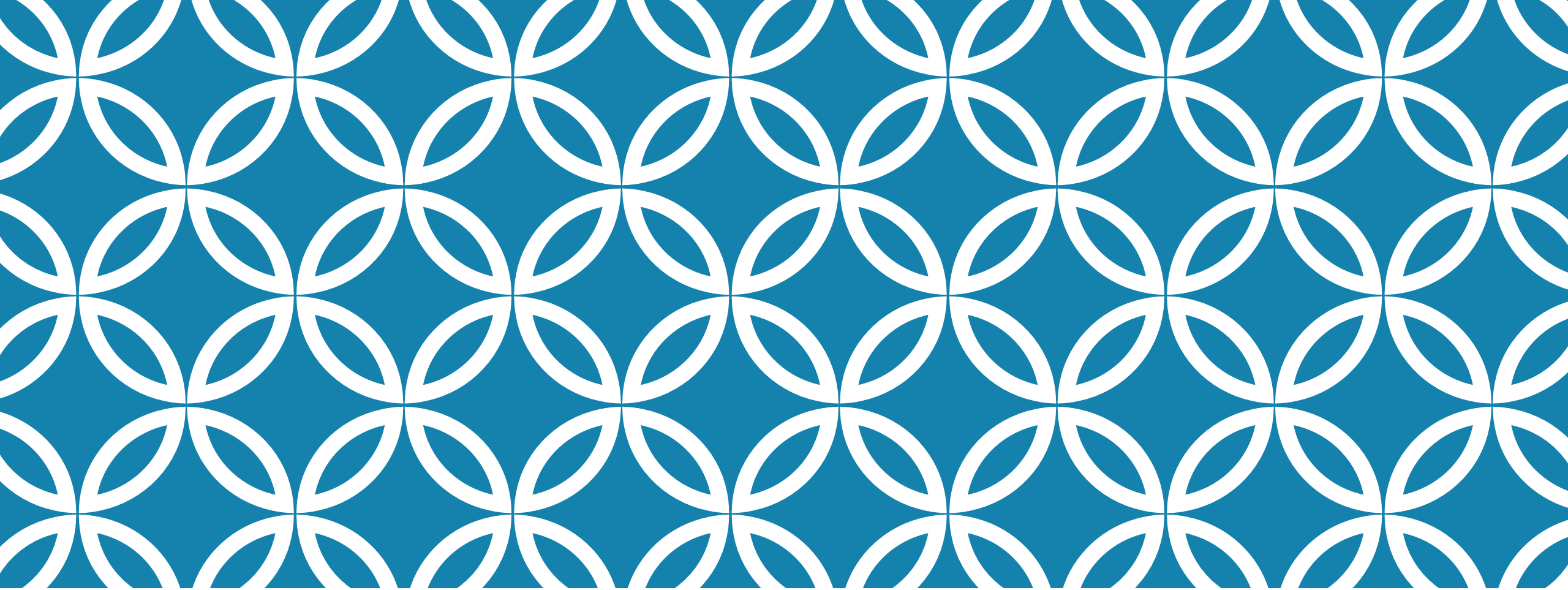
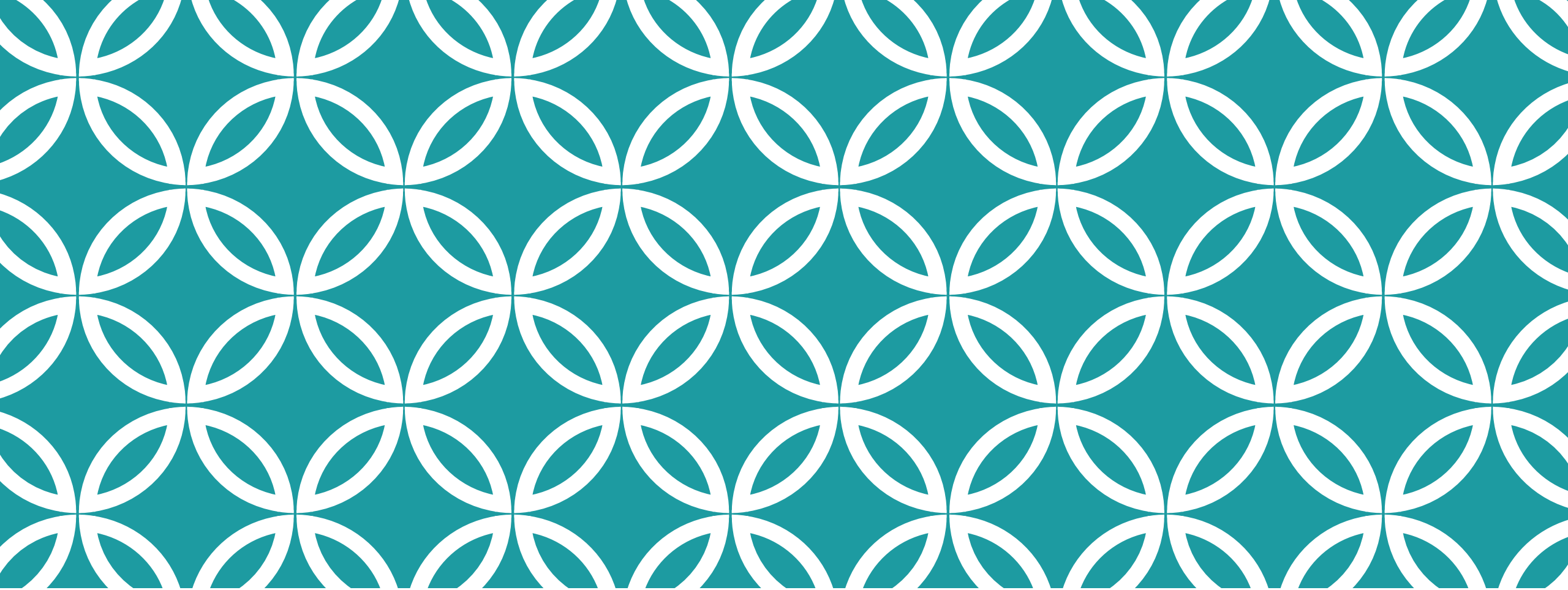


