**🧠 What Are Angular Services?**

**✅ Definition:**

A **service** is a class with reusable logic that can be **shared across components**.

It helps you:

* Separate **business logic** from UI
* **Reuse code** across components
* Keep components **clean and focused**

**📦 Common Uses of Services:**

| **Purpose** | **Example** |
| --- | --- |
| Data management | Storing or fetching todo items |
| API communication | Using HttpClient to fetch data |
| Auth & user state | Tracking login info |
| Shared logic | Utility functions, caching |

**⚙️ How to Create a Service**

bash

CopyEdit

ng generate service todo

This creates:

ts

CopyEdit

// todo.service.ts

@Injectable({

providedIn: 'root' // Auto-provided in root module

})

export class TodoService {

constructor() { }

getTodos() {

return [ { title: 'Learn Angular' }, { title: 'Build App' } ];

}

}

**💉 What is Dependency Injection (DI)?**

**✅ Definition:**

DI is a **design pattern** where Angular **automatically provides an instance** of a service to components or other services.

**🧊 Real-Life Analogy:**

Imagine you're ordering pizza 🍕:

* You **need cheese** (a dependency)
* Instead of making it yourself, the restaurant **injects** cheese into your pizza 🍕

Same way:  
Angular **injects TodoService** into your component, so you don’t create it manually.

**✅ How to Inject a Service**

ts

CopyEdit

constructor(private todoService: TodoService) { }

ngOnInit() {

this.todos = this.todoService.getTodos();

}

Angular looks at the constructor, sees you need TodoService, and **automatically provides it**.

**🌐 ProvidedIn vs NgModule**

**1. providedIn: 'root'**

* Automatically available everywhere in the app
* Recommended default

**2. Register manually in a module:**

ts

CopyEdit

@NgModule({

providers: [TodoService]

})

Not recommended unless you want **scoped services**.

**🛠 Service with HTTP Example**

ts

CopyEdit

@Injectable({ providedIn: 'root' })

export class TodoService {

constructor(private http: HttpClient) {}

getTodos() {

return this.http.get<Todo[]>('/api/todos');

}

}

In the component:

ts

CopyEdit

this.todoService.getTodos().subscribe(data => this.todos = data);

**💥 Singleton Nature of Services**

Services are usually **singletons**:

* One instance is shared across the app
* Keeps state consistent (e.g., login info, shopping cart)

**✅ Advanced: Hierarchical DI**

You can provide a service at:

* **Root level** → providedIn: 'root'
* **Module level**
* **Component level** → using providers: [MyService]

🔁 If provided in a component, a **new instance** is created for that component and its children.

**🔍 Questions**

1. **Why are services used in Angular?**

To separate business logic, reuse code, and handle shared data or API interactions.

1. **What is Dependency Injection?**

A design pattern where dependencies are automatically provided instead of being created manually.

1. **Difference between providing a service in root vs component?**

root: singleton (shared).  
component: new instance (scoped).

1. **Can you inject a service into another service?**

Yes. This is called **service chaining** or **service hierarchy**.

**🧪 Mini Exercise**

Create a QuoteService that:

* Returns a list of quotes (strings)
* Inject it into QuoteComponent
* Display quotes with \*ngFor

Q. what is instance

**💡 What is an *Instance*?**

An **instance** is a **specific copy of an object** created from a class.

**📦 In Simple Words:**

A **class** is like a **blueprint**.  
An **instance** is the **real object** built using that blueprint.

**🔧 Example in Code:**

ts

CopyEdit

class Car {

start() {

console.log("Engine started!");

}

}

const myCar = new Car(); // ← this is an instance

myCar.start(); // Output: Engine started!

Here:

* Car is the class (blueprint)
* myCar is an **instance** of that class

**🧠 In Angular Services:**

When you inject a service like this:

ts

CopyEdit

constructor(private todoService: TodoService) {}

Angular creates **one instance** of TodoService and gives it to any component that asks for it.

If the service is provided in root, that **same instance** is reused (singleton).  
If you provide it in a component, **a new instance** is created just for that component.

**🔁 Why It Matters:**

Using instances means:

* You can **store data** in services (like user info)
* And it **persists** as long as the instance exists

Let me know if you'd like a visual diagram or animation to explain it better!

**Q.** why injctable provided in root why can we use something else

**✅ Why We Use @Injectable({ providedIn: 'root' }) in Angular**

**📌 What it means:**

ts

CopyEdit

@Injectable({

providedIn: 'root'

})

export class MyService { }

This tells Angular:

"Create **one global instance** of this service and make it available **everywhere** in the app."

This is called a **singleton service**.

**🔍 What Are the Other Options?**

Besides 'root', you can use:

**1. 'root' (✅ Default & Recommended)**

* Angular creates one instance **shared across the app**.
* Automatically included in the build.
* No need to register it in any NgModule.

**2. 'platform' (👨‍🔬 Advanced)**

* Makes the service available across **multiple Angular apps** running on the same page.
* Used **very rarely** in large enterprise apps with micro frontends.

