

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 :

- ☒ ~~Vous avez réalisé totalement le TP10.~~
- ☒ ~~Vous 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

I. UTILISATION DE SPRING SECURITY STARTER

1. DANS LE POM.XML DE L'APPLICATION FRONT-END AJOUTER LE STARTER SUIVANT:

```
<dependency>  
  <groupId>org.springframework.boot</groupId>  
  <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;  
import org.springframework.security.config.annotation.authentication.builders.AuthenticationManagerBuilder;  
import
```

```
org.springframework.security.config.annotation.web.configuration.EnableWebSecurity;
import
org.springframework.security.config.annotation.web.configuration.WebSecurityConfigurerAdapter;
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">
</html>

<head>
    <meta charset="UTF-8"/>
    <link th:rel="stylesheet"
          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"
          href="https://use.fontawesome.com/releases/v5.6.3/css/all.css"
          integrity="sha384-UHRtZLI+pbxtHCWp1t77BillL4ZtigrqD80Kn4Z8NTSRyMA2Fd33n5dQ8lWUE00s/" 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"
    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">
    <ul class="navbar-nav ml-auto">
      <li sec:authorize="isAuthenticated()" class="nav-item">
        <a class="btn btn-outline-light"
th:href="@{/logout}">Logout</a>
      </li>
      <li class="nav-item active">
        <a class="nav-link" href="#">Home
          <span class="sr-only"> (current) </span>
        </a>
      </li>
      <li class="nav-item">
        <a class="nav-link" href="#">About</a>
      </li>
      <li class="nav-item">
        <a class="nav-link" href="#">Services</a>
      </li>
      <li class="nav-item">
        <a class="nav-link" href="#">Contact</a>
      </li>
    </ul>
  </div>
</div>
</nav>

<div class="container">
  <div class="row">
    <div class="col-sm-8">
      <form action="#" th:action="@{/add-client}"
        th:object="${abc}" method="post">
        <div class="form-group">
          <input type="text" th:field="*{id}" id="id"
            placeholder="Id" readonly class="form-control">
        </div>
        <div class="form-group">
          <label for="name">Name</label>
          <input type="text" th:field="*{name}" id="name"
            placeholder="Name" class="form-control">
        </div>
