Université Nord Américaine privée
Institut International Technologie

Atelier Framework Spring-06
Spring Security
Partie 01- Approche "Stateful"

Objectifs

Réaliser des fonctionnalités web sécurisées avec Spring Security (Approche «Stateful»)

- **Mettre en place une sécurité de base**
  - Utiliser la configuration par défaut de Spring Security
  - Utiliser des paramètres d'authentification personnalisés
- **Personnaliser la configuration de la sécurité en mémoire et en BD**
  - Définir les utilisateurs
  - Protéger les ressources (définir les régles d'accès)
  - Définir une page d'authentification
  - Gérer les rôles
  - Récupérer les informations de l'utilisateur courant
  - Gérer la déconnexion
  - Configurer Bootstrap

A. Mise en place d'une sécurité de base (Configuration par défaut)

1. Prendre une copie du projet «**jpa-spring-boot-ServiceWeb-REST»** et le nommer «**jpa-spring-boot-ServiceWeb-REST-Security-Partie-01-Stateful**».
2. Référencer dans le fichier « **application.properties** » vers une nouvelle base de données «**springJpaSWSecurity**».
3. Rendre l'option « **spring.jpa.hibernate.ddl-auto**» à la valeur «**create**»

```
spring.jpa.hibernate.ddl-auto=create
```

4. Lancer l'exécution du projet et tester avec l'outil les requêtes suivantes :
   - **GET** : http://localhost:8080/produits/
   - **POST** : http://localhost:8080/produits/
   - **PUT** : http://localhost:8080/produits/
   - **DELETE** : http://localhost:8080/produits/

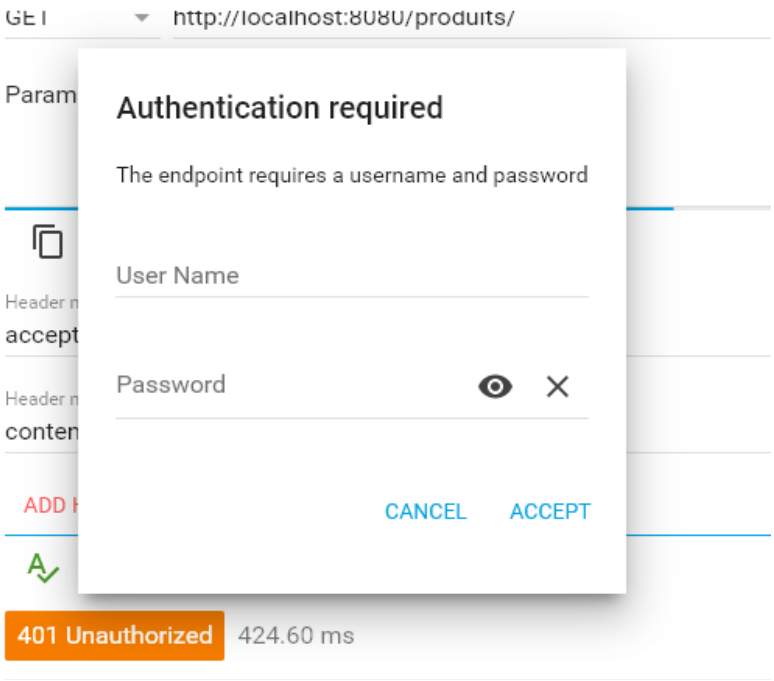
5. Ajouter la dépendance de « **Spring Security** » suivante dans le fichier « **pom.xml** » puis enregistrer :

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

6. Relancer l'exécution du projet et tester l'url suivante :

   - **GET** : http://localhost:8080/produits/

   Remarquer que l'accès à cette URL est **non autorisé** et une boite de dialogue qui s'affiche pour demander les paramètres de connexion :



7. Pour s'authentifier :

   a) Saisir la valeur «**user**» pour le champ « **User Name** »

   b) Copier le mot de passe de sécurité généré par Spring Security et la coller dans le champ « **Password** ». Aller à la console de « eclipse » pour récupérer le mot de passe générée :

```
Using generated security password: 94eac99e-2ed1-4f46-b898-31617091040b
```

8. Il est possible aussi de spécifier dans le fichier « **application.properties** » les paramètres d'authentification pour la configuration par défaut. Par exemple, donner la valeur «**admin**» à « **User Name** » et la valeur «**admin**» à « **Password** ».

Pour ceci, ajouter les lignes suivantes au début du fichier « **application.properties** » :

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

Redémarrer l'application et remarquer la non génération d'un mot de passe automatique dans la console de « **eclipse** ». Il suffit, maintenant, d'utiliser les valeurs «**admin**» «**admin**».

