Introduction to Angular Services

3 Comments / 5 minutes of reading / March 9, 2023

← Select Options | Angular Tutorial | Angular Dependency Injection →

In this **Angular Services** tutorial, we will show you how to build a simple component that fetches a list of products from an Angular Service and displays it in our template. Service is a class that has the purpose of Providing a Service to a Component, Directive, or to another Service. The Service may be fetching data from the back end, running a business logic, etc

Table of Contents

What is an Angular Service

Service is a piece of reusable code with a focused purpose. A code that you will use in many components across your application

Our components need to access the data. You can write data access code in each component, but that is very inefficient and breaks the rule of single responsibility. The Component must focus on presenting data to the user. The task of getting data from the back-end server must be delegated to some other class. We call such a class a Service class. Because it provides the service of providing data to every component that needs it.

What Angular Services are used for

1. Features that are independent of components such a logging services
2. Share logic or data across components
3. Encapsulate external interactions like data access

Advantageous of Angular Service

1. Services are easier to test.
2. They are easier to Debug.
3. We can reuse the service at many places.

How to create a Service in Angular

An Angular service is simply a Javascript function. All we need to do is to create a class and add methods & properties. We can then create an instance of this class in our component and call its methods.

One of the best uses of services is to get the data from the data source. Let us create a simple service, which gets the product data and passes it to our component.

You can download the source code from stackBlitz

Product Model

Create a new file under the folder src/app and call it product.ts

product.ts

```
1
2  export class Product {
3
4     constructor(productID:number,    name: string ,   price:number) {
5        this.productID=productID;
6        this.name=name;
7        this.price=price;
8     }
9
10    productID:number ;
11    name: string ;
12    price:number;
13
14 }
15
16
```

The Product class above is our domain model.

Product Angular Service

Next, let us build an Angular Service, which returns the list of products .

Create a new file under the folder src/app and call it product.service.ts

product.service.ts

```
1
2  import {Product} from './product'
3
4  export class ProductService{
5
```

```
    public getProducts() {

        let products:Product[];

        products=[
            new Product(1,'Memory Card',500),
            new Product(1,'Pen Drive',750),
            new Product(1,'Power Bank',100)
        ]

        return products;
    }
}

```

First, we import the `Product` model from the `product.ts`

Next, create `ProductService` class and export it. We need to export so that the Components & Other service class import it and use it

The `getProducts` method returns the collection of the `products`. In this example, we have hardcoded the `products`. In real life, you would send an HTTP GET request to your back end API to get the data

Now our service is ready.

Note that the above class is a simple JavaScript function. There is nothing Angular about it.

Invoking the ProductService

The Next step is to invoke the `ProductService` from the component. Open the `app.componet.ts` and add the following code.

app.component.ts

```
1
2  import { Component } from '@angular/core';
3
4  import { ProductService } from './product.service';
5  import { Product } from './product';
6
7  @Component({
8    selector: 'app-root',
9    templateUrl: './app.component.html',
10 })
11
12 export class AppComponent
13 {
14
15    products:Product[];
16    productService;
17
18    constructor(){
19      this.productService=new ProductService();
20    }
21
22    getProducts() {
23
24      this.products=this.productService.getProducts();
25    }
26
27 }
28
```

We start with importing `Product` & `ProductService`

We create an instance of `ProductSerivce` in the constructor of the `AppComponet`. In real-life Angular Apps, we use the Dependency Injection in Angular to inject the `ProductSerivce` in the constructor. We will learn that in the next tutorial.

The `getProducts` method calls the `getProducts` method of the `ProductService`. It returns a list of Products, which we store in the local variable `products`

