Services in Angular:

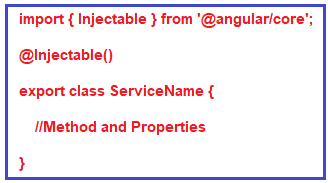
Why do we need a service in Angular?

Whenever you need to reuse the same data and logic across multiple components of your application, then you need to go for angular service. That means whenever you see the same logic or data-access code duplicated across multiple components, then you need to think about refactoring the same logic or data access code into a service.

There is a principle in software development i.e. DRY (Don’t Repeat Yourself) and using services in angular safely follow this principle. The logic or data is implemented in a services and the service can be used across multiple components of your angular application. So, services is a mechanism used to share the functionality between the components.

Step1: Creating Angular Service

The angular service is composed of three things. You need to create an export class and you need to decorate that class with @Injectable decorator and you need to import the Injectable decorator from the angular core library. The syntax to create angular service is given below.



Below is to explain the process of DI

Service Class

**import** **{** Injectable **}** from '@angular/core';

@Injectable**()**

**export** **class** StudentService **{**

getTitle**()**

**{**

**return** "Dependency Injection in Angular";

**}**

getStudents**()**: **any[]** **{**

**return** **[**

**{**

ID: 'std101', FirstName: 'Preety', LastName: 'Tiwary',

Branch: 'CSE', DOB: '29/02/1988', Gender: 'Female'

**}**,

**{**

ID: 'std102', FirstName: 'Anurag', LastName: 'Mohanty',

Branch: 'ETC', DOB: '23/05/1989', Gender: 'Male'

**}**,

**{**

ID: 'std103', FirstName: 'Priyanka', LastName: 'Dewangan',

Branch: 'CSE', DOB: '24/07/1992', Gender: 'Female'

**}**,

**]**;

**}**

**}**

Component.html file:

<h2>{{pageTitle}}</h2>

<table>

<thead>

<tr>

<th>ID</th>

<th>First Name</th>

<th>Last Name</th>

<th>Branch</th>

<th>DOB</th>

<th>Gender</th>

</tr>

</thead>

<tbody>

<tr \*ngFor='let student of students'>

<td>{{student.ID}}</td>

<td>{{student.FirstName}}</td>

<td>{{student.LastName}}</td>

<td>{{student.Branch}}</td>

<td>{{student.DOB}}</td>

<td>{{student.Gender}}</td>

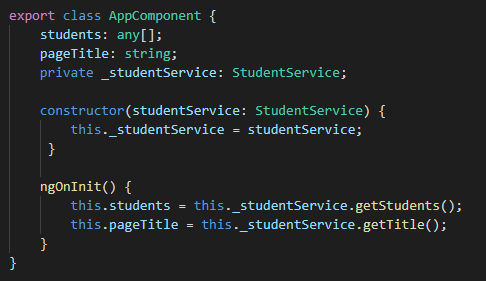
</tr>

</tbody>

</table>

Component.ts file:

Another approach of creating DI based property(apart from the method taught in class)



Here, in the AppComponent class, we need an instance of StudentService to call the the getStudents() and getTitle() method of Student Service to get the list of students and the title of the page which is required by App component.

But if you look at the above AppComponent code, we are not creating an instance of StudentService. Here, we declared a private field \_studentService of type StudentService. The constructor also has a parameter studentService of type StudentService. Then the constructor initializing the private class field \_studentService with it’s parameter studentService .

Then we are using this private field \_studentService to call the Student Service methods getStudents() and getTitle().

How are we getting an instance of the StudentService class.

How are we getting an instance of the StudentService class?

From the code of AppComponent class we can see that the constructor is provided with an instance of StudentService class, and then the constructor is assigning that instance to the private field \_studentService.

Who is creating and providing the instance to the constructor?

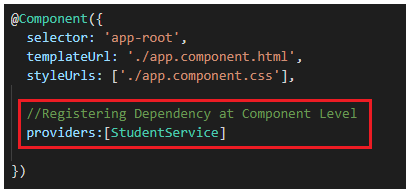
The answer is Angular Injector. When an instance of AppComponent class is created, the angular injector creates an instance of the StudentService class and provides it to the AppComponent constructor. The constructor then assigns that instance to the private field \_studentService. We then use this private field \_studentService to call the StudentService methods getStudents() and getTitle().

How does the angular injector know about StudentService?

For the Angular injector to be able to create and provide an instance of StudentService , first we need to register the StudentService with the Angular Injector. We register a service with the angular injector by using the providers property of @Component decorator or @NgModule decorator. We already know we decorate an angular component with @Component decorator and an angular module with @NgModule decorator.

At Component Level:

If you are registering a service using the providers property of the @Component decorator then you are registering the service with an angular injector at the component level. The service is then available to that component and all of it’s children. The syntax to register dependency at component level is given below. Here, you need to register the service within the Providers array of Component decorator.



At Module Level:

On the other hand if you register the service using the providers property of the @NgModule decorator then you are registering the service with an angular injector at the module level which is the root injector. The service registered with the root injector is then available to all the component across the entire application. The syntax to register at app level is given below. As you can see, you need to declare the service within the Providers array of your root module i.e. AppModule.



So, in our example, AppComponent has a dependency on StudentService. The AppComponent receives the dependency instance (i.e StudentService instance) from the the external source (i.e the angular injector) rather than creating the instance itself.

Advantages of Dependency Injection in Angular?

Create applications that are easy to write and maintain over time as the application evolves

Easy to share data and functionality as the angular injector provides a Singleton i.e a single instance of the service

Easy to write and maintain unit tests as the dependencies can be mocked

Difference between constructor and ngOnInit (life cycle hooks -other methods are ngOnChange(), ngDoCheck())

Whenever you create an instance of a class, the class constructor is automatically called. Like other programming languages, the class constructor in angular is also used to initialize the members of the class and it’s sub classes.

The ngOnInit is a life cycle hook method provided by Angular which is called after the constructor and is generally used to perform tasks related to Angular bindings. For example, ngOnInit is the right place to call a service method to fetch data from a remote server. We can also do the same using a class constructor, but the general rule of thumb is, tasks that are time consuming should use ngOnInit instead of the constructor. As fetching data from a remote server is time consuming, the better place for calling the service method is ngOnInit.

In our example, the dependency injection is done using the class constructor and the actual service method call is issued from ngOnInit life cycle hook.