Angular Authentication: Using the Http Client and Http Interceptors

Ryan Chenkie Jul 18, 2017 · 6 min read

Hey! I'm Ryan and I teach at Angularcasts. Follow me on Twitter and let me know what you're working on!

Angular 4.3 is here and with it comes a brand new set of HTTP tools with a bunch of useful features. Perhaps the most long-awaited feature addition is the `HttpInterceptor` interface. Until now, there was no way to intercept and modify HTTP requests globally. This has always been possible in AngularJS and the fact that it has been lacking in Angular 2+ has been a sticking point with developers.

So why are HTTP interceptors useful? There are many reasons, but one common use case is to automatically attach authentication information to requests. This can take several different forms but most often involves attaching a JSON Web Token (or other form of access token) as an `Authorization` header with the `Bearer` scheme.

Let's take a look at how to use Angular's `HttpInterceptor` interface to make authenticated HTTP requests.

> ***Quick note:*** *I've just released a book called* ***Securing Angular Applications****. It will teach you* ***everything*** *you need to know to properly add authentication and authorization to you Angular app.*

> *You'll also learn how to mitigate common security threats. Check it out if you're interested :)*

Make an Authentication Service

When handling authentication in an Angular app, it's generally best to put everything you need in a dedicated service. Any authentication service should have a few basic methods for allowing users to log in and log out. It should also include a method for retrieving a JSON Web Token from wherever it is stored on the client and a way to determine if the user is authenticated or not.

> This article assumes you already have an authentication setup in place and that you are storing JWTs in local storage. We won't get into a full details here.

One way we can check whether a JWT is expired is to use **angular2-jwt** to return a `boolean` after checking the `exp` claim.

```
npm i --save angular2-jwt
```

After installing **angular2-jwt**, use it in a service.

```
// src/app/auth/auth.service.ts

import { Injectable } from '@angular/core';
import decode from 'jwt-decode';

@Injectable()
export class AuthService {

  public getToken(): string {
    return localStorage.getItem('token');
  }

  public isAuthenticated(): boolean {
    // get the token
    const token = this.getToken();
    // return a boolean reflecting
    // whether or not the token is expired
    return tokenNotExpired(null, token);
  }

}
```

Create an Interceptor

The goal is to include the JWT which is in local storage as the `Authorization` header in any HTTP request that is sent. The first step is to create an interceptor. To do this, create an `Injectable` class which implements `HttpInterceptor`.

```
// src/app/auth/token.interceptor.ts

import { Injectable } from '@angular/core';
import {
  HttpRequest,
  HttpHandler,
  HttpEvent,
  HttpInterceptor
} from '@angular/common/http';
import { AuthService } from './auth/auth.service';
import { Observable } from 'rxjs/Observable';
```

```
@Injectable()
export class TokenInterceptor implements HttpInterceptor {

  constructor(public auth: AuthService) {}

  intercept(request: HttpRequest<any>, next: HttpHandler):
Observable<HttpEvent<any>> {

    request = request.clone({
      setHeaders: {
        Authorization: `Bearer ${this.auth.getToken()}`
      }
    });

    return next.handle(request);
  }
}
```

Any interceptor that we want to create needs to implement the `HttpInterceptor` interface. This means that our new class must have a method called `intercept` with `HttpRequest` and `HttpHandler` parameters. Using interceptors is all about changing outgoing requests and incoming responses, but we can't tamper with the original request–it needs to be immutable. To make changes we need to `clone` the original `request`.

As we `clone` the original `request` we can set the headers we want. In our case its very simple–we just want to add an `Authorization` header with an auth scheme of `Bearer` followed by the JSON Web Token in local storage which we get from a call to the `getToken` method from the `AuthService`.

Calling `next.handle` means that we are passing control to the next interceptor in the chain, if there is one.

Add the Interceptor to Providers

The interceptor needs to be added to the `HTTP_INTERCEPTORS` array. This is done by making the existing `HTTP_INTERCEPTORS` array use the new class we've created. Add this in the `providers` array for our application's module.

```
// src/app/app.module.ts

import { HTTP_INTERCEPTORS } from '@angular/common/http';
import { TokenInterceptor } from './../auth/token.interceptor';

@NgModule({
  bootstrap: [AppComponent],
  imports: [...],
  providers: [
    {
      provide: HTTP_INTERCEPTORS,
      useClass: TokenInterceptor,
      multi: true
    }
  ]
})
export class AppModule {}
```

Now when we make any HTTP request, the user's token will be attached automatically.

```
// src/app/ping/ping.component.ts

import { HttpClient } from '@angular/common/http';
// ...
export class AppComponent {
```

```
  constructor(public http: HttpClient) {}

  public ping() {
    this.http.get('https://example.com/api/things')
      .subscribe(
        data => console.log(data),
        err => console.log(err)
      );
  }

}
```

This request will include an `Authorization` header with a value of `Bearer ey...`.

