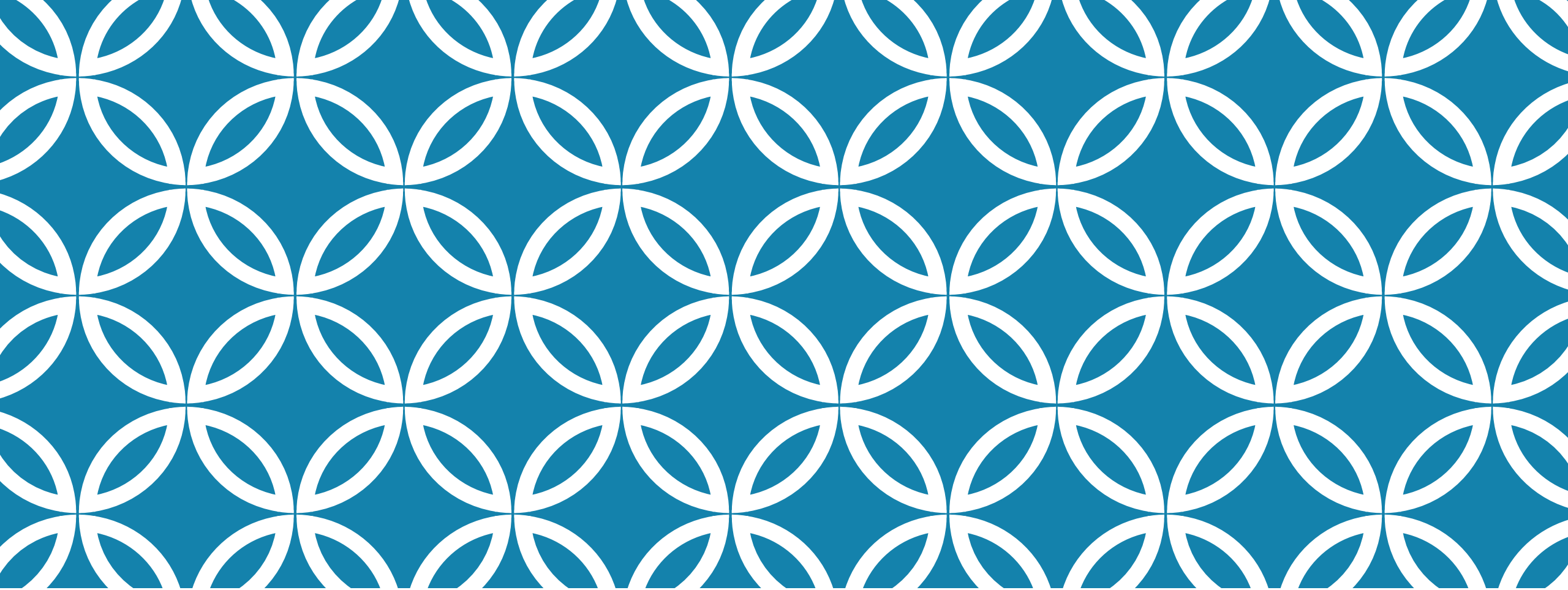
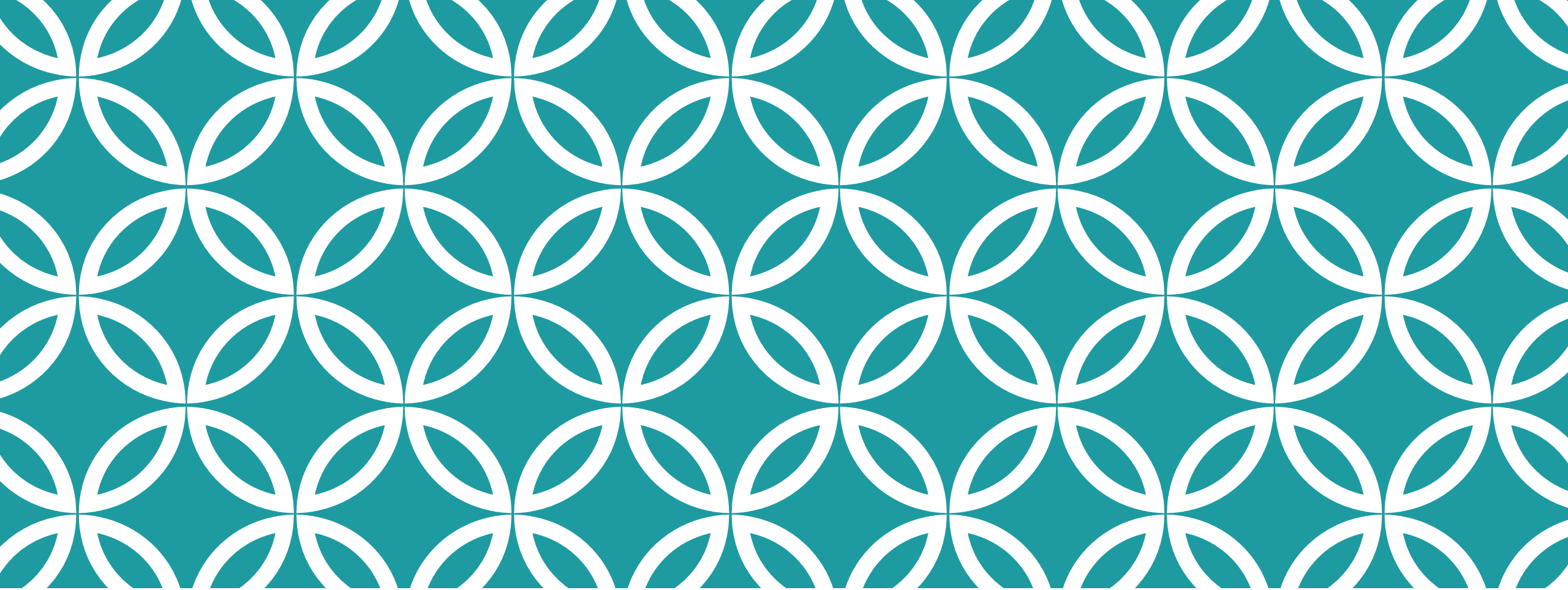


