(/)

Front-End App with Spring Security OAuth – Authorization Code Flow

Last modified: July 3, 2018

by baeldung (/author/baeldung/)

Security (/category/security-2/) **Spring (/category/spring/)** **+**

OAuth (/tag/oauth/)

1. Overview

In this tutorial, we'll continue our Spring Security OAuth series (/spring-security-oauth) by building a simple front end for Authorization Code flow.

Keep in mind that the focus here is the client-side; have a look at the Spring REST API + OAuth2 + AngularJS (/rest-api-spring-oauth2-angularjs) writeup – to review detailed configuration for both Authorization and Resource Servers.

2. Authorization Server

Before we get to our front end, we need to add our client details in our Authorization Server configuration:

```
@Configuration
@EnableAuthorizationServer
public class OAuth2AuthorizationServerConfig extends AuthorizationSer

    @Override
    public void configure(ClientDetailsServiceConfigurer clients) thr
        clients.inMemory()
                .withClient("fooClientId")
                .secret(passwordEncoder().encode("secret"))
                .authorizedGrantTypes("authorization_code")
                .scopes("foo", "read", "write")
                .redirectUris("http://localhost:8089/ (http://localhos
...
```

Note how we now have the Authorization Code grant type enabled, with the following, simple details:

- our client id is *fooClientId*
- our scopes are *foo*, *read* and *write*
- the redirect URI is *http://localhost:8089/* (we're going to use port 8089 for our front-end app)

3. The Front End

Now, let's start building our simple front-end application.

As we're going to use to use Angular 6 for our app here, we need to use the *frontend-maven-plugin* plugin in our Spring Boot application:

```xml
<plugin>
    <groupId>com.github.eirslett</groupId>
    <artifactId>frontend-maven-plugin</artifactId>
    <version>1.6</version>

    <configuration>
        <nodeVersion>v8.11.3</nodeVersion>
        <npmVersion>6.1.0</npmVersion>
        <workingDirectory>src/main/resources</workingDirectory>
    </configuration>

    <executions>
        <execution>
            <id>install node and npm</id>
            <goals>
                <goal>install-node-and-npm</goal>
            </goals>
        </execution>

        <execution>
            <id>npm install</id>
            <goals>
                <goal>npm</goal>
            </goals>
        </execution>

        <execution>
            <id>npm run build</id>
            <goals>
                <goal>npm</goal>
            </goals>

            <configuration>
                <arguments>run build</arguments>
            </configuration>
        </execution>
    </executions>
</plugin>
```

Note that, naturally, we need to install Node.js (https://nodejs.org/en/) first on our box; we'll use the Angular CLI to generate the base for our app:

```
ng new authCode
```

4. Angular Module

Now, let's discuss our Angular Module in detail.

Here's our simple *AppModule*:

```
import { BrowserModule } from '@angular/platform-browser';
import { NgModule } from '@angular/core';
import { HttpClientModule } from '@angular/common/http';
import { RouterModule }   from '@angular/router';
import { AppComponent } from './app.component';
import { HomeComponent } from './home.component';
import { FooComponent } from './foo.component';

@NgModule({
  declarations: [
    AppComponent,
    HomeComponent,
    FooComponent
  ],
  imports: [
    BrowserModule,
    HttpClientModule,
    RouterModule.forRoot([
     { path: '', component: HomeComponent, pathMatch: 'full' }], {on
  ],
  providers: [],
  bootstrap: [AppComponent]
})
export class AppModule { }
```

Our Module consists of three Components and one service, we'll discuss them in the following sections

4.1. App Component

Let's start with our *AppComponent* which is the root component:

```
import {Component} from '@angular/core';

@Component({
    selector: 'app-root',
    template: `<nav class="navbar navbar-default">
  <div class="container-fluid">
    <div class="navbar-header">
      <a class="navbar-brand" href="/">Spring Security Oauth - Author
    </div>
  </div>
</nav>
<router-outlet></router-outlet>`
})

export class AppComponent {}
```

4.2. Home Component

Next is our main component, *HomeComponent*:

```
import {Component} from '@angular/core';
import {AppService} from './app.service'

@Component({
    selector: 'home-header',
    providers: [AppService],
  template: `<div class="container" >
    <button *ngIf="!isLoggedIn" class="btn btn-primary" (click)="log
    <div *ngIf="isLoggedIn" class="content">
        <span>Welcome !!</span>
        <a class="btn btn-default pull-right"(click)="logout()" href=
        <br/>
        <foo-details></foo-details>
    </div>
</div>`
})

export class HomeComponent {
     public isLoggedIn = false;

    constructor(
        private _service:AppService){}

    ngOnInit(){
        this.isLoggedIn = this._service.checkCredentials();
        let i = window.location.href.indexOf('code');
        if(!this.isLoggedIn && i != -1){
            this._service.retrieveToken(window.location.href.substrin
        }
    }

    login() {
        window.location.href = 'http://localhost:8081/spring-security
    }

    logout() {
        this._service.logout();
    }
}
```

Note that:

- If the user is not logged in, only the login button will appear
- The login button redirect user to the Authorization URL
- When user is redirected back with the authorization code, we retrieve access token using this code

