
ngOnInit() {
  console.log('ngOnInit called');
}

// Angular ka internal change detection jab chale, toh ye custom method call hota hai
ngDoCheck() {
  console.log('ngDoCheck called');
}

// Jab projected content (ng-content) component mein load ho jaye tab yeh call hota hai
ngAfterContentInit() {
  console.log('ngAfterContentInit called');
}
```

```
// Jab projected content ka change detection ho jaye tab yeh call hota hai
ngAfterContentChecked() {
  console.log('ngAfterContentChecked called');
}

// Jab component ka view render ho jaye (matlab HTML DOM ready ho) tab call hota hai
ngAfterViewInit() {
  console.log('ngAfterViewInit called');
}

// Jab view ka change detection complete ho jaye tab call hota hai
ngAfterViewChecked() {
  console.log('ngAfterViewChecked called');
}

// Jab component destroy hone wala ho (e.g. *ngIf false ho gaya), cleanup yahan hota hai
ngOnDestroy() {
  console.log('ngOnDestroy called');
}
}
```

src > app > app.component.html > ...

(Parent Component Template)

```
<!--
  Child component ko render karte hain sirf agar showComponent true ho
  *ngIf = Angular directive hai jo component ko conditionally show/hide karta hai
  [name]="userName" => parent component se child ko 'userName' variable pass kar rahe hain
-->
<app-lifecycle-demo *ngIf="showComponent" [name]="userName"></app-lifecycle-demo>

<!--
   Ye button click hone par parent component ka 'changeName()' method call karta hai
  Isse userName ka value change hota hai -> jo @Input ke through child component ko milega
-->
<button (click)="changeName()">Change Name</button>

<!--
  Ye button click hone par 'destroyComponent()' call hota hai
  Isme showComponent false ho jata hai -> jisse component destroy ho jata hai
-->
<button (click)="destroyComponent()">Destroy Component</button>
```

(Child Component Template)

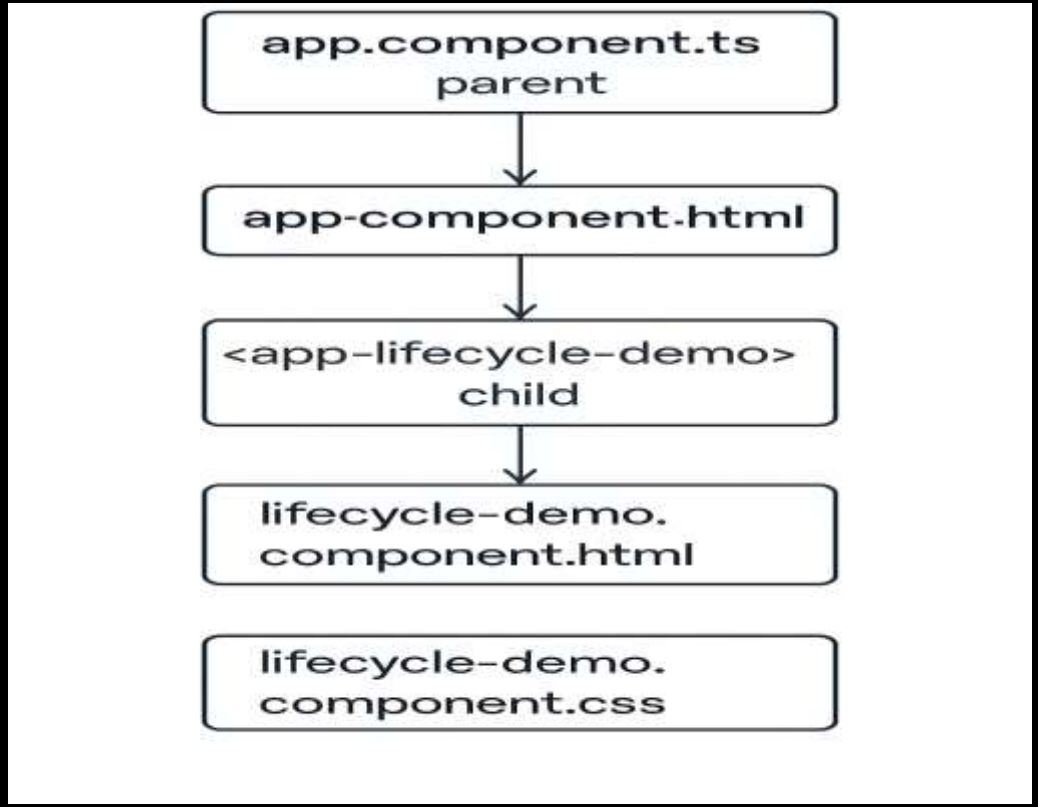
src > app > lifecycle-demo > lifecycle-demo.component.html > ...

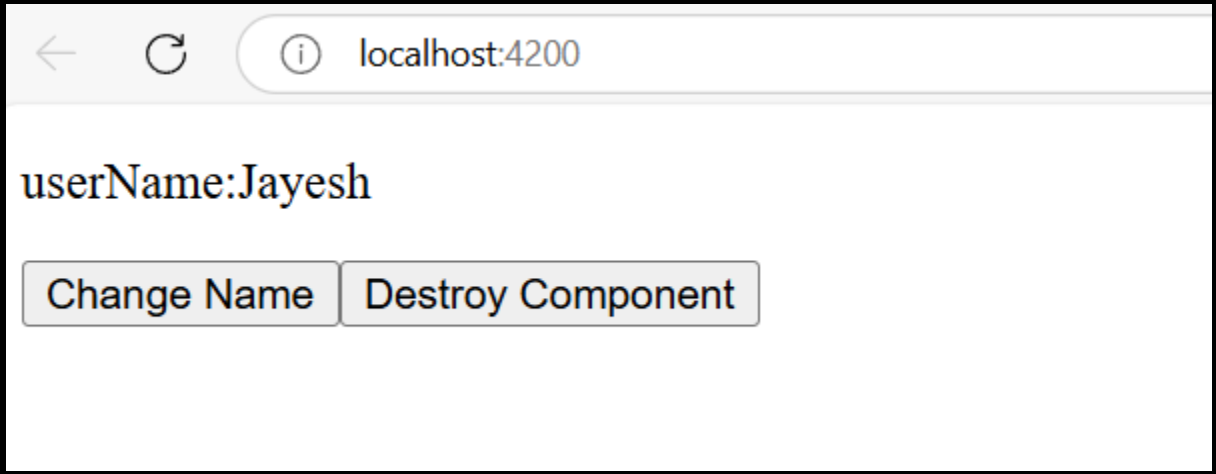
```
<!--
  Is paragraph mein parent se aayi value show ho rahi hai
  {{ name }} interpolation se value bind hoti hai
-->
<p>userName: {{ name }}</p>
```