SPRING



SPRING SECURITY

Introduction à
Spring Security

CONFIGURATION

Spring Security requiert 2 dépendances

- **spring-security-web**
- **spring-security-config**

CONFIGURATION

Utilisation d'un filtre Web **DelegatingFilterProxy**

- On active le filtre sur toutes les ressources (« /* »)
- On sécurise l'ensemble des pages

```
<filter>
  <filter-name>springSecurityFilterChain</filter-name>
  <filter-class>org.springframework.web.filter.DelegatingFilterProxy</filter-class>
</filter>

<filter-mapping>
  <filter-name>springSecurityFilterChain</filter-name>
  <url-pattern>/*</url-pattern>
</filter-mapping>
```

CONFIGURATION

La sécurité peut avoir un context différent de celui du Dispatcher

- SAUF dans le cas de l'activation des annotations

Différentes possibilités de configuration

- Configuration dans le fichier web.xml
- Configuration directement dans le fichier dispatcher-context.xml / application-context.xml
- Configuration dans le fichier dispatcher-context.xml / application-context.xml
  - Via un import de configuration XML
- Configuration par classe

CONFIGURATION

Définition des URL autorisées

```
<security:http pattern="/resources/**" security="none" />

<security:http auto-config="true">
  <security:intercept-url pattern="/**" access="hasAnyRole('ROLE_ADMIN', 'ROLE_USER')" />
</security:http>
```

Définition des utilisateurs et de leur rôles

- Avec le préfix {noop} pour préciser qu'on ne chiffre pas les mots de passe (NoOpPasswordEncoder)

```
<security:authentication-manager>
  <security:authentication-provider>
    <security:user-service>
      <security:user name="admin" password="{noop}admin123456" authorities="ROLE_ADMIN" />
      <security:user name="user" password="{noop}mdp123456" authorities="ROLE_USER" />
    </security:user-service>
  </security:authentication-provider>
</security:authentication-manager>
```

CONFIGURATION

Configuration par classe

```
@EnableWebSecurity
public class SecurityConfig extends WebSecurityConfigurerAdapter {
  @Override
  protected void configure(HttpSecurity http) throws Exception {
    http.authorizeRequests()
      .antMatchers("/resources/**").permitAll()
      .antMatchers("/**").hasAnyRole("ADMIN", "USER")
      .and()
      .formLogin();
  }

  @Override
  protected void configure(AuthenticationManagerBuilder auth) throws Exception {
    auth.inMemoryAuthentication()
      .withUser("admin").password("{noop}admin123456").roles("ADMIN")
      .and()
      .withUser("user").password("{noop}user123456").roles("USER");
  }
}
```

