



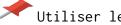
Objectifs de la séance 14

(05-06/03/2022)

Les objectifs de la séance d'aujourd'hui:

Objectif 14.1:

Utiliser Spring Security pour sécuriser les actions Front-End:



Utiliser le starters spring boot : Security starter



L'authentification en utilisant AuthenticationManager Builder



L'autorisation en utilisant Http Security

Dans ce TP on suppose que :

- *∀ous avez réalisé totalement le TP10.*
- *∀ous avez réalisé totalement le TP11.*
- ✓ Vous avez réalisé totalement le TP12.
- ✓ Vous avez réalisé totalement le TP13.

Si ce n'est pas le cas : Faire d'abord le TP10, TP11, et le TP12 d'urgence

Ι. Utilisation de Spring Security Starter

Dans le pom.xml de l'application Front-End ajouter le starter suivant:

```
<dependency>
 <groupId>org.springframework.boot
 <artifactId>spring-boot-starter-security</artifactId>
</dependency>
```

A. AUTHENTICATION : UTILISATION DU LOGIN/PASSWORD POUR ACCÉDER À L'APPLICATION

2. CRÉER UN NOUVEAU PACKAGE "ma.cigma.config.security" ET AJOUTER LA CLASSE DE CONFIGURATION SUIVANTE

```
package ma.cigma.config.security;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.context.annotation.Configuration;
org.springframework.security.config.annotation.authentication.builders.Auth
enticationManagerBuilder;
import
```





```
org.springframework.security.config.annotation.web.configuration.EnableWebS
ecurity;
import
org.springframework.security.config.annotation.web.configuration.WebSecurit
yConfigurerAdapter;
import org.springframework.security.crypto.password.PasswordEncoder;
@Configuration
@EnableWebSecurity
public class WebSecurityConfig extends WebSecurityConfigurerAdapter {
   @Autowired
  PasswordEncoder passwordEncoder;
   // Authentication
   @Override
  public void configure (AuthenticationManagerBuilder
authenticationManagerBuilder) throws Exception {
       authenticationManagerBuilder
         .inMemoryAuthentication()
         .passwordEncoder(passwordEncoder)
.withUser("user").password(passwordEncoder.encode("123456")).roles("USER")
               .and()
.withUser("admin").password(passwordEncoder.encode("123456")).roles("ADMIN
");
   }
```

3. DANS LA TEMPLATE THYMELEAF resources/templates/index-client.html, AJOUTER UN LIEN POUR SE DÉCONNECTER. CI-APRÈS LA NOUVELLE BOOTSTRAP NAVBAR POUR SATISFAIRE CE BESOIN

```
<!DOCTYPE HTML>
<html xmlns:th="http://www.thymeleaf.org"
   xmlns:sec="http://www.thymeleaf.org/extras/spring-security">
<head>
  <meta charset="UTF-8"/>
  <link th:rel="stylesheet"</pre>
         th:href="@{/webjars/bootstrap/4.0.0-2/css/bootstrap.min.css} "/>
  <link rel="stylesheet" type="text/css" th:href="@{/css/style.css}"/>
  <link rel="stylesheet"</pre>
         href="https://use.fontawesome.com/releases/v5.6.3/css/all.css"
integrity="sha384-UHRtZLI+pbxtHCWp1t77Bi1L4ZtiqrqD80Kn4Z8NTSRyMA2Fd33n5dQ
81WUE00s/" crossorigin="anonymous">
</head>
<body>
<!-- Navigation -->
<nav class="navbar navbar-expand-lg navbar-dark bg-dark static-top">
```





```
<div class="container">
      <a class="navbar-brand" href="/">Front Application</a>
      <button class="navbar-toggler" type="button"</pre>
             data-toggle="collapse" data-target="#navbarResponsive"
             aria-controls="navbarResponsive"
             aria-expanded="false" aria-label="Toggle navigation">
          <span class="navbar-toggler-icon"></span>
      </button>
      <div class="collapse navbar-collapse" id="navbarResponsive">
          sec:authorize="isAuthenticated()" class="nav-item">
                 <a class="btn btn-outline-light"
th:href="@{/logout}">Logout</a>
             <a class="nav-link" href="#">Home
                     <span class="sr-only">(current)</span>
                 </a>
             <a class="nav-link" href="#">About</a>
             <a class="nav-link" href="#">Services</a>
             <a class="nav-link" href="#">Contact</a>
             </div>
  </div>
</nav>
<div class="container">
  <div class="row">
      <div class="col-sm-8">
          <form action="#" th:action="@{/add-client}"</pre>
               th:object="${abc}" method="post">
             <div class="form-group">
                 <input type="text" th:field="*{id}" id="id"</pre>
                        placeholder="Id" readonly class="form-control">
             </div>
             <div class="form-group">
                 <label for="name">Name</label>
                 <input type="text" th:field="*{name}" id="name"</pre>
                        placeholder="Name" class="form-control">
             </div>
             <button type="submit" class="btn btn-success">Add
                 Client</button>
          </form>
      </div>
  </div>
  <!-- row end-->
```



