# Angular

Lecture 6

## Agenda

- Recap last lecture points
- Dependency injection
- Services
- Observable and subscribers
- Subjects
- Behavior subject
- http
- Questions !

## Dependency Injection

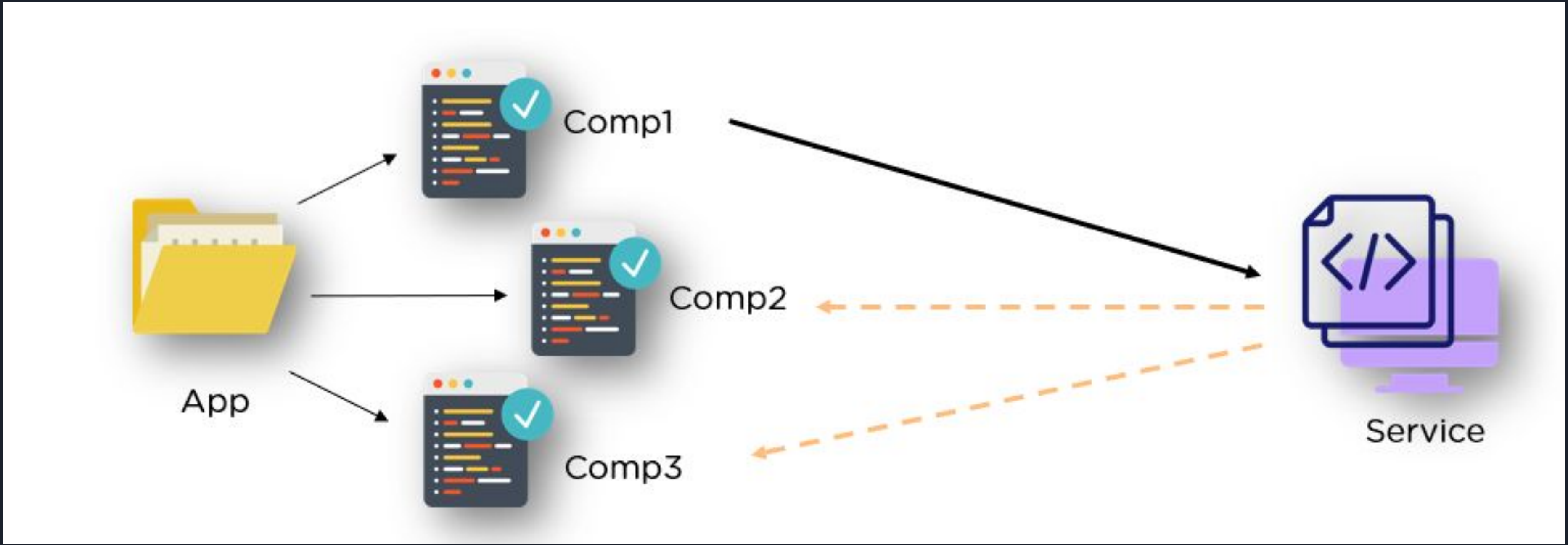
- Dependency or dependent means relying on something for support , Dependencies are services or objects that a class needs to perform its function. Dependency injection, or DI, is a design pattern in which a class requests dependencies from external sources rather than creating them.

- You can use Angular DI to increase flexibility and modularity in your applications.

## Services

- A service is typically a class with a narrow, well-defined purpose. It should do something specific and do it well. A component can delegate certain tasks to services, such as fetching data from the server , In Angular, a class with the @Injectable() decorator that encapsulates non-UI logic and code that can be reused across an application.
- The @Injectable() metadata allows the service class to be used with the dependency injection mechanism. The injectable class is instantiated by a provider. Injectors maintain lists of providers and use them to provide service instances when they are required by components or other services.

## Services



## Services

To generate a new service :

- `ng generate service service-name`

By default the value is set to ‘root’. This translates to the root injector of the application. Basically, setting the field to ‘root’ makes the service available anywhere.

## Observable

Observable is a stream of events or data , Subscribing "kicks off" the observable stream. Without a subscribe (or an async pipe) the stream won't start emitting values. It's similar to subscribing to a newspaper or magazine ... you won't start getting them until you subscribe.

## Subscribe

- A function that defines how to obtain or generate values or messages to be published. This function is executed when a consumer calls the subscribe() method of an observable.
- The subscribe() method takes a JavaScript object (called an observer) with up to three callbacks, one for each type of notification that an observable can deliver:
    - The **Success** notification sends a value such as a number, a string, or an object.
    - The **error** notification sends a JavaScript Error or exception.
    - The **complete** notification doesn't send a value, but the handler is called when the call completes. Scheduled values can continue to be returned after the call completes.

## Transfer data using services

- When passing data between components that lack a direct connection, such as siblings, grandchildren, etc, you should create a shared service. When you have data that should be  sync, BehaviorSubject very useful in this situation.

- BehaviorSubject holds data in a stream and to get data you need to subscribe to get most recent data.

- We use .next()  to update BehaviorSubject value that holds in our service.

