
 <i>Thymeleaf</i>	Atelier N°6 Spring Boot : Spring Security : Les fondamentaux	
--	---	---

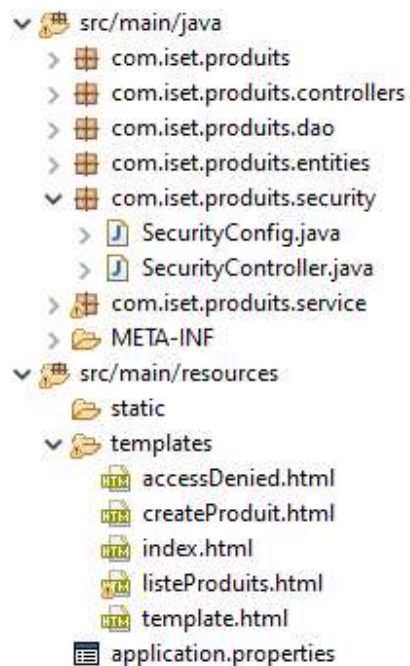
Objectifs	Temps alloué	Outils
<ul style="list-style-type: none"> • Ajouter Spring Security au projet • L'authentification basique, • L'authentification en utilisant des utilisateurs InMemory, • Sécuriser l'accès à l'application selon les rôles, • Crypter le password, • Contextualisation du menu selon l'utilisateur connecté. 	3 Heures	<ul style="list-style-type: none"> • STS 4 • MySQL Server

I. Travail demandé

Dans cet atelier on va ajouter la couche sécurité à notre application de gestion des produits en appliquant les règles de gestion suivantes :

- Toutes les opérations de l'application nécessitent une authentification,
- Les utilisateurs de l'application peuvent avoir les rôles suivants :
 - ADMIN : a le droit de faire toutes les opérations,
 - AGENT : a seulement le droit de consulter ou d'ajouter les produits, il ne peut ni les modifier ni les supprimer,
 - USER : a seulement le droit de consulter les produits.

Structure du projet:



Etape 1 : Ajouter Spring Security au projet

1. Ajouter la dépendance Spring security au fichier pom.xml :

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

2. Créer dans le dossier templates le fichier index.html :

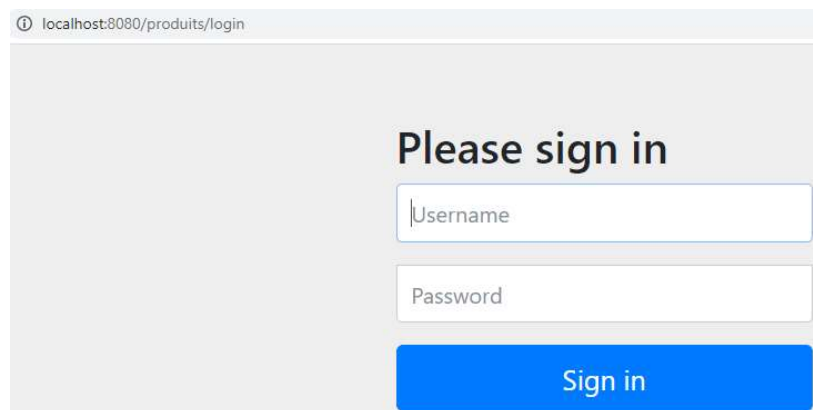
```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org"
      xmlns:layout="http://www.ultraq.net.nz/thymeleaf/layout"
      layout:decorator="template">

<link rel="stylesheet" type="text/css"
      href="/webjars/bootstrap/4.3.1/css/bootstrap.min.css" />
<head>
<meta charset="utf-8">
<title>Gestion des Produits</title>
</head>
<body>
  <div layout:fragment="Mycontent"></div>
</body>
```