SPRING

Jérémy PERROUAULT



SPRING SECURITY

Introduction et configuration de 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

- MAIS, dans le cas de l'activation des annotations, ce sera au Dispatcher d'avoir l'information

Différentes configurations

- Configuration Spring
  - Configuration directement dans le fichier *dispatcher-context.xml*
  - Configuration dans un fichier *security-context.xml*
    - + import de configuration XML possible
  - Configuration par classe **SecurityConfig**
    - + import de classe possible
- Configuration applicative (dans le fichier web.xml)
  - Déclaration de la nouvelle configuration, si non importée dans une configuration Spring déjà existante !

CONFIGURATION

Définition des URL autorisées

```
<security:http pattern="/assets/**" 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

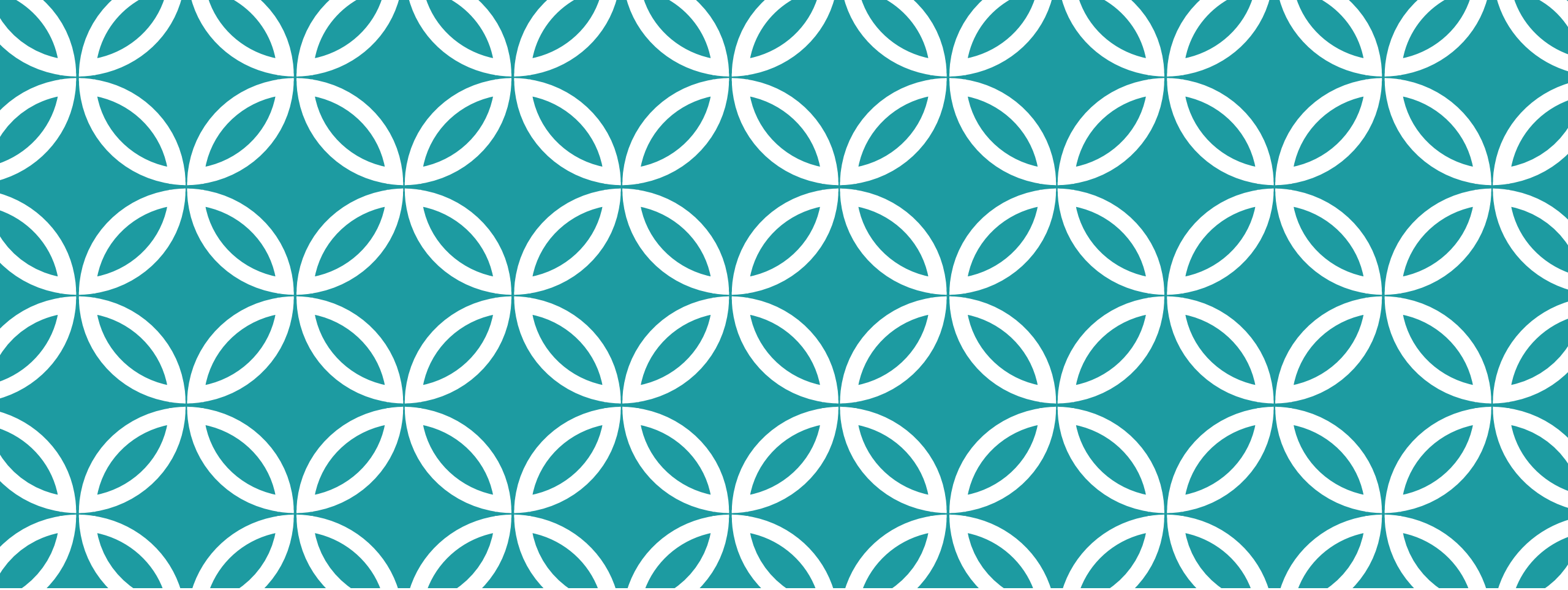
Configuration par classe

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

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

EXERCICE

Mettre en place la sécurité

- Tout est bloqué aux utilisateurs non connectés (sauf les ressources CSS, JS, …)



PAGE D'AUTHENTIFICATION

Personnaliser la page de connexion

PAGE D'AUTHENTIFICATION PERSONNALISÉE

Par défaut, Spring propose une page d'authentification

- Possible de la remplacer par une page personnalisée

PAGE D'AUTHENTIFICATION PERSONNALISÉE

Configuration de la page de login (XML)

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

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