```
src > app > app.component.ts
// Angular core library se Component decorator import kar rahe hain
import { Component } from '@angular/core';

// Component decorator se batate hain ki yeh class ek Angular component hai
@Component({
  selector: 'app-root',                    // HTML mein use hone wala tag name (custom HTML tag)
  templateUrl: './app.component.html',     // HTML file jisme component ka UI likha hoga
  standalone: false,                       // Yeh component standalone nahi hai (Angular v14+
  feature)
  styleUrl: './app.component.css'          // CSS file jisme styling hoti hai component ke liye
})

// AppComponent class define kar rahe hain jo is component ka logic handle karti hai
export class AppComponent {
  userName = 'Jayesh';        // Parent component ka property, initial value 'Jayesh'
  showComponent = true;       // Yeh flag batata hai ki child component dikhana hai ya nahi

  // Jab user "Change Name" button click karta hai
  changeName() {
    this.userName = 'JK Pathshala'; // userName ko update kar rahe hain
  }

  // Jab user "Destroy Component" button click karta hai
  destroyComponent() {
    this.showComponent = false; // child component ko hata dete hain (ngIf false ho jaata hai)
  }
}
```

(Parent Component)

📊 Angular Lifecycle Flow Diagram (File Name + Arrow Flow)




localhost:4200
userName:Jayesh
Change Name
Destroy Component

a) Give the simple layout of the Angular application with multiple components. Explain how to create and use components in Angular? **[9]**

a) Simple Layout of an Angular Application with Multiple Components:

An Angular application typically follows a component-based architecture. A basic layout with multiple components might look like this:

```
AppComponent
│
├── HeaderComponent
├── SidebarComponent
├── ContentComponent
│   ├── ArticleComponent
│   └── CommentsComponent
└── FooterComponent
```

Each of these components handles a part of the UI and logic independently.

How to Create and Use Components in Angular

1. Creating a Component:

You can create a component using Angular CLI:

```
ng generate component component-name
```

Example:

```
ng generate component header
```

This will create:

- header.component.ts (logic)
- header.component.html (template)
- header.component.css (styles)
- header.component.spec.ts (tests)

2. Using a Component:

To use a component inside another:
•Include its selector in the parent component’s HTML file.

Example:

```
<!-- app.component.html -->
<app-header></app-header>
<app-content></app-content>
<app-footer></app-footer>
```

- Make sure the component is declared in app.module.ts:

```
@NgModule({
  declarations: [
    AppComponent,
    HeaderComponent,
    ContentComponent,
    FooterComponent
  ],
  ...
})
```

3. Component Structure:

Each component has:
•**Template (HTML):** UI layout
•**Class (TS):** Logic and data handling
•**Styles (CSS):** Component-specific styling
•**Metadata (Decorator):** Links all together via @Component
Example:

```
@Component({
  selector: 'app-header',
  templateUrl: './header.component.html',
  styleUrls: ['./header.component.css']
})
export class HeaderComponent {
  title = "My Angular App";
}
```

Angular Modules

What is an Angular Module?

An **Angular Module** is like a **container** that holds parts of your app — such as components, services, pipes, etc.
It helps to keep everything **organized** and **works together**.

Types of Modules

1.Root Module (AppModule)

- The **main module** of your app.
- It starts and runs the app.

2.Feature Module

- A module for a **specific part** of the app.
- Example: UserModule for user-related features.

3.Shared Module

- A module to **share common things** (like buttons or pipes) across the app.

4.Core Module

- A module for **important services** used everywhere (only loaded once).

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

- declarations: Lists all components used.
- imports: Brings in other modules.
- bootstrap: The first component to load (usually AppComponent).

Creating a Module

To make a module using Angular CLI:

```
ng generate module user
```

Then you can add components to it, like UserProfileComponent.

Why Use Modules?

- Keep app parts **organized**
- Make big apps **easy to manage**
- Allow **lazy loading** (load parts only when needed)
- Help with **reusing** common things

Angular Data Binding

Angular Data Binding Explained:
Angular **data binding** is a powerful feature that connects the **component logic** and the **view (HTML template)**. It keeps the UI and the component data in sync.

There are **four main types** of data binding in Angular:

Interpolation	Component → View
Property Binding	Component → DOM element
Event Binding	DOM Event → Component
Two-Way Binding	Component ↔ View

Binding Type	Direction	Example
Interpolation	Component → View	{{ title }}
Property Binding	Component → DOM element	<img [src]="imagePath">
Event Binding	DOM Event → Component	<button (click)="doSomething()">
Two-Way Binding	Component ↔ View	<input [(ngModel)]="value">

1. Interpolation (One-way from Component to View)

Used to display data from the component in the template.

Syntax:{{ title }}

Example:

```
title = 'Welcome to Angular!';

<h1>{{ title }}</h1>
```

b) Explain event binding and property binding in angular with example. [6]

2. Property Binding (One-way from Component to DOM)
Used to bind component properties to HTML element properties.

Syntax:

```
[elementProperty]="componentProperty"
```

Example:

```
<img [src]="imagePath">
```

```
imagePath = 'assets/logo.png';
```

3. Event Binding (One-way from DOM to Component)

Used to listen for user actions like clicks, typing, etc.

Syntax:

```
(event)="methodName()"
```

Example:

html

```
<button (click)="handleClick()">Click Me</button>
```

Typescript

```
handleClick() {
alert('Button was clicked!');
}
```

4. Two-Way Data Binding (Component ↔ View)
Used with ngModel to create a two-way link between the component and the form input.

Syntax:

```
[(ngModel)]="property"
```

Example:

html
```
<input [(ngModel)]="username">
<p>Hello, {{ username }}!</p>
```

typescript
```
username = ‘ ‘;
```

Note: For two-way binding to work, you must import FormsModule in your module.

◆ Directives

Directives are instructions in the DOM that tell Angular how to manipulate elements.

Types of Directives:

Component Directives

Structural Directives

Attribute Directives

Component Directives

In Angular, a **component** is actually a **special type of directive** — one that **has its own HTML template**. **Directive** means: "a class that can modify the structure or behavior of the DOM."

- Technically directives with a template.
- Every component is a directive with its own view (template).
- Example:

```
•@Component({
•  selector: 'app-user',
•  templateUrl: './user.component.html'
•})
•export class UserComponent {}
```

) a) List and explain different types of structural directives in Angular. [6]

Structural Directives are Angular directives that modify the **structure of the DOM** by adding or removing elements.

1. *ngIf

- Used to display an element conditionally.
- Removes or includes elements based on a Boolean expression.

