STEP 14: Service In Angular

1. Create a service folder and execute the below command from within the service folder
   ng g s demo

2. Add a property message and a method to update this message
   ```
   message:string ;
   constructor() {
           this.message = 'from service';
           console.log('demo service')
       }
   setMessage(msg:string){
           this.message = msg
    }
   ```

3. Create a component within the service folder
   ng g c service --flat

4. Inject this service via constructor in service component as well as app component

   ```
   constructor(public service:DemoService){}
   ```

5. Add <app-service></app-service> in app html component

6. Update the html of service component as follows:

   ```
   <h1>Service Component</h1>
   <p>Service Message : {{service.message}}</p>
   <p><input type="text" [(ngModel)]="service.message"/></p>
   ```

7. Update app component html as below and **DO INJECT SERVICE IN CONSTRUCTOR of app component ts file**:

   ```
    <h1>App Component</h1>
   <p>{{service.message}}</p>
    <app-service></app-service>
   ```

8. Modifying the message in service component will modify the message in app component as well since angular injects a single instance of service throughout angular application

9. Now in service component, inject the demo service in providers array within the @Component decorator as follows:

   ```
   @Component({
     selector: 'app-serv',
     templateUrl: './serv.component.html',
     styleUrls: ['./serv.component.css'],
     providers:[DemoService]
   })
   ```

10. Now when the message value is changed by service component will not reflect in app component as now there are 2 different instances of service created.

Behaviour Subject an Observables:

- Even though services share data and modify the component view with the updated changes, it has to go through the DOM hierarchy to look for change detection and update the view. Also as soon as data changes in service it does update the view but what if we want to perform some action based on changed value?

- BehaviourSubject are helpful in the case, when component 2 wants to take some action based on the data change from component 1 or vice versa

- If you are not using any publish/subscribe patterns like Subject or BehaviorSubject, then it is difficult to observe the data change and perform an action.

- To perform logic every time the value changes. You could also do this in set function on your variable, but using RxJS gives you easy access powerful operators such as debounce and several mapping operators. Need to make an API call every time the value changes, but you want to wait until the user is done typing before hitting your API? RxJS is a fantastic solution to that.

- You can combine it with other observables. Maybe you have another stream of events. Making that variable an Observable, you can easily plug changes to the variable into that other stream.

- Better Angular performance. By using a BehaviorSubject (which is a type of observable), you can put a reference to it inside of the Angular template and use the async pipe to automatically subscribe and automatically mark it for change detection when the value changes. So it would look like this: service.isLoggedIn | async. If you do your whole component this way, you could eventually switch the ChangeDetectionStrategy to OnPush, which is far more performant. Or even better, you could switch to the ngrxPush pipe and move your application entirely out of Angular zones, improving performance even more

Example:

1. Add below code in demo service:

```
// declare and initialize the quote property which will be a BehaviorSubject
 qoute = new BehaviorSubject("Hello world");

 // expose the BehaviorSubject as an Observable
 currentQuote = this.qoute.asObservable();

 // function to update the value of the BehaviorSubject
 updateQuote(newQuote: string){
  this.qoute.next(newQuote);
 }
```

2. Update the service component and html as follows:

```
currentQuote: string = '';

 ngOnInit(): void {
  // Subscribe the currentQuote property of quote service to get real time value
  this.service.currentQuote.subscribe(
   // update the component's property
   quote => this.currentQuote = quote
  );
 }
}
```

```
<h2>{{currentQuote}}</h2>
```

3. Create another component within service folder as follows:

```
ng g c quote –flat
```

4. Update the quote component and html as follows:

```
constructor(private quoteService: QuoteService){}

 quote = '';

 // function to update the quote in the service
 submitHandler(){
  this.quoteService.updateQuote(this.quote);
  this.quote="";
 }
}
```

```
<div>
 <input type="text" [(ngModel)]="quote" placeholder="Write new quote" />
 <button (click)="submitHandler()">Submit</button>
</div>
```

Add a new quote , click submit and the quote will be updated in the service component html. This also provides a way to write any business logic or use any of observable operators within the service subscribe method as follows which is not possible with normal services:

```
  ngOnInit(): void {
    // Subscribe the currentQuote property of quote service to get real time value
    this.service.currentQuote
        .pipe(debounceTime(1000))
        .subscribe(
      // update the component's property
        if(quote.endsWith("!")
            quote => this.currentQuote = quote
    );
  }
}
```

https://stackoverflow.com/questions/57681288/angular-change-detection-for-service-variables-and-data