Fms Academy 2022
Formation Java Spring Angular
M10 : Spring Security

mohamed.el-babili@fms-ea.com

SOMMAIRE

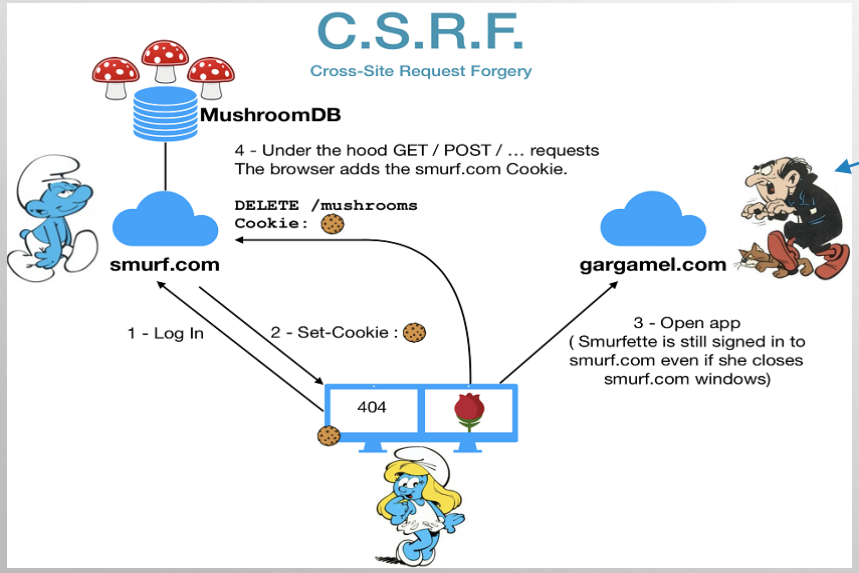
- POURQUOI LA SÉCURITÉ EST IMPORTANTE DANS UNE WEB APP ?
- TYPE DE SECURITE
- SPRING SECURITY FILTER
- MAVEN DEPENDANCY
- CLASSE DE CONFIGURATION DE LA SÉCURITÉ
- WEBSECURITYCONFIGURERADAPTER DEPRECATED
- AJOUTER UNE PAGE 403 POUR LES ACCÈS NON AUTORISÉS
- GESTION LOGOUT
- AFFICHER DES BALISES EN FONCTION DES DROITS D'ACCÈS
- DÉPLOIEMENT
- NECESSITE DE FAIRE DE LA VEILLE SUR LA SÉCURITÉ DES APP/WEB
- CONLUSION

Pourquoi la sécurité est importante dans une web app ?

Hacker

Dev

L'activité économique repose de + en + sur des applications web et les pirates sont attirés par l'appat du gain, ils profitent des vulnérabilités **ou failles de conception** : Cors, Xss, Csrf, SqlInjection, Xxe…



Ou failles humaines, on parle d'ingénierie ou piratage sociale lorsque les attaques se servent des comportements des individus (confiance, serviabilité, reconnaissance, anxiété, peur…) pour soutirer des informations de connexions et autre : phishing est inspiré de l'IS

Classifier les applications web

Appliquer le principe du moindre privilège

Appliquer une politique rigoureuse de gestion des mots de passe

Utiliser un scanner de vulnérabilité

Type de sécurité

Sécurité basée sur les cookies et les sessions : **statefull**

Les données de la session sont enregistrés côté serveur

Sécurité basée sur les tokens (Jwt) : **stateless**

Les données de la session sont enregistrés dans un jeton envoyé au client

Sécurité basée sur les blockchains : Décentralisation + consensus + crypto

SPRING SECURITY FILTER

- **Spring security filter** permet de jouer le rôle de filtre en amont du dispatcher servlet qui recevait jusqu'ici les requêtes avant d'orienter vers une méthode d'un contrôleur.

