**Angular Services**

* Services is a broad term refer to a function or group of functions designed to do something specific.
* Reusable piece of functionality shared across components.
* Should be responsible for single, discrete piece of functionality.
* Able to be delivered when and where it is needed. This is accomplished in angular by adding our services to dependency injection system of angular.
* Services gives the benefit of separation of concerns and we can say single responsibility principle as well.

# **How to Define an Angular Service?**

To define a class as a service in Angular, use the @[Injectable](https://angular.io/api/core/Injectable)() decorator to provide the metadata that allows Angular to inject it into a component as a *dependency*.

We need to provide a service before it can be available. This can be done in three ways:

* Via the service’s metadata passed to the @Injectable() decorator (The service will be available everywhere),
* Via the providers' array, in a specific module (The service is available only to the components and services of the module),
* Via the providers' array in a specific component (The service is available only to the component).

**Let’s now create a service by example.**

**$ cd your-angular-project**

**$ ng generate service my-example**

**import { HttpClient } from '@angular/common/http';**

**@Injectable({**  
 **providedIn: 'root'**  
**})**  
**export class MyExampleService {**

**constructor(private httpClient: HttpClient) { }**  
**}**

**There are some ways to inject a service in angular:**

1 — A service provided at a root level (angular creates a single shared instance of service singleton), so it's available in any class that needs it, we can think of this one as a global shared class.

2 — A service provided at a specific module which makes it available in the classes that this module has, it’s a singleton but scoped to the module that it was provided in (so everything this module declares or provides will have access to the same instance of this service).

3 — A service provided by a single component, which makes a single instance for the component that provided it (even if the component is dynamically created each component with has its own instance of the service).

4 — Using a factory provider, in a case when the service needs information that is only available in the run time.

**Dependency Injection**

Injecting the instance of a service in a component using constructor injection. This process is often called as dependency injection.

## **Why is dependency injection important.**

1. Loosely coupled code as the service is not dependent on the component.
2. More flexible code and easier to test.

## **Providers**

Providers tell the injector how to create a service. Provider basically has two parts, provider token and provider recipes.

* Provider token : is basically the token that a component will use to request an instance of a service from the injector.
* Provider recipe : indicates the method or process involved in creating the instance of that service.

## **Multiple ways of providing a service**

* ***Short-hand:*** This is simply listing the provider token in the providers array. By this method the injector will simply create an instance of that service with same name as the token and pass it to the component.

providers: [ **HttpClient**]

* ***UseClass:*** This method helps you have a different token and instance class name. You pass an object to providers array and give a token name to provide property and service class name in useclass property.

providers: [{ provide: httpClient, useClass: HttpClient}]

* ***UseExisting***: It similar to useClass but here you need to give the name of the service class name in useExisting if that service class name has already been provided in the providers so that injectors use the same instance for injection in any component or service in that module.

providers: [ ArticleService, { provide: DefaultService, UseExisting: ArticleService }]

* ***UseValue***: This method helps you provide an object literal directly against a token.

providers: [{ provide: DefaultService, useValue: {  
 log: (message) => console.log(message);  
 }}]

* ***UseFactory:*** This method helps you inject a service class instance which will be returned using a factory function. This method gives more flexibility to creating the instance. You define the factory function in useFactory property and the you define the dependencies of this factory function in the deps array property.

providers: [{ provide: DefaultService, useFactory: articleServiceFactory, deps:[LogService]} ]

## **Hierarchical Injectors**

Angular injectors are organised hierarchically. If a component requests a service. The task will be delegated to the components injector itself. If service is not provided in that component, the injector will delegate the task to parent component. If the service is not found in the parent injector also the task is delegated to the root injector.