## Transfer data using services: Behavior subject

In the service,

- we create a private BehaviorSubject that will hold the current value of the message.
- We define a currentValue variable handle this data stream as an observable that will be used by the components.
- Lastly, we create function that calls next on the BehaviorSubject to change its value.

## Transfer data using services: Behavior subject

```
import { BehaviorSubject } from 'rxjs';

export class DataService {
// new behavior subject with initial value
  private messageSource = new BehaviorSubject('default message');
//public observable value will be used by the components
  currentMessage = this.messageSource.asObservable();
//method to change value , so all components values will be
updated
  changeMessage(message: string) {
    this.messageSource.next(message)
  }
```

## Http Requests in Javascript

The XMLHttpRequest object can be used to exchange data with a server behind the scenes. This means that it is possible to update parts of a web page, without reloading the whole page.

```
var xhttp = new XMLHttpRequest();
  xhttp.onreadystatechange = function () {
    if (this.readyState == 4 && this.status == 200) {
      console.log(this.responseText)
    }
  };
  xhttp.open("GET", "API", true);
  xhttp.send();
```

## Http

Most front-end applications need to communicate with a server over the HTTP protocol, in order to download or upload data and access other back-end services. Angular provides a simplified client HTTP API for Angular applications, the HttpClient service class in @angular/common/http.

**Setup for server communication**

- import the Angular HttpClientModule in app module
  - `import { HttpClientModule } from '@angular/common/http';`

## Http

- In your service that calls apis import httpClient and create a new instance from it in constructor

```
import { HttpClient } from '@angular/common/http';

getList(): Observable<any> {
    return this.http.get('API');
}
```

## Subscribe

- Import your service in component and create instance in constructor then use this service methods

```
this.service.getData().subscribe(
  data => {
    this.data = data
  },
  error => {
    console.log('error: ', error);
  },
  () => {
    console.log('complete ', "compelete");
  }
);
```