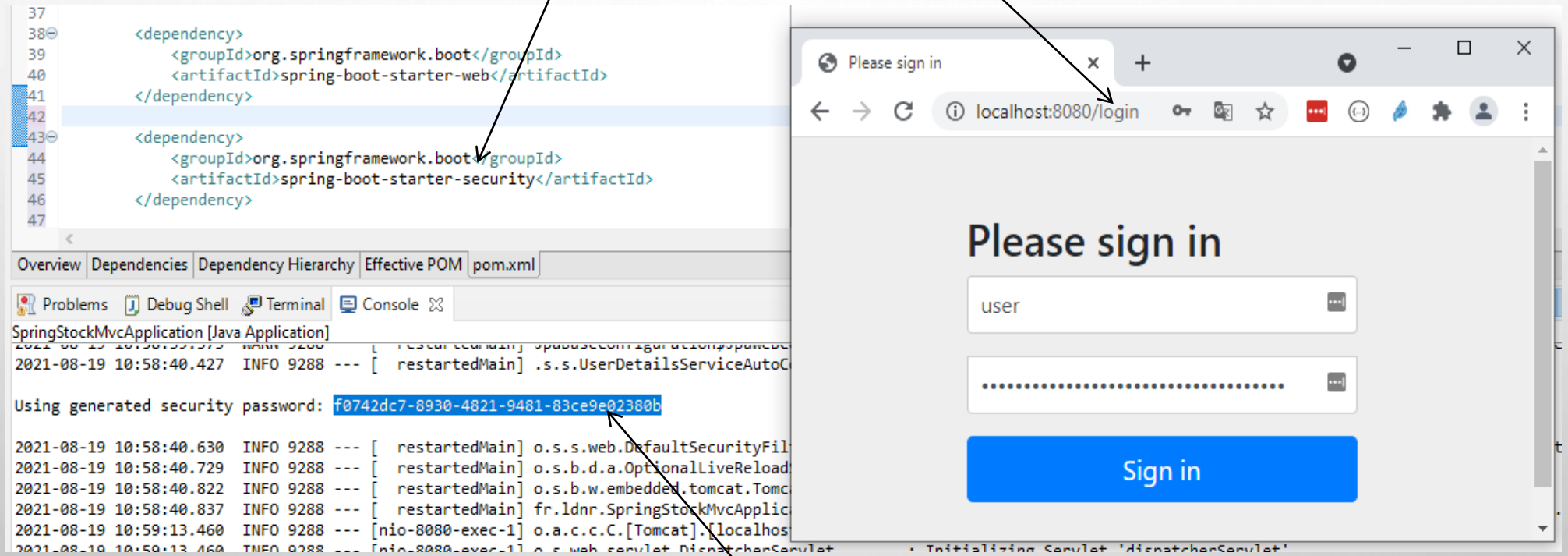
- Il vérifie donc chaque requête et voit si l'utilisateur est authentifié, si ce n'est pas le cas, il sera orienté vers un formulaire d'authentification
  - (On peut utiliser celui fourni par spring ou le réaliser nous même)

- L'utilisateur saisi donc id + pwd puis spring security vérifie en base ou en mémoire ses accès et ses droits/rôles/acteurs

- Si tout est ok, il aura accès à l'appli selon des droits spécifiques sinon l'user restera sur le formulaire d'authentification.

MAVEN DEPENDANCY

Dès qu'on ajoute la dépendance dans l'appli, au reboot, nous sommes redirigé vers un formulaire d'authentification fourni par Spring



On peut, par défaut, s'authentifier avec le mot de passe fourni par Spring sur l'id user

UTILISER UNE CLASSE DE CONFIGURATION DE LA SÉCURITÉ, POURQUOI ?

- Pour personnaliser la configuration de la sécurité et ne pas laisser Spring Security gérer celle-ci par défaut comme on l'a vu
- Pour indiquer si la gestion de la sécurité est en mémoire ou en base de données
- Pour sélectionner l'algo de cryptage choisi
- Pour attribuer les droits d'accès en fonction des rôles

