Lecture 16: Dependency Injection

What is Dependency Injection?

Dependency Injection is a design pattern used to manage dependencies in an application. It allows components, services, and other classes to request their dependencies rather than creating them manually.

Purpose: Decouple components for better reusability and testability.
Simplify the process of managing dependencies.

Why Use Dependency Injection?

Angular heavily relies on DI (Dependency Injection) to:

- Provide instances of services to components and other parts of the application
- Share services across components
- Manage lifecycle of services efficiently

Angular DI Workflow

- **Declare the Dependency**

  Use the **@Injectable( )** decorator on a class to make it available for injection

```
@Injectable({
    providedIn: 'root',
})
export class DataService {
  getData() {
    return 'Hello World';
  }
}
```

Angular DI Workflow

- **Provide the Dependency**

  Add the service to the provider list, if not using **provided: ‘root’**

```
@NgModule({
    providers: [DataService],
})
export class AppModule {}
```

Angular DI Workflow

- **Inject the Dependency**

  Use constructor injection to receive the service in a component or another class

```
@Component({
  selector: 'app-root',
  template: `<h1>{{ message }}</h1>`,
})
export class AppComponent {
  message: string;

  constructor(private dataService: DataService) {
    this.message = this.dataService.getData();
  }
}
```

Use of InjectionToken

For non-class dependencies like configuration data we use

InjectionToken.

```
export const CONFIG = new InjectionToken<string>('config');

@NgModule({
  providers: [
    { provide: CONFIG, useValue: 'App Config' },
  ],
})
export class AppModule {}
```

Use of InjectionToken

Inject the token into the constructor

```
constructor(@Inject(CONFIG) private config: string) {}
```