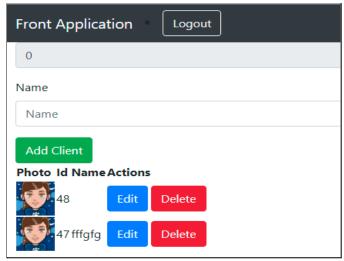


```
<div class="row">
     <div class="col-lg-12">
        <thead>
           >
              Photo
              Id
              Name
              Actions
           </thead>
           >
                 <div class="event-img">
                    <img
src="https://bootdey.com/img/Content/avatar/avatar3.png" alt=""/>
                 </div>
              <a class="btn btn-primary"</pre>
                    th:href="@{|/show-client/${c.id}|}">
                    <i class="fa fa-edit fa-lg"></i> Edit</a>
                 <a class="btn btn-danger"
                    th:href="@{|/delete-client/${c.id}|}"><i
class="fa fa-trash fa-lg"></i> Delete</a>
              </div>
     <!-- /col end-->
  </div>
</div>
<script th:src="@{/webjars/jquery/3.0.0/jquery.min.js}"></script>
th:src="@{/webjars/popper.js/1.12.9-1/umd/popper.min.js}"></script>
<script
th:src="@{/webjars/bootstrap/4.0.0-2/js/bootstrap.min.js}"></script>
</body>
```

- 4. Exécuter et tester l'accès à l'application par le login et le mot de passe configuré dans la classe Web SecurityConfig
- 5. TESTER LE LIEN LOGOUT







B. AUTHORIZATIONS : AFFECTATION DES ACTIONS AUX RÔLES: ADMIN ET USER

Nous supposons dans cette partie que notre application possède deux profils

ADMIN : PEUT

★ Afficher La LISTE DES CLIENTS

★ SUPPRIMER DES CLIENTS

USER : PEUT

* Afficher La LISTE DES CLIENTS

★ AJOUTER DES CLIENTS

★ Modifier des clients

* Afficher un client

1. AJOUTER À LA CLASSE Web SecurityConfig LA MÉTHODE SUIVANTE:

```
@Override
protected void configure(HttpSecurity http) throws Exception {
   http
            .authorizeRequests()
            .antMatchers(
                    "/add-client", "/show-client/**"
                   )
            .hasRole("USER")
            .antMatchers(
                    "/delete-client/**"
            .hasRole("ADMIN")
            .antMatchers(
                    "/clients","/"
            .permitAll()
            .anyRequest()
            .authenticated()
            .and()
            .formLogin()
```





.defaultSuccessUrl("/clients");

2. Se connecter à l'application par login => user et password => 123456 AJOUTER, MODIFIER ET SUPPRIMER UN CLIENT PAR CE ROLE : "USER" REMARQUER QUE LE ROLE "USER" PEUT AJOUTER ET MODIFIER UN CLIENT MAIS

LA SUPPRESSION D'UN DANS L'ERREUR 403 : FORBIDDEN





(i) localhost:8089/delete-client/48

Whitelabel Error Page

This application has no explicit mapping for /error, so you are seeing this as a fallback.

Sat Mar 05 19:49:42 WEST 2022

There was an unexpected error (type=Forbidden, status=403).