PAGE D'AUTHENTIFICATION PERSONNALISÉE

Configuration de la page de login

```
<security:http auto-config="true">
  <security:intercept-url pattern="/ma_page_de_login*" access="isAnonymous()" />
  <security:intercept-url pattern="/**" access="isAuthenticated()" />

  <security:form-login login-page="/ma_page_de_login"
    login-processing-url="/perform_login"
    default-target-url="/home"
    authentication-failure-url="/ma_page_de_login?error=true" />

  <security:logout logout-success-url="/ma_page_de_deconnexion" />
</security:http>
```

PAGE D'AUTHENTIFICATION PERSONNALISÉE

Configuration de la page de login

```
@Override
protected void configure(HttpSecurity http) throws Exception {
  http
    .authorizeRequests()
      .antMatchers("/resources/**").permitAll()
      .antMatchers("/**").hasAnyRole("ADMIN", "USER")
    .and()
    .formLogin()
      .loginPage("/ma_page_de_login")
      .loginProcessingUrl("/perform_login")
      .defaultSuccessUrl("/home", true)
      .failureUrl("/ma_page_de_login?error=true")
      .permitAll()
    .and()
    .logout()
      .logoutSuccessUrl("/ma_page_de_deconnexion")
      .permitAll();
}
```

PAGE D'AUTHENTIFICATION PERSONNALISÉE

Par défaut, CSRF est activé (Cross-Site Request Forgery)

- Cette protection demande un jeton à chaque envoie de formulaire
- Cette protection empêche les requêtes provenant d'autres domaines
- Il faut donc préciser ce jeton dans les paramètres envoyés via le formulaire
  - On peut utiliser un champ caché pour que ce ne soit pas visible

```
<input type="hidden" th:name="${_csrf.parameterName}" th:value="${_csrf.token}" />
```

EXERCICE

Mettre en place la sécurité

- Tout est bloqué aux utilisateurs non connectés (sauf les ressources CSS, JS, …)
- Page de connexion personnalisée
  - Afficher un message d'erreur sur la page de connexion si les identifiants saisis sont faux

UTILISATEURS EN BASE DE DONNÉES

Nous avons jusqu'ici défini nos utilisateurs dans la configuration

Pour utiliser une liste d'utilisateur en base de données

- Définition d'un **@Service** qui implémente **UserDetailsService**
- Son rôle est de rechercher un utilisateur par son nom d'utilisateur
  - La cohérence du mot de passe se fera après avoir récupéré l'utilisateur

```
@Service
public class AuthService implements UserDetailsService {
  @Autowired
  IDAOUtilisateur daoUtilisateur;

  @Override
  public UserDetails loadUserByUsername(String username) throws UsernameNotFoundException {
    //...
  }
}
```

UTILISATEURS EN BASE DE DONNÉES

On indique dans la configuration qu'on utilise ce service

- NOTE : le bean est déjà déclaré puisque la classe est annotée de @**Service**
  - Par XML

```
<security:authentication-manager>
  <security:authentication-provider user-service-ref="authService" />
</security:authentication-manager>
```

  - Par classe

```
@Autowired
private AuthService authService;

@Override protected void configure(AuthenticationManagerBuilder auth) throws Exception {
  auth.userDetailsService(this.authService);
}
```

UTILISATEURS EN BASE DE DONNÉES

UserDetailsService manipule la classe **UserDetails**

- C'est cette classe qui joue le rôle de l'utilisateur à authentifier
- Il faut créer une classe qui encapsule le modèle Utilisateur, et qui implémente **UserDetails**

```
public class UtilisateurPrincipal implements UserDetails {
  private Utilisateur utilisateur;

  public UtilisateurPrincipal(Utilisateur utilisateur) {
    if (utilisateur == null) {
      throw new UsernameNotFoundException("L'utilisateur n'existe pas.");
    }

    this.utilisateur = utilisateur;
  }
}
```

UTILISATEURS EN BASE DE DONNÉES

Il faut ensuite modifier le comportement des méthodes de l'interface

Méthode	Fonction	Valeur de retour
getAuthorities	Liste des autorisations, des rôles	Une liste de rôles
getPassword	Récupérer le mot de passe	Le mot de passe
getUsername	Récupérer le nom d'utilisateur	Le nom d'utilisateur
isAccountNonExpired	Vérifie que le compte n'a pas expiré	Vrai
isAccountNonLocked	Vérifie que le compte n'est pas verrouillé	Vrai
isCredentialsNonExpired	Vérifie que les identifiants n'ont pas expirés	Vrai
isEnabled	Vérifie que le compte est actif	Vrai

