Angular Services and Dependency Injection

23/02/25

What We'll Cover:

What Are Services? (Reusable Logic)

Creating a Service (ng generate service)

Injecting Services into Components

Sharing Data Across Components

Understanding Dependency Injection in Angular

What Are Services?

Concept: Services are classes that hold reusable logic

Purpose:

- Avoid duplicating code in components
- Centralize business logic, data fetching, or utilities

Examples:

- Fetching data from an API
- Storing app-wide state (e.g., user info)

Analogy: A librarian who manages books (data/logic) for everyone

Creating a Service

How: Use Angular CLI to generate a service

Command: ng generate service task

Output: Creates task.service.ts in src/app

Example:

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

@Injectable({
  providedIn: 'root' // Singleton, available app-wide
})
export class TaskService {
  getTasks() {
        return ['Task 1', 'Task 2', 'Task 3'];
  }
}
```

Key Point: @Injectable makes it ready for injection

Injecting Services into Components

Concept: Use dependency injection to access services

Steps:

1. Add service to component’s constructor
2. Angular provides the instance automatically

Example:

```
import { Component } from '@angular/core';
import { TaskService } from './task.service';
@Component({
  selector: 'app-task-list',
  standalone: true,
  template: `<ul>@for (task of tasks; track task) { <li>{{ task }}</li> }</ul>`
})
export class TaskListComponent {
  tasks: string[];
  constructor(private taskService: TaskService) {
        this.tasks = this.taskService.getTasks();
  }
}
```

Output: List of tasks from the service

Sharing Data Across Components

Concept: Services act as a single source of truth

Example: Share a task list between two components

Service:

```
@Injectable({
  providedIn: 'root'
})
export class TaskService {
  private tasks = ['Learn Angular', 'Build App'];
  getTasks() {
        return this.tasks;
  }
  addTask(task: string) {
        this.tasks.push(task);
  }
}
```

Component 1 (Display):

```
template: `<ul>@for (task of tasks; track task) { <li>{{ task }}</li> }</ul>`

tasks = this.taskService.getTasks();
```

Component 2 (Add):

```
template: `<input #taskInput (keyup.enter)="add(taskInput.value)">`

add(task: string) { this.taskService.addTask(task); }
```

Result: Adding in one updates the other

Understanding Dependency Injection

Concept: Angular’s way of providing dependencies

How It Works:

- Angular’s injector creates and manages service instances
- Services are injected via constructor parameters

Benefits:

- Loose coupling (components don’t create services)
- Easy testing (mock dependencies)

Scope:

- providedIn: 'root' = Singleton, app-wide
- Can scope to modules or components for specific use

Putting It Together

Mini-Project: Task manager with a service

Service:

```
@Injectable({
  providedIn: 'root'
})
export class TaskService {
  private tasks: string[] = [];
  getTasks() { return this.tasks; }
  addTask(task: string) { this.tasks.push(task); }
}
```

Component:

```
@Component({
  standalone: true,
  imports: [CommonModule, FormsModule],
  template: `
      <input [(ngModel)]="newTask" (keyup.enter)="addTask()">
      <ul>@for (task of tasks; track task) { <li>{{ task }}</li> }</ul>
      @if (tasks.length === 0) { <p>No tasks yet!</p> }
  `
})
```

```
export class TaskManagerComponent {
  tasks: string[];
  newTask = '';
  constructor(private taskService: TaskService) {
        this.tasks = this.taskService.getTasks();
  }
  addTask() {
        if (this.newTask) {
        this.taskService.addTask(this.newTask);
        this.newTask = '';
        }
  }
}
```

Key Takeaways

- Services hold reusable logic and data
- Create them with ng generate service
- Inject them via constructors
- Share data app-wide with singleton services
- Dependency Injection = Angular’s magic for modularity

Practice & Next Steps

- **Practice**:
    - Add a deleteTask method to the service
    - Create a second component to display tasks differently
- **Next Lesson**: Routing and Navigation
- **Question**: How might you use a service in your app?