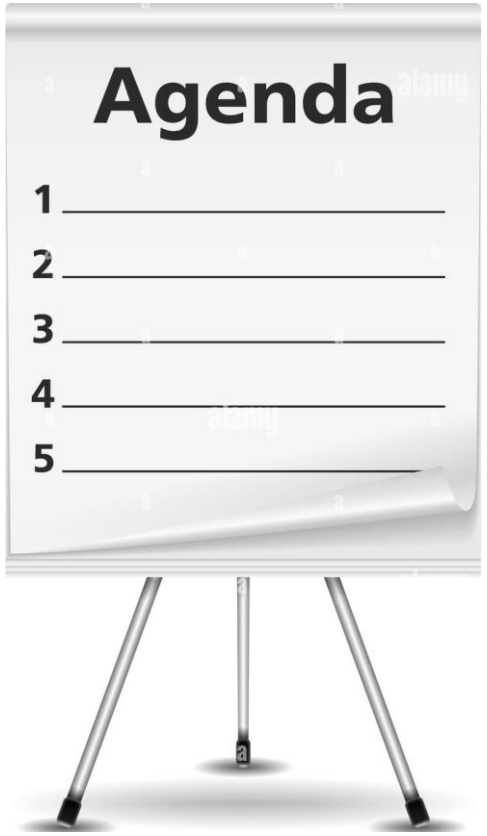
Welcome to the Apprenticeship Program – Session 27

Agenda

- Angular Directives: Built-in & Custom
- Pipes: Data Formatting and Transformation
- Angular Services
- Dependency Injection (DI)
- Sharing Data Across Components



Angular Directives

- **Definition:** Directives are markers on elements that change their appearance or behavior.
- Angular provides built-in directives, and you can create custom ones.
- Three types:
  - Component directives
  - Structural directives – alter layout (*ngIf, *ngFor)
  - Attribute directives – change appearance/behavior ([ngClass], [ngStyle]).

Structural Directives – Examples

- **These control DOM structure at runtime**
  - ***ngIf –** Conditionally show/hide elements

```
<p *ngIf="isVisible">Now you see me</p>
```

  - ***ngFor –** Loop through lists

```
<li *ngFor="let item of items">{{ item }}</li>
```

Attribute Directives – Examples

- **These enhance appearance without altering the structure.**
  - [ngClass] – Dynamically assign classes

```
<div [ngClass]="{active: isActive}"></div>
```

  - [ngStyle] – Bind inline styles

```
<p [ngStyle]="{'color': textColor}">Styled Text</p>
```

Creating a Custom Directive

- **Great for reusable behavior.**

```
@Directive({
  selector: '[appHighlight]'
})
export class HighlightDirective {
  constructor(el: ElementRef) {
    el.nativeElement.style.backgroundColor = 'yellow';
  }
}
```

- Usage

```
<p appHighlight>This text is highlighted</p>
```

DEMO

Angular Pipes

- Pipes transform data in the template.
  - Syntax:

```
{{ value | pipeName }}
```

Common Built-In Pipes

Pipe	Purpose
date	Formats dates
uppercase	Transforms text to UPPERCASE
lowercase	Transforms text to lowercase
currency	Formats numbers as currency
json	Outputs JSON as string

```
{{ birthday | date:'fullDate' }}
{{ price | currency:'USD' }}
```

Creating a Custom Pipe

- **Ideal for simple, reusable text/data formatting..**

```
@Pipe({ name: 'exclaim' })
export class ExclaimPipe implements PipeTransform {
  transform(value: string): string {
    return value + '!!!';
  }
}
```

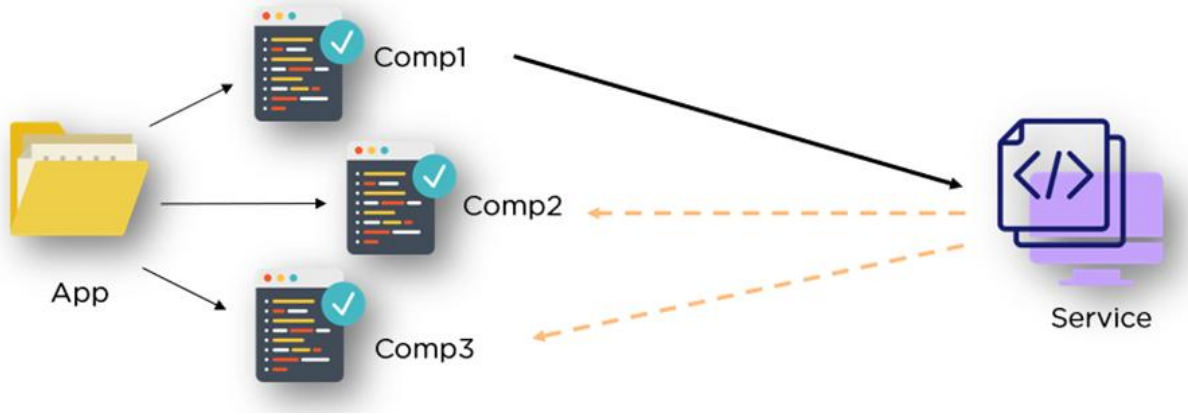
- Usage

```
<p>{{ 'Hello' | exclaim }}</p> <!-- Output: Hello!!! -->
```

DEMO

What is a Service?

- Service = A reusable class for logic or data
- Examples:
  - Fetch data from APIs
  - Store application state
  - Share data across components
- Keeps components clean and focused on the UI.



Creating a Simple Service

✅ @Injectable marks it for dependency injection.

```
@Injectable({ providedIn: 'root' })
export class LoggerService {
  log(msg: string) {
    console.log('LOG:', msg);
  }
}
```

Using the Service in a Component

✅ Angular injects the service automatically when needed.

```
constructor(private logger: LoggerService) {}

ngOnInit() {
  this.logger.log("Component initialized");
}
```

What is Dependency Injection (DI)?

- DI is a design pattern where dependencies are provided, not created manually.
- Angular handles this using the Injector.:

Benefits:

- Decouples components and services
- Easier to test
- Promotes modular architecture

Sharing Data Across Components

- Use a shared service with a shared data property.
- Subscribe to that data using Observables or Subject.

Example Service:

```
export class SharedService {
  message = new BehaviorSubject<string>('Hello');
}
```

All components using the service get real-time updates.

DEMO

Conclusion and Q&A

- **Key Takeaways:**
  - Use directives for layout & style changes.
  - Use pipes to format data in templates.
  - Services help organize business logic.
  - Angular’s dependency injection makes your code clean and testable.

Questions? 👀