PnT

Spring MVC

Yohan Boichut

Version 2.0

Sommaire

Who am i

Yohan Boichut, yohan.boichut@univ-orleans.fr

@YohanBoichut

yohanboichut

http://bit.ly/Yoh4n

Chapter 1. Introduction à Spring Boot et au modèle MVC

1.1. Objectifs

- Comprendre le rôle de Spring Boot dans une application web MVC
- Créer une application Spring Boot minimale
- Afficher une première vue HTML avec données dynamiques

1.2. Présentation de Spring Boot

- Spring Boot est un framework basé sur Spring qui facilite le développement d'applications Java autonomes et prêtes à l'emploi. Il propose :
  - Une configuration automatique (auto-configuration)
  - Une structure de projet simplifiée
  - Un serveur embarqué (Tomcat par défaut)

1.3. Structure d'un projet Spring Boot

- Un projet Spring Boot généré via https://start.spring.io contient généralement :
  - `src/main/java` : le code source Java
  - `src/main/resources` : les ressources (fichiers HTML, CSS, application.properties)
  - `pom.xml` : fichier de configuration Maven
- L'entrée principale de l'application contient :

```
@SpringBootApplication
public class Application {
public static void main(String\[] args) {
SpringApplication.run(Application.class, args);
}
}
```

1.4. Première vue avec Thymeleaf

- Spring Boot utilise Thymeleaf comme moteur de template par défaut.
- Les fichiers HTML doivent être placés dans `src/main/resources/templates`.

```
@Controller
public class AccueilController {
```

```
@GetMapping("/")
public String accueil(Model model) {
    model.addAttribute("message", "Bienvenue dans Spring MVC !");
    return "accueil";
}


}
```

Fichier `accueil.html` :

```
<!DOCTYPE html>

<html xmlns:th="http://www.thymeleaf.org">
<head><title>Accueil</title></head>
<body>
    <h1 th:text="${message}">Message par défaut</h1>
</body>
</html>
```

1.5. Hello World MVC

1. Créer un projet sur start.spring.io avec les dépendances : Spring Web, Thymeleaf
2. Créer un contrôleur avec une route `/`
3. Passer une variable au modèle
4. Créer une vue `accueil.html` pour l'afficher

Chapter 2. Contrôleurs, routes et paramètres

2.1. Objectifs

- Comprendre le rôle du contrôleur dans Spring MVC
- Manipuler les routes avec paramètres GET
- Utiliser `Model` pour transmettre des données aux vues

2.2. Le rôle du contrôleur (`@Controller`)

- Un contrôleur gère les requêtes HTTP et choisit quelle vue retourner. Il agit comme intermédiaire entre le modèle (données) et la vue (interface).

2.3. Routage avec `@GetMapping` et paramètres

- Spring permet de gérer les routes HTTP avec des annotations simples
  - `@GetMapping` : pour les requêtes GET
  - `@PostMapping` : pour les requêtes POST
  - `@RequestParam` : pour récupérer les paramètres de requête
  - `@PathVariable` : pour récupérer des informations dans l'URL (par exemple, `/salut/{prenom}`)
  - `@Controller` : pour déclarer une classe comme contrôleur

2.4. Classes à connaître

- Pour une fonction d'un contrôleur, on peut utiliser des paramètres connus qui sont injectés par Spring
  - `Model` : pour ajouter des attributs à la vue (ou en récupérer)
  - `HttpSession` : pour accéder à la session HTTP
  - `ModelAttribute` : pour accéder directement à un attribut de modèle. Utilisé en particulier pour lier des données de formulaire à un objet Java

2.5. Exemple

```
@Controller
public class BonjourController {


@GetMapping("/bonjour")
public String direBonjour(@RequestParam(defaultValue = "inconnu") String nom, Model
```

```
model) {
    model.addAttribute("nom", nom);
    return "bonjour";
}


}
```

Vue `bonjour.html` :

```
<!DOCTYPE html>

<html xmlns:th="http://www.thymeleaf.org">
<head><title>Bonjour</title></head>
<body>
<p th:text="'Bonjour ' + ${nom} + ' !'"></p>
</body>
</html>
```