# Thank you

## Lab

## Products App

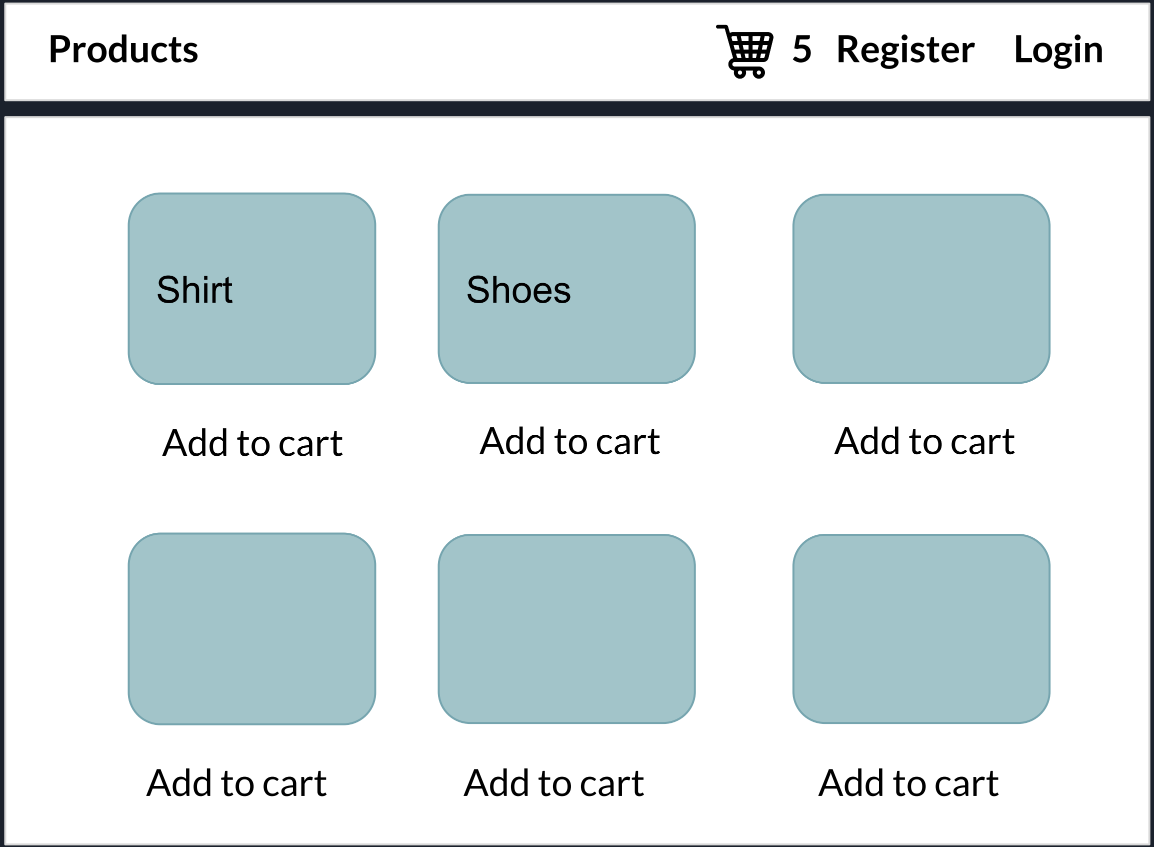
**As a user I would like to:**

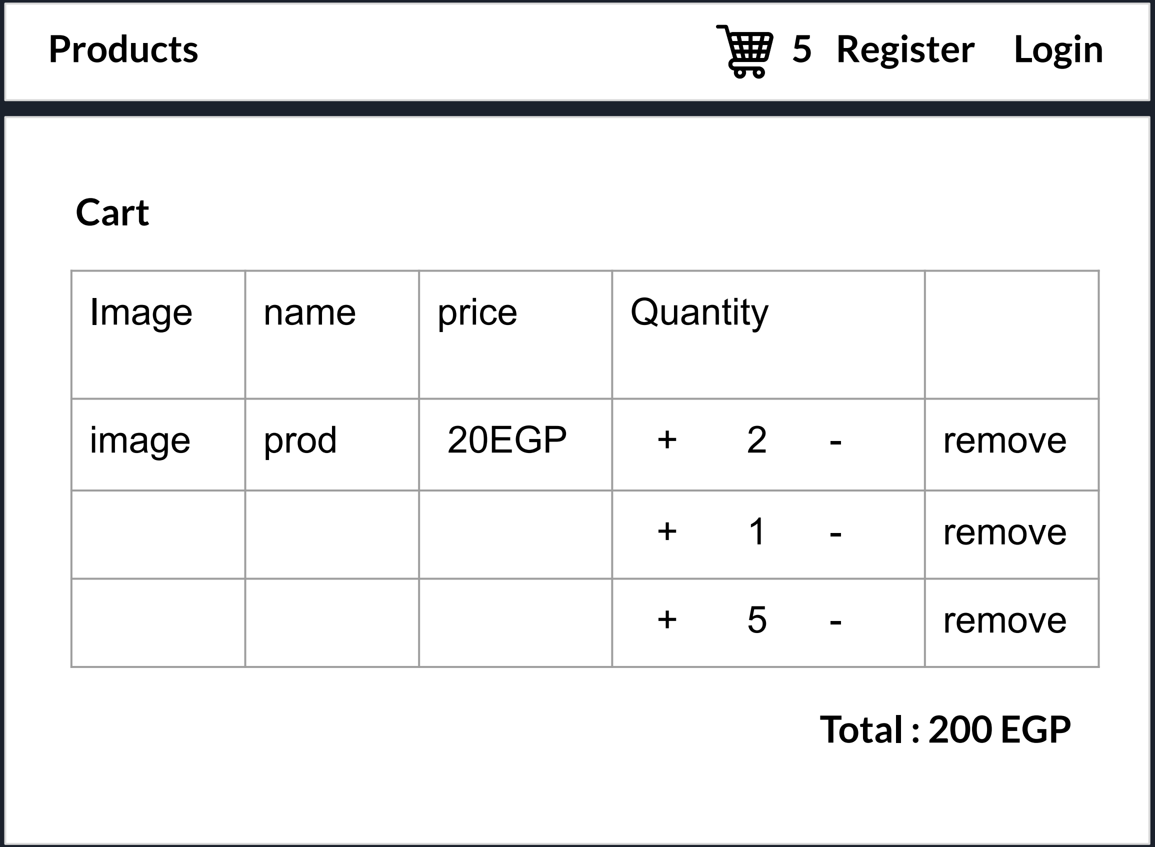
Show list of the products using http Modules instead of static array

Each product card item should have:

- Product images
- Product name
- Product category
- Product price [Use custom pipe to format the price to be shown as the following : **20 EGP**]
- “Add to cart” button

Product card will be clickable to show detailed view for the item clicked


Products
5
Register
Login
Shirt
Shoes
Add to cart
Add to cart
Add to cart
Add to cart
Add to cart
Add to cart



Products 🛒 5 Register Login

Cart

| Image | name | price | Quantity | |
|---|---|---|---|---|
| image | prod | 20EGP | + 2 - | remove |
| | | | + 1 - | remove |
| | | | + 5 - | remove |

Total : 200 EGP

## Products App

**As a user I would like to:**

Create a routing module for Cart page in the navbar:

Shopping cart that shows count value of added and selected items

- When user click on the cart icon show the following:
    - If counter = 0 : show in shopping cart page ( Empty cart )
    - If count > 0 : show items count in page

Your needed APIS will be :

- [Products List : https://fakestoreapi.com/products](https://fakestoreapi.com/products)
- [Product Details : https://fakestoreapi.com/products/id](https://fakestoreapi.com/products/id)

## Products App

Using angular routing module navigation between Master view (products list)/ Detail view (Single item view once item clicked) and the page url parameters updated based on the selected product.

Availability to get back to home from a navigation bar [when click on Products]

- **Show selected products in cart page with option to +/- item count and remove items from cart**

Using Services update guard file to be able to view the protected page after user is logged in.[Bonus]