PAGE D'AUTHENTIFICATION PERSONNALISÉE

Configuration de la page de login (Java)

```
protected void configure(HttpSecurity http) throws Exception {
  http
    .authorizeRequests()
      .antMatchers("/assets/**").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()
      .logoutUrl("/ma_page_de_deconnexion")
      .logoutSuccessUrl("/ma_page_de_login")
      .permitAll();
}
```

PAGE D'AUTHENTIFICATION PERSONNALISÉE

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

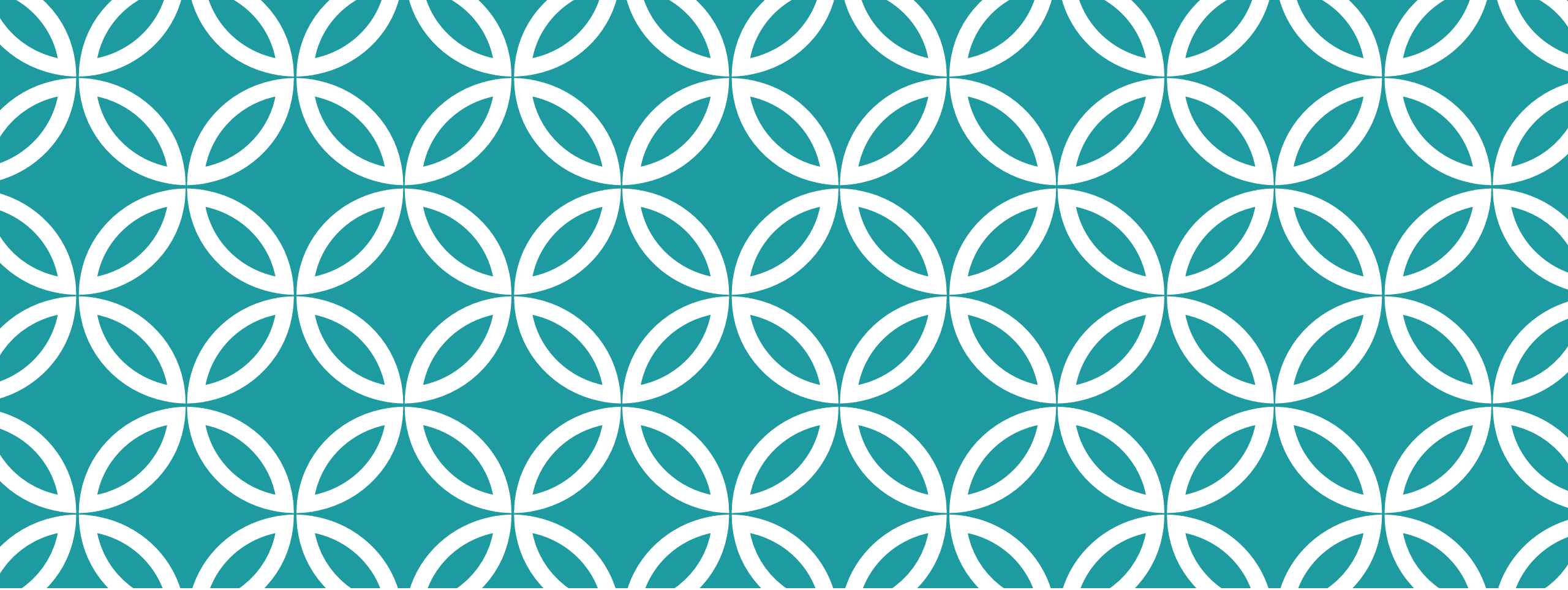
- Cette protection demande un jeton à chaque envoie de formulaire
- Cette protection empêche les requêtes provenant d'autres domaines
- Il faut donc préciser ce jeton dans les paramètres envoyés via le formulaire
  - On peut utiliser un champ caché pour que ce ne soit pas visible

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

EXERCICE

Page de connexion personnalisée

- Afficher un message d’erreur sur la page de connexion si les identifiants saisis sont faux



UTILISATEURS & ROLES

Stocker les utilisateurs et leurs rôles en base de données

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
  IUtilisateurDao daoUtilisateur;

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

UTILISATEURS EN BASE DE DONNÉES

On indique dans la configuration qu'on utilise ce service

- NOTE : vérifier que le package dans lequel le **@Service AuthService** est placé est bien scanné
  - Par XML

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

  - Par classe

