Angular
Lab 7
Dependency Injection (DI) and Services

1 Lab setup

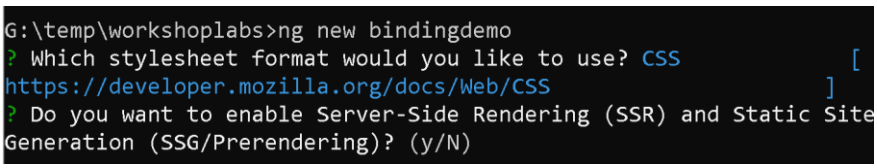
The setup here is identical to that in Lab 1

2 Generating a new Angular project

In either a separate command prompt (or shell terminal) or in an embedded terminal in Visual Studio Code, create a new Angular project with:

```
ng new servicesdidemo
```

Press enter to accept the default values for all the question prompts that follow regarding stylesheet format (CSS) and enabling Server side rendering (No)



Navigate into the root project folder from the terminal with:

```
cd servicesdidemo
```

Build the app and serve it via Angular's live development server by typing:

```
ng serve
```

The final output line from this should indicate that the compilation process was successful and identify the port that the live development server is currently serving up the Angular app dynamically from (by default this will be port 4200)

Open a browser tab at:
http://localhost:4200/

You should be able to see the default landing page for all new autogenerated Angular projects.

To stop the development server, type Ctrl+C in the terminal window that it is running in. You can restart again anytime by typing `ng serve` in the project root folder.

3 Motivation for services

The source code for this lab can be found in the `changes` subfolder of `ServicesDI-Demo`

Modify / add the following files in `src/app` with the latest update from `changes`:

`app.component-v1.html`

`app.component-v1.ts`

This simple scenario illustrates a common computation performed on many e-commerce sites. The implementation above performs the calculation of the sales tax on the sales total in the component class. While this functionally fine, it violates a key principle of software design: the Single Responsibility Principle (SRP) of software engineering.

https://www.freecodecamp.org/news/solid-principles-single-responsibility-principle-explained/

Components should only implement UI-specific logic (i.e. related to what users see in a view and how they interact with it). Any other functionality and logic should be delegated to a separate class, which in Angular is called a service.

Services are classes with basic methods that are intended to be reused at various points in the application. Their functionality is very focused and specific. Examples of functionality that might be implemented by a service class are:

a) Reusable core business logic (for e.g. calculating sales tax on a purchase)
b) Logging output to track key events in the lifecycle of a complex application
c) Retrieving / storing data from / to a backend server application via HTTP calls

Services are known generally as dependencies, which is simply anything a class needs in order to run correctly. Angular has a comprehensive dependency injection (DI) framework which ensures that service classes are created correctly within the classes that require their use (this process is called injection).

3.1 Generating and using a service

In a command prompt in the root project folder, you can use the Angular CLI to generate a new service with:

```
ng generate service TaxCalculator
```

A short form for the above uses only the first letter of the commands, i.e.

```
ng g s service-name
```

This creates a new `tax-calculator.service.ts` file in `src/app`.

Modify / add the following files in `src/app` with the latest update from `changes`:

`tax-calculator.service-v2.ts`

`app.component-v2.html`

`app.component-v2.ts`

Some key points:

a) The service class is marked by the @Injectable decorator which has the default value of root. This means the service class is available for injection into any of the components that require it in in the component hierarchy.
b) The service class is introduced into the component that requires it through a process called constructor injection, which has a clear explicit format that distinguishes it from how constructors are normally used.
c) Once injected , the service class object / instance can be used in a usual way, for e.g. by invoking methods in that instance and passing it suitable arguments.
d) The component no longer implements any business domain logic (e.g. computation of total prices taking into account sales tax); all of this is segregated into the service class. This allows its functionality to be reused in other components without duplicating code.

Modify / add the following files in `src/app` with the latest update from `changes`:

`app.component-v2-2.ts`

Angular 17 provides an alternative syntax to constructor injection to introduce a service dependency into a component. This syntax makes it more clearly that a service class instance is being introduced.

4 Services that use other services

In a command prompt in the root project folder, you can use the Angular CLI to generate a new service with:

```
ng generate service Logger
```

A short form for the above uses only the first letter of the commands, i.e.

```
ng g s service-name
```

Modify the following files in `src/app` with the latest update from `changes`:

```
logger.service-v3.ts

tax-calculator.service-v3.ts
```

Here, the TaxCalculatorService introduces the LoggerService through injection, in the same way that is being done in the component.

When the app is reloaded, you should be able to see the various log messages appearing in the Console tab of Chrome Developer Tools.

5 Services are singleton by default

In the command prompt in the root project folder `servicesdidemo`, generate 2 new components using the Angular CLI:

```
ng generate component firstChild

ng generate component secondChild
```

We generate a new service with:

```
ng generate service Basic
```

Modify the following files in `src/app` with the latest update from `changes`:

```
app.component-v4.html

app.component-v4.ts

basic.service-v4.ts

first-child.component-v4.ts

first-child.component-v4.html

second-child.component-v4.ts

second-child.component-v4.html
```

The service BasicService is accessed in both the FirstChild and SecondChild component via constructor injection. Notice that when you set a new value for the single number property in BasicService in either component, calling getNum on the service in other component returns the newly set value. In other words, there is only a single instance of BasicService that is created

The dependencies / services are singletons within the scope of an injector. When the injector gets a request for a particular service for the first time, it creates a new instance of the service. For all the subsequent requests, it will return the already created instance.

References:

https://www.tektutorialshub.com/angular/angular-singleton-service/