



Angular | Lecture 6

Marina Magdy



Agenda

- Recap last lecture points
- Dependency injection & Services
- Observable and subscribers
- Behavior subject
- Http Module
- Deploy your app
- Questions !





Dependency Injection & Services





Dependency Injection

A hand-drawn diagram showing a rectangular box labeled 'Component'. Inside the box, the text '{Constructor(service)}' is written. A green arrow points from the word 'service' to the 'Component' box, indicating a dependency.

- 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.

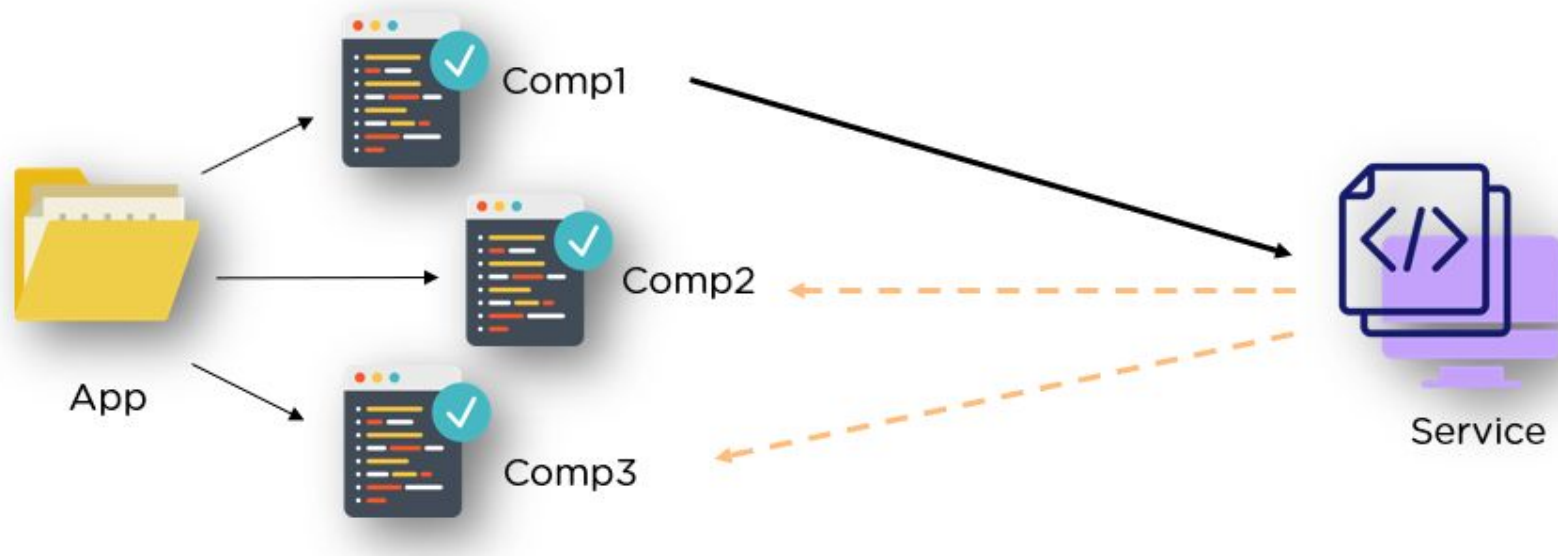


Why Services !

- Components shouldn't fetch or save data directly and they certainly shouldn't knowingly present fake data. They should focus on presenting data and delegate data access to a service.
- When you provide the service at the root level, Angular creates a single, shared instance of this service and injects into any class that asks for it.
- Registering the provider in the `@Injectable` metadata also allows Angular to optimize an application by removing the service if it turns out not to be used after all.



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.



Injectable

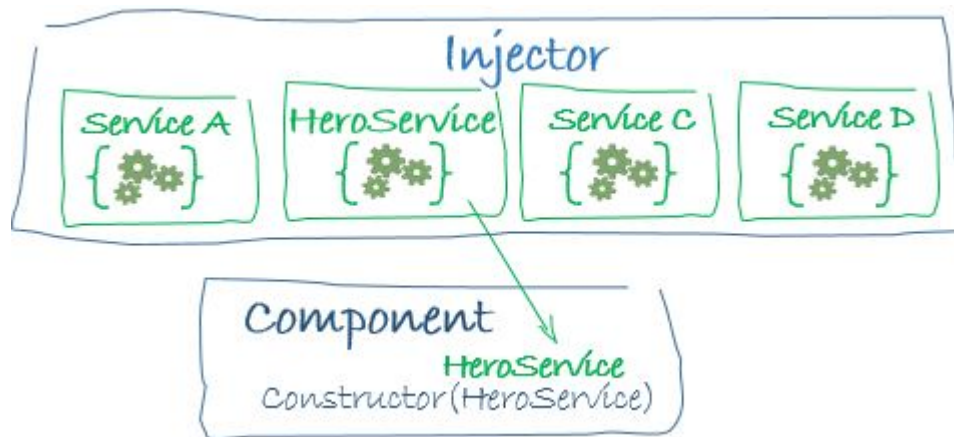
A dependency can be provided in multiple places:

- **Component level:** When you register a provider at the component level, you get a new instance of the service with each new instance of that component.
- **Module level:** When you register a provider with a specific NgModule, the same instance of a service is available to all applicable components, directives and pipe
- **Root level:** Angular creates a single, shared instance of the HeroService and injects it into any class that asks for it.