Ex1 : Mémoire

Version 2.6.0 < 2.7.0

```
@Configuration
@EnableWebSecurity      //désactive le formulaire d'authentification par défaut de spring
                        //et active notre stratégie de sécurité
public class SecurityConfig extends WebSecurityConfigurerAdapter {

    @Override  //cette méthode précise si les users sont en base, dans un fichier, en mémoire comme ci-dessous
    protected void configure(AuthenticationManagerBuilder auth) throws Exception {

        //Il est impératif de toujours stocké en mémoire ou en base des mots de pass crypté
        //création d'utilisateurs en mémoire avec mot de passe crypté et des rôles distincts
        PasswordEncoder pe = passwordEncoder();
        auth.inMemoryAuthentication().withUser("mohamed").password(pe.encode("12345")).roles("ADMIN","USER");
        auth.inMemoryAuthentication().withUser("aymene").password(pe.encode("12345")).roles("USER");
        //indique à Spring l'algo utilisé pour le cryptage des pwd
        auth.inMemoryAuthentication().passwordEncoder(new BCryptPasswordEncoder());
    }

    @Bean   //annotation permettant à cet objet d'être inscrit dans le contexte de Spring
            //et delors peut être utilisé ailleurs dans l'appli via @Autowired
    PasswordEncoder passwordEncoder() {
        return new BCryptPasswordEncoder();
    }

    @Override
    protected void configure(HttpSecurity http) throws Exception {
        http.formLogin();
        //attribution des accès aux pages en fonction des rôles
        http.authorizeRequests().antMatchers("/index","/save","/delete","/edit","/article").hasRole("ADMIN");
        http.authorizeRequests().antMatchers("/index").hasRole("USER");
    }
}
```

- Dans ce 2ème exemple, nous allons travailler avec une base de donnée pour gérer les utilisateurs, pour cela, il faut les ajouter avec les droits associés dans une table d'association comme ci-dessous :

```
-- ------------------------------------------------------------------------
-- - Gestion des droits d'accès                                         ---
-- ------------------------------------------------------------------------
USE Stock;

-- ------------------------------------------------------------------------
-- - Construction de la table des Users                                 ---
-- ------------------------------------------------------------------------
CREATE TABLE T_Users (
    username            varchar(25)     PRIMARY KEY,
    password            varchar(250),
    active              boolean
) ENGINE = InnoDB;

INSERT INTO T_Users (username, password, active) VALUES ( 'mohamed',  '$2a$12$A.
INSERT INTO T_Users (username, password, active) VALUES ( 'aymene',  '$2a$12$/Fx

SELECT * FROM T_Users;

-- ------------------------------------------------------------------------
-- - Construction de la table avec 2 Roles principaux                   ---
-- ------------------------------------------------------------------------
CREATE TABLE T_Roles (
    role        varchar(25)     PRIMARY KEY
) ENGINE = InnoDB;

INSERT INTO T_Roles (role) VALUES ('ADMIN');
INSERT INTO T_Roles (role) VALUES ('USER');

-- ------------------------------------------------------------------------
-- - Construction de la table des rôles par utilisateur                 ---
-- ------------------------------------------------------------------------
CREATE TABLE T_Users_Roles (
    username        varchar(25),
    role            varchar(25),
    PRIMARY KEY(username,role)
) ENGINE = InnoDB;

INSERT INTO T_Users_Roles (username,role) VALUES ('mohamed','ADMIN');
INSERT INTO T_Users_Roles (username,role) VALUES ('mohamed','USER');
INSERT INTO T_Users_Roles (username,role) VALUES ('aymene','USER');
```

```
MySQL Client (MariaDB 10.3 (x64)) - mysql  -u root -p
| Tables_in_stock |
+-----------------+
| article         |
| t_roles         |
| t_users         |
| t_users_roles   |
+-----------------+
4 rows in set (0.053 sec)

MariaDB [Stock]> select * from T_Users;
+----------+--------------------------------------------------------------+--------+
| username | password                                                     | active |
+----------+--------------------------------------------------------------+--------+
| aymene   | $2a$12$/FxJ4RIYiIcO8eZp6wT.1e54T9q5uk4HVtHmUteGZqW2XGKs0RMRm |      1 |
| mohamed  | $2a$12$A.1omyeduJjn9BulU5TVxuLmvfC6FFiqUQieW2Y8Nc2xGwr44p5N2 |      1 |
+----------+--------------------------------------------------------------+--------+
2 rows in set (0.000 sec)

MariaDB [Stock]> select * from T_Users_Roles;
+----------+-------+
| username | role  |
+----------+-------+
| aymene   | USER  |
| mohamed  | ADMIN |
| mohamed  | USER  |
+----------+-------+
3 rows in set (0.000 sec)

MariaDB [Stock]>
```

- S'agissant du mot de passe crypté avec Bcrypt, il est possible de le générer en ligne **ici**
- Vous pouvez aussi utiliser HeidiSql pour générer les tables et les insertions.

Toujours dans la classe SecurityConfig, remplacer la gestion en mémoire par la gestion en base de donnée comme ci dessous puis vérifier par des tests

```java
import javax.sql.DataSource;

@Configuration
@EnableWebSecurity      //désactive le formulaire d'authentification par défaut de spring
                        //et active notre stratégie de sécurité
public class SecurityConfig extends WebSecurityConfigurerAdapter {
//  @Autowired
//  BCryptPasswordEncoder bCryptPasswordEncoder;

    @Autowired
    DataSource dataSource;  //pointe vers la base de donnée

    @Override  //cette méthode précise si les users sont en base, dans un fichier, en mémoire comme ci-dessous
    protected void configure(AuthenticationManagerBuilder auth) throws Exception {
        PasswordEncoder pe = passwordEncoder();
        //Il est impératif de toujours stocké en mémoire ou en base des mots de pass crypté
        //création d'utilisateurs en mémoire avec mot de passe crypté et des rôles distincts
        //auth.inMemoryAuthentication().withUser("mohamed").password(pe.encode("12345")).roles("ADMIN","USER");
        //auth.inMemoryAuthentication().withUser("aymene").password(pe.encode("12345")).roles("USER");
        //indique à Spring l'algo utilisé pour le cryptage des pwd
        //auth.inMemoryAuthentication().passwordEncoder(new BCryptPasswordEncoder());

        auth.jdbcAuthentication()
            .dataSource(dataSource)         //Spring va ici vérifier si l'utilisateur existe ou pas, si oui il compare les pwd, si ok il enchaine
            .usersByUsernameQuery("select username as principal, password as credentials, active from T_Users where username=?")
            .authoritiesByUsernameQuery("select username as principal, role as role from T_Users_Roles where username=?") //chargement des rôles pour username
            .rolePrefix("ROLE_")        //ajout d'un prefix, par ex si le role est ADMIN => ROLE_ADMIN
            .passwordEncoder(passwordEncoder());    //indique l'algo utilisé pour crypter les pwd
    }

    @Bean   //annotation permettant à cet objet d'être inscrit dans le contexte de Spring
            //et delors peut être utilisé ailleurs dans l'appli via @Autowired
    PasswordEncoder passwordEncoder() {
        return new BCryptPasswordEncoder();
    }
```

Bdd

WebSecurityConfigurerAdapter deprecated (Spring boot 2.7.0)

```java
@Configuration
@EnableWebSecurity
public class SecurityConfig {

    @Autowired
    DataSource dataSource;

    @Bean
    protected InMemoryUserDetailsManager configureAuthentication() {
        List<UserDetails>  userDetails = new ArrayList<>();

        List<GrantedAuthority> adminRoles = new ArrayList<>();
        adminRoles.add(new SimpleGrantedAuthority("ADMIN"));
        userDetails.add(new User("admin","$2a$12$A.1omyeduJjn9BulU5TVxuLmvfC6FFiqUQieW2Y8Nc2xGwr44p5N2",adminRoles));

        List<GrantedAuthority> userRoles = new ArrayList<>();
        userRoles.add(new SimpleGrantedAuthority("USER"));
        userDetails.add(new User("user","$2a$12$A.1omyeduJjn9BulU5TVxuLmvfC6FFiqUQieW2Y8Nc2xGwr44p5N2",userRoles));

        return new InMemoryUserDetailsManager(userDetails);
    }

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

    @Bean
    protected SecurityFilterChain filterChain(HttpSecurity http) throws Exception {
        http.formLogin().loginPage("/login");
        http.authorizeRequests().antMatchers("/confirm","/porder","/order","/save","/delete","/edit","/article").hasAuthority("ADMIN");
        http.authorizeRequests().antMatchers("/confirm","/porder","/order").hasAnyAuthority("USER","ADMIN");

        http.exceptionHandling().accessDeniedPage("/403");
        return http.build();
    }
}
```

Ajouter une page 403 pour les accès non autorisés

```java
protected void configure(HttpSecurity http) throws Exception {
    http.formLogin();
    //attribution des accès aux pages en fonction des rôles
    http.authorizeRequests().antMatchers("/index","/save","/delete","/edit","/article").hasRole("ADMIN");
    http.authorizeRequests().antMatchers("/index","/edit").hasRole("USER");

    http.exceptionHandling().accessDeniedPage("/403");  //au cas ou un utilisateur tente d'accéder à une page non authorisée
}
```

1

```java
@GetMapping("/403")
public String error() {
    return "403";
}
```

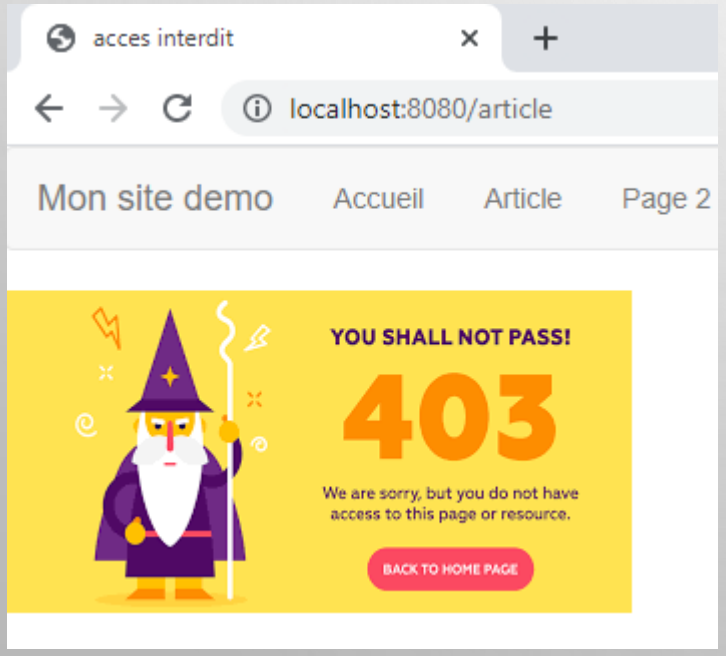
2

3

```html
<!DOCTYPE html>
<html xmlns:th="http://thymeleaf.org"
    xmlns:layout="http://www.ultraq.net.nz/thymeleaf/layout"
    layout:decorate="layout">
<head>
<meta charset="utf-8">
<title>acces interdit</title>
</head>

<body>
    <div layout:fragment="content">
        <h2>Vous n'avez pas les droits d'accès</h2>
    </div>
</body>

</html>
```