B. Personnalisation de la configuration de sécurité (Gérer des utilisateurs en mémoire)

9. Créer, dans le package «**spring.jpa**», une classe «**SecurityConfig**» qui :
   - Présente l'annotation **@Configuration** pour être prise en considération au démarrage.
   - Définit deux utilisateurs :
     - Utilisateur (name=**admin**, password=**admin**) ayant les rôles (**ADMIN** et **USER**).
     - Utilisateur (name=**user**, password=**user**) ayant uniquement le rôle (**USER**).
   - Impose l'authentification pour toutes les requêtes

Voici le code de la classe « **SecurityConfig** » :

```
package spring.jpa;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.security.config.annotation.authentication.builders.AuthenticationManagerBuilder;
import org.springframework.security.config.annotation.web.builders.HttpSecurity;
import org.springframework.security.web.SecurityFilterChain;

@Configuration
public class SecurityConfig
{
	@Autowired // pour l'injection de dépendances
	//Définir les utilisateurs et leurs rôles
	public void globalConfig(AuthenticationManagerBuilder auth) throws Exception {

auth.inMemoryAuthentication().withUser("admin").password("{noop}admin").roles("ADMIN","USER");
auth.inMemoryAuthentication().withUser("user").password("{noop}user").roles("USER");

	}



	@Bean
	public SecurityFilterChain filterChain(HttpSecurity http) throws Exception
```

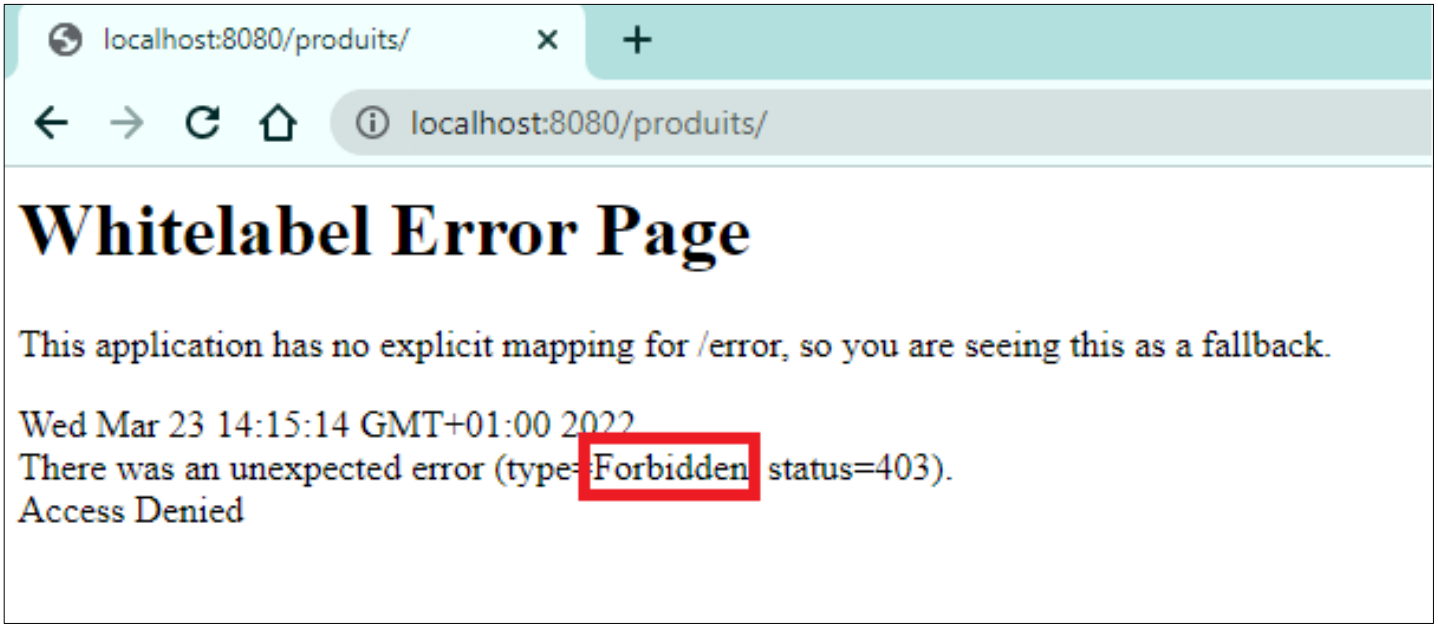
```
        {
                http.authorizeHttpRequests()
                        .anyRequest()
                                .authenticated();

            return http.build();
        }
}
```

10. Mettre en commentaire la configuration basique de l'utilisateur dans le fichier « **application.properties** » :

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

11. Redémarrer le projet et exécuter, dans le navigateur web, l'url :
GET : http://localhost:8080/produits/

Remarquer que l'accès est interdit (sans authentification) :



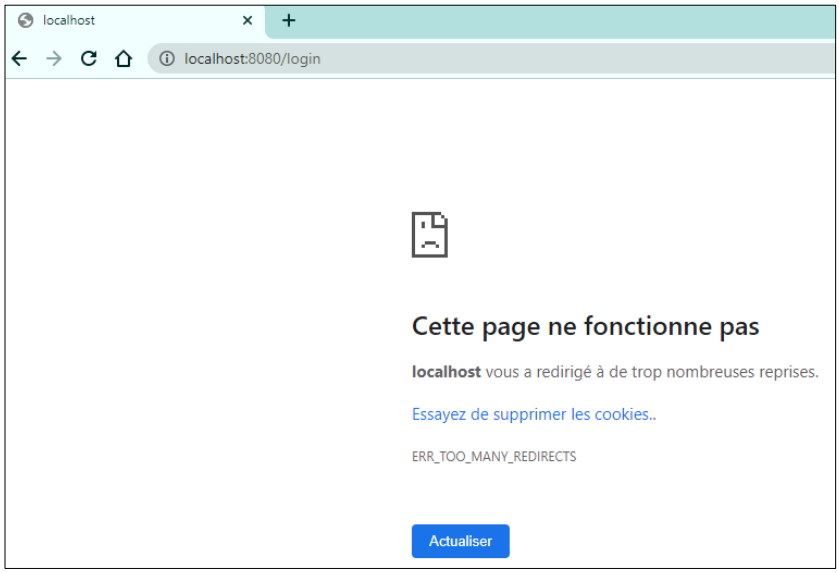
12. Il est nécessaire de spécifier le formulaire d'authentification (via l'URL « **/login**»). Pour ceci, modifier la méthode « **filterChain**» de la classe « **SecurityConfig** » comme suit :

```
@Bean
	public SecurityFilterChain filterChain(HttpSecurity http) throws
Exception {
		http
			.authorizeRequests()
				.anyRequest()
					.authenticated() //il faut s'authentifier pour accéder à toutes les URLs
						.and()
					.formLogin()
						.loginPage("/login");//page d'authentification (template)

	  return http.build();
	}
```

