Multiple Interceptors in Angular

Hossein Mousavi Follow

Aug 22 · 3 min read

One of the amazing features that Angular provides is the **interceptors,** but what does an Interceptor do, and can we implement multiple of them in our Angular project?

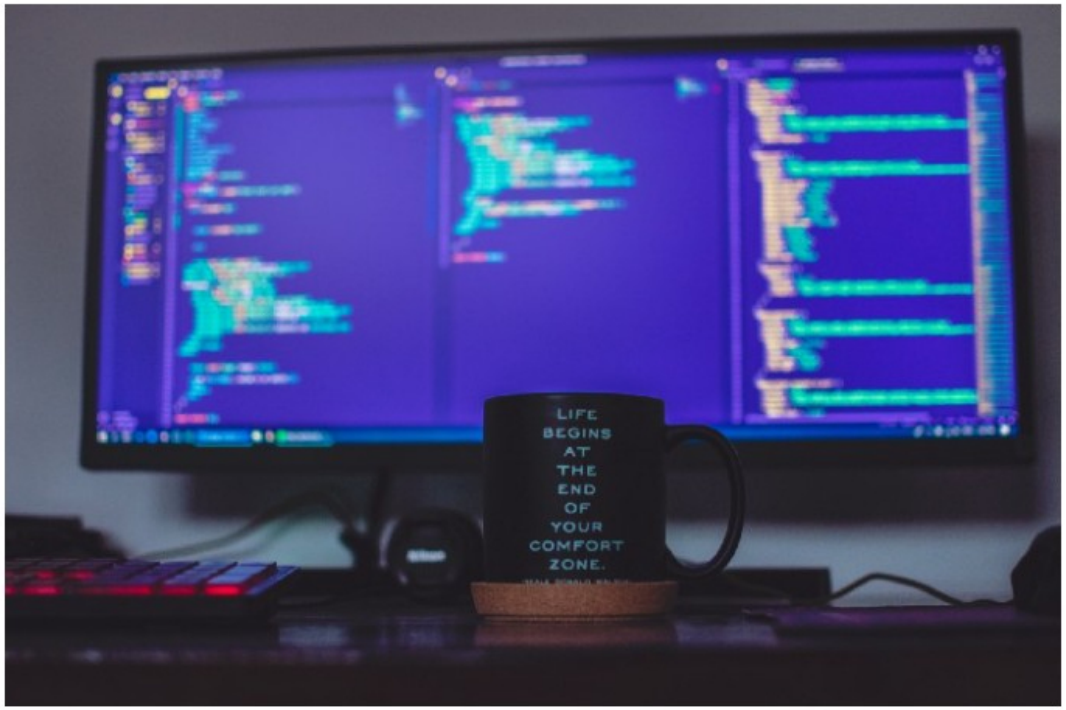
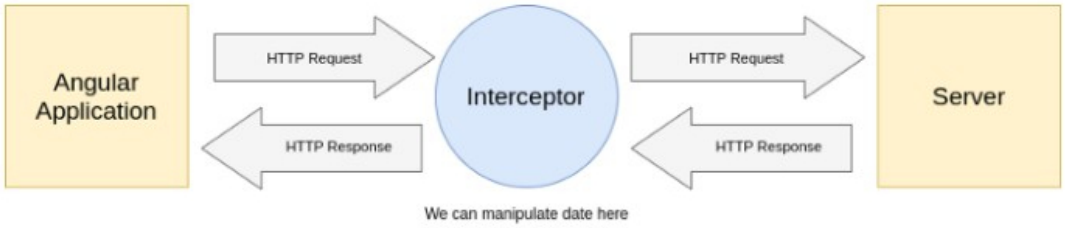


Photo by Tudor Baciu on Unsplash

. . .

What is Interceptor?

We can use Interceptors to intercept the incoming and outgoing HTTP requests and manipulate them using Angular `HttpClient`.



Interceptor is in the middle of Angular application and server (image created using draw.io)

One of the most use cases for adding an interceptor to our Angular application is adding the **token** to the header of outgoing HTTP requests.

Let's implement an interceptor in our Angular application!

We need to import `HTTP_INTERCEPTORS` from `@angular/common/http` in our `app.module` to make sure this will work for all of our requests. Then we should provide that in our `providers`. We should also import our interceptor and our module and set the value of `useClass` to it.

```
import { HTTP_INTERCEPTORS } from '@angular/common/http';
import { Interceptor } from './interceptor';

@NgModule({

   providers: [
     {
```

```
      provide: HTTP_INTERCEPTORS, useClass: Interceptor, multi: true
    },
],})
```

After providing `HTTP_INTERCEPTORS` we need to inform the class we are going to implement our interceptor into by using `useClass.`

> *Setting **multi*** to true, makes sure that you can have multiple interceptors in your project.

Now we create `interceptor.ts` in `/src/app` and since interceptors are services we need to use `@Injectable()` decorator in our file:

```
import { HttpEvent, HttpHandler, HttpHeaders, HttpInterceptor, HttpRequest } from '@angular/comm
import { Injectable } from '@angular/core';
import { Observable } from 'rxjs';
@Injectable()
export class Interceptor implements HttpInterceptor {
  constructor() {}

  token = localStorage.getItem('token');

  intercept(req: HttpRequest<any>, next: HttpHandler): Observable<HttpEvent<any>> {
    const auth = req.clone({ headers: new HttpHeaders().set(`Authorization`, `Bearer ${this.toke
    return next.handle(auth);
  }
}
```

interceptor.ts hosted with ❤ by GitHub view raw

> *Keep in mind that Interceptors are **Services.***

After every single HTTP request, the Interceptor calls the `intercept()` method. In `Intercept()` method, we clone the HttpRequest we are going to

send and add the token to its HttpHeaders, and once it's done we call `next.handle` to return manipulated requests to the application.

Adding the second interceptor

It would be nice to have a loading component wrapper for the whole project and once the requests from the server resolve, we can remove the loading from the layout of the project.

First, we need to create `loadingInterceptor.ts` file like bellow:

```ts
import { HttpEvent, HttpHandler, HttpInterceptor, HttpRequest } from '@angular/common/http';
import { Injectable } from '@angular/core';
import { Observable, Subscription } from 'rxjs';
import { finalize } from 'rxjs/operators';
import { LoadingService } from './shared/services/loading.service';

@Injectable()
export class LoadingInterceptor implements HttpInterceptor {
  constructor(private readonly loadingService: LoadingService) {}

  intercept(req: HttpRequest<any>, next: HttpHandler): Observable<HttpEvent<any>> {
    const spinnerSubscription: Subscription = this.loadingService.spinner$.subscribe();
    return next.handle(req).pipe(finalize(() => spinnerSubscription.unsubscribe()));
  }
}
```

loadingInterceptor.ts hosted with ❤ by GitHub view raw

We will cover the `LoadingService` later, but for now, let's check the `loadingInterceptor` first. We are creating a `Subscription` and then after every HTTP request is handled and *finalized* we are unsubscribing `spinnerSubscription` which we are going to use in out `loadingService` for indicating a spinner loading.

Now let's take a look at `loadingService.ts`

```ts
import { Overlay, OverlayRef } from '@angular/cdk/overlay';
import { ComponentPortal } from '@angular/cdk/portal';
import { Injectable } from '@angular/core';
import { defer, NEVER, Subject } from 'rxjs';
import { finalize, share } from 'rxjs/operators';
import { LoadingComponent } from '../components/loading/loading.component';

@Injectable({ providedIn: 'root' })
export class LoadingService {
  private overlayRef?: OverlayRef;
  isLoading = new Subject<boolean>();

  constructor(private overlay: Overlay) {}

  public show(): void {
    this.isLoading.next(true);
    Promise.resolve(null).then(() => {
      this.overlayRef = this.overlay.create({
        positionStrategy: this.overlay.position().global().centerHorizontally().centerVertically
        hasBackdrop: true,
      });
      this.overlayRef.attach(new ComponentPortal(LoadingComponent));
    });
  }

  public readonly spinner$ = defer(() => {
    this.show();
    return NEVER.pipe(finalize(() => this.hide()));
  }).pipe(share());

  public hide(): void {
    this.isLoading.next(false);
    this.overlayRef!.detach();
    this.overlayRef = undefined;
  }
}
```

LoadingService.ts hosted with ❤ by GitHub view raw

We can add and remove an overlay with a spinner component (line 22) in

our Angular application with the above service until the subscription has not being unsubscribed. The `loadingComponent` can be anything you want, a GIF, a spinner, or a progress bar. I have set this to be a `mat-progress-bar` like this:

```
<mat-progress-bar color="warn" mode="indeterminate">
</mat-progress-bar>
```

Do not forget to import the second interceptor in your `app.module.ts` file:

```
providers: [
  {
    provide: HTTP_INTERCEPTORS, useClass: Interceptor, multi: true
  },
  {
    provide: HTTP_INTERCEPTORS, useClass: LoadingInterceptor,
multi:true
  }
],
```

> ***Important note:*** *The order of providing interceptors does matter. For example if you provide three interceptors called A, B, C in that order, as the Angular Docs say: "…requests flow in A->B->C and responses flow out C->B->A."*

. . .

The source code of this project is available at GitHub.

Share with your friends! Clap 👏 as max as 50 times.

If you have any problem with implementing Angular Interceptors, don't hesitate to comment down below. You can also reach me out on Twitter or GitHub.

Read more from me:

Husky 6 Lint (prettier + eslint) and commitlint for JavaScript Projects

Programming is a teamwork job so we must assure that our codebase is clean and usable for everyone in the team with the...

medium.com

Angular forms (Reactive Form) including Angular Material and Custom Validator

Forms are major parts of every Angular project and in this article, we want to implement a Reactive Angular form with a...

medium.com

Take a good look at filter, map, and reduce in JavaScript

In JavaScript we have These three methods as parts of Array.prototype methods, but what are the differences between...

blog.usejournal.com

Angular JavaScript Interceptors Typescript

WRITTEN BY

Hossein Mousavi

Software Developer | Linux Enthusiastic | HMousavi.dev

Follow

CodeX

Everything connected with Tech & Code. Follow to join our 500K+ monthly readers

Follow

More From Medium

More from CodeX

60 Best Gifts for Programmers, Developers, and all Tech Lovers [2021]

Lindsey Tam
Nov 19 · 20 min read ★

579

More from CodeX

Intel's Secret Plan to Destroy ARM

Erik Engheim
Nov 15 · 7 min read ★

717

More from CodeX

Complete Self-Hosted Bitwarden for Raspberry Pi

mr.smashy
Nov 27 · 10 min read ★

4

Learn more.

Medium is an open platform where 170 million readers come to find insightful and dynamic thinking. Here, expert and undiscovered voices alike dive into the heart of any topic and bring new ideas to the surface. Learn more

Make Medium yours.

Follow the writers, publications, and topics that matter to you, and you'll see them on your homepage and in your inbox. Explore

Write a story on Medium.

If you have a story to tell, knowledge to share, or a perspective to offer — welcome home. It's easy and free to post your thinking on any topic. Start a blog

---