

Nom & prénom : DAHMANE walid

Classe : 4IIR G2

Site : CENTRE



Rapport Java JEE

Objet : *Spring Security*

❖ Code source :

<https://github.com/DAHMANE-git/GestionDesPatients>

❖ Présentation :

1 – Definition:

Spring Security est un framework qui se concentre sur la fourniture à la fois d'authentification et d'autorisation aux applications Java. Comme tous les projets Spring, la vraie puissance de Spring Security réside dans la facilité avec laquelle il peut être étendu pour répondre aux exigences personnalisées

2 – Spring Security Maven dependencies:

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

3 – Implementation:

Dans notre projet on va utiliser Spring Security pour appliquer la restriction des droits d'accès ainsi que la contextualisation. En plus des fonctionnalités SignIn et SignUp.

Nom & prénom : DAHMANE walid

Classe : 4IIR G2

Site : CENTRE

4- classe Configuration :

4.1 – Prototype est héritage :

On déclare un contrôleur de sécurité nommé SecurityConfig qui va hériter de WebSecurityConfigurerAdapter, c'est une classe SpingSecurity qui prend en charge la gestion d'authentification, en l'héritant on peut surcharger son comportement par défaut !

```
@Configuration
public class SecurityConfig extends WebSecurityConfigurerAdapter{
    @Autowired
    private DataSource dataSource;

    public PasswordEncoder passwordEncoder() {

    protected void configure(AuthenticationManagerBuilder auth) throws Exception {
    protected void configure(HttpSecurity http) throws Exception {
```

4.2 – Création d'utilisateur grâce a InMemoryAuthentication :

La création des utilisateurs en premier temps va se faire en uniquement dans la mémoire c'est-à-dire que nos users ne vont pas être stocké dans la base.

```
PasswordEncoder passwordEncoder = passwordEncoder();
auth.inMemoryAuthentication().withUser("user1").password("{noop}1234").roles("USER");//c'est une strategie de s:
auth.inMemoryAuthentication().withUser("user1").password(passwordEncoder.encode("1234")).roles("USER");
auth.inMemoryAuthentication().withUser("user2").password(passwordEncoder.encode("1234")).roles("USER");
auth.inMemoryAuthentication().withUser("admin").password(passwordEncoder.encode("1234")).roles("USER", "ADMIN");
```

4.3 – Création d'utilisateur grâce a JDBC Authentication :

4.3.1 – Code :

```
@Autowired
private DataSource dataSource;
```

```
auth.jdbcAuthentication().dataSource(dataSource).usersByUsernameQuery("select username as principal, password as credentials, active f
.authoritiesByUsernameQuery("select username as principal, role as role from users_roles where username=?")
.passwordEncoder(passwordEncoder).rolePrefix("ROLE_");
```

On a utiliser l'injection des dependances pour instancier l'interface DataSource pour l'utiliser dans notre traitement

Nom & prénom : DAHMANE walid

Classe : 4IIR G2

Site : CENTRE

4.3.2 – Conception :

Table	Action
<input type="checkbox"/> patients	★ Parcourir Structure Rechercher Insérer Vider Supprimer
<input type="checkbox"/> roles	★ Parcourir Structure Rechercher Insérer Vider Supprimer
<input type="checkbox"/> users	★ Parcourir Structure Rechercher Insérer Vider Supprimer
<input type="checkbox"/> users_roles	★ Parcourir Structure Rechercher Insérer Vider Supprimer

4.4 – Restriction et permissions :

```
@Override
protected void configure(HttpSecurity http) throws Exception {
    http.formLogin();
    //http.httpBasic();
    http.authorizeRequests().antMatchers("/save**/**", "/delete**/**", "/form**/**").hasRole("ADMIN");
    http.authorizeRequests().antMatchers("/patients**/**").hasRole("USER");
    //http.authorizeRequests().anyRequest().authenticated();
    http.csrf();
    http.exceptionHandling().accessDeniedPage("/notAuthorized");
}
```

Ici on a itliser lles methodes de La classe HttpSecurity pourspecifier les roles necessqires a chaque user pour qu'il puissent acceder a certains pages.

➔ **http.formLogin()** : permet de mettre en valeur le formulaire de connexion par défaut.

Please sign in

Username

Password

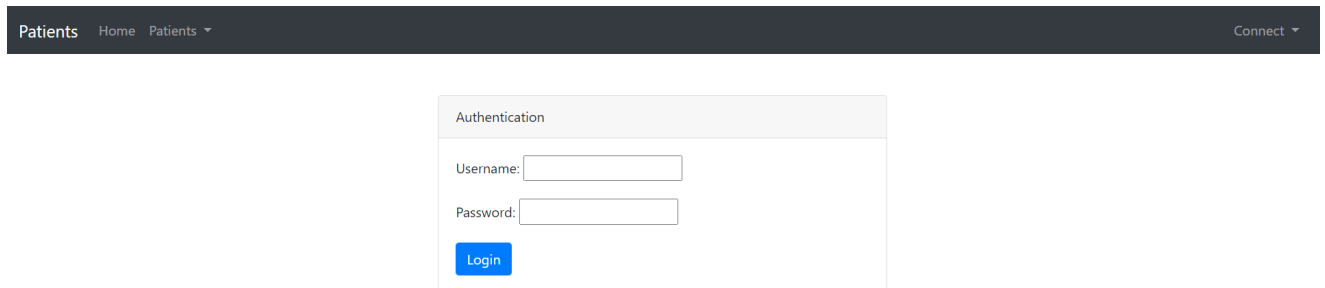
Sign in

Nom & prénom : DAHMANE walid

Classe : 4IIR G2

Site : CENTRE

→ `http.formLogin().loginPage("url")` : permet de surcharger le formulaire par défaut par celui qu'on spécifie dans le chemin.



The screenshot shows a web application interface. At the top, there is a dark navigation bar with the text 'Patients', 'Home', and 'Patients' with a dropdown arrow, followed by a 'Connect' button. Below the navigation bar, there is a light gray box titled 'Authentication'. Inside this box, there are two input fields: 'Username:' and 'Password:'. Below these fields is a blue button labeled 'Login'.

→ `http.authorizeRequests().anyRequest().authenticated()` : Interdire tout acces si le user est anonymous.

→ `http.authorizeRequests().anyRequest().permitAll()` : autoriser toutes les requetes peu importe le user.

4.5 – Contextualisation :

4.5.1 – Explication :

C'est le faite de cacher/afficher certains composants de notre page selon le user qui a envoyé la requete.

4.5.2 – Package :

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

4.5.3 – Exemple :

Ici on n'autorise l'affichage des boutons Delete & Edit que pour les utilisateurs avec le rôle ADMIN.

```
<td sec:authorize="hasRole('ROLE_ADMIN')">
  <a onclick="return confirm('Etes vous sur de vouloir supprimer?')"
    class="btn btn-danger"
    th:href="@{deletePatient2(id=${p.id},page=${currentPage},keyword=${keyword},size=${size})}">
    Delete</a>
  <a th:href="@{editPatient(id=${p.id})}" class="btn btn-success">
    Edit</a>
</td>
```