Redémarrer le projet et exécuter, dans le navigateur web, l'url suivante:

GET : http://localhost:8080/produits/

L'erreur suivante s'affiche à cause d'une non autorisation :



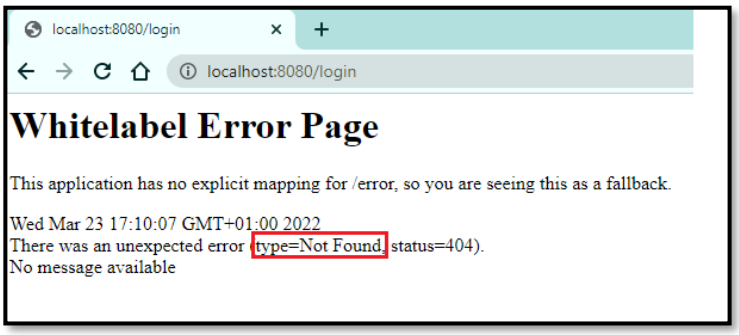
13. Ajouter la permission (**permitAll() ;**) comme suit dans la méthode « **filterChain**» de la classe «**SecurityConfig** » comme suit :

```
@Bean
	public SecurityFilterChain filterChain(HttpSecurity http) throws
Exception {
		http
			.authorizeRequests()
				.anyRequest()
					.authenticated() //il faut s'authentifier pour accéder à toutes les URLs
						.and()
					.formLogin()
						.loginPage("/login") //page d'authentification (template)
							.permitAll() ;
	  return http.build();
	}
```

14. Redémarrer le projet et exécuter, dans le navigateur web, l'url suivante :

GET : http://localhost:8080/produits/

La ressource est non trouvée :



15. Créer , sous « **src/main/resources/templates** », la page « **login.html** » (qui contient un formulaire de connexion) :

```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org">
<head>
<meta charset="utf-8" />
<title>Authentification</title>
</head>
<body>
                    <div>
                         <form th:action="@{/login}" method="POST" >

                              <div>
                                   <label>Login:</label>
                                   <input type="text" name="username"/>
                              </div>
                              <div>
                                   <label>Password:</label>
                                   <input type="password" name="password"/>
                              </div>
                              <div>
                                   <input  type="submit">S'inscrire</button>
                              </div>
                         </form>
                    </div>
</body>
</html>
```

Explication :

- Suite à la soumission du formulaire, Spring Security récupère les valeurs des deux paramètres « **username** » et « **password** » et réalise la validation
- La méthode du formulaire doit être « **POST** »
- L'action doit être « **/login** »
- Le paramètre du nom de l'utilisateur doit être « **username** »
- Le paramètre du mot de passe doit être « **password** »

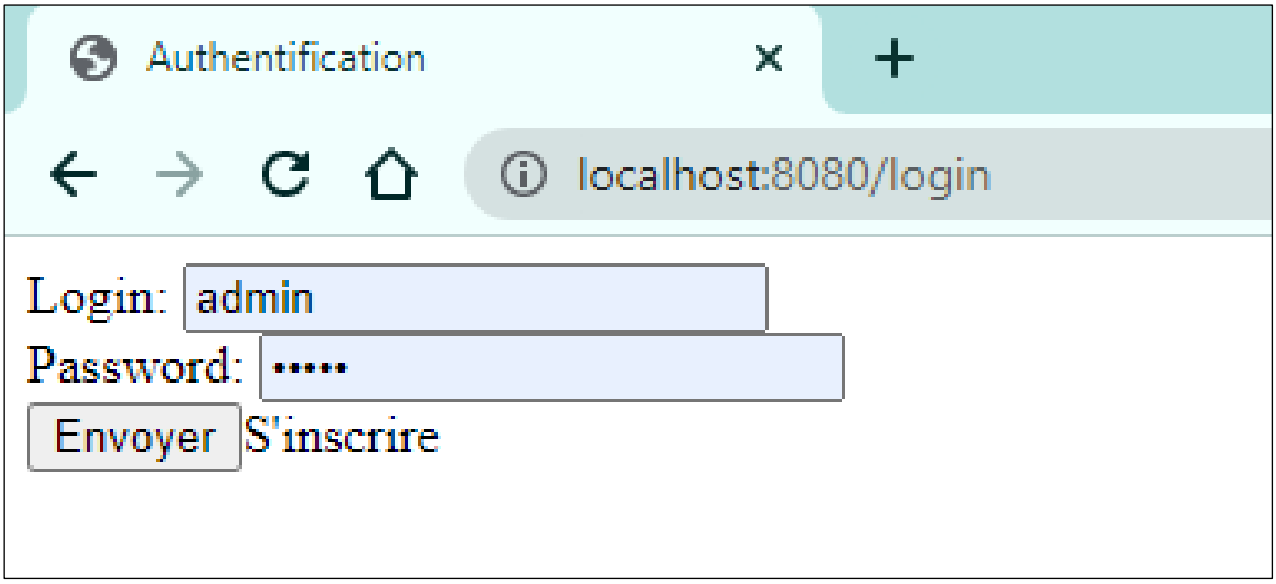
16. Il est nécessaire d'établir le mapping entre l'url « **/login** » et la page « **login.html** ». Il existe deux solutions :