```
@Autowired
private AuthService authService;

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

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

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

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

```
public boolean isAccountNonExpired() {
  return true;
}

public boolean isAccountNonLocked() {
  return true;
}

public boolean isCredentialsNonExpired() {
  return true;
}

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

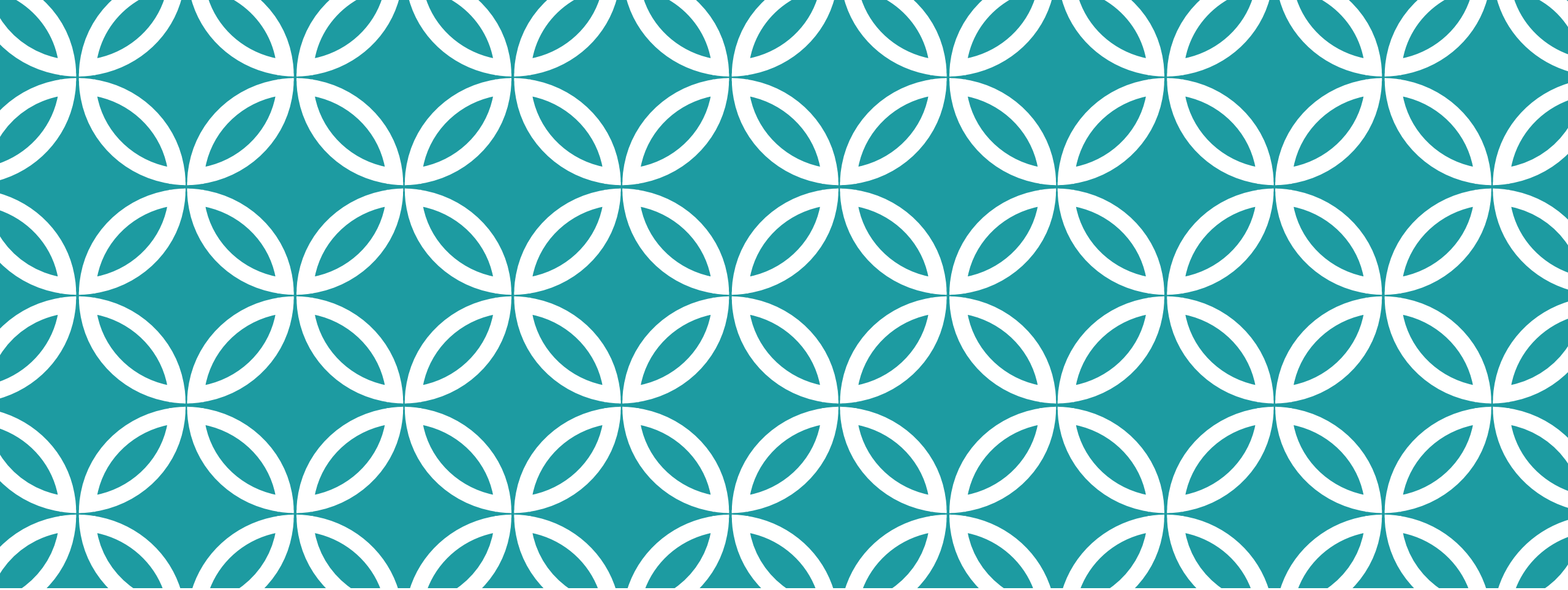
Pour générer un mot de passe Blowfish

