Templates and Data Binding

Angular 19

23/02/25

What We'll Cover:

1. Writing Templates (HTML in Components)
2. One-Way Data Binding (Interpolation, Property Binding)
3. Event Binding (Responding to User Actions)
4. Two-Way Data Binding with ngModel
5. Using Directives (*ngIf, *ngFor)

Writing Templates

Concept: Templates define the UI of your Angular components

Where: Written in the template property or a separate .html file

Example:

```
@Component({
  selector: 'app-hello',
  template: `<h1>Hello, Angular!</h1>`
})
export class HelloComponent {}
```

Key Point: HTML + Angular syntax = Dynamic UI

One-Way Data Binding - Interpolation

Concept: Display data from component to template

Syntax: {{ expression }}

Example:

```
@Component({
  template: `<p>Welcome, {{ name }}!</p>`
})
export class AppComponent {
  name = 'User';
}
```

Output: "Welcome, User!"

Use Case: Show dynamic text (e.g., user names, titles)

One-Way Data Binding - Property Binding

Concept: Bind component data to HTML properties

Syntax: [property]="expression"

Example:

```
@Component({
  template: `<img [src]="imageUrl">`
})
export class AppComponent {
  imageUrl = 'https://example.com/image.jpg';
}
```

Key Point: Safer than plain HTML attributes, reactive updates

Visual: Image displayed via binding

Event Binding

Concept: Respond to user actions (clicks, inputs, etc.)

Syntax: (event)="handler()"

Example:

```
@Component({
  template: `<button (click)="sayHello()">Click Me</button>`
})
export class AppComponent {
  sayHello() {
        alert('Hello, Angular!');
  }
}
```

Use Case: Buttons, form submissions, mouse events

Demo: Click button → Alert pops up

Two-Way Data Binding with ngModel

Concept: Sync data between component and template both ways

Requirement: Import FormsModule in app.component.ts (standalone)

Syntax: [(ngModel)]="property"

Example:

```
@Component({
  standalone: true,
  imports: [FormsModule],
  template: `<input [(ngModel)]="name"> <p>{{ name }}</p>`
})
export class AppComponent { name = 'Type here';}
```

Output: Typing updates name instantly

Visual: Live demo of input changing text

Using Directives - @if

Concept: Conditionally show/hide elements with modern control flow

Syntax: @if (condition) { content }

Example:

```
@Component({
  template: `
            <button (click)="toggle()">Toggle</button>
            @if (isVisible) {
            <p>I'm visible!</p>
            }
  `
})
export class AppComponent {
  isVisible = false;
  toggle() { this.isVisible = !this.isVisible; }}
```

Use Case: Cleaner, more readable conditionals

Note: *ngIf still works but @if is the future

Using Directives - @for

Concept: Loop over arrays with modern control flow

Syntax: @for (item of items; track item) { content }

Example:

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

})
export class AppComponent { tasks = ['Learn Angular', 'Build App', 'Have Fun'];}
```

Output: Bulleted list of tasks

Key Point: track replaces trackBy, improves performance

Putting It Together

Mini-Project: Build a simple to-do list with new syntax

Code:

```
@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 AppComponent {
  tasks: string[] = [];
  newTask = '';
  addTask() {
        if (this.newTask) {
        this.tasks.push(this.newTask);
        this.newTask = '';
        }
  }
}
```

Explanation of Changes

- **Old Syntax (*ngIf, *ngFor)**: These are structural directives from earlier Angular versions. They're still supported for backward compatibility but are considered legacy in Angular 19.
- **New Syntax (@if, @for)**: Introduced in Angular 17, this block-based control flow is more intuitive, aligns with modern JavaScript, and offers better performance. The @for loop requires a track expression (similar to trackBy) to optimize rendering.
- **Impact**: The functionality remains the same, but the code looks cleaner and is future-proof.

Key Takeaways:

Templates define your UI with HTML and Angular magic

One-way binding: Data → UI (interpolation, properties)

Event binding: UI → Data (user actions)

Two-way binding: Data ↔ UI (ngModel)

Modern directives (@if, @for) add logic to templates