UTILISATEURS EN BASE DE DONNÉES

Exemple

```java
@Override
public Collection<? extends GrantedAuthority> getAuthorities() {
  List<GrantedAuthority> myAuthorities = new ArrayList<GrantedAuthority>();
  myAuthorities.add(new SimpleGrantedAuthority("ROLE_USER"));
  return myAuthorities;
}

@Override
public String getPassword() {
  return this.utilisateur.getPassword();
}

@Override
public String getUsername() {
  return this.utilisateur.getUsername();
}
```

```java
@Override
public boolean isAccountNonExpired() {
  return true;
}

@Override
public boolean isAccountNonLocked() {
  return true;
}

@Override
public boolean isCredentialsNonExpired() {
  return true;
}

@Override
public boolean isEnabled() {
  return true;
}
```

UTILISATEURS EN BASE DE DONNÉES

Il faut ensuite indiquer comment sont chiffrés les mots de passe

- En utilisant un **PasswordEncoder**

  - Pas de chiffrement

```
<bean id="passwordEncoder" class="org.springframework.security.crypto.password.NoOpPasswordEncoder" />
```

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

  - Chiffrement Blowfish (bcrypt)

```
<bean id="passwordEncoder" class="org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder" />
```

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

UTILISATEURS EN BASE DE DONNÉES

Pour générer un mot de passe Blowfish

```
BCryptPasswordEncoder bcrypt = new BCryptPasswordEncoder();
System.out.println(bcrypt.encode("le mot de passe"));
```

EXERCICE

Modifier la sécurité

- Les utilisateurs sont stockés en base de données
- Leur mot de passe est chiffré !

AUTORISATIONS – ANNOTATIONS

@Secured

@PreAuthorize / @PostAuthorize

Annotations à utiliser

- Sur une classe
- Sur une méthode

AUTORISATIONS – ANNOTATIONS

@PreAuthorize / @PostAuthorize

- Se base sur Spring Expression Language (SpEL)
- @**PreAuthorize**("hasRole(‘ROLE_1’) and hasRole(‘ROLE_2’)")
- ROLE_1 ET ROLE_2

@Secured

- Se base sur une liste de rôles
- @**Secured**({ "ROLE_1", "ROLE_2" })
- ROLE_1 OU ROLE_2

AUTORISATIONS – ANNOTATIONS

Il faut activer les annotations dans la configuration (du Dispatcher !)

- En XML

```
<security:global-method-security pre-post-annotations="enabled" secured-annotations="enabled" />
```

- En classe

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

AUTORISATIONS – ANNOTATIONS

@PreAuthorize

- hasRole("role")
- hasAnyRole("role1", "role2")
- isAnonymous()
- isAuthentificated()
- hasPermission()

AUTORISATIONS – ANNOTATIONS

Il est possible d'utiliser des meta-annotations

- Pour regrouper plusieurs annotations
- Pour éviter d'écrire plusieurs fois les mêmes autorisations

```
@Retention(RetentionPolicy.RUNTIME)
@PreAuthorize("hasRole('ROLE_ADMIN')")
public @interface IsAdmin {

}
```

- Dans cet exemple, l'utilisation de **@IsAdmin** remplacera l'annotation **@PreAuthorize**(…)

EXERCICE

Sécuriser un contrôleur **HomeController** par des annotations

- Seul l'administrateur a accès à la vue *home*

AUTORISATIONS – VUES

Si utilisation des JSP

- Nécessite la dépendance **spring-security-taglibs**
- Nécessite ces taglibs dans chaque JSP

```
xmlns:sec="http://www.springframework.org/security/tags"
```

```
<sec:authorize access="hasRole('ROLE_USER')">
  <!-- -->
</sec:authorize>
```

AUTORISATIONS – VUES

Si utilisation de Thymeleaf

- Nécessite la dépendance **thymeleaf-extras-springsecurity4**
- Nécessite l'ajout du dialect **SpringSecurityDialect**
- Nécessite l'ajout de l'espace de nom XML

AUTORISATIONS – VUES

Configuration XML

```
<bean id="templateEngine" class="org.thymeleaf.spring5.SpringTemplateEngine">
  <property name="templateResolver" ref="templateResolver" />
  <property name="enableSpringELCompiler" value="true" />
  <property name="additionalDialects">
    <set>
      <bean class="nz.net.ultraq.thymeleaf.LayoutDialect" />
      <bean class="org.thymeleaf.extras.springsecurity4.dialect.SpringSecurityDialect" />
    </set>
  </property>
</bean>
```