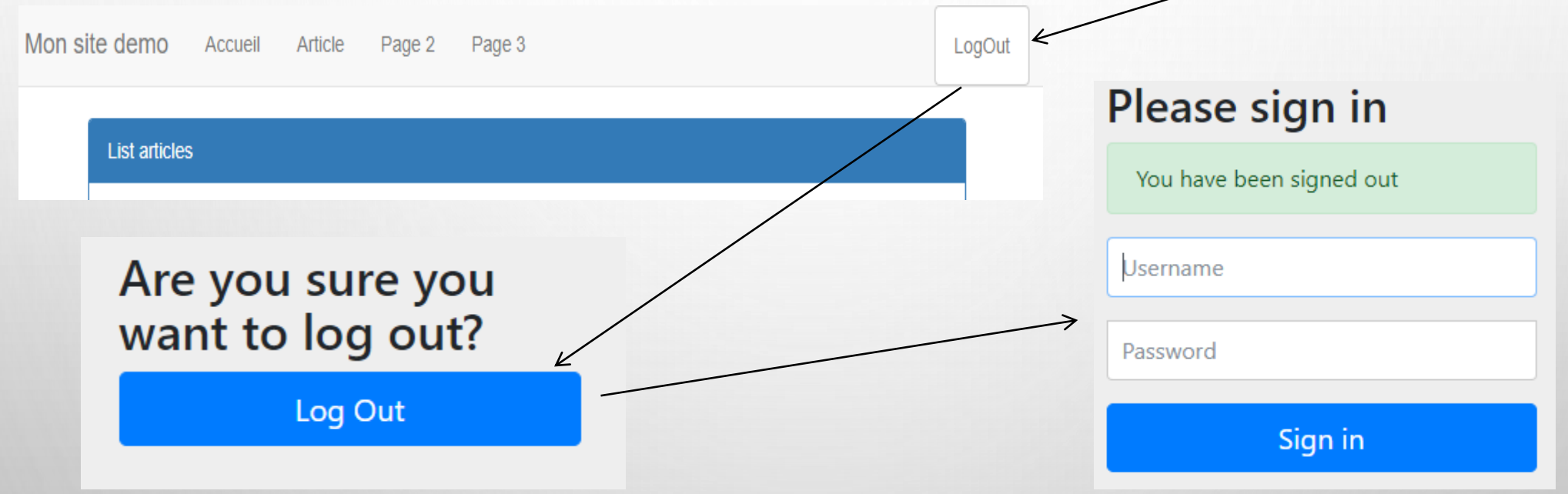
4

Gestion Logout

Tout comme pour la connexion, nous pouvons utiliser le gestionnaire de déconnexion fourni par Spring, il faut ajouter le lien dans la barre de navigation.

```
<ul class="nav navbar-nav">
  <li><a th:href="@{/index}">Accueil</a></li>
  <li><a th:href="@{/article}">Article</a></li>
  <li><a href="#">Page 2</a></li>
  <li><a href="#">Page 3</a></li>
</ul>

<ul class="nav navbar-nav pull-right">
    <li><a class="btn btn-default" th:href="@{/logout}">LogOut</a></li>
</ul>
```



NB : Il est possible de réaliser votre page login
- il faut l'indiquer à spring

```
@Override
protected void configure(HttpSecurity http)
    http.formLogin().loginPage("/login");
```

- ajouter la méthode dans le contrôleur
- réaliser le formulaire dans une page html (username,password)

AFFICHER DES BALISES EN FONCTION DES DROITS D'ACCÈS

En effet, vous souhaitez permettre à Admin uniquement de voir des éléments graphiques que User ne pourra pas visualiser.
De même, permettre à User (connecté) d'avoir accès à des données qu'un utilisateur non connecté ne pourra pas accèder, il faut :

1/ Ajouter la dépendance

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-thymeleaf</artifactId>
</dependency>
<dependency>
    <groupId>nz.net.ultraq.thymeleaf</groupId>
    <artifactId>thymeleaf-layout-dialect</artifactId>
</dependency>
<dependency>
<groupId>org.thymeleaf.extras</groupId>
    <artifactId>thymeleaf-extras-springsecurity5</artifactId>
</dependency>
```

2/ Ajouter le lien sur la page html visée

```
<!DOCTYPE html>
<html xmlns:th        = "http://thymeleaf.org"
      xmlns:layout    = "http://www.ultraq.net.nz/thymeleaf/layout"
      layout:decorate = "mylayout"
      xmlns:sec="http://www.thymeleaf.org/extras/spring-security5">
```

3/ Ajouter le test dans la balise concernée
Elle apparaîtra si le rôle est concerné

```
<td sec:authorize="hasRole('ROLE_ADMIN')">
    <a class="btn btn-info" th:href="@{ /edit (id=${a.id}) }" >Edit</a>
</td>
```

```
sec:authorize="hasRole('ROLE_USER')">
```

Déploiement avec maven

Sous Linux, il faut d'abord installer maven :
→ sudo apt-get update + sudo apt install maven
→ mvn package : compile + test + génére le .jar dans target

Sous Windows, fichier pom.xml/clic droit/ run as / maven build…

Dans le dossier Target donc vous pouvez récupérer « SpringShopMvc-0.0.1-SNAPSHOT.jar » puis le copier coller sur votre bureau et fermer Eclipse avant de l'exécuter.

Puis rendez vous sur un navigateur pour utiliser l'application normalement

Il est possible sur un même réseaux local de faire un appel depuis un poste client vers un poste dit serveur contenant l'appli qui tourne. Pour ce faire, à la place de localhost, insérer l'adresse IP du poste serveur.

Commande pour obtenir votre Ip sous Linux : hostname –i

Sous windows : ipconfig

NB : marche pas avec les pc airbus hyper verrouillé ;)

NECESSITE DE FAIRE DE LA VEILLE SUR LA SÉCURITÉ DES APP/WEB

https://owasp.org/

https://donnees-rgpd.fr/definitions/privacy-by-design/

https://openclassrooms.com/fr/courses/6179306-securisez-vos-applications-web-avec-lowasp/6520583-testez-la-securite-de-votre-application

Après Log4J, une nouvelle vulnérabilité dans notre écosystème : spring4shell
https://cyberwatch.fr/cve/spring4shell-tout-savoir-sur-la-vulnerabilite-0-day-liee-a-java-spring/
……..

CONCLUSION

En synthèse, qu'est-ce qu'on peut retenir ?