Inject service

The most common way to inject a dependency is to declare it in a class constructor. When Angular creates a new instance of a component, directive, or pipe class, it determines which services or other dependencies that class needs by looking at the constructor parameter types.





Rxjs

- Stands for Reactive Extensions Library for JavaScript
- Reactive Programming is simply the idea of designing your program to wait on events and react to them when they happen.
- RxJS is a library for composing asynchronous and event-based programs.

<https://rxjs.dev/>

<https://www.youtube.com/watch?v=T9wOu11uU6U&list=PL55RiY5tL51pHpagYcrN9ubNLVXF8rGV>



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.

Observables are streams of data over time. They represent a potential sequence of values, which can be emitted over time.

They are lazy, meaning they don't start emitting values until someone subscribes to them.



Observable

Observables

Think of an Observable as a river. It's a continuous flow of data, which can be anything from numbers, strings, objects, or even events. The key point is that it's a potential stream of values that might be emitted over time.

Observers

An Observer is like a person standing on the bank of the river (Observable). They are interested in the water (data) flowing by.

Subscription

To start receiving data, an Observer must subscribe to the Observable.



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.
- You can get a value from BehaviorSubject using the method `asObservable()`
- We use `.next()` to update the BehaviorSubject value that holds in our service.



Transfer data using services:

Behavior subject

Observable: A fundamental building block in RxJS, representing a potential sequence of values over time. It's like a river, flowing continuously.

BehaviorSubject: A specialized type of Observable that always has a current value. It's like a river with a constant water level, and new observers immediately see the current level.



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 Request





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 `provideHttpClient()` in the array providers in `appConfig` file.



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
- Call the service in ngOnInit for API callings for example <https://angular.io/tutorial/toh-pt4#call-it-in-ngoninit>

Example :

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



Interceptors [Extra]



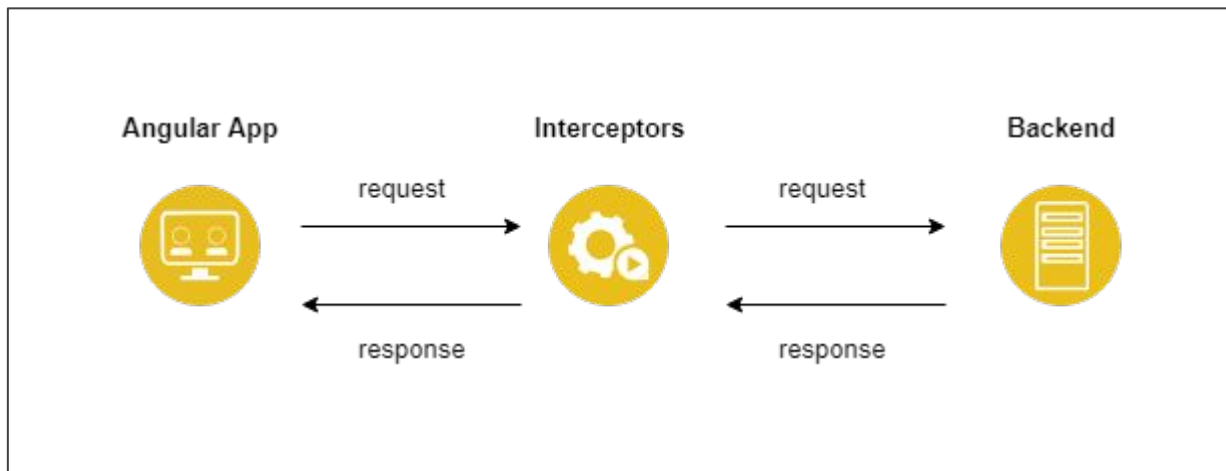


Interceptors

- Interceptors provide a mechanism to intercept and/or mutate outgoing requests or incoming responses.
- It's a simple way provided by the framework to intercept and modify the application's http requests globally before they are sent to the server.
- In order to implement an Interceptor, you need to create a class that implements the intercept method of the `HttpInterceptor` interface.



Interceptors





Interceptors

Providing the Interceptor in app config:

```
provideHttpClient(withInterceptors([loaderInterceptor])),
```



Interceptors



```
1 export const loaderInterceptor: HttpInterceptorFn = (req, next) => {  
2   const newReq = req.clone({  
3     headers: new HttpHeaders({  
4       token: "TOKEEEEN"  
5     })  
6   })  
7   return next(newReq).pipe(finally(() => alert('REQUEST FINISHED')));  
8 };
```



Environments & Deployment





Angular Deployment

Use command line :

`ng build` //Read from environment.development.ts

to generate a build that used for deployment

`ng build --configuration=production` //Read from environment.ts

To generate an optimized build version for production

You can use vercel to deploy your application to live server and get url for your application

<https://vercel.com/solutions/angular>



Environments Variables

What is Angular Environment Variable

The Environment Variable are those variables, whose value changes as per the environment we are in. This will help us to change some behavior of the App based on the environment.