a) Utiliser un contrôleur :

- Editer la classe suivante dans le package « **spring.jpa.controller** » :

```
package spring.jpa.controller;

import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;

@Controller
public class WebController {
    @RequestMapping(value="/login",method=RequestMethod.GET)
      public String login()
      {
         return "login";
      }
}
```

- Redémarrer le projet et exécuter, dans le navigateur web, l'url :
  GET : http://localhost:8080/produits/



- Saisir les paramètres de connexion pour un administrateur et visualiser le résultat :

b) Utiliser une classe de configuaration :

- Commenter l'annotation **@Controller** dans la classe « **WebController** » pour désactiver la première solution.
- Créer la classe de configuration «**WebMVCConfig**» dans le package « **spring.jpa** » :

```
package spring.jpa;

import org.springframework.context.annotation.Configuration;
import org.springframework.web.servlet.config.annotation.ViewControllerRegistry;
import org.springframework.web.servlet.config.annotation.WebMvcConfigurer;

@Configuration
public class WebMVCConfig implements WebMvcConfigurer{
    @Override
```

```
    public void addViewControllers(ViewControllerRegistry registry)
{
        registry.addViewController("/login").setViewName("login");
    }
}
```

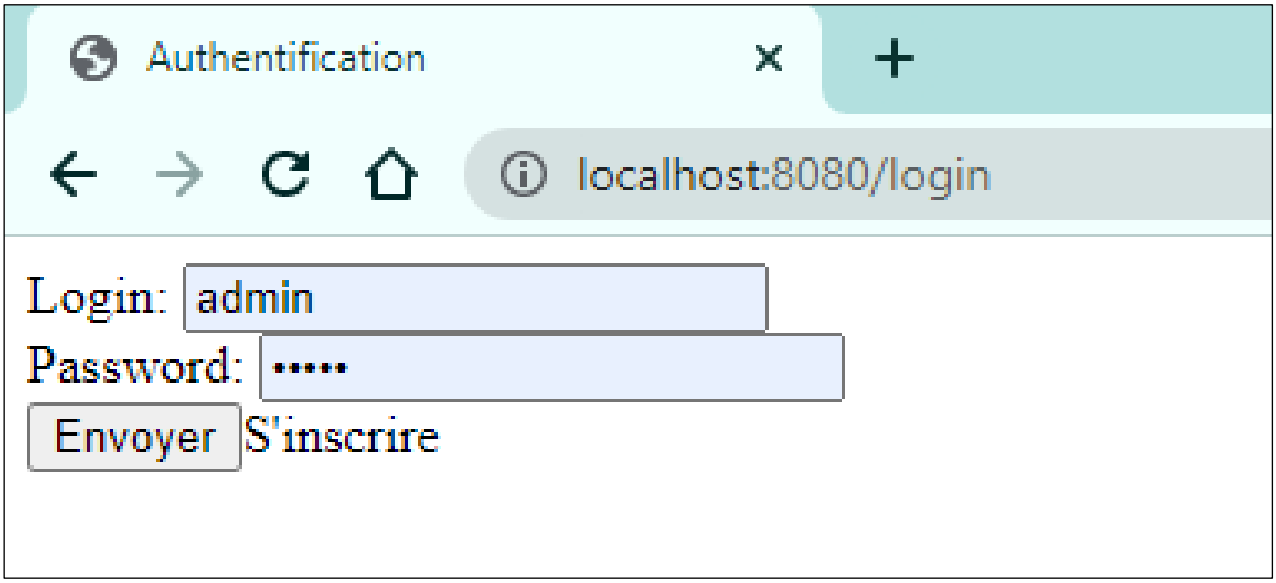
Explication :

Cette classe nouvelle classe :

- ✓ implémente l'interface « **WebMvcConfigurer** »
- ✓ Redéfinie la méthode « **addViewControllers** » pour définir un mapping entre une URL « **/login** » et une template HTML nommée « **login** »

- Redémarrer le projet et exécuter, dans le navigateur web, l'url :

GET : http://localhost:8080/produits/



- Saisir les paramètres de connexion pour un administrateur et visualiser le résultat (la liste de tous les produits)

