**Services**

services in Angular, code or functionalities that are then accessible and reusable in many other components in Angular project. Services help with the abstraction of logic and data that is hosted independently but can be shared across other components.

The service class has a single, well-defined function, helping make application structure very modular. It is different from other classes in Angular because of the injection process. Dependency injection is the concept that makes it possible for you to receive dependencies from one class to another.

One of the biggest use cases for Angular services is managing or manipulating or even storing data.

# Angular Promises Vs Observables

Both observables and promises help us work with asynchronous functionality in JavaScript. Promises deal with one asynchronous event at a time, while observables handle a sequence of asynchronous events over a period of time.

|  |  |
| --- | --- |
| **Observables** | **Promises** |
| Emit multiple values over a period of time. | Emit a single value at a time. |
| Are lazy: they’re not executed until we subscribe to them using the subscribe() method. | Are not lazy: execute immediately after creation. |
| Have subscriptions that are cancellable using the unsubscribe() method, which stops the listener from receiving further values. | Are not cancellable. |
| Provide the map for for Each, filter, reduce, retry, and retryWhen operators. | Don’t provide any operations. |
| Deliver errors to the subscribers. | Push errors to the child promises. |

|  |  |  |
| --- | --- | --- |
| **Operations** | **Observables** | **Promises** |
| Creation | const obs = new Observable((observer) => {  observer.next(10);  }) ; | const promise = new Promise(() => {  resolve(10);  }); |
| Transform | Obs.pipe(map(value) => value \* 2); | promise.then((value) => value \* 2); |
| Subscribe | const sub = obs.subscribe((value) => {  console.log(value)  }); | promise.then((value) => {  console.log(value)  }); |
| Unsubscribe | sub.unsubscribe(); | Can’t unsubscribe |

**Pure and Impure Pipes:**In Angular, Pipes are of two types.

* Pure Pipes
* Impure Pipes

**Pure Pipes:**

* Pure pipes in angular are the pipes that execute when it detects a *pure change* in the input value.
* A pure change is when the change detection cycle detects a change to either a primitive input value (such as String, Number, Boolean, or Symbol) or object reference (such as Date, Array, Function, or Object).

**Impure Pipes:**

* Impure pipes in angular are the pipes that execute when it detects an *impure change* in the input value.
* An impure change is when the [change detection](https://angular.io/guide/glossary#change-detection) cycle detects a change to composite objects, such as adding an element to the existing array. Let’s take the same example that we took earlier.

|  |
| --- |
| <div> {{ user | myPipe }} </div> Angular Decorators In AngularJS, decorators are functions that allow a **service, directive,** or **filter** to be modified before it is used.  **There are four main types of angular decorators:**   * Class decorators, such as **@Component** and **@NgModule** * Property decorators for properties inside classes, such as **@Input** and **@Output** * Method decorators for methods inside classes, such as **@HostListener** * Parameter decorators for parameters inside class constructors, such as **@Inject**  What is the Difference between Constructor and ngOnInit in AngularJS ? **Constructor:** Constructor is the default method for a class that is created when a class is installed and ensures the proper execution of the roles in the class and its subsections. Angular are preferably the Dependency Injector (DI), analyzes the builder’s components and when creating a new feature by calling the new MyClass() tries to find suppliers that match the builder’s parameter types, resolve them and pass them to similar components.  new MyClass(someArg);  **ngOnInit:** OnInit is a life cycle widget called Angular to show that Angular is made to create a component. We have to import OnInit like this to use it (actually using OnInit is not mandatory but it is considered good).  import {Component, OnInit} from '@ angular / core';  **Difference between ngOnInit and Constructor:**   * We mostly use ngOnInit in every startup/announcement and avoid things to work in builders. The constructor should only be used to start class members but should not do the actual “work”. * So you should use the constructor() to set Dependency Injection and not much. ngOnInit() is a better “starting point” – this is where / when component combinations are solved. * We use constructor() for all the initialization/declaration. * It’s better to avoid writing actual work in the constructor. * The constructor() should only be used to initialize class members but shouldn’t do actual “work”. * So we should use constructor() to set up Dependency Injection, Initialization of class fields, etc. * ngOnInit() is a better place to write “actual work code” that we need to execute as soon as the class is instantiated. * Like loading data from Database — to show the user in your HTML template view. Such code should be written in ngOnInit().  High-level Differences between Template-driven and Reactive Forms Below are some of the high-level differences between the two types:   1. Template-driven forms make use of the "FormsModule", while reactive forms are based on "ReactiveFormsModule". 2. Template-driven forms are asynchronous in nature, whereas Reactive forms are mostly synchronous. 3. In a template-driven approach, most of the logic is driven from the template, whereas in reactive-driven approach, the logic resides mainly in the component or typescript code. Let us get started by generating a component and then we'll update our form code. |