```
<div *ngIf="isLoggedIn">Welcome User!</div>
```

2. *ngFor

•Used to repeat a portion of the DOM based on a collection.

```
<ul>
  <li *ngFor="let item of items">{{ item }}</li>
</ul>
```

3. *ngSwitch

•Used to conditionally display elements based on multiple conditions (like a switch-case).

```
<div [ngSwitch]="status">
  <p *ngSwitchCase='"active"'>Active</p>
  <p *ngSwitchCase='"inactive"'>Inactive</p>
  <p *ngSwitchDefault>Unknown</p>
</div>
```

Attribute Directives

- Change the **appearance or behavior** of an element.
- Built-in: ngClass, ngStyle

Custom example:

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

Q9 b) What is Pipe? Demonstrate the code for pipes in Angular

Angular mein **Pipe** ek aisa feature hota hai jo template mein data ko **transform** karta hai. Pipe ek input leta hai aur usko format karke output deta hai. Pipe ka use | (pipe symbol) se hota hai.

Built-in Pipes ke Examples:

```
<p>{{ 3.14159 | number:'1.2-2' }}</p>      <!-- Output: 3.14 -->
<p>{{ today | date:'fullDate' }}</p>      <!-- Output: Sunday, May 11, 2025 -->
<p>{{ 0.75 | percent }}</p>               <!-- Output: 75% -->
<p>{{ 2500 | currency:'INR' }}</p>        <!-- Output: ₹2,500.00 -->
```

- date: Date ko format karta hai.
- number: Number ko format karta hai.
- currency: Amount ko currency format mein dikhata hai.
- percent: Value ko percent format mein dikhata hai.

◆ What is Pipe in Angular?

Pipe Angular ka ek feature hai jo data ko format karne ke liye use hota hai **HTML template ke andar hi** — bina TypeScript logic likhe.

Pipes use hote hain:

- Date formatting
- Uppercase/lowercase conversion
- Currency, percent formatting
- Custom transformations

🔷 Built-in Pipes Example:

```
<!-- Inside component.html -->
<p>{{ today | date:'fullDate' }}</p>        <!-- Date formatting -->
<p>{{ name | uppercase }}</p>               <!-- UPPERCASE -->
<p>{{ amount | currency:'INR' }}</p>        <!-- ₹ Currency -->
```

🔹 Custom Pipe Banana:

👉 **Step 1: Generate Pipe**

ng generate pipe custom-uppercase

👉 Step 2: Define Pipe Logic

```
// custom-uppercase.pipe.ts
import { Pipe, PipeTransform } from '@angular/core';

@Pipe({
  name: 'customUppercase'
})
export class CustomUppercasePipe implements PipeTransform {
  transform(value: string): string {
    return value.toUpperCase();
  }
}
```

👉 Step 3: Use Pipe in Template

```
<!-- component.html -->
<p>{{ 'hello world' | customUppercase }}</p>
<!-- Output: HELLO WORLD -->
```

Pipe Type	Example	Purpose
uppercase	`{{ name	uppercase }}`
date	`{{ today	date }}`
currency	`{{ amount	currency }}`
Custom Pipe	`{{ value	customPipe }}`

Angular Services and Dependency Injection (DI)

◆ **Angular Services kya hote hain?**
Service ek **class hoti hai jisme logic likha jaata hai** — jaise ki data fetch karna, authentication, ya reusable business logic.

Features of Services:
•Reusable code
•Maintain separation of concerns (component ka code clean rehta hai)
•Easy to test
•Use kiya ja sakta hai multiple components mein

◈ Service ka Example:

```
@Injectable({                    // Service globally available ho jaati hai
  providedIn: 'root'  // Angular will create a single instance of this service
})
export class DataService {
  getData() {
    return ['Item 1', 'Item 2', 'Item 3'];
  }
}
```

➤ **Filename: data.service.ts**
Is file mein service class likhi jaati hai.

➤ **Filename: list.component.ts**
Yahan service ko inject kiya jaata hai (DI ke through)

- Service ko Component mein kaise use karein?

```
import { Component } from '@angular/core';    Duplicate identifier 'Component'
import { DataService } from './data.service';    Cannot find module './data.se
```

// Service import karni zaroori hai

```
@Component({
  selector: 'app-list',
  template: `<ul><li *ngFor="let item of items">{{ item }}</li></ul>`
})
export class ListComponent {
  items: string[];

  constructor(private dataService: DataService) {
    this.items = this.dataService.getData();  // Service se data mil raha hai
  }
}
```

// ← DI yahan ho raha hai

3. **Component Template File**

➤ Filename: list.component.html

Yeh file service se aaye data ko dikhane ke liye hoti hai.

```
<!-- list.component.html -->
<ul>
  <li *ngFor="let item of items">{{ item }}</li>
</ul>
```

4. App Module File

➤ Filename: app.module.ts

Normally agar service providedIn: 'root' likha ho to isme manually add karne ki zarurat nahi hoti. Lekin agar nahi diya hai, to yahan service ko providers mein register karna padta hai.

ts
Copy
Edit

```ts
// app.module.ts
import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';

import { AppComponent } from './app.component';
import { ListComponent } from './list/list.component';
import { DataService } from './data.service';