17. Redémarrer le projet et exécuter, dans le navigateur web, l'url :

GET : http://localhost:8080/produits/

- Afficher le code source du formulaire d'authentification et remarquer que Spring Security injecte du code source qui permet d'insérer un paramètre caché (nommé « _csrf ») qui joue le rôle d'un jeton qui sera envoyé par toutes les requêtes qui suivent l'opération d'authentification. Si une requête en dehors de cette application était reçue et n'ayant pas ce jeton, elle serait refusée :

```
<!DOCTYPE html>
<html>
▶ <head>…</head>
▼ <body>
  ▼ <div>
    ▼ <form action="/login" method="POST">
        <input type="hidden" name="_csrf" value="183e8a13-34cc-4732-986d-aa2a156e8422"> == $0
      ▼ <div>
          <label>Login:</label>
          <input type="text" name="username">
        </div>
      ▼ <div>
          <label>Password:</label>
          <input type="password" name="password">
        </div>
      </form>
    </div>
  ▼ <div>
      <input type="submit">
      "S'inscrire "
    </div>
  </body>
```

C'est une solution contre les attaques de type ***cross-site request forgery (csrf)***

- Désactiver la protection contre CSRF en modifiant la méthode « **`filterChain`**» de la classe «**SecurityConfig** » comme suit :

```
@Bean
        public SecurityFilterChain filterChain(HttpSecurity http) throws
Exception {
                    http
                            .csrf().disable() //désactiver la protection contre CSRF
                            .authorizeRequests()
                                    .anyRequest()
                                            .authenticated() //il faut s'authentifier pour accéder à toutes les URLs
                                                    .and()
                                    .formLogin()
                                            .loginPage("/login") //page d'authentification (template)
                                                    .permitAll();
            return http.build();
        }
```

- Redémarrer le projet et exécuter, dans le navigateur web, l'url :
  GET : http://localhost:8080/produits/
- Afficher le code source du formulaire d'authentification et remarquer la disparition du paramètre caché « _csrf ».

```
<form action="/login" method="POST" >

    <div>
        <label>Login:</label>
        <input type="text" name="username"/>
    </div>
    <div>
        <label>Password:</label>
        <input type="password" name="password"/>
    </div>
    <div>
        <input  type="submit">S'inscrire</button>
    </div>
</form>
```

18. Passer maintenant à définir une page d'accueil en cas du succès d'authentification.

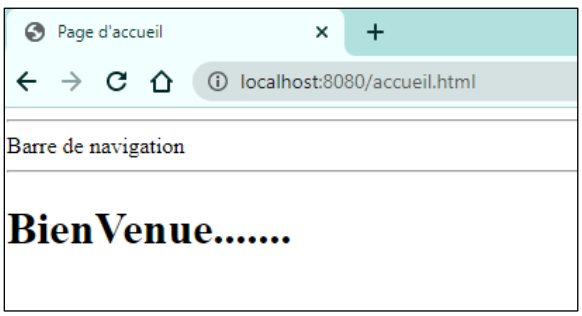
- Editer une page HTML statique sous « **src/main/resources/static** » nommée « **accueil.html** » :

```
<!DOCTYPE html>
<html>
    <head>
    <meta charset="ISO-8859-1">
    <title>Page d'accueil</title>
    </head>
    <body>
            <hr> Barre de navigation
            <hr>
            <h1>BienVenue.......</h1>
    </body>
</html>
```

- Modifier la méthode « **filterChain**» de la classe «**SecurityConfig** » comme suit :

```
@Bean
      public SecurityFilterChain filterChain(HttpSecurity http) throws
Exception {
                http
                      .csrf().disable() //désactiver la protection contre CSRF
                      .authorizeRequests()
                            .anyRequest()
                                  .authenticated() //il faut s'authentifier pour accéder à toutes les URLs
                                        .and()
                            .formLogin()
                                  .loginPage("/login") //page d'authentification (template)
                                        .permitAll()
                            .defaultSuccessUrl("/accueil.html",true);//page d'accueil

         return http.build();
      }
```

- Redémarrer le projet et exécuter, dans le navigateur web, l'url :
  GET : http://localhost:8080/produits/
- Visualiser la page d'accueil :



19. Passer, maintenant, à **sécuriser l'accès aux méthodes des services web** (càd gérer les rôles des utilisateurs) :

- Attribuer aux rôles (**ADMIN** et **USER**) toutes les opérations de lecture (**consultation**). Pour cela, ajouter l'annotation suivante, avant toutes les méthodes correspondantes du service web « **ProduitRESTController** » :

```
@Secured(value = {"ROLE_ADMIN","ROLE_USER"})
```

- Attribuer uniquement au rôle (**ADMIN**) toutes les opérations d'écriture (**ajout, mise à jour et suppression**). Pour cela, ajouter l'annotation suivante, avant toutes les méthodes correspondantes du service web « **ProduitRESTController** » :

```
@Secured(value = {"ROLE_ADMIN"})
```

- Pour activer cette sécurité, ajouter avant la définition de la classe de configuration «**SecurityConfig**», l'annotation suivante :

```
@EnableGlobalMethodSecurity(securedEnabled = true)
```

- Redémarrer le projet et exécuter, dans un navigateur web (une première fois pour l'utilisateur « **admin** » et une deuxième fois pour l'utilisateur « **user** ») les URLs suivantes :
  - GET : http://localhost:8080/produits/
  - GET : http://localhost:8080/produits/4
  - GET : http://localhost:8080/produits/delete/4