It should be noted that Angular's new `HttpClient` from `@angular/common/http` is being used here and not the `Http` class from `@angular/http`. If we try to make requests with the traditional `Http` class, the interceptor won't be hit.

Top highlight

Looking for Unauthorized Responses

When tokens expire we will generally get a `401 Unauthorized` response back from the server. This gives us an indication that we need the user to log in again to get a new token.

We have some choices to make at this point. Do we want to redirect to a specific route that has a login form? Do we want to show a modal? Do we want to attempt to refresh the token?

Either way, we need to set up the intereptor to handle responses. The `intercept` method returns an observable which means we can capture the success and error channels for a response and operate on them however we like. This is the perfect place to do any kind of logging we might want to do. We can also check for `401 Unauthorized` responses and prompt the user to log in again.

```
// src/app/auth/jwt.interceptor.ts

// ...
import 'rxjs/add/operator/do';

export class JwtInterceptor implements HttpInterceptor {

  constructor(public auth: AuthService) {}

  intercept(request: HttpRequest<any>, next: HttpHandler):
Observable<HttpEvent<any>> {

    return next.handle(request).do((event: HttpEvent<any>) => {
      if (event instanceof HttpResponse) {
        // do stuff with response if you want
      }
    }, (err: any) => {
      if (err instanceof HttpErrorResponse) {
        if (err.status === 401) {
          // redirect to the login route
          // or show a modal
        }
      }
    });
  }
}
```

This is also a great spot to cache any failed requests. This comes in handy if we have token refreshing in place and we want to retry the requests

once we have a new token.

```
// src/app/auth/auth.service.ts

import { HttpRequest } from '@angular/common/http';

// ...
export class AuthService {

cachedRequests: Array<HttpRequest<any>> = [];

public collectFailedRequest(request): void {
    this.cachedRequests.push(request);
  }

public retryFailedRequests(): void {
    // retry the requests. this method can
    // be called after the token is refreshed
  }

}
```

The `collectFailedRequests` method can now be used in the interceptor.

```
// src/app/auth/jwt.interceptor.ts

// ...
intercept(request: HttpRequest<any>, next: HttpHandler):
Observable<HttpEvent<any>> {
  return next.handle(req).do((event: HttpEvent<any>) => {
    if (event instanceof HttpResponse) {
      // do stuff with response if you want
    }
  }, (err: any) => {
    if (err instanceof HttpErrorResponse {
      if (err.status === 401) {
        this.auth.collectFailedRequest(request);
      }
    }
  });
}
```

With this in place, we have the option of calling `retryFailedRequests` after the user's token is refreshed to fire off the previously-failed requests. This is just a small addition that can help to greatly improve UX, especially if you have tokens with a very short lifetime.

Wrapping Up

Angular 4.3 offers a brand new set of features for working with HTTP requests. Perhaps one of the most useful is the new `HttpInterceptor` interface which allows us to modify outgoing requests and incoming responses. This feature greatly simplifies a lot of previously tricky operations and removes the need for a lot of class wrappers that have been around since the early days of Angular 2.

In fact, a library that I wrote called **angular2-jwt** has, up until now, relied on wrapping all of the methods offered by Angular's `Http` class. I'm now rewriting it to use `HttpInterceptor` and it is resulting in **much** less library code.

If you want to learn everything you need to know to properly secure an Angular application, be sure to check out the book that I'll be launching this fall: **Securing Angular Applications**.

JavaScript Angular2 Angularjs Angular Angular 4

8.5K 77

More from Ryan Chenkie

Follow

Mostly JavaScripting with Angular and Node. Screencasting at https://angularcasts.io , Google Developer Expert. Formerly at @auth0.

Dec 15, 2016

Angular CLI Deployment: Host Your Angular 2 App on Heroku

Hey! I'm Ryan and I teach at Angularcasts. Follow me on Twitter and let me know what you're working on!

The Angular CLI is all around awesome and gives us a ton of time-saving features out-of-the-box. One that I love is the development server it comes with. If you've used the Angular CLI, there's a good chance you've run `ng serve` and then visited `localhost:4200` to see your app.

This is great for development purposes, but what do we do when it comes time to put our apps into production? The CLI comes with a command to deploy to GitHub...

Read more · 8 min read

1.5K 71

More From Medium

Understand Angular's forRoot and forChild

Heloise Parein in JavaScript in Plain English

Concepts of JavaScript—Part 3

Anil Kumar

How to create fluent interfaces the easy way with vanilla JavaScript

Jason Barr in Simply Web Development

Don't use bootstrap

Sho Mukai

When color theory meets technology: auto-generated accessible color schemes with VueJs

Krystal Campioni in Vue.js Developers

JavaScript Increment and Decrement operators

Kesk -*- in JavaScript in Plain English

Are you still running jQuery 1.x? Why?

Todd Peters in Website Upgrade and Patching Blog

What Are React Hooks?

Valeria Hutsuliak