Accès via : `/bonjour?nom=Alice`

2.6. Paramètres dans l'URL avec `@PathVariable`

```
@GetMapping("/salut/{prenom}")
public String salut(@PathVariable String prenom, Model model) {
model.addAttribute("prenom", prenom);
return "salut";
}
```

URL : `/salut/Jean`

2.7. Retour d'une méthode de contrôleur Spring MVC

2.8. String : nom de la vue

- C'est le cas classique : le nom retourné correspond au fichier `.html` (ou `.jsp`) dans `src/main/resources/templates`.

```
return "accueil"; // affichera accueil.html via Thymeleaf
```

2.9. String commençant par "redirect:"

- Effectue une redirection HTTP vers une autre URL.

```
return "redirect:/accueil"; // déclenche une autre requête qui sera traitée par
                            // le contrôleur gérant /accueil
```

2.10. String commençant par "forward:"

- Transfert interne à une autre route (sans changement d'URL côté client).

```
return "forward:/accueil"; // fait suivre la requête sur une autre URI
                           // sans redirection HTTP
```

2.11. ModelAndView

- Permet de spécifier à la fois :
  - le nom de la vue,
  - et les données (modèle) à y injecter.

```
return new ModelAndView("accueil").addObject("nom", "Alice");
```

2.12. void

- Par défaut, Spring déduit le nom de la vue à partir de l'URL de la requête.

```
@GetMapping("/home")
public void afficher(Model model) {
model.addAttribute("msg", "Hello");
}
// View attendue : home.html
```

2.13. Récapitulatif

Retour	Comportement
"vue"	Affiche la vue nommée vue.html
"redirect:/x"	Redirige vers /x
"forward:/x"	Transfert interne vers /x
ModelAndView	Combine modèle + nom de vue
void	Vue déduite de l'URL

2.14. Exemple : Page de salutation

1. Créer un contrôleur `BonjourController`
2. Ajouter une route `/bonjour` qui accepte un paramètre `nom`
3. Afficher ce nom dans la vue avec Thymeleaf
4. Bonus : route `/salut/{prenom}` avec `@PathVariable`

Chapter 3. Écrire des vues

3.1. Introduction aux vues

- Les vues dans Spring MVC sont responsables du rendu de l'interface utilisateur.
- Spring MVC support plusieurs technologies de vue, chacune avec ses avantages.

3.2. Technologies de vue supportées

- **JSP** : Pages JavaServer traditionnelles
- **Thymeleaf** : Moteur de template moderne
- **Freemarker** : Moteur de template flexible
- **Mustache** : Template logic-less
- **Velocity** : Moteur de template (déprécié)
- **JSON/XML** : Pour les API REST

3.3. Thymeleaf

3.4. Configuration pour Thymeleaf

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-thymeleaf</artifactId>
</dependency>
```

3.5. Définir un hyperlien avec Thymeleaf

```
<!--Permet de définir une URL sans paramètres-->
<a th:href="@{/chemin}">Texte du lien</a>

<!--Permet de définir une URL avec paramètres-->
<a th:href="@{/chemin(param1=val1&...&parami=vali)}">Texte du lien</a>

<!--Permet de définir une URL dynamiquement, en particulier ici
des paramètres, en fonction de la valeur stockée dans la variable x -->
<a th:href="@{chemin(id=${e.identifiant})}">Texte du lien</a>

<!--Permet de définir une URL dynamiquement (ici un chemin)
en fonction de la valeur stockée dans la variable x -->
<a th:href="@{chemin/${e.identifiant}}">Texte du lien</a>
```

3.6. th:if

- Cet attribut permet de conditionner l'apparition de n'importe quel tag HTML

```
<!--Le bloc div apparaîtra côté client uniquement si erreur n'est pas null et n'est
pas vide -->
<div th:if="${erreur != null && erreur != ''}" class="error-message" th:text=
"${erreur}"></div>
```

3.7. th:text

- Cet attribut permet d'insérer dynamiquement du texte dans un élément HTML, en remplaçant son contenu.

```
<!-- Affiche la valeur de l'attribut nom -->
<p th:text="${utilisateur.nom}">Nom par défaut</p>
```

3.8. th:each

- Permet d'itérer sur une collection d'objets et de répéter un élément HTML pour chaque élément.

```
<!-- Itère sur la liste utilisateurs et affiche le nom de chacun -->
<ul>
  <li th:each="u : ${utilisateurs}" th:text="${u.nom}">Nom</li>
</ul>
```

3.9. th:value

- Utilisé pour définir la valeur d'un champ de formulaire (`input`, `textarea`, etc.).

```
<input type="text" th:value="${utilisateur.nom}" />
```

3.10. th:action

- Permet de spécifier dynamiquement l'attribut `action` d'un formulaire (l'URL vers laquelle il est soumis).

```
<form th:action="@{/soumettre}" method="post">
  ...
</form>
```

3.11. th:object et *{}

- Permet de lier un objet formulaire global à un formulaire HTML, avec un accès simplifié aux propriétés via `*{…}`.

```
<form th:action="@{/soumettre}" th:object="${utilisateur}" method="post">
  <input type="text" th:field="*{nom}" />
  <input type="email" th:field="*{email}" />
</form>
```

3.12. th:field

- Alternative pratique à `th:value`, utilisée avec `th:object`, elle définit automatiquement le `name` et la `value` du champ.

```
<input type="text" th:field="*{nom}" />
```

3.13. Exemple de vue Thymeleaf

```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org">
<head>
    <title>Liste des utilisateurs</title>
    <meta charset="UTF-8">
    <link rel="stylesheet" th:href="@{/css/style.css}">
</head>
<body>
    <h1>Utilisateurs</h1>

    <div th:if="${message}" class="alert alert-success" th:text="${message}"></div>

    <table>
        <thead>
            <tr>
                <th>ID</th>
                <th>Nom</th>
                <th>Email</th>
                <th>Actions</th>
            </tr>
        </thead>
        <tbody>
            <tr th:each="user : ${users}">
                <td th:text="${user.id}">1</td>
                <td th:text="${user.name}">John Doe</td>
                <td th:text="${user.email}">john@example.com</td>
                <td>
```

```
                    <a th:href="@{/users/{id}(id=${user.id})}">Voir</a>
                    <a th:href="@{/users/{id}/edit(id=${user.id})}">Modifier</a>
                    <form th:action="@{/users/{id}(id=${user.id})}" method="post"
style="display:inline;">
                        <input type="hidden" name="_method" value="delete">
                        <button type="submit" onclick="return confirm('Êtes-vous sûr?
')">Supprimer</button>
                    </form>
                </td>
            </tr>
        </tbody>
    </table>

    <a th:href="@{/users/new}" class="btn btn-primary">Nouvel utilisateur</a>
</body>
</html>
```

3.14. Formulaires avec Thymeleaf

```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org">
<head>
    <title>Formulaire utilisateur</title>
    <meta charset="UTF-8">
</head>
<body>
    <h1>Créer un utilisateur</h1>

    <form th:action="@{/users}" th:object="${user}" method="post">
        <div>
            <label for="name">Nom:</label>
            <input type="text" id="name" th:field="*{name}" />
            <span th:if="${#fields.hasErrors('name')}" th:errors="*{name}" class=
"error"></span>
        </div>

        <div>
            <label for="email">Email:</label>
            <input type="email" id="email" th:field="*{email}" />
            <span th:if="${#fields.hasErrors('email')}" th:errors="*{email}" class=
"error"></span>
        </div>

        <div>
            <label for="age">Âge:</label>
            <input type="number" id="age" th:field="*{age}" />
            <span th:if="${#fields.hasErrors('age')}" th:errors="*{age}" class="error
"></span>
        </div>
```

```
        <div>
            <label for="country">Pays:</label>
            <select id="country" th:field="*{country}">
                <option value="">Sélectionner...</option>
                <option th:each="country : ${countries}"
                        th:value="${country.code}"
                        th:text="${country.name}">France</option>
            </select>
        </div>

        <div>
            <input type="checkbox" id="active" th:field="*{active}" />
            <label for="active">Actif</label>
        </div>

        <button type="submit">Enregistrer</button>
        <a th:href="@{/users}">Annuler</a>
    </form>

    <script th:src="@{/js/form-validation.js}"></script>
</body>
</html>
```

3.15. Utilitaires Thymeleaf courants (préfixe #)

- Thymeleaf propose des objets utilitaires accessibles via le préfixe `#` dans les expressions (souvent utilisés dans `th:if`, `th:each`, `th:text`, etc.).

3.16. #dates

- Permet de manipuler les dates (formatage, comparaison...).

```
<!-- Affiche la date du jour formatée -->
<span th:text="${#dates.format(today, 'dd/MM/yyyy')}"></span>
```

3.17. #numbers

- Permet de formater des nombres (entiers, décimaux, monétaires...).

```
<!-- Affiche le prix avec deux décimales -->
<span th:text="${#numbers.formatDecimal(prix, 1, 2)}"></span>
```

3.18. #strings

- Fonctions sur les chaînes de caractères (longueur, majuscules, tests, etc.).

```
<!-- Affiche le nom en majuscules -->
<span th:text="${#strings.toUpperCase(utilisateur.nom)}"></span>
```

3.19. #lists

- Opérations sur les listes (tri, fusion, recherche…).

```
<!-- Itère sur la liste triée -->
<ul>
  <li th:each="e : ${#lists.sort(utilisateurs)}" th:text="${e.nom}"></li>
</ul>
```

3.20. #sets

- Utilitaire pour manipuler des ensembles (`Set` Java).

```
<!-- Crée un set -->
<div th:with="monSet=${#sets.set('A', 'B', 'C')}">
  <span th:each="x : ${monSet}" th:text="${x}"></span>
</div>
```

3.21. #maps

- Opérations sur les maps (`Map` Java).

```
<!-- Affiche une valeur de map -->
<span th:text="${#maps.get(mesDonnees, 'clé')}"></span>
```

3.22. #bools

- Fonctions logiques utilitaires.

```
<!-- Vérifie si une valeur est vraie -->
<span th:if="${#bools.isTrue(flag)}">Activé</span>
```

3.23. #arrays

- Fonctions utiles sur les tableaux Java.

```
<!-- Affiche la taille du tableau -->
<span th:text="${#arrays.length(monTableau)}"></span>
```

3.24. #objects

- Fonctions génériques (nullité, classe, etc.).

```
<!-- Vérifie si l'objet est vide -->
<span th:if="${#objects.isEmpty(objet)}">Vide</span>
```

3.25. #ctx

- Contexte Thymeleaf : donne accès à des infos globales sur le contexte de rendu.

```
<!-- Affiche le nom de l'application -->
<span th:text="${#ctx.contextPath}"></span>
```

3.26. #request, #session, #servletContext

- Accès direct aux objets `HttpServletRequest`, `HttpSession`, et `ServletContext`.

```
<!-- Affiche un paramètre de requête -->
<span th:text="${#request.getParameter('id')}"></span>

<!-- Accès à un attribut de session -->
<span th:text="${#session.getAttribute('user')}"></span>
```

3.27. #execInfo

- Informations sur le template en cours d'exécution.

```
<!-- Affiche le nom du template courant -->
<span th:text="${#execInfo.templateName}"></span>
```

3.28. Layout et templates

3.29. Utiliser thymeleaf-layout-dialect pour gérer les layouts avec Thymeleaf

3.30. Introduction

- Thymeleaf ne propose pas nativement une fonctionnalité de layout comme `@extends` en Twig ou `th:replace` avec gestion d'héritage.
- Le dialecte `thymeleaf-layout-dialect` ajoute cette capacité, permettant de créer des **templates maîtres réutilisables** avec des blocs (`layout:fragment`) que les vues filles peuvent remplacer.

3.31. Dépendance Maven

- Ajoutez la dépendance suivante dans votre `pom.xml` :

```
<dependency>
    <groupId>nz.net.ultraq.thymeleaf</groupId>
    <artifactId>thymeleaf-layout-dialect</artifactId>
</dependency>
```

3.32. Configuration (Spring Boot)

- Spring Boot configure automatiquement le dialecte si la dépendance est présente.
- Pour Spring MVC classique, ajoutez-le manuellement à votre moteur Thymeleaf :

```
@Bean
public SpringTemplateEngine templateEngine() {
    SpringTemplateEngine engine = new SpringTemplateEngine();
    engine.setTemplateResolver(templateResolver());
    engine.addDialect(new LayoutDialect()); // ajout du dialecte ici
    return engine;
}
```

3.33. Créer un layout principal (ex: `layout.html`)

```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org"
      xmlns:layout="http://www.ultraq.net.nz/thymeleaf/layout">
<head>
    <meta charset="UTF-8">
    <title layout:title-pattern="$DECORATOR_TITLE - Mon Site">Mon Site</title>
    <link rel="stylesheet" href="/css/styles.css"/>
</head>
<body>
```

```
<header>
    <h1>Mon site</h1>
</header>

<nav>
    <!-- menu commun -->
</nav>

<main layout:fragment="content">
    <!-- contenu par défaut ou vide -->
</main>

<footer>
    &copy; 2025 - Mon site
</footer>

</body>
</html>
```

3.34. Utiliser le layout dans une vue (ex: `home.html`)

```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org"
      xmlns:layout="http://www.ultraq.net.nz/thymeleaf/layout"
      layout:decorate="~{layout}">
<head>
    <title layout:title="Accueil">Accueil</title>
</head>
<body>

<section layout:fragment="content">
    <h2>Bienvenue !</h2>
    <p>Contenu spécifique à la page d'accueil.</p>
</section>

</body>
</html>
```

3.35. Résultat

Lors du rendu

- Le contenu de `home.html` sera injecté dans le `<main layout:fragment="content">…</main>` du `layout.html`.
- Le `<title>` de la vue s'insère dans celui du layout grâce à `layout:title-pattern`.

3.36. Bonnes pratiques

- Créez un layout général (`layout.html`) et des sous-layouts si nécessaire (`admin-layout.html`, etc.)
- Utilisez des fragments réutilisables pour les entêtes, menus, messages flash, etc.
- Combinez avec `th:if` et `th:replace` pour adapter dynamiquement certains éléments.

3.37. Exemple de structure de fichiers

```
src/
  └──── main/
      └──── resources/
          └──── templates/
              ├──── layout.html
              ├──── home.html
              ├──── fragments/
              │     └──── navbar.html
              └──── admin/
                    └──── dashboard.html
```

3.38. Références pour les layouts

- https://ultraq.github.io/thymeleaf-layout-dialect
- https://github.com/ultraq/thymeleaf-layout-dialect

Chapter 4. Internationalisation (i18n) dans Spring MVC

4.1. Objectifs du chapitre

À la fin de ce chapitre, vous serez capable de :

- Configurer l'internationalisation dans Spring MVC
- Créer et organiser les fichiers de messages
- Utiliser les résolveurs de locale
- Implémenter le changement de langue dynamique
- Internationaliser les messages d'erreur et de validation
- Formater les dates, nombres et devises selon la locale

4.2. Introduction à l'internationalisation

L'internationalisation (i18n) est le processus de conception d'une application pour qu'elle puisse être adaptée à différentes langues et régions sans modifications du code. Spring MVC fournit un support complet pour l'i18n.

4.3. Concepts clés

- **Locale** : Combinaison de langue et région (ex: fr_FR, en_US)
- **MessageSource** : Interface pour résoudre les messages
- **LocaleResolver** : Détermine la locale courante
- **LocaleChangeInterceptor** : Permet de changer la locale dynamiquement

4.4. Configuration de base

4.5. Configuration Java

```
@Configuration
@EnableWebMvc

public class WebConfig implements WebMvcConfigurer {

    /**
     * Configuration du MessageSource pour les messages i18n
     */
    @Bean
    public MessageSource messageSource() {
```

```
        ResourceBundleMessageSource messageSource = new ResourceBundleMessageSource();
        messageSource.setBasename("messages");
        messageSource.setDefaultEncoding("UTF-8");
        messageSource.setCacheSeconds(3600); // Cache pendant 1 heure
        messageSource.setFallbackToSystemLocale(false);
        return messageSource;
    }

    /**
     * Résolveur de locale basé sur les cookies
     */
    @Bean
    public LocaleResolver localeResolver() {
        CookieLocaleResolver resolver = new CookieLocaleResolver();
        resolver.setDefaultLocale(Locale.FRENCH);
        resolver.setCookieName("app-locale");
        resolver.setCookieMaxAge(3600 * 24 * 30); // 30 jours
        resolver.setCookiePath("/");
        return resolver;
    }

