École Doctorale en Sciences Economique, Gestion et Informatique
Faculté des Sciences Économiques et de Gestion de Sfax
Département Informatique

Auditoire

Mastère Professionnel Audit et Sécurité Informatique

Architectures des Systèmes d'Information (JavaEE)

Enseignante

AÏDA KHEMAKHEM

Année Universitaire

2019 – 2020

Chapitre 4 :

Spring security :

la gestion d'utilisateurs

I. Introduction

II. La connexion basique

III. La connexion statique

IV. Attribution des rôles pour l'action sécurisée

V. La connexion dynamique

I. Introduction : La sécurité

- L'objectif : empêcher un utilisateur d'accéder à une ressource à laquelle il n'a pas droit

- Deux étapes
  - **Qui** veut accéder ?
  - Est-ce qu'il a **le doit d'accéder** à cette ressource ?

- Configuration selon **trois méthodes** qui dépend de la source des données :
  - Le fichier de configuration **application.properties**
  - Des données **statiques** (en mémoire, dans le code)
  - Des données **dynamiques** provenant d'une BD

- Les packages nécessaires : on utilise un module de Spring

org.springframework.security

Les dépendances dans le pom.xml

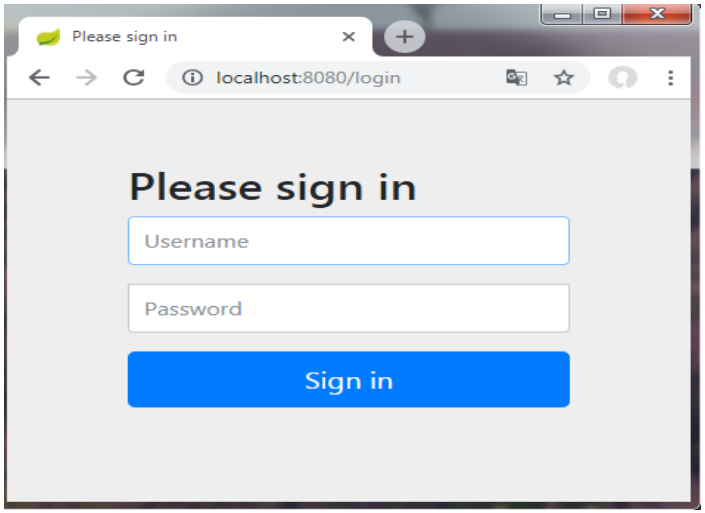
- Dans le fichier pom.xml, créer les deux dépendances suivantes :

pom.xml

```
<dependency><groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-web</artifactId>
</dependency>
<dependency><groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-security</artifactId>
</dependency>
<dependency>
    <groupId>org.springframework.security</groupId>
    <artifactId>spring-security-config</artifactId>
    <version>5.1.1.RELEASE</version>
</dependency>
```