@NgModule({
  declarations: [AppComponent, ListComponent],
  imports: [BrowserModule],
  providers: [DataService],  // ← Optional if not using providedIn: 'root'
  bootstrap: [AppComponent]
})
export class AppModule {}
```

File Name	Role
data.service.ts	Service logic likhne ke liye
list.component.ts	Component class + service inject karna
list.component.html	Template jo service se aaye data ko show karta hai
app.module.ts	App ke modules aur providers manage karta hai

◆ Dependency Injection (DI) kya hai?

Dependency Injection ek design pattern hai jisme Angular automatically aapke required services ya objects ko **inject** karta hai jahan zarurat ho.

Benefits:

- Loose coupling
- Code reuse
- Easy testing and maintenance

◈ Real-Life Analogy:

Jaise ek ghar ka plumber automatically aapke water tank se pipe connect karta hai bina aapko har jagah plumbing karne ke — waise hi Angular DI automatically service ko component ke constructor mein inject kar deta hai.

Term	Meaning
Service	Reusable logic store karne wali class
DI (Injection)	Automatically provide karna required service ko

Angular Routers

◆ What is Angular Router?

Angular Router allow karta hai aapko ek **single-page application (SPA)** banane mein jahan **different components** browser URL ke basis pe load hote hain **without reloading the page**.

Features:

- Navigate between views/components
- Pass route parameters
- Lazy loading
- Guards for route protection

Concept	Description
RouterModule	Angular routing enable karta hai
Routes[]	Path-component mapping list
routerLink	Navigation link
router-outlet	Jahan routed component render hota hai
redirectTo	Default or fallback path

Example: Angular Routing Step-by-Step

```
src/
├── app/
│   ├── home/
│   │   └── home.component.ts
│   ├── about/
│   │   └── about.component.ts
│   ├── app-routing.module.ts    ←   Routing yahan define hota hai
│   └── app.module.ts
```

src > app > app.module.ts > AppRoutingModule

```
// app-routing.module.ts
import { NgModule } from '@angular/core';
import { RouterModule, Routes } from '@angular/router';
import { HomeComponent } from './home/home.component';
import { AboutComponent } from './about/about.component';    Cannot find

const routes: Routes = [
  { path: 'home', component: HomeComponent },
  { path: 'about', component: AboutComponent },
  { path: '', redirectTo: '/home', pathMatch: 'full' }, // Default route
];

@NgModule({
  imports: [RouterModule.forRoot(routes)],
  exports: [RouterModule]
})
export class AppRoutingModule {}
```

```html
<!-- app.component.html -->
<nav>
  <a routerLink="/home">Home</a> |
  <a routerLink="/about">About</a>
</nav>

<router-outlet></router-outlet>
```

```
// app.module.ts
import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';

import { AppComponent } from './app.component';
import { HomeComponent } from './home/home.component';
import { AboutComponent } from './about/about.component';    Cannot find modu
import { AppRoutingModule } from './app-routing.module';

@NgModule({
  declarations: [AppComponent, HomeComponent, AboutComponent],
  imports: [BrowserModule, AppRoutingModule],  // ← Routing module included
  bootstrap: [AppComponent]
})
export class AppModule {}
```

◆ What are Angular Forms?

Angular forms help you to **handle user inputs**, **validate data**, and **submit** it in a clean and reactive way.

Type	Description
Template-driven	Simple and HTML-based form handling
Reactive forms	Programmatic, more control and scalability

1. Template-driven Forms (Basic, HTML mein likhte hain)

◈ **Step-by-step:**

☞ **app.module.ts**

```
import { FormsModule } from
'@angular/forms';  // Add this
@NgModule({
  imports: [FormsModule]
})
```

👉 HTML Template (form.component.html)

```html
<form #userForm="ngForm" (ngSubmit)="onSubmit(userForm)">
  <input name="username" ngModel required placeholder="Username">
  <button type="submit">Submit</button>
</form>
```

👉 Component Class (form.component.ts)

```typescript
export class FormComponent {
  onSubmit(form: any) {
    console.log('Form Data:', form.value);
  }
}
```

2. Reactive Forms (Structured, used in bigger apps)

◈ Step-by-step:

👉 app.module.ts

```
import { ReactiveFormsModule } from
'@angular/forms';  // Add this
@NgModule({
  imports: [ReactiveFormsModule]
})
```

☞ Component Class (reactive-form.component.ts)

```
import { Component } from '@angular/core';
import { FormGroup, FormControl } from '@angular/forms';

@Component({
  selector: 'app-reactive-form',
  templateUrl: './reactive-form.component.html'
})
export class ReactiveFormComponent {
  userForm = new FormGroup({
    username: new FormControl(''),
    email: new FormControl('')
  });

  onSubmit() {
    console.log(this.userForm.value);
  }
}
```

👉 HTML Template (reactive-form.component.html)

```html
<form [formGroup]="userForm" (ngSubmit)="onSubmit()">
  <input formControlName="username" placeholder="Username">
  <input formControlName="email" placeholder="Email">
  <button type="submit">Submit</button>
</form>
```

Feature	Template-driven	Reactive
Setup	HTML-based	Code-based
Flexibility	Less	More
Validation	Directive-based	Programmatic
Use case	Simple forms	Complex/large forms

Angular – Final Summary

🔷 **Angular** is a powerful **TypeScript-based front-end framework** developed by Google for building **single-page applications (SPA)**. It provides a complete ecosystem including:

Concept	Use/Role
Components	Building blocks of UI (HTML + CSS + TS)
Modules	Organize related components, directives, services, etc.
Templates	HTML with Angular syntax to display data
Directives	Modify DOM behavior (*ngIf, *ngFor, etc.)
Pipes	Format data in templates (date, uppercase, etc.)
Services & DI	Reusable logic with automatic injection into components
Routing	Navigation between components using URLs
Forms	Handle user input with validation (template & reactive)

◆ DOM (Document Object Model) kya hai?

DOM ek **tree structure** hai jo HTML elements ko represent karta hai. Jab bhi hum kisi web page ko browser me dekhte hain, HTML DOM us page ka structure banata hai.

Example:

```
<div>
  <h1>Hello</h1>
  <p>Welcome</p>
</div>
```

Yeh DOM tree me aise dikhega:

