6 Months Training TR-104 Full Stack Development

January, 2026

Day 2 Report

The second day of the training focused on strengthening core Angular fundamentals and moving toward real-world application structure. The session emphasized understanding Angular's event-handling mechanism, improving code cleanliness, and implementing component-based architecture through parent–child communication. Practical hands-on exercises were carried out to ensure clarity in how Angular coordinates between templates and component logic.

Revision of Angular Fundamentals

The session began with a revision of previously covered Angular concepts, including components, signals, interpolation, and event binding. Special focus was placed on understanding how Angular acts as a framework that coordinates between HTML (template) and TypeScript (component logic).

The separation of responsibilities was reinforced:

- HTML handles display and user interaction
- TypeScript handles logic and state
- Angular manages synchronization between the two

Improved Event Handling Techniques

The concept of cleaner event handling was introduced to reduce dependency on low-level browser event objects.

Template Reference Variables

Participants learned how to use template reference variables (#variable) to directly access DOM element values inside templates. This approach allows passing only the required data to component methods instead of the entire event object.

Key benefits discussed:

- Cleaner and more readable code
- Reduced TypeScript complexity
- Better separation between DOM handling and business logic

Understanding Angular as a Framework

A detailed explanation was provided on how Angular functions as a middleman framework between HTML and TypeScript. The complete flow of data and events was analyzed:

- TypeScript sends data to HTML for display
- HTML detects user actions and notifies TypeScript
- Angular tracks state changes and updates the UI automatically

This clarified why Angular applications feel reactive without manual DOM manipulation.

Component-Based Architecture

The session progressed to building applications using multiple components, which is essential for real-world Angular projects.

Participants learned:

- Why large applications are divided into smaller, reusable components

- How Angular applications are structured using parent and child components
- The importance of maintaining a clear data flow between components

Parent to Child Communication (@Input)

The concept of passing data from a parent component to a child component was introduced using the @Input decorator.

Key points:

- The parent component owns the data
- The child component receives data for display purposes
- Data flow is one-way (top to bottom)

A header component was created and successfully integrated into the main application using property binding.

Child to Parent Communication (@Output)

The session further covered communication from child to parent components using the @Output decorator and EventEmitter.

Key points:

- Child components cannot directly modify parent data
- Child components emit events to notify the parent
- Parent components decide how to respond to those events

A button click in the header component was used to trigger a state change in the parent component, demonstrating controlled and predictable communication.

Hands-On Tasks Implemented

- Revised event binding using (click) and (input)
- Implemented cleaner input handling using template reference variables
- Practiced updating signals using .update() and .set()
- Created a standalone Header component using Angular CLI
- Passed data from parent to child using @Input
- Emitted custom events from child to parent using @Output
- Updated parent state based on child-generated events
- Observed automatic UI updates without manual DOM manipulation

These tasks reinforced Angular's unidirectional data flow and component-driven architecture.

Key Learnings

- Angular coordinates between templates and component logic
- Events trigger actions, while data drives logic
- Template reference variables simplify event handling
- Components should be small, reusable, and focused
- Parent components own and control data
- Child components communicate through inputs and outputs
- Angular enforces predictable and maintainable architecture

Conclusion

The second day of training significantly advanced understanding of Angular's internal workflow and component-based design. Participants gained hands-on experience with cleaner event-handling patterns and implemented structured communication between parent and child

components. These concepts form the backbone of real-world Angular applications and prepare participants to work confidently on internship-level projects involving modular UI design and controlled data flow.