- Interpréter les résultats

20. Il est possible de récupérer les informations qui sont relatives à l'utilisateur courant.

- Ajouter, pour ceci, la méthode suivante dans la classe « **ProduitRESTController** » :

```
@RequestMapping(value="getInfosUser",
                produces = {MediaType.APPLICATION_JSON_VALUE })

Map<String,Object> getInformationsUtilisateurCourant(HttpServletRequest
request)
     {
          //préparer un objet HashMap
          Map<String,Object> m = new HashMap<String, Object>();
          //Récupérer la session
          HttpSession session =request.getSession();
          //Récupérer le contexte
```

```
            SecurityContext ctx =(SecurityContext)
session.getAttribute("SPRING_SECURITY_CONTEXT");
            //Récupérer le nom de l'utilisateur courant
            String username=ctx.getAuthentication().getName();
//Préparer une liste pour récupérer les noms des rôles de l'utilisateur courant
            List<String> roles= new ArrayList<String>();
            for (GrantedAuthority ga: ctx.getAuthentication().getAuthorities())
            {
                  roles.add(ga.getAuthority());
            }
            //stocker le nom de l'utilisateur
            m.put("username", username);
            //stocker les roles
            m.put("roles",roles);
            return m;
      }
```

- Redémarrer le projet et exécuter, dans un navigateur web l'URL suivante :
  GET : **http://localhost:8080/produits/getInfosUser**

- Visualiser les résultats :

  - Une première fois pour l'utilisateur « **admin** » :

  - Et une deuxième fois pour l'utilisateur « **user** » :

C. Personnalisation de la configuration de sécurité (Gérer des utilisateurs en Base de données)

21. Dans ce cas, les utilisateurs et les rôles sont gérés dans la base de données. Chaque utilisateur peut avoir plusieurs rôles et un rôle peut être affecté à plusieurs utilisateurs.

- Ainsi, définir une entité « **Role** » :

```
package spring.jpa.model;

import java.io.Serializable;

import jakarta.persistence.Entity;
import jakarta.persistence.Id;

@Entity
public class Role implements Serializable {
	@Id
	private String role;
	private String description;

	@Override
	public String toString() {
		return "Role [role=" + role + ", description=" + description + "]";
	}

	public Role(String role, String description) {
		super();
		this.role = role;
		this.description = description;
	}
	public Role() {
		super();
		// TODO Auto-generated constructor stub
	}

	public String getRole() {
		return role;
	}
	public void setRole(String role) {
		this.role = role;
	}
	public String getDescription() {
		return description;
	}
	public void setDescription(String description) {
		this.description = description;
	}
}
```

- Définir une entité « **User** » :

```
package spring.jpa.model;

import java.io.Serializable;
import java.util.Collection;

import jakarta.persistence.Entity;
import jakarta.persistence.Id;
import jakarta.persistence.JoinTable;
import jakarta.persistence.ManyToMany;
import jakarta.persistence.Table;

@Entity
@Table(name ="users")
public class User  implements Serializable{

	@Id
	private String username;
	private String password;
	private boolean actived;

	public boolean isActived() {
		return actived;
	}

	public void setActived(boolean actived) {
		this.actived = actived;
	}

	@ManyToMany
	@JoinTable(name="USERS_ROLES")
	private Collection <Role> roles;

	public User(String username, String password, boolean actived, Collection<Role>
roles) {
		super();
		this.username = username;
		this.password = password;
		this.actived = actived;
		this.roles = roles;
	}

	@Override
	public String toString() {
		return "User [username=" + username + ", password=" + password + ",
actived=" + actived + ", roles=" + roles
				+ "]";
	}
	public User() {
		super();
		// TODO Auto-generated constructor stub
	}

	public String getUsername() {
```

```
        return username;
    }

    public void setUsername(String username) {
        this.username = username;
    }

    public String getPassword() {
        return password;
    }

    public void setPassword(String password) {
        this.password = password;
    }

    public Collection<Role> getRoles() {
        return roles;
    }

    public void setRoles(Collection<Role> roles) {
        this.roles = roles;
    }
}
```

- Redémarrer le projet et remarquer la création des trois tables (**role**, **users** et **users_roles**)
- Insérer directement dans la base de données :
  - Un utilisateur («**ali**» , «**ali**» , 1)
  - Un utilisateur («**sami**» , «**sami**», 1)
  - Un utilisateur («**mohamed**» , «**mohamed**», 1)
  - Un rôle («**USER**» , «**simple utilisateur**»)
  - Un rôle («**ADMIN**» , «**super utilisateur**»)

- Affecter à « **ali** » le rôle « **USER** »
- Affecter à « **sami** » le rôle « **ADMIN** »
- Affecter à « **mohamed** » les deux rôles « **USER** » et « **ADMIN** »

- Modifier, dans le fichier « **application.properties** » la propriété suivante à update :

```
spring.jpa.hibernate.ddl-auto=update
```

22. Passer, maintenant à configurer les utilisateurs dans la classe de configuration « **SecurityConfig** » à partir de la base de données.