```
div
 ├── h1 → "Hello"
 └── p → "Welcome"
```

❌ **Problem:**
Jab hum real DOM me koi change karte hain (e.g., innerHTML, textContent), poora DOM update hota hai — chahe sirf ek line badli ho. Ye slow process hai.

◆ **Virtual DOM kya hai?**
Virtual DOM ek **lightweight copy** hota hai real DOM ka. React isse **memory me** create karta hai.

Kaise kaam karta hai?

1.React pura UI ka virtual DOM create karta hai.
2.Jab koi data/state change hota hai:
    1. React new virtual DOM banata hai.
    2. Usme aur purane virtual DOM me difference (diff) check karta hai.
    3. Sirf **badle hue parts** ko real DOM me update karta hai.

⚡ **Benefit:**
•Fast updates
•Efficient rendering
•Better performance (especially large apps)

- **Analogy:**

Imagine **DOM** as a real house.

•Real DOM = Pure brick house (slow to modify)

•Virtual DOM = Paper model of house (easy to modify)

•React modifies the paper first, then applies **only the changed parts** to the real house.

Feature	Real DOM	Virtual DOM
Performance	Slower	Faster
Updates	Whole tree	Only changed nodes
Memory usage	Normal	Less (virtual in memory)
Used by	All browsers	React and similar libraries

React.js

ReactJS: Introduction to ReactJS, React Components, Inter Components Communication, Components Styling, Routing, Redux- Architecture, Hooks- Basic hooks, useState() hook, useEffect() hook useContext() hook.

React.js

◆ Introduction to React.js

React.js ek popular **JavaScript library** hai jo **user interfaces (UI)** banane ke liye use hoti hai — specially for **Single Page Applications (SPA)**. Yeh Facebook ne banaya tha aur open-source hai.

Feature	Description
Component-Based	UI is broken into small reusable components.
JSX Syntax	JavaScript + HTML-like syntax used in components.
Virtual DOM	Efficient UI updates using a virtual copy of the DOM.
One-way Data Flow	Data flows from parent to child (unidirectional).
Declarative	You describe *what* to show, not *how* to update DOM.

◆ **Why Use React?**

- Fast performance
- Reusable components
- Easy to integrate with other libraries
- Huge community and ecosystem (Redux, React Router, etc.)

```javascript
// App.js
import React from 'react';

function App() {
  return (
    <div>
      <h1>Welcome to React!</h1>
      <p>This is a simple React component.</p>
    </div>
  );
}

export default App;
```

◆ **Real-world Usage:**

- Facebook, Instagram
- WhatsApp Web
- Netflix UI
- E-commerce dashboards

◈ Steps to Install React.js

React run karne ke liye **Node.js** aur **npm (node package manager)** chahiye.

Download from: https://nodejs.org

Check version:

```
node -v
npm -v
```

Create a New React App

```
npx create-react-app my-app
```

```
cd my-app
```

```
npm start
```

◈ 2. React Components – Step by Step

🔹 Step 1: React App Setup

```
npx create-react-app my-app
cd my-app
npm start
```

Step 2: Component kya hota hai?

React mein **component ek reusable code block** hota hai jo UI banata hai. Har component ek function ya class ho sakta hai.

Types of Components

Component Type	Description	Syntax Style	Modern Use
1 Functional Component	JavaScript function ke form mein hota hai. Simple, fast aur hooks compatible.	Function based	Recommended
2 Class Component	JavaScript class hoti hai, render() method use karti hai.	Class based	Less used (old style)

◆ 1. Functional Component (Recommended)

📌 **Features:**

- Lightweight and simple
- Can use **React Hooks** (like useState, useEffect)
- Better performance
- Used in modern React apps

```
import React from 'react';

function Greet(props) {
  return <h2>Hello, {props.name}</h2>;
}

export default Greet;
```

Use in App:

```
<Greet name="Ravi" />
```

◆ 2. Class Component (Old Style)

📌 **Features:**

- Uses render() method to return JSX
- Can hold state and lifecycle methods
- Bigger syntax, less preferred now

```
import React, { Component } from 'react';

class Greet extends Component {
  render() {
    return <h2>Hello,
{this.props.name}</h2>;
  }
}

export default Greet;
```

Feature	Functional Component	Class Component
Syntax	Function	Class + render()
State Handling	via Hooks (useState)	via this.state
Lifecycle Methods	via useEffect	using built-in methods
Performance	High	Comparatively Lower
Code Simplicity	Simple	Verbose
Usage Today	Most Used	Rarely Used

◆ What is Inter-Component Communication in React?

React mein jab ek component doosre component ke saath **data share** karta hai, ya unke beech **interaction** hota hai, use hum **Inter-Component Communication** kehte hain.

4 Main Types of Communication:

Type	Description
1 Parent → Child	Via **props**
2 Child → Parent	Via **callback function (props)**
3 Sibling ↔ Sibling	Via **common parent**
4 Unrelated Components	Via **Context API** or **Redux**

Output: Hello Ravi from Child Component!

1. **Parent to Child Communication (via props)**

📝 Parent.js

```
import React from 'react';
import Child from './Child';

function Parent() {
  return (
    <div>
      <h1>Parent Component</h1>
      <Child name="Ravi" />
    </div>
  );
}

export default Parent;
```

📝 Child.js

```
import React from 'react';

function Child(props) {
  return <p>Hello {props.name} from Child
Component!</p>;
}

export default Child;
```

🔷 2. **Child to Parent Communication (via callback props)**

🔧 **Step-by-Step**

📝 Parent.js

```
import React from 'react';
import Child from './Child';

function Parent() {
  const greetParent = (childName) => {
    alert(`Hello from ${childName} to Parent!`);
  };

  return <Child onGreet={greetParent} />;
}

export default Parent;
```

📝 Child.js

```
import React from 'react';

function Child(props) {
  return (
    <button onClick={() => props.onGreet("Child Component")}>
      Greet Parent
    </button>
  );
}

export default Child;
```

Output: Button click par alert aayega →
"Hello from Child Component to Parent!"

Output: A ka button click karne par B mein message dikhai dega.

🔷 3. **Sibling Components Communication (via common parent)**

📝 Parent.js

```
import React, { useState } from 'react';
import SiblingA from './SiblingA';
import SiblingB from './SiblingB';

function Parent() {
  const [message, setMessage] = useState('');

  return (
    <div>
      <SiblingA sendMessage={setMessage} />
      <SiblingB message={message} />
    </div>
  );
}

export default Parent;
```

📝 SiblingA.js

```
function SiblingA({ sendMessage }) {
  return <button onClick={() => sendMessage("Hello from A")}>Send to B</button>;
}
export default SiblingA;
```

📝 SiblingB.js

```
function SiblingB({ message }) {
  return <p>Message from A: {message}</p>;
}
export default SiblingB;
```

Note: Property names camelCase format mein honi chahiye (e.g., backgroundColor).

Component Styling in React

◆ **1. Inline Styling**

•Style object ko JavaScript mein define karke JSX ke element mein directly use karte hain.