II. Connexion basique

- Spring offre une configuration par défaut
- Authentification basique



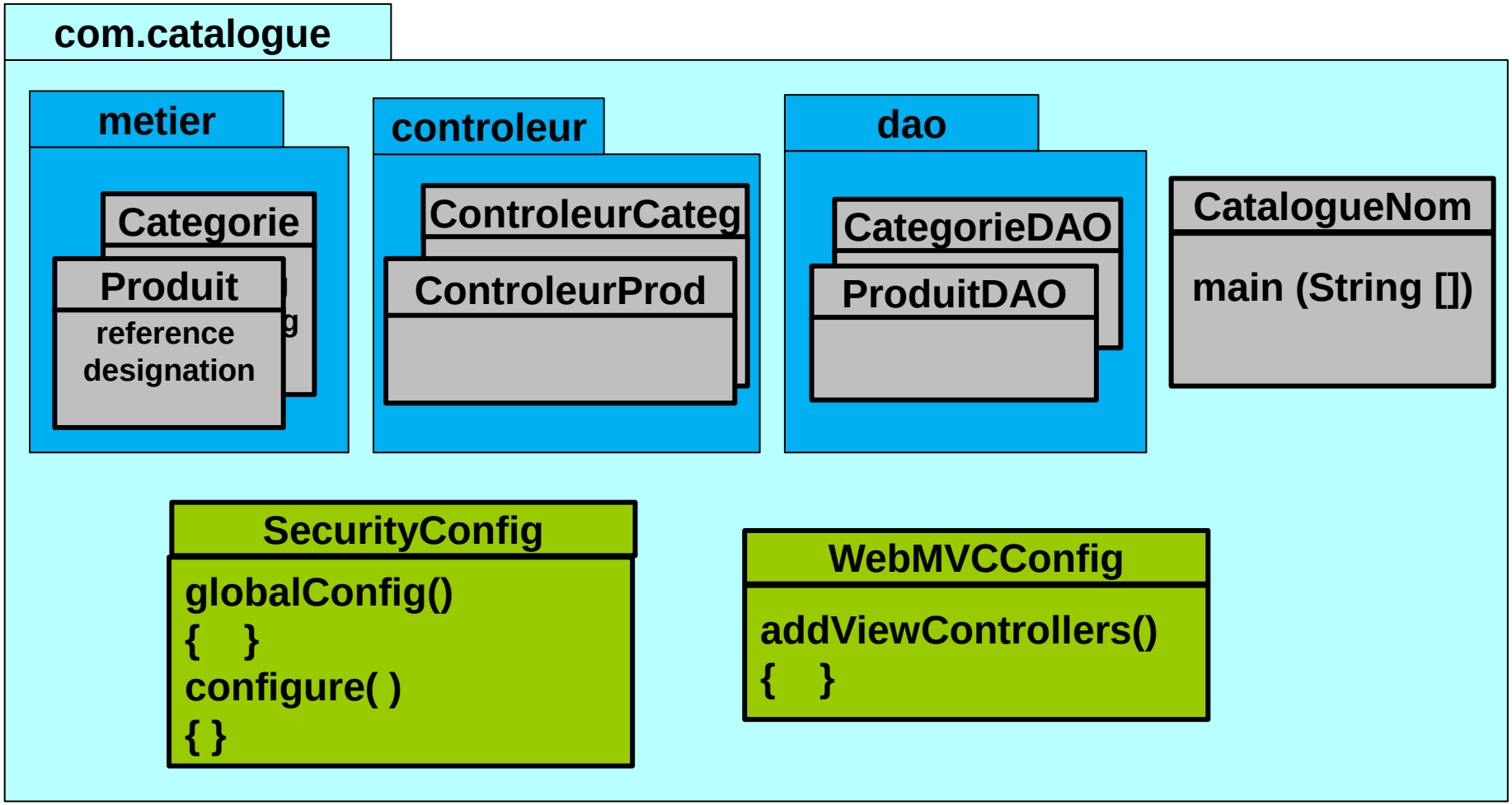
- Ajouter dans le fichier de configuration **application.properties**
- Un seul utilisateur

application.properties

```
spring.security.user.name=aida
spring.security.user.password=aaa
…
```