</html>

3. Redémarrer l'application et tester :

<http://localhost:8080/produits/>



Connectez-vous en tant qu'user, le mot de passe vous le trouvez dans la console au démarrage de l'application :

```
2022-04-12 12:13:36.138 INFO 12996 --- [ restartedMain] .s.s.UserDetailsServiceAuto
Using generated security password: 752ca8c0-5922-4f7e-97ce-44645441116b
2022-04-12 12:13:36.300 INFO 12996 --- [ restartedMain] o.s.s.web.DefaultSecurityFi
2022-04-12 12:13:36.421 INFO 12996 --- [ restartedMain] o.s.b.d.a.OptionalLiveReloa
2022-04-12 12:13:36.470 INFO 12996 --- [ restartedMain] o.s.b.w.embedded.tomcat.Tom
```

NB : Le mot de passe change à chaque démarrage.

Etape2 : L'authentification basique

On se propose à présent de créer les méthodes nécessaires pour interroger les données des produits en fournissant le nom du produit comme critère de recherche

1. Ajouter des utilisateurs dans le fichier application.properties pour cela éditer le fichier application.properties et ajouter les lignes suivantes :

```
spring.security.user.name=admin
spring.security.user.password=123
```

2. puis testez, de nouveau le lien : <http://localhost:8080/produits/>

Etape 3 : L'authentification en utilisant des utilisateurs InMemory

1. Créer la classe SecurityConfig, placez la dans le package security :

```

@EnableWebSecurity
public class SecurityConfig extends WebSecurityConfigurerAdapter {

    @Override
    protected void configure(AuthenticationManagerBuilder auth) throws Exception {
        // TODO Auto-generated method stub

        auth.inMemoryAuthentication().withUser("admin").password("{noop}123").roles("ADMIN");

        auth.inMemoryAuthentication().withUser("Najla").password("{noop}123").roles("AGENT", "USER");

        auth.inMemoryAuthentication().withUser("user1").password("{noop}123").roles("USER");
    }

    @Override
    protected void configure(HttpSecurity http) throws Exception {
        // TODO Auto-generated method stub
        http.authorizeRequests().anyRequest().authenticated();
        http.formLogin();
    }
}

```

2. Commentez les lignes suivantes dans le fichier **application.properties**

```

#spring.security.user.name=admin
#spring.security.user.password=123

```

3. Avec un navigateur, authentifiez-vous, puis testez, de nouveau le lien :
<http://localhost:8080/produits/login>

Etape4 : Sécuriser l'accès à l'application selon les rôles

On se propose ici d'autoriser l'accès aux méthodes de la classe ProduitController selon le rôle de l'utilisateur authentifié. Et ce en appliquant les règles de gestion annoncées au début de l'atelier :

- Les utilisateurs de l'application peuvent avoir les rôles suivants :
 - ADMIN : a le droit de faire toutes les opérations,
 - AGENT : a seulement le droit de consulter ou d'ajouter les produits, il ne peut ni les modifier ni les supprimer,
 - USER : a seulement le droit de consulter les produits.

Modifier la méthode `configure(HttpSecurity http)` comme suit :

```

http.authorizeRequests().antMatchers("/showCreate", "/saveProduit").hasAnyRole("ADMIN", "AGENT");

    http.authorizeRequests().antMatchers("/ListeProduits")
        .hasAnyRole("ADMIN", "AGENT", "USER");

    http.authorizeRequests()
        .antMatchers("/supprimerProduit", "/modifierProduit", "/updateProduit")
        .hasAnyRole("ADMIN");

    http.authorizeRequests().anyRequest().authenticated();
    http.formLogin();

```

Etape5 : Personnaliser la page Access Denied

1. Créer la classe contrôleur SecurityController dans le package security :

```

@Controller
public class SecurityController {
    @GetMapping("/accessDenied")
    public String geterror() {
        return "accessDenied";
    }

    @PostMapping("/accessDenied")
    public String posterror() {
        return "accessDenied";
    }
}

```

2. Créer la page accessDenied.html :

```

<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org"
      xmlns:layout="http://www.ultraq.net.nz/thymeleaf/layout"
      layout:decorator="template">
<head>
<meta charset="utf-8">
<title>Error</title>
</head>
<body>
    <div layout:fragment="Mycontent">
        <div class="container">
            <h3 class="text-danger">Vous n'êtes pas autorisé</h3>
        </div>
    </div>
</body>
</html>

```

3. Ajouter la ligne suivante à la méthode `configure(HttpSecurity http)`

```
http.exceptionHandling().accessDeniedPage("/accessDenied");
```

4. Testez l'application

Etape6 : Ajouter crypter le Password

1. Ajouter, à la classe `SecurityConfig`, la méthode `passwordEncoder` :

```
@Bean
public PasswordEncoder passwordEncoder () {
    return new BCryptPasswordEncoder();
}
```

2. Modifier la méthode `configure` :

```
@Override
protected void configure(AuthenticationManagerBuilder auth) throws Exception {
    // TODO Auto-generated method stub
    PasswordEncoder passwordEncoder = passwordEncoder ();
    auth.inMemoryAuthentication().withUser("admin")
        .password(passwordEncoder.encode("123")).roles("ADMIN");
    auth.inMemoryAuthentication().withUser("Najla")
        .password(passwordEncoder.encode("123")).roles("AGENT", "USER");
    auth.inMemoryAuthentication().withUser("user1")
        .password(passwordEncoder.encode("123")).roles("USER");
}
```

Etape 7 : Contextualisation du menu selon l'utilisateur connecté

1. Ajouter la dépendance qui permet à Thymeleaf de communiquer avec Spring Security :

```
<dependency>
    <groupId>org.thymeleaf.extras</groupId>
    <artifactId>thymeleaf-extras-springsecurity5</artifactId>
</dependency>
```

2. Ajouter le namespace xml suivant au fichier **template.html**

```
xmlns:sec=https://www.thymeleaf.org/thymeleaf-extras-springsecurity5
```

3. Modifier le fichier `template.html` comme suit :

```
<ul class="navbar-nav ml-auto">
    <li class="nav-item dropdown">
        <a class="nav-link dropdown-toggle" href="#" id="navbardrop"
            data-toggle="dropdown">
            <span sec:authentication="name"></span>
        </a>
```

```

<div class="dropdown-menu">
  <a sec:authorize="!isAuthenticated()" class="dropdown-item"
    th:href="@{login}">login</a>
  <a sec:authorize="isAuthenticated()" class="dropdown-item"
th:href="@{logout}">logout</a>
  <a class="dropdown-item" href="@{login}">Profile</a>
</div></li>
</ul>

```

Etape 7 : Cacher les boutons Supprimer et Editer pour les utilisateurs non autorisés

1. Modifier le fichier listeProduits.html comme suit :

```

<td sec:authorize="hasRole('ADMIN')" ><a onclick="return confirm('Etes-vous sûr ?')"
class="btn btn-danger" th:href="@{supprimerProduit(id=${p.idProduit},
page=${currentPage},size=${size})}">Supprimer</a></td>
<td sec:authorize="hasRole('ADMIN')" ><a class="btn btn-success"
th:href="@{modifierProduit(id=${p.idProduit})}">Editer</a></td>

```

2. Modifier également le fichier template.html pour cacher le menu « Ajouter » aux utilisateurs n'ayant pas le rôle ADMIN, comme suit :

```

<!-- Dropdown -->
<li class="nav-item dropdown"><a class="nav-link dropdown-toggle" href="#"
id="navbardrop" data-toggle="dropdown"> Produits </a>
<div class="dropdown-menu">
  <a sec:authorize="hasRole('ADMIN')" class="dropdown-item"
th:href="@{showCreate}">Ajouter</a>
  <a class="dropdown-item" th:href="@{ListeProduits}">Lister</a>
</div></li>

```