Full Stack Spring Boot Angular

Contents

Basic Authentication with Spring Boot and Spring Security

Introduction

- The easiest way you can make something secure, is by adding a very simple dependency, spring-boot-starter-security.
- It is the spring boot starter for implementing security in web applications, as well as RESTful services.
- By default spring-boot-starter-security enables both **basic authentication** as well as form based authentication with username and password.
- When we use basic authentication from browser, it prompts us to enter username and password, that is why it is called as **form based authentication**.
- Default username is 'user' and default password is generated and printed in the console when application is started. You can override it via application.properties file using below properties -

```
spring.security.user.name=user
spring.security.password=password
```

- When you hit the rest service having basic authentication from browse directly, it is called form based authentication as it asks for username and password.
- When you hit the rest service having basic authentication via rest client, we need to send authentication information via "Basic" Authentication header. So header name as "Authorization" and value as "Basic xxxxxxxxxxxxxxx" which is an encoded form of the userid and password (userid and password separated by a colon).

Form Based Authentication

- Once you enter the user id and password with form based authentication, a session cookie is set.
- So there is a session for you which is created on the server side, and a cookie is now registered in your browser, and that cookie is sent along with every request.
- So as long as you are in the same browser and keep opening tabs, all requests would continue succeeding.
- This is why it is called form based authentication, and this is enabled by default by spring security.

Difference between Basic Authentication and Form based authentication

- Form based authentication is based on a session which is created on the server and a cookie which is in your browser. So this cookie identifies the session on the browser, and that will be used as the authentication mechanism.
- However a form based authentication needs an additional session which is created on the server side. So that's an overhead.
- In the case of a basic authentication, what we do is we keep sending something called a basic authorization header as part of each request. In this case, you don't need a session created for you at the backend.

Adding Basic Authentication Header in Angular

- E.g.

```
let basicAuthHeaderString = 'Basic ' + window.btoa(username + ':' + password);

let headers = new HttpHeaders({
     Authorization: basicAuthHeaderString
})

return this.http.get<HelloWorldBean>(
     `http://localhost:8080/hello-world/path-variable/${name}`,
     {headers}
);
```

- Instead of adding above code for each http call in the service, we can keep the code at one place by implementing HttpInterceptor in Angular so that it will be added automatically for each request. (Make sure you add it to providers array of the module for HTTP_INTERCEPTORS.

  E.g.

```
@Injectable({
  providedIn: 'root'
})
export class HttpIntercepterBasicAuthService implements HttpInterceptor{

  constructor(
    private basicAuthenticationService : BasicAuthenticationService
  ) { }

  intercept(request: HttpRequest<any>, next: HttpHandler){
    let basicAuthHeaderString =
this.basicAuthenticationService.getAuthenticatedToken();
```

```
    let username = this.basicAuthenticationService.getAuthenticatedUser()



    if(basicAuthHeaderString && username) {
      request = request.clone({
        setHeaders : {
            Authorization : basicAuthHeaderString
          }
        })
    }
    return next.handle(request);
  }
}
```

Configure Spring Security to disable CSRF and enable OPTIONS request

- Whenever you add authorization credentials on a GET request, or a POST request, what happens is a **preflight request** is sent. So before sending the actual GET/POST request, an OPTIONS request is sent to the server to check if you have the right permissions.
- **`WebSecurityConfigurerAdapter`** – contains the default security country creation for spring security.
- **Spring Boot code –**

```
@Configuration // to enable spring configuration.
@EnableWebSecurity // to tell Spring Security that this file has security configuration.
public class SpringSecurityConfigurationBasicAuth extends
WebSecurityConfigurerAdapter{

	@Override
	protected void configure(HttpSecurity http) throws Exception {
		http
		// disable CSRF for now.
		.csrf().disable()
		.authorizeRequests()
		// allow OPTIONS request
		.antMatchers(HttpMethod.OPTIONS,"/**").permitAll()
				.anyRequest().authenticated()
				.and()
			// Comment out form-based authentication to disable it
			//.formLogin().and()
			.httpBasic();
	}
}
```

Connecting Spring Security and Spring Boot with JWT

JWT

- JWT library dependency – io.jsonwebtoken:jjwt:0.9.1
- Refer – https://github.com/in28minutes/full-stack-with-angular-and-spring-boot#core-jwt-components
- For each REST request, we need to pass Authorization header with this JWT token as a Bearer.
  E.g. Header as "Authorization", value as "Bearer sdsjdslsldjldjlxxxxxxxxxxxxxx"

Tips and Tricks

- Repositories –
  - https://github.com/sameerbhilare/spring-boot-angular-fullstack-examples
  - https://github.com/sameerbhilare/full-stack-with-angular-and-spring-boot
  - https://github.com/sameerbhilare/Full-Stack-Spring-Boot-Angular
- ng lint – to check for coding standard errors based on rules in tslint.json file.
- ng test – to run unit tests (.spec.ts) created by jasmin framework one by one. Uses karma configuration. Starting point is test.ts file.
- ng e2e – to test entire application by launching it. Uses protractor configuration.
- Pre-flight request is an HTTP OPTIONS request.