

- 1) What is Dependency Injection (DI) in Angular, and how does it help in application development?

Q1) What is Dependency Injection (DI) in Angular, and how does it help in application development?

→ Dependency Injection allows classes with Angular decorators like decorators, components, directives, pipes and injectables if they need.

By using dependency injection we can inject services to other components.

It makes code reusable and more testable.

- 2) How do you create and register a service in Angular? Explain the role of @Injectable() and providedIn.

Q2) How do you create and register a service in Angular? Explain the role of `@Injectable()` and `provide`.

→ Creating a Service :-

Open CLI and command is `ng g s service`

It creates two files i) `servicename.ts`
ii) `servicename.spec.ts`

by defining `providedIn : 'root'` in `@Injectable()` ~~decorator~~ we can register a service.

`@Injectable` :

`@Injectable` is a ~~decorator~~ it says like it is ready for Dependency Injection wherever you required it.

`providedIn` :

By using `providedIn` we can provide service in Application root level.

3. What is the difference between providing a service in root, a module, or a component? How does it affect the service's scope and instance?

Q3) What is the difference between providing a service in root, a module, or a component? How does it affect the service's scope and instance?

→ Providing a service in root:

We can provide a service in root by using `ProvidedIn` and `ApplicationConfig`.
Scope: We can use this service in whole application.

Instance: Singleton.

→ Module:

If we provide a service in module we can use the service within the module and components which are declared or imported in module.

by using `providers[]` we can declare services.

Scope: It available only to components declared or imported in the module.

Instance: Singleton within that module.

→ Component:

We can provide service in component at `@component` decorator's `providers` field.

Ex:- `@Component({`

`providers: [service name]`

scope:- Available in only component and its child components.

Instance:

It creates a new instance every time component.

4. Explain the concept of hierarchical injectors in Angular with an example.

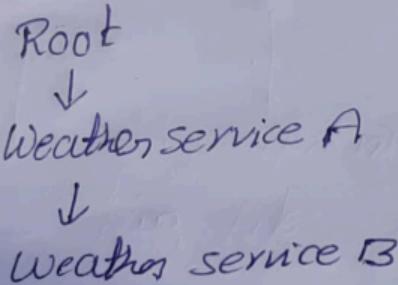
Q4) Explain the concept of hierarchical injectors in Angular with an example.

→ Angular creates tree like structure injectors hierarchical way.

→ Angular starts looking for a dependency from the component's injector and moves up the tree

like child → parent → root

Example:-



It starts checking if the service is available in Weather service B or not. If not it will go for its parent service. If not present it will search in Root level.

5. What is the purpose of the @Inject() decorator in Angular DI? When would you use it instead of constructor injection alone?

Q5) What is the purpose of the `@Inject()` decorator in Angular DI? When would you use it instead of constructor injection alone?

→ `@Inject()` Decorator:

Angular uses constructor parameter types to identify which dependency to inject. But in some special cases, the type information is either not available, or not enough - that's where `@Inject()`.

Use constructor injections for regular services and classes.

We can use `@Inject()` when injecting a custom injection tokens, Interfaces and Abstract classes.

6. What is the difference between `@Optional()` and `@Self()` in Angular dependency injection? Provide use cases.

Q6) What is the difference between @Optional and @self in Angular dependency injection?

→ @Optional is allows injection of dependency if available. If it is not available it returns null, instead of throwing an error.

@self is restricts angular to look only in the current injector. It will throw an error if not found at the current level.

@optional usecases:-

- Optional configuration services
- Avoid crashing the app when a dependency is missing.

@self:-

- Custom form controls
- Avoiding using dependency from a parent when you expect a local instance.

7. What are the differences between using useClass, useValue, useFactory, and useExisting in service providers? Provide examples.

Q7) What are the ~~instance~~ differences between `useClass`, `useValue`, `useFactory` and `useExisting` in service providers?

→ `useClass` :-

Default class based provider

Provides a class by `new` keyword.

Provides a service by creating a new instance of a class.

`useValue`:

Provides a fixed value (object, string, number).
use for configurations, constants or static data.

`useFactory` :-

Uses a factory function to returns the value or object

Used when the value needs to be computed or configured dynamically.
You can inject dependencies into the factory function also.

Use Existing:-

Maps one token to another already existing providers.

Makes one token act as alias for another.

8. How can you override a service provided in the root injector at a component level? What are the implications?

Q8) How can you override a service provided in the root injector at a component level? What are the implications.

→ Sometimes we want a component (or sub) use a different instance or custom implementation of a service separate from global one.

When you want a separate service instance per component.

For testing, mocking, or shadowing the service.

Scope is only the component and its children use the overridden instance.