Monday: Register a Service

Register a Service

Now that we have created a Goal Service, Angular needs to know that this service is available so it can inject it into components or other services that need it. To do this, we need to register a **provider**. A **provider** is something that can create or deliver a service, in our case, it instantiates the **GoalService** class to provide the service.

There are 3 ways in which we can register providers.

- In the metadata of the service (in the @Injectable() decorator)
- In the @NgModule() decorator (inside the app.module file)
- In the @component() metadata (in the component where you want to inject the service)

Note that if you choose one of the methods above then you don't have to do the other two methods. Let's look at each method.

In the @Injectable() Decorator

The Angular CLI, by default, registers a provider with the **root** injector after running the command $ ng generate service <service-name>. This happens when the CLI automatically supplies the provider metadata in the **@Injectable** decorator.

src/app/goal-service/goal.service.ts

```
....
@Injectable({
  providedIn: 'root'
})
....
```

The **providedIn** metadata specifies that our service is available in the application at root level. When you provide a service at root level, Angular creates a single shared instance of the service and injects it into any class that asks for it. Registering a provider in the **@Injectable** metadata is best practice since it allows Angular to optimize the app by removing the service if it turns out not to be used at all.

In the @NgModule() decorator

When you register a provider with a specific NgModule, such as the one inside the app.module.ts, the same instance of a service is available to all components in that NgModule. To register at this level, use the `providers` property of the `@NgModule()` decorator. For example

src/app/app.module.ts

```
........
import { GoalService } from './goal-service/goal.service';
@NgModule({
  declarations: [
    AppComponent,
    GoalComponent,
    GoalDetailComponent,
    StrikethroughDirective,
    DateCountPipe,
    GoalFormComponent,
    NavbarComponent
  ],
.........
  providers: [GoalService],
  bootstrap: [AppComponent]
})
.......
```

In the @component() metadata

When you register a provider at the component level, you get a new instance of the service with each new instance of that component. At the component level, register a service provider in the `providers` property of the `@Component()` metadata. For example

src/app/goal/goal.component.ts

```
import { Component, OnInit } from '@angular/core';
import { GoalService } from '../goal-service/goal.service';

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

Note: You should always provide your service in the root injector unless there is a case where you want the service to be available only if the consumer imports a particular `@NgModule`.

Injecting the GoalService into the Goal Component

Now, we can finally inject the GoalService into our goal component

src/app/goal/goal.component.ts

```
import { Component, OnInit } from '@angular/core';
import { Goal } from '../goal';
import { GoalService } from '../goal-service/goal.service';
....
export class GoalComponent implements OnInit {

  goals:Goal[];

  constructor(goalService:GoalService) {
    this.goals = goalService.getGoals()
  }
....
}
```

At the top, we import our **GoalService** and also get rid of the **Goals** array since it is the service that will inject the goals from now on. In the component class, we create a property **goals** and assign it a type. We then tweak the Goal component class to consume the GoalService with the constructor. Inside the constructor function, we use the **getGoals()** method of the goal service to supply the goals.

If your local server is still running, you'll notice that our application still displays our goals which means that the service works as expected.

alert Service

At the moment, when the user deletes a goal, a prompt asks the user to confirm whether or not they want to delete the goal. Let's create another service that alerts the user once they delete a goal.

On our terminals, let's create a new service with this command:

```
$ ng generate service alert-service/alert
```

Let's write the code to describe what the service will do:

src/app/alert-service/alert.service,ts

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

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

  alertMe(message:string){
    alert(message)
  }

  constructor() { }
}
```

Our service is available at root level so we can inject it anywhere we like. Inside the service class, we have created the **alertMe()** method which accepts a message of string type. The method should use the alert function in javascript to display the message it receives.

Let's inject this service in our goal component.

src/app/goal/goal.component.ts

```
...
import { AlertService } from '../alert-service/alert.service';
...
export class GoalComponent implements OnInit {

  goals:Goal[];
  alertService:AlertService;

....
  deleteGoal(isComplete, index){
    if (isComplete) {
      let toDelete = confirm(`Are you sure you want to delete ${this.goals[index].name}?`)

      if (toDelete){
        this.goals.splice(index,1)
        this.alertService.alertMe("The goal has been deleted")
      }
    }
  }

....
  constructor(goalService:GoalService, alertService:AlertService) {
    this.goals = goalService.getGoals()
    this.alertService = alertService;
  }
...
}
```

At the top, we have imported the **AlertService**. Inside the component class, we have created a property **alertService** and assigned it our AlertService type. In the **deleteGoal()** function, we have added code that uses the alertMe() method from the alert service to display the message inside after the user has confirmed to delete a goal.

To make the service available in the component, we have added it to the constructor function and instantiated it inside the constructor function. If our server is still running, we can delete a goal to see the alert service at work.