- Voici une nouvelle version de la classe :

```
package spring.jpa;

import javax.sql.DataSource;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.security.config.annotation.authentication.builders.AuthenticationManagerBuilder;
import org.springframework.security.config.annotation.method.configuration.EnableGlobalMethodSecurity;
import org.springframework.security.config.annotation.web.builders.HttpSecurity;
import org.springframework.security.config.annotation.web.configuration.EnableWebSecurity;
import org.springframework.security.crypto.password.NoOpPasswordEncoder;
import org.springframework.security.crypto.password.PasswordEncoder;
import org.springframework.security.web.SecurityFilterChain;

@Configuration
@EnableWebSecurity
@EnableGlobalMethodSecurity(securedEnabled = true)
public class SecurityConfig
{

    @Bean
    public PasswordEncoder passwordEncoder() {
        return NoOpPasswordEncoder.getInstance();
    }

    @Autowired // pour l'injection de dépendances
    //Définir les utilisateurs et leurs rôles
    public void
globalConfig(AuthenticationManagerBuilder auth ,
DataSource dataSource) throws Exception {
        // configuration en mémoire
    /*
        auth.inMemoryAuthentication().withUser("admin").password("{noop}admin").roles("ADMIN","USER");
        auth.inMemoryAuthentication().withUser("user").password("{noop}user").roles("USER");
     */
        //configuration en BD

auth.jdbcAuthentication()
    .dataSource(dataSource)
    .usersByUsernameQuery("select username as principal" +
",  password as credentials, true from users where username =? ")
```

```
    .authoritiesByUsernameQuery("select user_username as" +
" principal , roles_role as role from users_roles where " +
"user_username =?")
    .rolePrefix("ROLE_");
    }

      @Bean
      public SecurityFilterChain
filterChain(HttpSecurity http) throws Exception {
            http
                .csrf().disable() //désactiver la protection contre CSRF
                .authorizeRequests()
                    .anyRequest()
                        .authenticated() // il faut s'authentifier pour accéder à toutes les URLs
                            .and()
                    .formLogin()
                        .loginPage("/login") //page d'authentification (template)
                            .permitAll()

    .defaultSuccessUrl("/accueil.html",true)//page d'accueil
                            ;
        return http.build();
      }
}
```

- Redémarrer le projet et exécuter, dans un navigateur web (pour chacun des trois utilisateurs enregistrés dans la base de données : **ali**, **sami** et **mohamed**) les URLs suivantes :
  - GET : http://localhost:8080/produits/getInfosUser
  - GET : http://localhost:8080/produits/
  - GET : http://localhost:8080/produits/4
  - GET : http://localhost:8080/produits/delete/4

- Visualiser les résultats

D. Configuration de bootstrap

23. Améliorer la mise en page de la page d'authentification en utilisant Bootstrap

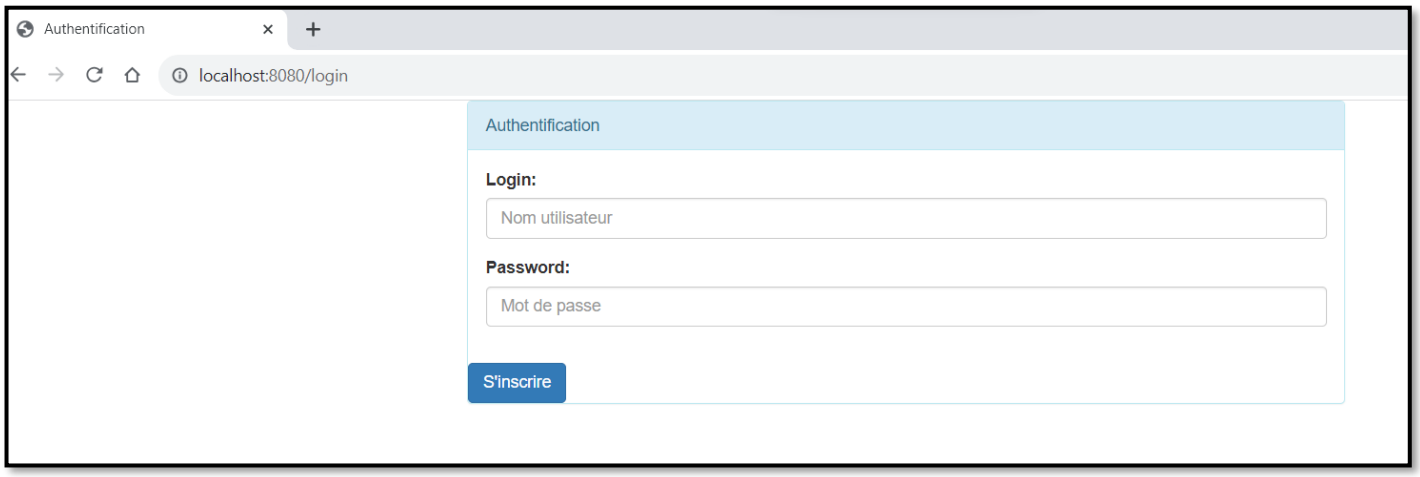
- Prendre une nouvelle version de la template « **login.html** » :

```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org">
<head>
<meta charset="utf-8" />
<title>Authentification</title>
<link rel="stylesheet" type="text/css"
     href="../static/css/bootstrap.min.css"
     th:href="@{/css/bootstrap.min.css}" />
<link rel="stylesheet" type="text/css"
     href="../static/css/mon_style.css" th:href="@{/css/mon_style.css}" />
</head>


<body>
     <div class="col-md-6 col-sm-6 col-xs-12 spacer col-md-offset-3">
          <div class="panel panel-info">
               <div class="panel-heading">Authentification</div>
                 <div class="panel-body">
                    <form th:action="@{/login}" method="POST">
                         <div class="form-group">
                         <label class="control-label">Login:</label>
<input      class="form-control" type="text" name="username"
                                   placeholder="Nom utilisateur" />
                         </div>
                         <div class="form-group">
                              <label class="control-label">Password:</label>
<input class="form-control" type="password" name="password"
                                   placeholder="Mot de passe" />
                         </div>
               </div>
               <div>
<input class="btn btn-primary" type="submit" value="S'inscrire">
               </div>
               </form>
          </div>
       </div>
      </div>
     </div>
   </body>
</html>
```