4.3. Foo Component

Our third and final component is the *FooComponent*; this displays the *Foo* resources – obtained from Resource Server:

```
import { Component } from '@angular/core';
import {AppService, Foo} from './app.service'

@Component({
  selector: 'foo-details',
  providers: [AppService],
  template: `<div class="container">
    <h1 class="col-sm-12">Foo Details</h1>
    <div class="col-sm-12">
        <label class="col-sm-3">ID</label> <span>{{foo.id}}</span>
    </div>
    <div class="col-sm-12">
        <label class="col-sm-3">Name</label> <span>{{foo.name}}</spar
    </div>
    <div class="col-sm-12">
        <button class="btn btn-primary" (click)="getFoo()" type="subr
    </div>
</div>`
})

export class FooComponent {
    public foo = new Foo(1,'sample foo');
    private foosUrl = 'http://localhost:8082/spring-security-oauth-re

    constructor(private _service:AppService) {}

    getFoo(){
        this._service.getResource(this.foosUrl+this.foo.id)
         .subscribe(
            data => this.foo = data,
            error =>  this.foo.name = 'Error');
    }
}
```

4.4. App Service

Now, let's take a look at the *AppService*:

```
import {Injectable} from '@angular/core';
import { Cookie } from 'ng2-cookies';
import { HttpClient, HttpHeaders } from '@angular/common/http';
import { Observable } from 'rxjs/Observable';
import 'rxjs/add/operator/catch';
import 'rxjs/add/operator/map';
```

```

export class Foo {
  constructor(
    public id: number,
    public name: string) { }
}

@Injectable()
export class AppService {
   public clientId = 'fooClientId';
   public redirectUri = 'http://localhost:8089/ (http://localhost:808

  constructor(
    private _http: HttpClient){}

  retrieveToken(code){
    let params = new URLSearchParams();
    params.append('grant_type','authorization_code');
    params.append('client_id', this.clientId);
    params.append('redirect_uri', this.redirectUri);
    params.append('code',code);

    let headers = new HttpHeaders({'Content-type': 'application/x-www
     this._http.post('http://localhost:8081/spring-security-oauth-ser
    .subscribe(
      data => this.saveToken(data),
      err => alert('Invalid Credentials')
    );
  }

  saveToken(token){
    var expireDate = new Date().getTime() + (1000 * token.expires_in)
    Cookie.set("access_token", token.access_token, expireDate);
    console.log('Obtained Access token');
    window.location.href = 'http://localhost:8089 (http://localhost:8
  }

  getResource(resourceUrl) : Observable<any>{
    var headers = new HttpHeaders({'Content-type': 'application/x-www
    return this._http.get(resourceUrl,{ headers: headers })
                   .catch((error:any) => Observable.throw(error.json(
  }

  checkCredentials(){
    return Cookie.check('access_token');
  }

  logout() {
    Cookie.delete('access_token');
    window.location.reload();
  }
}
```

Let's do a quick rundown of our implementation here:

- *checkCredentials()*: to check if user is logged in
- *retrieveToken()*: to obtain access token using authorization code
- *saveToken()*: to save Access Token in a cookie
- *getResource()*: to get Foo details using its ID
- *logout()*: to delete Access Token cookie

5. Run the Application

To run our application and make sure everything is working properly, we need to:

- First, run Authorization Server on port 8081
- Then, run the Resource Server on port 8082
- Finally, run the Front End

We'll need to build our app first:

```
mvn clean install
```

Then change directory to src/main/resources:

```
cd src/main/resources
```

Then run our app on port 8089:

```
npm start
```

6. Conclusion

We learned how to build a simple front end client for Authorization Code flow using Spring and Angular 6.

And, as always, the full source code is available over on GitHub (https://github.com/Baeldung/spring-security-oauth).

Leave a Reply

Start the discussion...

Subscribe

CATEGORIES

SPRING (/CATEGORY/SPRING/)
REST (/CATEGORY/REST/)
JAVA (/CATEGORY/JAVA/)
SECURITY (/CATEGORY/SECURITY-2/)
PERSISTENCE (/CATEGORY/PERSISTENCE/)
JACKSON (/CATEGORY/JACKSON/)
HTTPCLIENT (/CATEGORY/HTTP/)
KOTLIN (/CATEGORY/KOTLIN/)

SERIES

JAVA “BACK TO BASICS” TUTORIAL (/JAVA-TUTORIAL)
JACKSON JSON TUTORIAL (/JACKSON)
HTTPCLIENT 4 TUTORIAL (/HTTPCLIENT-GUIDE)
REST WITH SPRING TUTORIAL (/REST-WITH-SPRING-SERIES/)
SPRING PERSISTENCE TUTORIAL (/PERSISTENCE-WITH-SPRING-SERIES/)
SECURITY WITH SPRING (/SECURITY-SPRING)

ABOUT

ABOUT BAELDUNG (/ABOUT/)
THE COURSES (HTTP://COURSES.BAELDUNG.COM)
CONSULTING WORK (/CONSULTING)
META BAELDUNG (HTTP://META.BAELDUNG.COM/)
THE FULL ARCHIVE (/FULL_ARCHIVE)
WRITE FOR BAELDUNG (/CONTRIBUTION-GUIDELINES)
CONTACT (/CONTACT)
EDITORS (/EDITORS)
MEDIA KIT (PDF) (HTTPS://S3.AMAZONAWS.COM/BAELDUNG.COM/BAELDUNG+-+MEDIA+KIT.PDF)