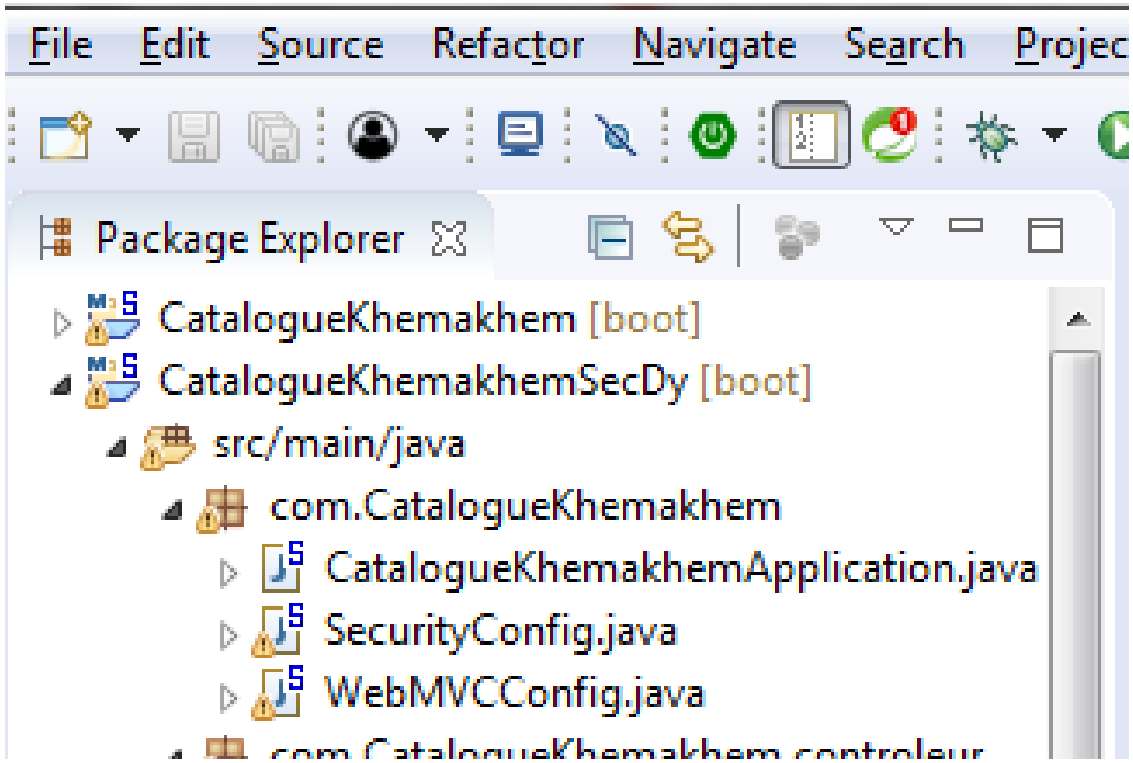
III. Connexion statique

Architecture de l'application



III.1. La classe SecurityConfig

- La classe **SecurityConfig** sert à configurer des rôles et des fonctionnements des pages (page) qui sont l'autorité d'accéder à ce rôle



La classe **SecurityConfig**

```
@Configuration
@EnableWebSecurity
public class SecurityConfig extends WebSecurityConfigurerAdapter
{

@Autowired
public void globalConfig(AuthenticationManagerBuilder auth) throws Exception
{
auth.inMemoryAuthentication().withUser("admin").password("{noop}aaa").roles("ADMIN","CLIE
NT");
auth.inMemoryAuthentication().withUser("cl1").password("{noop}111").roles("CLIENT");
auth.inMemoryAuthentication().withUser("cl2").password("{noop}112").roles("CLIENT");
auth.inMemoryAuthentication().withUser("vend1").password("{noop}333").roles("VENDEUR");
auth.inMemoryAuthentication().withUser("vend2").password("{noop}332").roles("VENDEUR");
}
}
```

Suite de la classe **SecurityConfig**

```
@Override
protected void configure(HttpSecurity http) throws Exception
{
http
    .authorizeRequests()
    .anyRequest().authenticated() //toutes les requêtes
                                  //doivent être authent
    .and()
    .formLogin().loginPage("/login")//page d'auth voir diap10
    .permitAll()     //autoriser l'accée à la page d'auth
    .defaultSuccessUrl("/menu");//url qui s'affiche après l'auth

}
} //fin class SecurityConfig
```

S'il y a les méthodes POST, désactiver CSRF

- La protection CSRF est activée par défaut avec la configuration Java
- Si vous souhaitez désactiver CSRF, vous trouverez ci-dessous la configuration Java correspondante

```
@Configuration
@EnableWebSecurity
public class SecurityConfig extends WebSecurityConfigurerAdapter
{//……..
@Override
protected void configure(HttpSecurity http) throws Exception
{
http
    .csrf().disable();
}
} //fin class SecurityConfig
```

III.2. La classe WebMVCConfig

```
@Configuration
public class WebMVCConfig extends WebMvcConfigurerAdapter
{
@Override
public void addViewControllers(ViewControllerRegistry
registry)
{
registry.addViewController("/login").setViewName("login");
registry.addViewController("/logout").setViewName("login");
}
}
```

Fichier login.html

```
<!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>
<table align="center">
<tr><td> Utilisateur :</td>
    <td><input type="text" name="username"/></td></tr>
<tr><td> Mot de passe : </td>
    <td><input type="password" name="password"/></td></tr>
</table>
</div>
<div><input type="submit" value="Login" /></div>
</form></div></body></html>*
```