- Redémarrer le projet et exécuter l'URL :

  GET : **http://localhost:8080/login**

- Interpréter le résultat :



24. Changer maintenant la page d'accueil pour qu'elle utilise la notion de layouts :

- Définir une nouvelle page « **accueil.html** » sous « **src/main/resources/templates** » ayant le code suivant :

```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org"
xmlns:layout="http://www.ultraq.net.nz/thymeleaf/layout"
    layout:decorate="~{layout}">
<html>
<head>
<meta charset="utf-8" />
<title>Page d'accueil</title>
<link rel="stylesheet" type="text/css"
href="/css/bootstrap.min.css"
    th:href="@{../static/css/bootstrap.min.css}" />
<link rel="stylesheet" type="text/css"
href="/css/mon_style.css"
    th:href="@{../static/css/mon_style.css}" />
</head>
<body>
<div layout:fragment="content">
    <div class=" spacer ">
        <div class="panel panel-info">

            <div class="panel-body">
                <table class="table table-striped">
```

```
                    <tr>
                        <td><a
href="/produits/getInfosUser"> Infos (JSON)</a></td>
                        <td><a href="/produits/">
                liste des produits (XML)</a></td>

                    </tr>
                </table>

            </div>
        </div>
    </div>
</div>
</body>
</html>
```

- Prendre une nouvelle version de la template « **layout.html** » pour ajouter l'action de déconnexion :

```
<!DOCTYPE html>
<html
 xmlns:th="http://www.thymeleaf.org"
 xmlns:layout="http://www.ultraq.net.nz/thymeleaf/layout">
<head>
<meta charset="utf-8">
<title>Gestion de stock</title>
<link rel= "stylesheet"
      type ="text/css"
      href= "../static/css/bootstrap.min.css"
      th:href="@{css/bootstrap.min.css}"
      />
<link rel= "stylesheet"
      type ="text/css"
      href= "../static/css/style.css"
      th:href="@{css/style.css}"
      />
</head>
<body>

    <header>
```

```
        <div class ="navbar navbar-default">
            <div class ="container-fluid">
                <ul class="nav navbar-nav">
<li> <a th:href="@{/produit/index}">Consultation de produits (Tableau)</a></li>
<li> <a th:href="@{/produit/form}">Fiche Produit (Fiche)</a> </li>
<li> <a th:href="@{/accueil}">Accueil</a> </li>
<li> <a href="/logout" onclick="return confirm('Voulez vous vraiment quitter ?')">
                                        Déconnexion</a></li>
                </ul>
            </div>
        </div>
    </header>
    <section layout:fragment="content">
    </section>
    <footer class="navbar-fixed-bottom">
        <hr/>
        <div class ="container">
            <small>
            ----Gestion de Stock----
            </small>
        </div>
    </footer>
</body>
</html>
```

- Modifier la méthode « **filterChain**» de la classe «**SecurityConfig** » comme suit :

```
@Bean
public SecurityFilterChain filterChain(HttpSecurity
http) throws Exception {
            http
                .csrf().disable()//désactiver la protection contre CSRF
                .authorizeRequests()
                    .anyRequest()
                        .authenticated()  // il faut s'authentifier pour accéder à toutes les URLs
                            .and()
                .formLogin()
                    .loginPage("/login") //page d'authentification (template)
                        .permitAll()

    .defaultSuccessUrl("/accueil",true)//page d'accueil
                        .and()
                .logout()
```

```
                        .invalidateHttpSession(true)
                        .logoutUrl("/logout")
                        .permitAll()
                            ;
        return http.build();
    }
```

- Mettre à jour le mapping (url/view) dans la classe de configuration « **WebMVCConfig** » comme suit :

```
package spring.jpa;

import org.springframework.context.annotation.Configuration;
import org.springframework.web.servlet.config.annotation.ViewControllerRegistry;
import org.springframework.web.servlet.config.annotation.WebMvcConfigurer;

@Configuration
public class WebMVCConfig implements WebMvcConfigurer{
    @Override
    public void addViewControllers(ViewControllerRegistry registry) {
        registry.addViewController("/login").setViewName("login");

    registry.addViewController("/logout").setViewName("login");

    registry.addViewController("/accueil").setViewName("accueil");
    }
}
```

- Redémarrer le projet et exécuter l'URL :
  - GET : http://localhost:8080/login
- S'authentifier: