6 Months Training TR-104 Full Stack Development

January, 2026

Day 4 Report

The fourth day of the training focused on completing the remaining modern Angular template control flow concepts and introducing one of the most important architectural fundamentals of Angular: Dependency Injection and Services. The session emphasized building a strong conceptual understanding of how Angular applications manage shared logic and state, rather than rushing into advanced features. Practical demonstrations were carried out using a fresh Angular project to ensure clarity and correctness.

Completion of Modern Angular Template Control Flow

The session began by completing the remaining modern Angular template control flow constructs that were pending from earlier discussions.

Multiple Conditional Rendering (@switch, @case, @default)

Participants learned how to handle multiple UI states using the ‘@switch’ syntax. This approach was introduced as a cleaner and more maintainable alternative to deeply nested ‘@if’ statements when handling more than two conditions.

Key points:

- **@switch:** evaluates a single value
- **@case:** matches specific values
- **@default:** handles unmatched scenarios
- Control flow affects UI rendering only, not application logic

Empty List Handling (@empty)

The ‘@empty’ block was introduced in conjunction with ‘@for’ to handle scenarios where a list has no items.

Key learnings:

- ‘@empty’ displays UI content when a list is empty
- Eliminates the need for additional conditional checks
- Improves template readability and cleanliness

With this, modern Angular template control flow was considered complete.

Introduction to Dependency Injection

A detailed conceptual explanation of Dependency Injection (DI) was provided before any implementation. The concept was explained from first principles, focusing on why Angular applications rely on DI for scalability and maintainability.

Key concepts:

- A dependency is something a class needs to function
- Injection means providing that dependency instead of creating it manually
- Angular manages object creation and lifecycle
- Components should never create services using new

Dependency Injection was established as a core design pattern used by Angular to decouple components from shared logic.

Introduction to Angular Services

The session introduced services as a foundational Angular concept used to store shared data and reusable logic outside of components.

Key points:

- A service is a TypeScript class decorated with '@Injectable'
- Services do not contain UI or templates
- Services are used to centralize logic and application state
- Angular creates and injects services using its dependency injection system
- **providedIn: 'root'** creates a single shared instance across the application

Hands-On: Creating the First Service

Using a fresh Angular project, a simple service was created to demonstrate real-world usage of dependency injection.

Service Implementation

- A service was generated using Angular CLI
- Reactive state was stored inside the service using signals
- A method was implemented to update the service state

Service Injection

- The service was injected into the root component using the constructor
- No manual instantiation was used
- Angular handled service creation and lifecycle management

Template Usage

- Component templates accessed service state directly
- UI updated automatically when service data changed
- Logic remained inside the service rather than the component

This demonstrated a complete data flow from service → component → template.

Key Learnings

- Modern Angular template control flow is declarative and UI-focused
- @switch and @empty complete the control flow toolkit
- Dependency Injection allows Angular to manage shared dependencies
- Services are used to store shared logic and state
- Components should remain focused on UI and interaction
- Angular automatically injects and shares service instances
- Signals inside services enable reactive UI updates
- Clean separation of concerns improves scalability and maintainability

Conclusion

The fourth day of training strengthened architectural understanding of Angular by completing modern template control flow concepts and introducing dependency injection and services from the ground up. Participants gained clarity on how Angular manages shared state and logic through services and how components interact with services without tight coupling. This session laid a critical foundation for upcoming topics such as forms, API integration, and larger application structures, ensuring participants are prepared to build scalable and maintainable Angular applications.