```
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 !



LES ANNOTATIONS

Sécuriser les accès via annotations

AUTORISATIONS – ANNOTATIONS

@Secured

@PreAuthorize / @PostAuthorize

@PreFilter / @PostFilter

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 si plusieurs contextes !)

- En XML

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

- En classe

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

AUTORISATIONS – ANNOTATIONS

@PreAuthorize / @PostAuthorize

- hasRole("role")
- hasAuthority("authorite")
- hasAnyRole("role1", "role2")
- hasAnyAuthority("authorite1", "authorite2")
- isAnonymous()
- isAuthenticated()
- hasPermission()
  - #arg, returnObject

A placer sur une méthode

- D'un contrôleur
- D'un service
- Eventuellement d'une **DAO**

AUTORISATIONS – ANNOTATIONS

@PreFilter / @PostFilter

- hasRole("role")
- hasAuthority("authorite")
- hasAnyRole("role1", "role2")
- hasAnyAuthority("authorite1", "authorite2")
- isAnonymous()
- isAuthenticated()
- hasPermission()
  - #arg, filterObject

A placer sur une méthode

- D'un contrôleur
- D'un service
- Eventuellement d'une **DAO**

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 le contrôleur **ProduitController** par des annotations

- Seul l'administrateur a accès au CRUD

LES VUES, SELON LES RÔLES

Contenu selon les rôles

AUTORISATIONS – VUES

Si utilisation des JSP

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

```
<%@ taglib uri="http://www.springframework.org/security/tags" prefix="sec" %>

<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

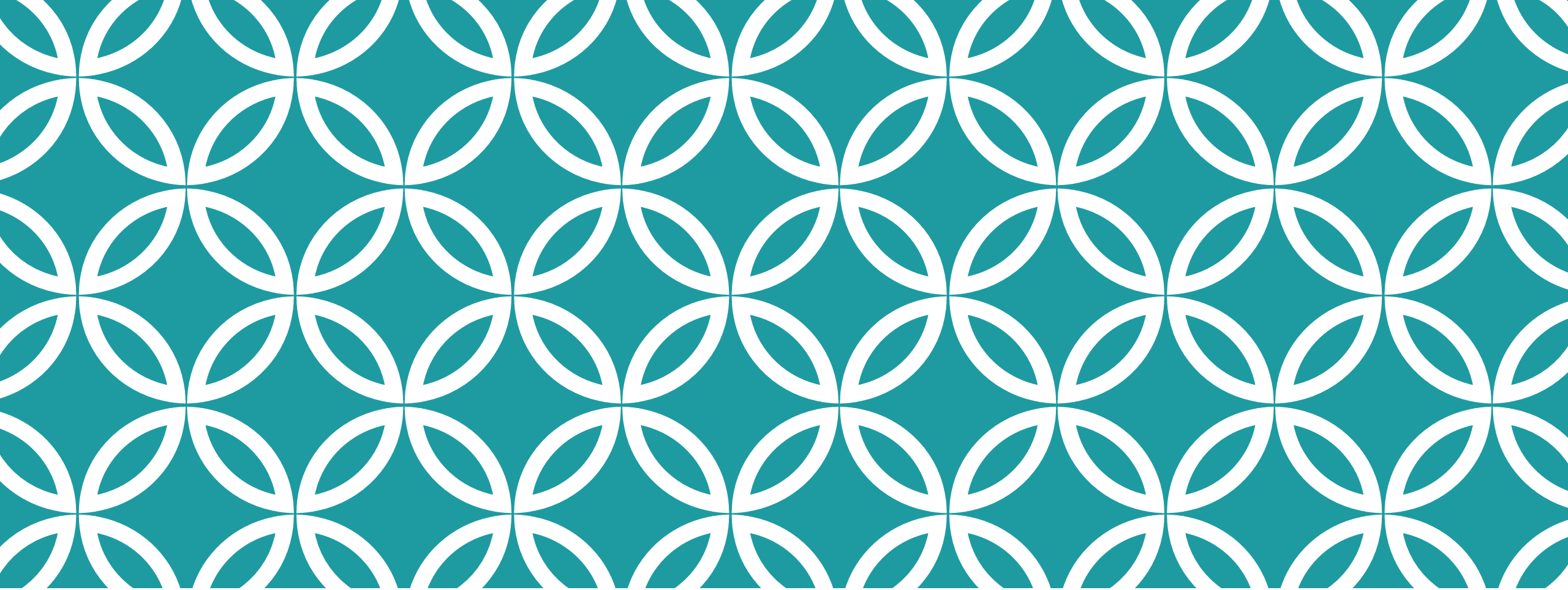
Dans la vue de **HomeController**, afficher

- « Bonjour administrateur » si c’est un administrateur
- « Bonjour utilisateur » si c’est un utilisateur

EXERCICE

Modifier les autorisations pour **ProduitController**

- Les utilisateurs peuvent voir la liste des produits
  - Sans les boutons d'action
  - Sans le lien « ajouter un produit »



LES PERMISSIONS

Aller plus loin avec les permissions

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

AUTORISATIONS – HASPERMISSION

Plutôt que d'utiliser

```
@PreAuthorize("hasAuthority('PRODUIT_READ_PERMISSION')")
public String findAll() {
    //...
}
```

On pourra utiliser

```
@PreAuthorize("hasPermission(null, 'Produit', 'read')")
public String findAll() {
    //...
}
```

```
@PreAuthorize("hasPermission(#id, 'Produit', 'read')")
public Produit findById(int id) {
    //...
}
```

```
@PostAuthorize("hasPermission(returnObject, 'read')")
public Produit findById(int id) {
    //...
}
```

AUTORISATIONS – HASPERMISSION

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

```
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

```
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

```
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

- Tout d'abord au niveau de *GlobalMethodSecurity*

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

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

AUTORISATIONS – HASPERMISSION

```
@Configuration
@EnableGlobalMethodSecurity(prePostEnabled=true)
public class SecurityMethodConfig extends GlobalMethodSecurityConfiguration {
  protected MethodSecurityExpressionHandler createExpressionHandler() {
    DefaultMethodSecurityExpressionHandler expressionHandler = new DefaultMethodSecurityExpressionHandler();

    expressionHandler.setPermissionEvaluator(new FormationPermissionEvaluator());
    return expressionHandler;
  }
}
```

AUTORISATIONS – HASPERMISSION

- Puis au niveau Sécurité globale si on veut l'activer au niveau JSP

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

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

AUTORISATIONS – HASPERMISSION

```
@Configuration
@EnableWebSecurity
public class SecurityConfig extends WebSecurityConfigurerAdapter {
  protected void configure(WebSecurity web) {
    DefaultWebSecurityExpressionHandler webExpressionHandler = new DefaultWebSecurityExpressionHandler();

    webExpressionHandler.setPermissionEvaluator(new FormationPermissionEvaluator());
    web.expressionHandler(webExpressionHandler);
  }

  //...
}
```

AUTORISATIONS – HASPERMISSION

Exemple global

```
@Configuration
@EnableWebSecurity
public class SecurityConfig extends WebSecurityConfigurerAdapter {
  protected void configure(WebSecurity web) {
    DefaultWebSecurityExpressionHandler webExpressionHandler = new DefaultWebSecurityExpressionHandler();

    webExpressionHandler.setPermissionEvaluator(permissionEvaluator());
    web.expressionHandler(webExpressionHandler);
  }

  @Bean
  public PermissionEvaluator permissionEvaluator() {
    return new FormationPermissionEvaluator();
  }

  //...
}
```

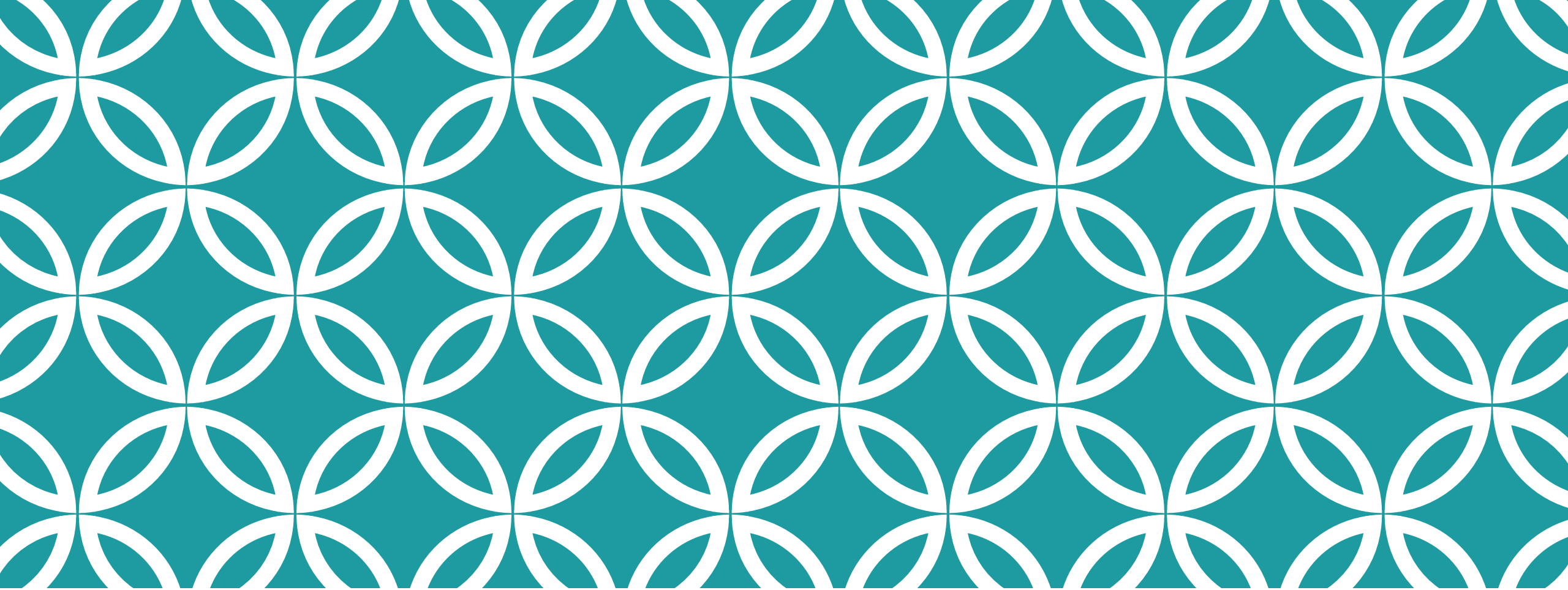
AUTORISATIONS – HASPERMISSION

```
@Configuration
@EnableGlobalMethodSecurity(prePostEnabled=true)
public class SecurityMethodConfig extends GlobalMethodSecurityConfiguration {
  @Autowired
  private PermissionEvaluator permissionEvaluator;

  protected MethodSecurityExpressionHandler createExpressionHandler() {
    DefaultMethodSecurityExpressionHandler expressionHandler = new DefaultMethodSecurityExpressionHandler();

    expressionHandler.setPermissionEvaluator(permissionEvaluator);
    return expressionHandler;
  }
}
```

EXERCICE

Mettre en place la sécurité via des permissions

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



LES EXPRESSIONS

Evaluer ses expressions

AUTORISATIONS – EXPRESSIONS

Il est possible, parce qu'on est limité dans la syntaxe des permissions

- De créer de nouvelles expressions à utiliser en complément de *hasPermission*

Créer une nouvelle classe qui

- Hérite de **SecurityExpressionRoot**
- Implémente **MethodSecurityExpressionOperations**
- Eventuellement avec les attributs *filterObject*, *returnObject*, *target*, *method*
- Défini son constructeur en appelant le constructeur parent (car classe mère abstraite)
- Défini les méthodes de l'interface
- Défini les méthodes supplémentaires qui seront les nouvelles expressions