        <button type="submit" class="btn btn-success">Add
          Client</button>
      </form>
    </div>
  </div>
<!-- row end -->
</div>
```

```
<div class="row">
  <div class="col-lg-12">
    <table>
      <thead>
        <tr>
          <th scope="col">Photo</th>
          <th scope="col">Id</th>
          <th scope="col">Name</th>
          <th scope="col">Actions</th>
        </tr>
      </thead>
      <tbody>
        <tr th:each="c : ${key1}">
          <td>
            <div class="event-img">
              
            </div>
          </td>
          <td th:text="${c.id}"></td>
          <td th:text="${c.name}"></td>
          <td>
            <a class="btn btn-primary"
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>
          </td>
        </tr>
      </tbody>
    </table>
  </div>
<!-- /col end-->
</div>
</div>
<script th:src="@{/webjars/jquery/3.0.0/jquery.min.js}"></script>
<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



Front Application
Logout

0

Name

Name

Add Client

Photo	Id	Name	Actions
	48		Edit Delete
	47	fffgfg	Edit Delete

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

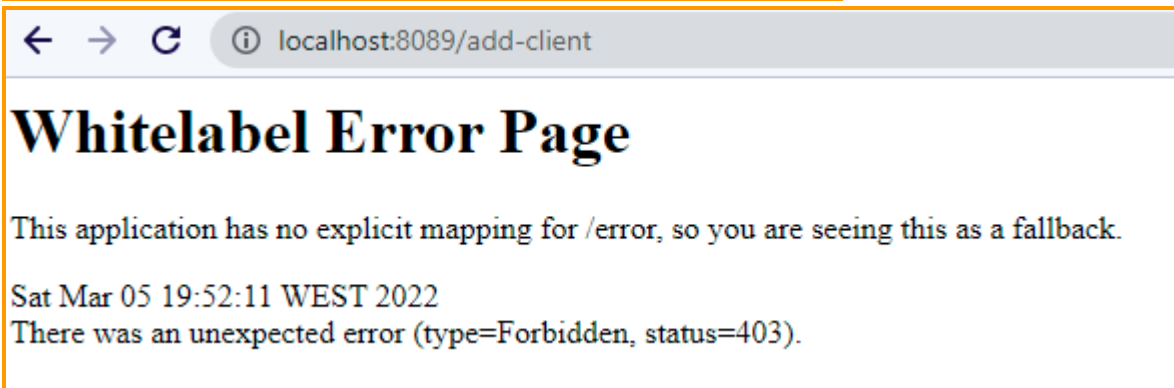


3. SE CONNECTER À L'APPLICATION PAR LOGIN => admin ET PASSWORD => 123456

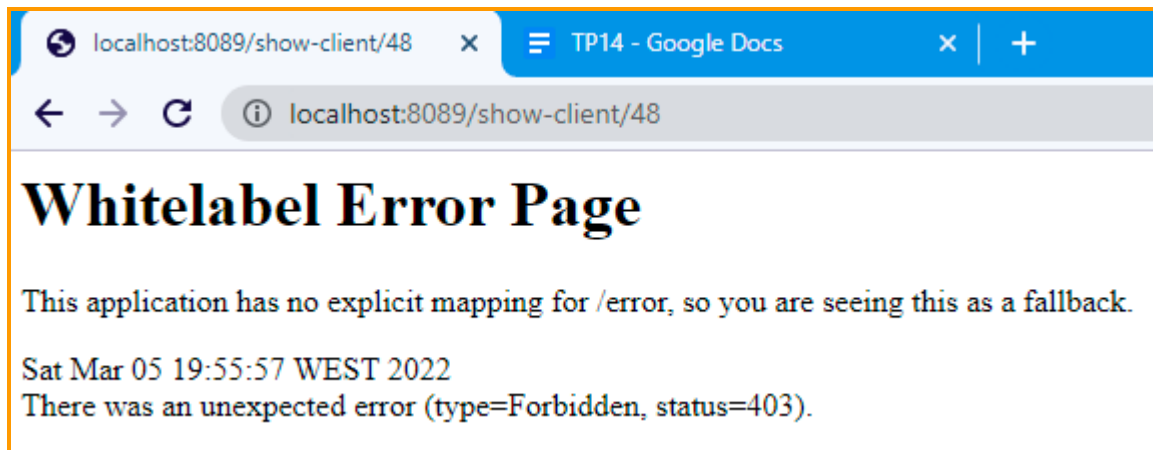
AJOUTER, MODIFIER ET SUPPRIMER UN CLIENT PAR CE ROLE : "User"

REMARQUER QUE LE ROLE "ADMIN" PEUT SUPPRIMER UN CLIENT MAIS

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



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

```
<!-- https://mvnrepository.com/artifact/io.jsonwebtoken/jjwt -->
<dependency>
  <groupId>io.jsonwebtoken</groupId>
  <artifactId>jjwt</artifactId>
  <version>0.9.1</version>
</dependency>
<dependency>
  <groupId>javax.xml.bind</groupId>
  <artifactId>jaxb-api</artifactId>
  <version>2.3.0</version>
</dependency>
```

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
```

6. EXECUTER LA CLASSE `JWTDEMO` POUR VOIR LE TOKEN `JWT` SELON LA STRUCTURE SUIVANTE:

7. AJOUTER UNE METHODE POUR DECODER LE TOKEN JWT:

Dossier des travaux pratiques. Module 3 : JEE and Fwks .Années scolaire 2021/2022. Niveau : Licence FST Settat

8/9


```
Claims claims = Jwts.parser()  
    .setSigningKey(SECRET_KEY.getBytes())  
    .parseClaimsJws(jwt).getBody();  
return claims;  
}
```

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  
}
```