3. SE CONNECTER À L'APPLICATION PAR LOGIN => admin ET PASSWORD => 123456 AJOUTER, MODIFIER ET SUPPRIMER UN CLIENT PAR CE ROLE : "USER" REMARQUER QUE LE ROLE "ADMIN" PEUT SUPPRIMER UN CLIENT MAIS

L'AJOUT D'UN NOUVEAU CLIENT DANS L'ERREUR 403 : FORBIDDEN







← → C ① localhost:8089/add-client

Whitelabel Error Page

This application has no explicit mapping for /error, so you are seeing this as a fallback.

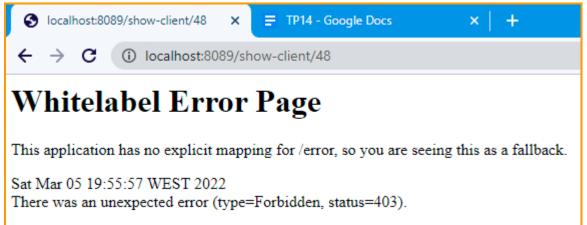
Sat Mar 05 19:52:11 WEST 2022

There was an unexpected error (type=Forbidden, status=403).

La modification d'un nouveau client dans aussi l'erreur 403 : Forbidden







II. CREATION DES TOKENS JWT

4. AJOUTER L'API JJWT à VOTRE POM.XML

5. Créer une classe pour la génération des tokens Jwt Appeler la JwtDemo

```
On
package ma.cigma.config.security;

import io.jsonwebtoken.Claims;
import io.jsonwebtoken.JwtBuilder;
import io.jsonwebtoken.Jwts;
import io.jsonwebtoken.SignatureAlgorithm;

import javax.crypto.spec.SecretKeySpec;
import java.security.Key;
import java.util.Base64;
import java.util.Date;

public class JWTDemo {

// The secret key. This should be in a property file NOT under source
```





```
// control and not hard coded in real life. We're putting it here for
   // simplicity.
  private static String SECRET KEY =
"oeRaYY7Wo24sDqKSX3IM9ASGmdGPmkTd9jo1QTy4b7P9Ze";
  public static void main(String[] args) {
      String token=createJWT( "id", "issuer", "subject", 100000001);
      System.out.println(token);
   //Sample method to construct a JWT
  public static String createJWT(String id, String issuer, String
subject, long ttlMillis) {
       //The JWT signature algorithm we will be using to sign the token
       SignatureAlgorithm signatureAlgorithm = SignatureAlgorithm.HS256;
      long nowMillis = System.currentTimeMillis();
      Date now = new Date(nowMillis);
       //We will sign our JWT with our ApiKey secret
      byte[] apiKeySecretBytes = SECRET KEY.getBytes();
      Key signingKey = new SecretKeySpec(apiKeySecretBytes,
signatureAlgorithm.getJcaName());
       //Let's set the JWT Claims
       JwtBuilder builder = Jwts.builder().setId(id)
               .setIssuedAt(now)
               .setSubject(subject)
               .setIssuer(issuer)
               .signWith(signatureAlgorithm, signingKey);
       //if it has been specified, let's add the expiration
       if (ttlMillis >= 0) {
          long expMillis = nowMillis + ttlMillis;
          Date exp = new Date(expMillis);
          builder.setExpiration(exp);
       }
       //Builds the JWT and serializes it to a compact, URL-safe string
      return builder.compact();
  }
```

6. Executer la classe JwtDemo pour voir le token Jwt selon la structure suivante:

<mark>eyJhBGciOiJIUzI1NiJ9</mark>.eyJqpGkiOiJpZCIsImLhDCI6MTY0NjUxMjE5OSwic3ViIjoic3Vi amVjdCIsImLzcyI6ImLzc3VLciIsImV4cCI6MTY0NjUyMjE5OX0.aJLSSQh-ooDv1W4LH39 vwWVA6LBno1epTsZfWznrgCY

7. AJOUTER UNE METHODE POUR DECODER LE TOKEN JWT:

```
public static Claims decodeJWT(String jwt) {
    //This line will throw an exception if it is not a signed
JWS (as expected)
```





8. Appeler cette methode dans main

```
public static void main(String[] args) {
   String token=createJWT( "id", "issuer", "subject",
100000001);
   System.out.println(token);
   Claims claims=decodeJWT(token);
   System.out.println(claims);
}
```

9. Voir le resultat dans la console

```
jti=id,
iat=1646512437,
sub=subject,
iss=issuer,
exp=1646522437
}
```