```
function InlineStyledComponent() {
  const headingStyle = {
    color: 'blue',
    backgroundColor: 'lightyellow',
    padding: '10px',
    borderRadius: '5px'
  };

  return <h1 style={headingStyle}>This is Inline Styled</h1>;
}
export default InlineStyledComponent;
```

◆ **2. CSS Stylesheet**

•Ek alag CSS file banate hain aur component mein import karte hain.

📝 **App.css**

```
.title {
  color: green;
  font-size: 24px;
  text-align: center;
}
```

📝 App.js

```
import './App.css';

function App() {
  return <h1 className="title">Styled using CSS File</h1>;
}
export default App;
```

◆ 3. CSS Modules (Scoped CSS)

•Localized CSS – file ke scope mein hi kaam karega, dusre components ko affect nahi karega.

Message.module.css

```
.success {
  color: white;
  background-color: green;
  padding: 10px;
}
```

Message.js

```
import styles from './Message.module.css';

function Message() {
  return <div className={styles.success}>This is a success message</div>;
}
export default Message;
```

◆ 4. Styled-Components (using a library)

•Yeh ek external library hai jisme hum components ke andar hi CSS likh sakte hain.

npm install styled-components

```
import styled from 'styled-components';

const StyledButton = styled.button`
  background-color: purple;
  color: white;
  padding: 10px;
  border: none;
  border-radius: 5px;
`;

function StyledComponent() {
  return <StyledButton>Click Me</StyledButton>;
}
export default StyledComponent;
```

Method	Scope	Reusability	CSS Separation	Recommended?
Inline Styling	Local	✗	✗	✗ Small cases
CSS File	Global	✓	✓	✓
CSS Module	Local	✓	✓	✓ Modern React
Styled Components	Component	✓	✓ (inline CSS)	✓ Advanced use

React Routing

React mein **Routing** ka use hota hai ek single-page application (SPA) mein multiple pages ya views handle karne ke liye **without page reload**. React ke liye sabse commonly used routing library hai **React Router**.

React Routing – Step by Step

Step 1: Install React Router

```
npm install react-router-dom
```

Step 2: Basic Folder Structure

```
/src
  /components
    Home.js
    About.js
    Contact.js
  App.js
  index.js
```

⬍ Step 3: Create Components (Example)

📝 Home.js

```
function Home() {
  return <h2>Welcome to Home Page</h2>;
}
export default Home;
```

About.js

```
function About() {
  return <h2>This is About Page</h2>;
}
export default About;
```

About.js

```
function About() {
  return <h2>This is About Page</h2>;
}
export default About;
```

Contact.js

```
function Contact() {
  return <h2>Contact us </h2>;
}
export default Contact;
```

Step 4: Set up Routes in App.js

```
import { BrowserRouter, Routes, Route, Link } from 'react-router-dom';
import Home from './components/Home';
import About from './components/About';
import Contact from './components/Contact';

function App() {
  return (
    <BrowserRouter>
      <nav>
        <Link to="/">Home</Link> | {" "}
        <Link to="/about">About</Link> | {" "}
        <Link to="/contact">Contact</Link>
      </nav>

      <Routes>
        <Route path="/" element={<Home />} />
        <Route path="/about" element={<About />} />
        <Route path="/contact" element={<Contact />} />
      </Routes>
    </BrowserRouter>
  );
}
export default App;
```

App.js

◆ Step 5:📝 index.js

```
// React library ko import karte hain taaki hum JSX use kar saken
import React from 'react';

// ReactDOM se hum React component ko actual browser DOM mein render karte hain
import ReactDOM from 'react-dom/client';

// App component ko import kar rahe hain (ye hamara main component hai)
import App from './App';

// HTML file ke andar jisme <div id="root"></div> hota hai, usko target kar rahe hain
const root = ReactDOM.createRoot(document.getElementById('root'));

// App component ko render karte hain root element ke andar
root.render(<App />);
```

📝 *Note*: Make sure aapki index.html file mein id="root" wala div present ho:

```
<div id="root">
</div>
```

◆ What is Redux?

Redux is a **state management library** used mostly with React.
It helps you manage **global state** — i.e., data that multiple components need access to (like user info, cart items, login status, etc.).

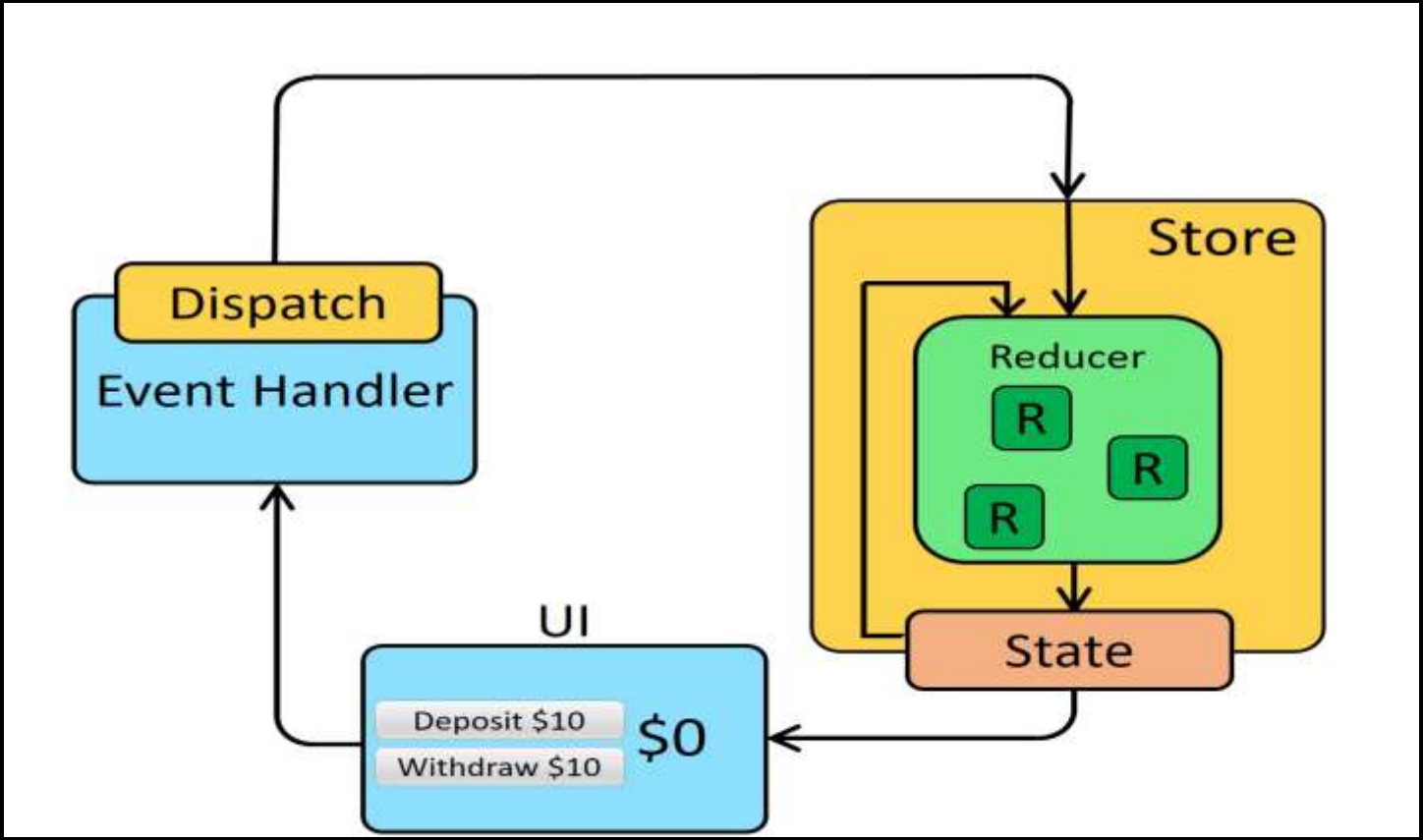
◈ Why use Redux?

React ke components apna **local state** rakh sakte hain (useState) —
lekin jab multiple components
ko ek shared state chahiye hoti hai, tab Redux ka use hota hai.

⬘ Key Concepts of Redux:

Concept	Description
Store	Global state container — yahan puri app ka state hota hai
Action	Plain JS object — describes **what happened**
Reducer	Function that decides **how state changes** based on action
Dispatch	Method to send action to reducer
Selector	Function to get specific data from the store

Redux Architecture (Basic Flow)



🔄 Step-by-Step Explanation:

🔹 1. UI (User Interface)

•Yahan par user koi action karta hai, jaise button dabata hai — Deposit $10 ya Withdraw $10.
•Ye event kisi function (event handler) ko trigger karta hai.

🔹 2. Event Handler + Dispatch

•Ye event handler dispatch() function ko call karta hai.
•dispatch() ek **action** ko Redux ke paas bhejta hai — example: { type: "DEPOSIT", amount: 10 }.

🔹 3. Store

•**Store** Redux ka main object hota hai jisme pura state (data) hota hai.
•Jab dispatch action karta hai, store us action ko **Reducer** ke pass bhejta hai.

3. Store

•**Store** Redux ka main object hota hai jisme pura state (data) hota hai.
•Jab dispatch action karta hai, store us action ko **Reducer** ke pass bhejta hai.

4. Reducer

•Reducer ek pure function hota hai jo current state aur action ko use karke naya state banata hai.
•Multiple reducers ho sakte hain (diagram me teen R dikh rahe hain).
•Reducer update karta hai state based on action type.

5. State

•Reducer se naya state milta hai aur wo store me update ho jata hai.
•Ye naya state firse UI tak pahunchta hai.

🔄 6. UI Updated

•UI ko naya state milta hai aur firse render ho jata hai, jaise: balance ab $10 dikh raha ho.

🔁 **Ye Cycle Continuous Hai:**

1. UI → Dispatch
2. Dispatch → Reducer
3. Reducer → State Update
4. State → UI Update

💡 **Summary :**

•Redux me data sirf ek direction me flow karta hai — **one-way data flow**.

•User action → Dispatch → Reducer → State → UI.

Component	Description
BrowserRouter	Main wrapper for routing
Routes	Container for all Route components
Route	Defines path and component to render
Link	Used to navigate (like <a> but without reload)
useNavigate()	Programmatic navigation (inside events or functions)

◈ Step-by-Step Example (Simple Counter App)

Step 1: Install Redux and React-Redux

```
npm install redux react-redux
```

Step 2: Create Actions – actions.js

```
// actions.js
export const increment = () => {
  return { type: 'INCREMENT' };
};

export const decrement = () => {
  return { type: 'DECREMENT' };
};
```

Step 3: Create Reducer – counterReducer.js

```
// counterReducer.js
const initialState = { count: 0 };

const counterReducer = (state = initialState, action) => {
  switch(action.type) {
    case 'INCREMENT':
      return { count: state.count + 1 };
    case 'DECREMENT':
      return { count: state.count - 1 };
    default:
      return state;
  }
};

export default counterReducer;
```

Step 4: Combine Reducers (optional for large apps) – rootReducer.js

```
// rootReducer.js
import { combineReducers } from 'redux';
import counterReducer from
'./counterReducer';

export const rootReducer = combineReducers({
  counter: counterReducer
});
```

Step 5: Create Store – store.js

```
// store.js
import { createStore } from 'redux';
import { rootReducer } from
'./rootReducer';

const store = createStore(rootReducer);
export default store;
```

Step 6: Provide Store in App – index.js

```
import React from 'react';
import ReactDOM from 'react-dom';
import App from './App';
import { Provider } from 'react-
redux';
import store from './store';

ReactDOM.render(
  <Provider store={store}>
    <App />
  </Provider>,
  document.getElementById('root')
);
```

Step 7: Use Redux in Component – Counter.js

```
import React from 'react';
import { useSelector, useDispatch } from 'react-redux';
import { increment, decrement } from './actions';

const Counter = () => {
  const count = useSelector(state => state.counter.count);
  const dispatch = useDispatch();

  return (
    <div>
      <h1>Count: {count}</h1>
      <button onClick={() => dispatch(increment())}> +
</button>
      <button onClick={() => dispatch(decrement())}> -
</button>
    </div>
  );
};
```

- 🔄 Redux uses **one global store**.
- 📤 You **dispatch** actions.
- 🧠 Reducers update the store based on action type.
- 🧩 Components subscribe to store via useSelector.

⬙ React Hooks - Basic Overview

Hooks React 16.8 ke baad aaye the jo functional components me bhi features jaise state aur lifecycle use karne dete hain — bina class banaye.

◈ 1. useState() Hook

Ye React me state banane ke liye hota hai functional component me.

const [count, setCount] = useState(0);

```
import React, { useState } from 'react';

function Counter() {
  const [count, setCount] = useState(0);

  return (
    <>
      <p>You clicked {count} times</p>
      <button onClick={() => setCount(count + 1)}>Click</button>
    </>
  );
}
```

```
useEffect(() => {
  // logic
}, [dependencies]);
```

◈ 2. useEffect() Hook

Ye hook side effects ke liye use hota hai (e.g., API call, timer, updating title).

```
import React, { useState, useEffect } from 'react';

function Timer() {
  const [count, setCount] = useState(0);

  useEffect(() => {
    document.title = `Clicked ${count} times`;
  }, [count]);

  return <button onClick={() => setCount(count +
1)}>Click</button>;
}
```

◈ 3. useContext() Hook

Global data (theme, user, language) ko access karne ke liye use hota hai bina props drill kiye.

```
const MyContext = React.createContext();
```

```
import React, { useContext } from 'react';

const ThemeContext = React.createContext("light");

function ThemedButton() {
  const theme = useContext(ThemeContext);
  return <button className={theme}>I am {theme} theme</button>;
}
```

⬘ React Hooks Mini Project (Theme + Counter)

Step 1: Project Setup

```
npx create-react-app hooks-demo
cd hooks-demo
npm start
```

Step 2: Folder Structure

```
src/
├── App.js
├── ThemeContext.js
└── components/
    ├── Counter.js
    └── ThemeToggler.js
```

Step 3: ThemeContext.js (Context setup)

```
import React from 'react';

const ThemeContext = React.createContext();

export default ThemeContext;
```

Step 4: Counter.js

```
import React, { useState, useEffect, useContext } from 'react';
import ThemeContext from '../ThemeContext';

function Counter() {
  const [count, setCount] = useState(0);
  const theme = useContext(ThemeContext);

  useEffect(() => {
    document.title = `Clicked ${count} times`;
  }, [count]);

  return (
    <div style={{ background: theme === 'dark' ? '#333' : '#eee', padding: '20px' }}>
      <h2>Count: {count}</h2>
      <button onClick={() => setCount(count + 1)}>Increment</button>
    </div>
  );
}

export default Counter;
```

Step 5: ThemeToggler.js

```
import React, { useContext } from 'react';
import ThemeContext from '../ThemeContext';

function ThemeToggler({ toggleTheme }) {
  const theme = useContext(ThemeContext);

  return (
    <button onClick={toggleTheme}>
      Switch to {theme === 'light' ? 'Dark'
: 'Light'} Theme
    </button>
  );
}

export default ThemeToggler;
```

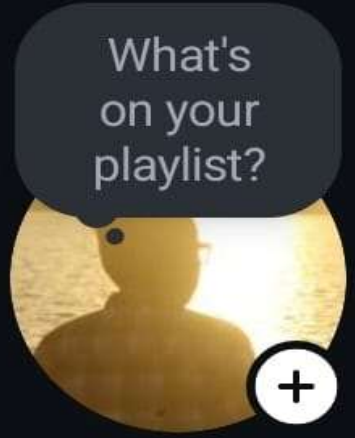
Step 6: App.js

```
import React, { useState } from 'react';
import ThemeContext from './ThemeContext';
import Counter from './components/Counter';
import ThemeToggler from './components/ThemeToggler';

function App() {
  const [theme, setTheme] = useState('light');

  const toggleTheme = () =>
    setTheme((prevTheme) => (prevTheme === 'light' ? 'dark' : 'light'));

  return (
    <ThemeContext.Provider value={theme}>
      <div style={{ padding: '30px' }}>
        <ThemeToggler toggleTheme={toggleTheme} />
        <Counter />
      </div>
    </ThemeContext.Provider>
  );
}

export default App;    A module cannot have multiple default exports.
```

📌 **What This Project Demonstrates:**

- useState ➤ Counter + Theme toggle
- useEffect ➤ Update document title
- useContext ➤ Share theme globally

jayesh_kande_



Jayesh Kande

16	275	276
posts	followers	following

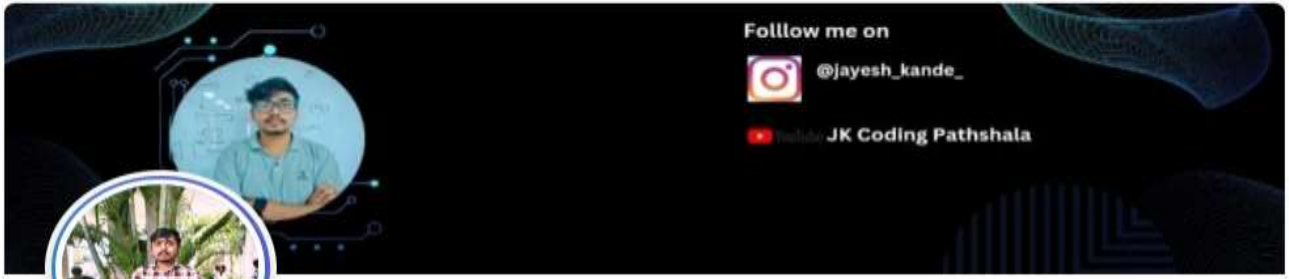
23

रास्ते बदलो, मंजिल नहीं

yt.openinapp.co/0y0qd

LinkedIn

Articles | People | Learning | Jobs | Games



Jayesh Kande

Third-Year IT Engineering Student | Aspiring Web Developer | Java Enthusiast | Data Structures & Algorithms Learner | Proficient in C, C++, Java, and MERN Stack | AI + Web Development Project Enthusiast

Nashik, Maharashtra, India · Contact Info

494 followers · 495 connections

See your mutual connections

Join to view profile | Message

Kbt engineering college nashik

✨ Thank You for Watching! ✨

➔📱 Follow us on Instagram:**@jayesh_kande_**
🔗 Connect with us on LinkedIn:[**Jayesh Kande]**