To generate environments: [ng generate environments](#)

First, import the default environment in the component. Note that you should not import any other environment files like environment.prod, but only the default environment file.

```
import { environment } from '../environments/environment';  
  
this.key = environment.key;
```

<https://angular.io/guide/build#configuring-application-environments>



General



Resources

- https://www.youtube.com/playlist?list=PL1w1q3fL4pmjaiypkxQ9-k_CPRsU1tatf
- https://www.youtube.com/watch?v=3BluwVnddG0&list=PL1BztTYDF-QNlGo5-g65Xj1mINHYk_FM9
- <https://www.youtube.com/watch?v=JWhRMyyF7nc>
- <https://blog.angular.io/>
- <https://angular-university.io/>
- <https://www.youtube.com/watch?v=2LCo926NFLI&t=94s>
- https://www.youtube.com/watch?v=T9wOu11uU6U&list=PL55RiY5tL51pHpagYcrN9ubNLVXF8rGV_i
- <https://www.youtube.com/watch?v=ewcoEYS85Co&t=95s>
- FullCourse : <https://www.youtube.com/watch?v=3qBXWUpoPHo>
- <https://www.youtube.com/watch?v=K6FA6f8fohk&list=PL1w1q3fL4pmjgBCa3R1T25LowreF5XfeA>
- <https://egghead.io/q/angular>
- <https://www.youtube.com/watch?v=d56mG7DezGs>
- Unit testing:
https://www.youtube.com/watch?v=VnaDJOucT-4&list=PL_euSNU_eLbcqJ6_Z3FOZJ8mLZ2oMFItV
- NGRX[Extra]:https://www.youtube.com/watch?v=3WI5BEXVkmE&list=PL_euSNU_eLbdq0gKbR8zmVJb4xLqHR7BX



Thank you



Lab



Products App

As a user I would like to:

- Show list of the products using http Modules instead of static array
- Product card will be clickable to show detailed view for the item clicked
- User can redirect to cart page.

Your needed APIS will be :

- Products List : <https://dummyjson.com/products>
- Product Details : <https://dummyjson.com/products/{id}>



Welcome to our shopping website , start browsing...



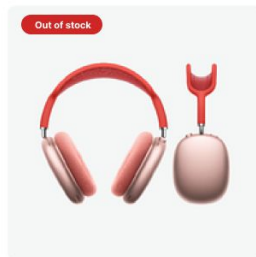
Wireless Earbuds, IPX8

\$89.⁰⁰

Organic Cotton, fairtrade certified

★★★★★

Add to Cart



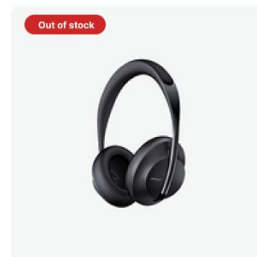
AirPods Max

\$559.⁰⁰

A perfect balance of high-fidelity audio

★★★★★

Add to Cart



Bose BT Earphones

\$289.⁰⁰

Table with air purifier, stained venner/black

★★★★★

Add to Cart



VIVEFOX Headphones

\$39.⁰⁰

Wired Stereo Headsets With Mic

★★★★★

Add to Cart



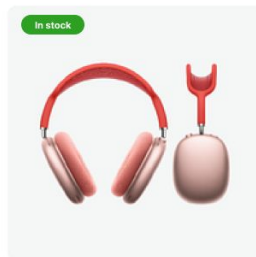
Wireless Earbuds, IPX8

\$89.⁰⁰

Organic Cotton, fairtrade certified

★★★★★

Add to Cart



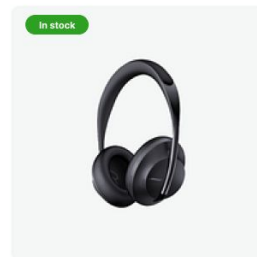
AirPods Max

\$559.⁰⁰

A perfect balance of high-fidelity audio

★★★★★

Add to Cart



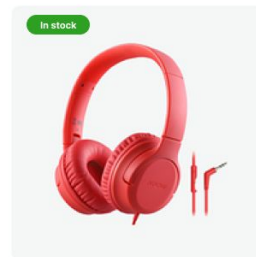
Bose BT Earphones

\$289.⁰⁰

Table with air purifier, stained venner/black

★★★★★

Add to Cart



VIVEFOX Headphones

\$39.⁰⁰


Wired Stereo Headsets With Mic

★★★★★

Add to Cart



Cart

Description		Quantity	Remove	Price
	Headphones Product Code: MLSB	<div><div>+</div><div>1</div><div>-</div></div>	<div>×</div>	£55
	Headphones Product Code: MLSB	<div><div>+</div><div>1</div><div>-</div></div>	<div>×</div>	£55
Total				£110.00





Products App

As a user I would like to:

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

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





Products App

[Bonus]

Change product quantity from the cart page will be limited to the stock value , should not exceed the stock value.

[Bonus]

Pagination for products list

https://dummyjson.com/docs/products#products-limit_skip