AUTORISATIONS – VUES

Configuration Java

```
@Bean
public SpringTemplateEngine templateEngine(SpringResourceTemplateResolver templateResolver) {
  SpringTemplateEngine templateEngine = new SpringTemplateEngine();

  templateEngine.setTemplateResolver(templateResolver);
  templateEngine.setEnableSpringELCompiler(true);
  templateEngine.addDialect(new SpringSecurityDialect());
  templateEngine.addDialect(new LayoutDialect());

  return templateEngine;
}
```

AUTORISATIONS – VUES

Dans chaque vue

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

```
<div sec:authorize="hasRole('ROLE_USER')">
  <!-- -->
</div>
```

EXERCICE

Modifier la sécurité

- Tout le monde a accès au contrôleur **HomeController**
- Dans la vue *home*, afficher le message
  - « Bonjour administrateur » si c'est un administrateur
  - « Bonjour utilisateur » si c'est un utilisateur

AUTORISATIONS – HASPERMISSION

Pour aller plus loin dans la configuration de la sécurité

- Configuration de permissions avec hasPermission
- Permet de définir des permissions sur un objet

Création d'une classe qui implémente **PermissionEvaluator** (Security Access)

- Implémentation des méthodes hasPermission
  - Permettent de déterminer si l'utilisateur a les permissions nécessaires pour accéder à l'objet

AUTORISATIONS – HASPERMISSION

```
@Override
public boolean hasPermission(Authentication auth, Object targetDomainObject, Object permission) {
  if ((auth == null) || (targetDomainObject == null) || !(permission instanceof String)) {
    return false;
  }

  String targetType = targetDomainObject.getClass().getSimpleName().toUpperCase();
  return this.hasPrivilege(auth, targetType, permission.toString().toUpperCase());
}
```

AUTORISATIONS – HASPERMISSION

```
@Override
public boolean hasPermission(Authentication auth, Serializable targetId, String targetType, Object permission) {
  if ((auth == null) || (targetType == null) || !(permission instanceof String)) {
    return false;
  }

  return this.hasPrivilege(auth, targetType.toUpperCase(), permission.toString().toUpperCase());
}
```

AUTORISATIONS – HASPERMISSION

```
private boolean hasPrivilege(Authentication auth, String targetType, String permission) {
  for (GrantedAuthority grantedAuth : auth.getAuthorities()) {
    if (grantedAuth.getAuthority().startsWith(targetType)) {
      if (grantedAuth.getAuthority().contains(permission)) {
        return true;
      }
    }
  }

  return false;
}
```

AUTORISATIONS – HASPERMISSION

Exemple d'autorisations pour un utilisateur **UserDetails**

- On lui donne le rôle USER
- On lui donne les privilèges de lecture et d'écriture sur les produits

```
@Override
public Collection<? extends GrantedAuthority> getAuthorities() {
  List<GrantedAuthority> myAuthorities = new ArrayList<GrantedAuthority>();

   myAuthorities.add(new SimpleGrantedAuthority("ROLE_USER"));
   myAuthorities.add(new SimpleGrantedAuthority("PRODUIT_READ_PRIVILEGE"));
   myAuthorities.add(new SimpleGrantedAuthority("PRODUIT_WRITE_PRIVILEGE"));

  return myAuthorities;
}
```

AUTORISATIONS – HASPERMISSION

Il faut ensuite configurer Spring Security pour prendre en compte cette classe

- Sécurité liée au Dispatcher (Annotations)

```
<security:global-method-security pre-post-annotations="enabled">
  <security:expression-handler ref="expressionHandler" />
</security:global-method-security>

<bean id="permissionEvaluator" class="fr.conviviance.security.ConviviancePermissionEvaluator" />
<bean id="expressionHandler"
    class="org.springframework.security.access.expression.method.DefaultMethodSecurityExpressionHandler">
  <property name="permissionEvaluator" ref="permissionEvaluator" />
</bean>
```

- Sécurité globale pour activer cette classe au niveau JSP (voir slide suivante)

AUTORISATIONS – HASPERMISSION

```
<security:http use-expressions="true">
  <security:intercept-url pattern="/login*" access="isAnonymous()" />
  <!-- ... -->
  <security:expression-handler ref="webExpressionHandler" />
</security:http>

<bean id="permissionEvaluator" class="fr.conviviance.security.ConviviancePermissionEvaluator" />
<bean id="webExpressionHandler"
    class="org.springframework.security.web.access.expression.DefaultWebSecurityExpressionHandler">
  <property name="permissionEvaluator" ref="permissionEvaluator" />
</bean>
```

EXERCICE

Mettre en place la sécurité via des permissions

- Les utilisateurs peuvent voir les produits
- Les administrateurs peuvent voir et modifier les produits