Template

The next step is to display the Products to user

Open the app.component.html file and add the following code

app.componenet.html

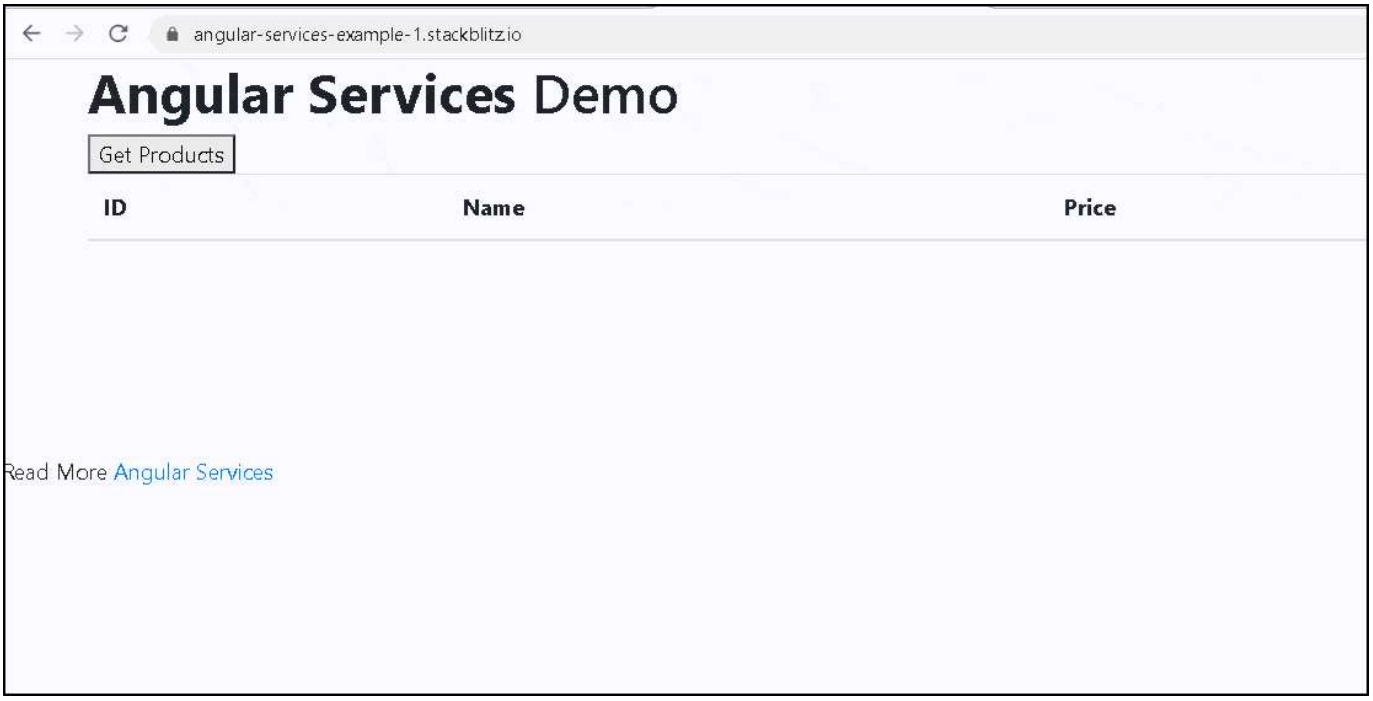
```
<div class="container">
   <h1 class="heading"><strong>Services </strong>Demo</h1>

   <button type="button" (click)="getProducts()">Get Products</button>

    <div class='table-responsive'>
        <table class='table'>
           <thead>
              <tr>
                 <th>ID</th>
                 <th>Name</th>
                 <th>Price</th>
              </tr>
           </thead>
           <tbody>
              <tr *ngFor="let product of products;">
                 <td>{{product.productID}}</td>
                 <td>{{product.name}}</td>
                 <td>{{product.price}}</td>
              </tr>
           </tbody>
        </table>
     </div>

</div>

```

We are using the bootstrap 4 to style our template here. You will find the link in the index.html .

The button Get Products , calls the getProducts method of the component class via event binding

We loop through the products via ngFor directive and display it in a table.

Now, you can run the code and click on the Get Product button. You will see the List of Products



Injecting Services into Component

In the example, we instantiated the `productService` in the Component directly as shown below

```
1
2 this.productService=new ProductService();
3
```

Directly instantiating the service, as shown above, has many disadvantageous

1. The `ProductService` is tightly coupled to the Component. If we change the `ProductService` class definition, then we need to update every code where service is used

2. If we want to change `ProductService` with `BetterProductService`, then we need to search wherever the `ProductService` is used and manually change it
3. Makes Testing difficult. We may need to provide `mockProductService` for testing and use the `ProductService` for Production.

We can solve this by using the Angular Dependency injection, which is our next tutorial.

References

Architecture Services

Read More

1. Angular Tutorial
2. Services
3. Dependency injection
4. Injector, @Injectable & @Inject
5. Providers
6. Injection Token
7. Hierarchical Dependency Injection
8. Angular Singleton Service
9. ProvidedIn root, any & platform
10. @Self, @SkipSelf & @Optional Decorators
11. @Host Decorator in Angular
12. ViewProviders
13. Introduction to Angular Modules
14. Lazy Loading in Angular
15. Preloading Strategy