IV. Attribution des rôles pour l'action sécurisée

- Ajouter dans la classe **SecurityConfig,** l'annotation suivante

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

```
@Configuration
@EnableWebSecurity
@EnableGlobalMethodSecurity(securedEnabled=true)
public class SecurityConfig extends WebSecurityConfigurerAdapter{.....}
```

- Ajouter dans les classes de **@Controller** et avant l'action sécurisée, l'annotation

```
@Secured(value={"ROLE_XXX","ROLE_YYY"})
```

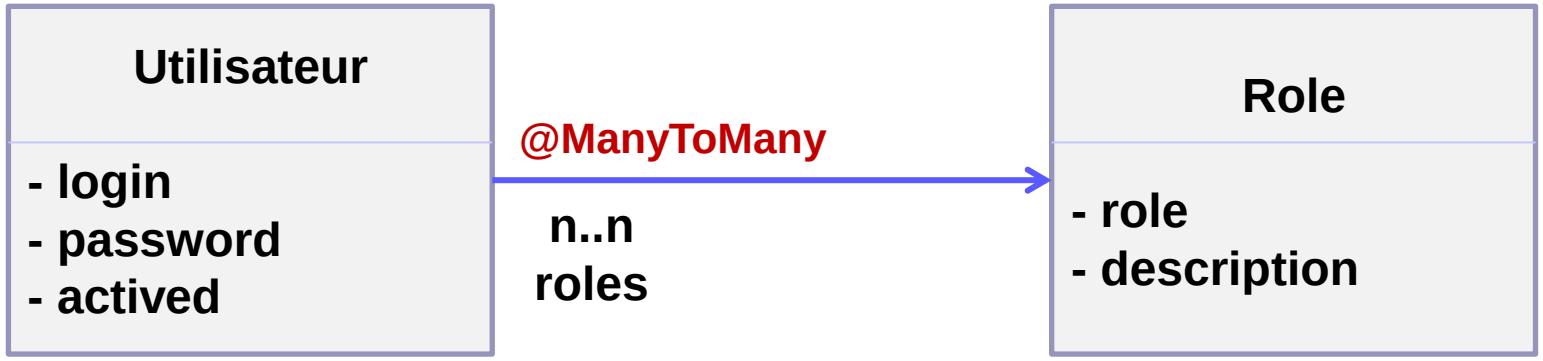
```
@Secured(value= {"ROLE_VENDEUR","ROLE_ADMIN"})
@RequestMapping(value="/gestionProduits", method=RequestMethod.GET)
public String gestionProduit(Model modele){...}
```

V. Connexion dynamique

Les principales étapes

- Modifier le fichier **SecurityConfig**
- Créer deux entités **Utilisateur** et **Role** et les **Repository** respectifs pour la gestion des utilisateurs (enregistrés dans la base de données)
- Préparer la couche métier qui va permettre de gérer l'accès à l'application
- Adapter la vue pour les messages d'erreurs

Les entités du couche métier



La classe SecurityConfig

```
 import javax.sql.DataSource; //
@Configuration
@EnableWebSecurity
public class SecurityConfig extends WebSecurityConfigurerAdapter{
@Autowired
public void globalConfig(AuthenticationManagerBuilder auth, DataSource
dataSource) throws Exception
{auth.jdbcAuthentication()
.dataSource(dataSource).passwordEncoder(passwordEncoder())//v page suivante
.usersByUsernameQuery("select login as principal, password as credentials,
actived from utilisateur where login = ?")
.authoritiesByUsernameQuery("select utilisateur_login as principal, roles_role
as role from utilisateur_roles where utilisateur_login = ?")
.rolePrefix("ROLE_");  }
```

Suite de la classe **SecurityConfig**

```
@Bean
public NoOpPasswordEncoder passwordEncoder() {
return (NoOpPasswordEncoder) NoOpPasswordEncoder.getInstance();
}
@Override
protected void configure(HttpSecurity http) throws Exception {

http
.authorizeRequests() //voir p 9
...
}
}
```