**3. 'any' (Scoped instance)**

* A **new instance** is created for each lazy-loaded module or component that uses the service.
* Useful when **you don't want to share state globally**.

ts

CopyEdit

@Injectable({

providedIn: 'any'

})

**4. No providedIn — Manual Registration**

You can also omit providedIn and register the service manually in a module:

ts

CopyEdit

@NgModule({

providers: [MyService]

})

export class AppModule { }

This is **less preferred** because:

* It’s more boilerplate
* You can accidentally register services multiple times

**💡 When to Use What?**

| **providedIn Value** | **Instance Scope** | **Use When...** |
| --- | --- | --- |
| 'root' | Global singleton | ✅ Default. Works for 95% of cases |
| 'any' | Lazy-loaded instances | You want **isolated state** per module |
| 'platform' | Shared across platforms | Rare, micro-frontends |
| No providedIn | Manual | Legacy apps or strict control needed |

**🧠 Summary:**

* ✅ Use providedIn: 'root' for most services.
* Use 'any' if you want **separate instances** per module.
* Avoid manual providers: [] unless needed.

Q. if i have and loggerService which has something common in both loggerService A and loggerService B

Let’s say you have:

* LoggerServiceA
* LoggerServiceB
* Both of them share **some common logic** (e.g. formatting, saving logs, sending logs to server)

Instead of duplicating that code in both services, you can move the **shared logic** to a **base class** or a **shared service**.

**✅ Option 1: Use a Base Class**

**🧱 Step 1: Create a Base Logger**

ts

CopyEdit

export class BaseLogger {

formatMessage(message: string): string {

const timestamp = new Date().toISOString();

return `[${timestamp}] ${message}`;

}

}

**🧱 Step 2: Extend It in A and B**

ts

CopyEdit

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

@Injectable({ providedIn: 'root' })

export class LoggerServiceA extends BaseLogger {

log(message: string) {

console.log(this.formatMessage('A: ' + message));

}

}

@Injectable({ providedIn: 'root' })

export class LoggerServiceB extends BaseLogger {

log(message: string) {

console.log(this.formatMessage('B: ' + message));

}

}

Now both services inherit the formatMessage() method from BaseLogger.

**✅ Option 2: Use a Shared Utility/Helper Service**

If you're not using inheritance, you can make a **SharedLoggerUtilityService** and inject it into both.

**🔧 SharedLoggerService:**

ts

CopyEdit

@Injectable({ providedIn: 'root' })

export class SharedLoggerService {

format(message: string) {

const time = new Date().toLocaleString();

return `[${time}] ${message}`;

}

}

**🔧 Use it inside LoggerServiceA and B:**

ts

CopyEdit

@Injectable({ providedIn: 'root' })

export class LoggerServiceA {

constructor(private sharedLogger: SharedLoggerService) {}

log(msg: string) {

console.log(this.sharedLogger.format('A: ' + msg));

}

}

ts

CopyEdit

@Injectable({ providedIn: 'root' })

export class LoggerServiceB {

constructor(private sharedLogger: SharedLoggerService) {}

log(msg: string) {

console.log(this.sharedLogger.format('B: ' + msg));

}

}

**🧠 Which Approach Should You Use?**

| **Scenario** | **Use This** |
| --- | --- |
| You want to **extend functionality** | Inheritance |
| You want **composition + flexibility** | Shared service |
| You want **easy testing/replacement** | Shared service |

Q. can i use the specific component instead of root

Yes, you **absolutely can** provide a service to a **specific component** instead of using providedIn: 'root' — and this is a **very powerful feature** in Angular called **Component-Level Dependency Injection**.

Let’s break it down simply:

**✅ What Does It Mean?**

Normally, when you use:

ts

CopyEdit

@Injectable({ providedIn: 'root' })

➡ Angular creates **one global (singleton) instance** of that service shared across the whole app.

But when you **register a service in a component**, Angular creates a **new instance** of the service **just for that component and its children**.

**🧱 How to Provide a Service in a Component**

**👇 Example:**

ts

CopyEdit

@Component({

selector: 'app-my-widget',

templateUrl: './my-widget.component.html',

providers: [MyService] // ✅ Scoped service

})

export class MyWidgetComponent {

constructor(private myService: MyService) {}

}

✅ This creates a **new instance** of MyService only for MyWidgetComponent (and its children).

**🧠 Why Use Component-Scoped Services?**

| **Scenario** | **Why It’s Helpful** |
| --- | --- |
| You want **isolated state** per component | e.g., separate counters, logs, configs |
| You want **different behavior** in different places | e.g., LoggerA vs LoggerB |
| You want to **avoid global pollution** | keeps app modular and testable |
| You want **multiple instances** of same service | each component has its own |

**🛑 Warning:**

If you provide a service both:

* in @Injectable({ providedIn: 'root' }) AND
* in a @Component.providers[]

Then Angular uses the **component one**, because it’s **closer in the DI hierarchy**.

**✅ Real Use Case: Logger A & B**

You can do:

ts

CopyEdit

@Component({

selector: 'app-a',

providers: [{ provide: LoggerService, useClass: LoggerA }]

})

export class ComponentA {}

@Component({

selector: 'app-b',

providers: [{ provide: LoggerService, useClass: LoggerB }]

})

export class ComponentB {}

Now LoggerService behaves differently in ComponentA and ComponentB 🔥

**🔁 Summary**

| **Provided In** | **Scope** | **Instance Behavior** |
| --- | --- | --- |
| 'root' | Whole app | Singleton (one copy) |
| Component | That component only | New instance per component |
| Module | Module-level | Depends on module structure |