AUTORISATIONS – EXPRESSIONS

Plutôt que d'utiliser

```
@PreAuthorize("hasRole('ADMIN') or hasPermission(#id, 'Produit', 'read')")
public Produit findById(int id) {
    //...
}
```

On pourra utiliser (par exemple)

```
@PreAuthorize("isAdmin() or isReadable(#id, 'Produit')")
public Produit findById(int id) {
    //...
}
```

```
@PostAuthorize("isAdmin() or isReadable(returnObject)")
public Produit findById(int id) {
    //...
}
```

AUTORISATIONS – HASPERMISSION

```
public class FormationMethodSecurityExpressionRoot extends SecurityExpressionRoot
    implements MethodSecurityExpressionOperations {
  private Authentication authentication;
  private Object target;
  private Object filterObject;
  private Object returnObject;
  private Method method;

  public FormationMethodSecurityExpressionRoot(Authentication authentication) {
    super(authentication);
    this.authentication = authentication;
  }

  //...

  public boolean isAdmin() {
    //... authentication.getAuthorities()...
  }

  public boolean isReadable(Object object) {
    //...
  }
}
```

AUTORISATIONS – EXPRESSIONS

Si on veut rendre cette expression disponible dans les annotations des méthodes

- Créer une nouvelle classe qui hérite de **DefaultMethodSecurityExpressionHandler**
- Instancier cette classe dans la méthode *createExpressionHandler* de la configuration

```
@Configuration
@EnableGlobalMethodSecurity(prePostEnabled=true)
public class SecurityMethodConfig extends GlobalMethodSecurityConfiguration {
  @Autowired
  private PermissionEvaluator permissionEvaluator;

  protected MethodSecurityExpressionHandler createExpressionHandler() {
    DefaultMethodSecurityExpressionHandler expressionHandler = new FormationMethodSecurityExpressionHandler();

    expressionHandler.setPermissionEvaluator(permissionEvaluator);
    return expressionHandler;
  }
}
```

AUTORISATIONS – EXPRESSIONS

```
public class FormationMethodSecurityExpressionHandler extends DefaultMethodSecurityExpressionHandler {
  protected MethodSecurityExpressionOperations createSecurityExpressionRoot(Authentication authentication,
      MethodInvocation invocation) {
    FormationMethodSecurityExpressionRoot root = new FormationMethodSecurityExpressionRoot(authentication);

    root.setThis(invocation.getThis());
    root.setPermissionEvaluator(getPermissionEvaluator());
    root.setTrustResolver(getTrustResolver());
    root.setRoleHierarchy(getRoleHierarchy());
    root.setDefaultRolePrefix(getDefaultRolePrefix());
    root.setMethod(invocation.getMethod());

    return root;
  }
}
```

AUTORISATIONS – EXPRESSIONS

Si on veut rendre cette expression disponible dans les JSP

- Créer une nouvelle classe qui hérite de **DefaultWebSecurityExpressionHandler**
- Instancier cette classe dans la méthode *configure* de la configuration

```
@Configuration
@EnableWebSecurity
public class SecurityConfig extends WebSecurityConfigurerAdapter {
  protected void configure(WebSecurity web) {
    DefaultWebSecurityExpressionHandler webExpressionHandler = new FormationWebSecurityExpressionHandler();

    webExpressionHandler.setPermissionEvaluator(permissionEvaluator());
    web.expressionHandler(webExpressionHandler);
  }

  //...
}
```

AUTORISATIONS – EXPRESSIONS

```
public class FormationWebSecurityExpressionHandler extends DefaultWebSecurityExpressionHandler {
  private AuthenticationTrustResolver trustResolver = new AuthenticationTrustResolver();
  private String defaultRolePrefix = "ROLE_";

  protected SecurityExpressionOperations createSecurityExpressionRoot(Authentication authentication,
      FilterInvocation fi) {
    FormationMethodSecurityExpressionRoot root = new FormationMethodSecurityExpressionRoot(authentication);

    root.setPermissionEvaluator(getPermissionEvaluator());
    root.setTrustResolver(trustResolver);
    root.setRoleHierarchy(getRoleHierarchy());
    root.setDefaultRolePrefix(defaultRolePrefix);

    return root;
  }
}
```