    /**
     * Intercepteur pour changer la locale
     */
    @Bean
    public LocaleChangeInterceptor localeChangeInterceptor() {
        LocaleChangeInterceptor interceptor = new LocaleChangeInterceptor();
        interceptor.setParamName("lang");
        return interceptor;
    }

    @Override
    public void addInterceptors(InterceptorRegistry registry) {
        registry.addInterceptor(localeChangeInterceptor());
    }
}
```

4.6. Fichiers de messages

4.7. Structure des fichiers

Créez les fichiers de propriétés dans `src/main/resources/` :

```
messages.properties           # Fichier par défaut
messages_fr.properties        # Français
messages_en.properties        # Anglais
messages_de.properties        # Allemand
messages_es.properties        # Espagnol
```

4.8. Contenu des fichiers de messages

4.8.1. messages.properties (par défaut)

```
# Navigation
nav.home=Accueil
nav.products=Produits
nav.about=À propos
nav.contact=Contact
nav.login=Connexion
nav.logout=Déconnexion

# Messages génériques
welcome=Bienvenue
hello=Bonjour {0}
goodbye=Au revoir
yes=Oui
no=Non
save=Enregistrer
cancel=Annuler
delete=Supprimer
edit=Modifier

# Messages d'erreur
error.required=Ce champ est obligatoire
error.email.invalid=Adresse email invalide
error.password.mismatch=Les mots de passe ne correspondent pas
error.user.notfound=Utilisateur non trouvé
error.access.denied=Accès refusé

# Messages de validation
validation.size.min=Minimum {0} caractères requis
validation.size.max=Maximum {0} caractères autorisés
validation.range=La valeur doit être entre {0} et {1}

# Formats
date.format=dd/MM/yyyy
datetime.format=dd/MM/yyyy HH:mm
currency.symbol=€
```

4.8.2. messages_en.properties

```
# Navigation
nav.home=Home
nav.products=Products
nav.about=About
nav.contact=Contact
nav.login=Login
```

```
nav.logout=Logout

# Messages génériques
welcome=Welcome
hello=Hello {0}
goodbye=Goodbye
yes=Yes
no=No
save=Save
cancel=Cancel
delete=Delete
edit=Edit

# Messages d'erreur
error.required=This field is required
error.email.invalid=Invalid email address
error.password.mismatch=Passwords do not match
error.user.notfound=User not found
error.access.denied=Access denied

# Messages de validation
validation.size.min=Minimum {0} characters required
validation.size.max=Maximum {0} characters allowed
validation.range=Value must be between {0} and {1}

# Formats
date.format=MM/dd/yyyy
datetime.format=MM/dd/yyyy HH:mm
currency.symbol=$
```

4.8.3. messages_de.properties

```
# Navigation
nav.home=Startseite
nav.products=Produkte
nav.about=Über uns
nav.contact=Kontakt
nav.login=Anmelden
nav.logout=Abmelden

# Messages génériques
welcome=Willkommen
hello=Hallo {0}
goodbye=Auf Wiedersehen
yes=Ja
no=Nein
save=Speichern
cancel=Abbrechen
delete=Löschen
```

```
edit=Bearbeiten

# Messages d'erreur
error.required=Dieses Feld ist erforderlich
error.email.invalid=Ungültige E-Mail-Adresse
error.password.mismatch=Passwörter stimmen nicht überein
error.user.notfound=Benutzer nicht gefunden
error.access.denied=Zugang verweigert

# Messages de validation
validation.size.min=Mindestens {0} Zeichen erforderlich
validation.size.max=Höchstens {0} Zeichen erlaubt
validation.range=Wert muss zwischen {0} und {1} liegen

# Formats
date.format=dd.MM.yyyy
datetime.format=dd.MM.yyyy HH:mm
currency.symbol=€
```

4.9. Utilisation dans les contrôleurs

4.10. Injection du MessageSource

```
@Controller
public class HomeController {

    @Autowired
    private MessageSource messageSource;

    @RequestMapping("/")
    public String home(Model model, Locale locale) {
        String welcomeMessage = messageSource.getMessage("welcome", null, locale);
        String helloMessage = messageSource.getMessage("hello",
                                                       new Object[]{"Jean"},
                                                       locale);

        model.addAttribute("welcomeMessage", welcomeMessage);
        model.addAttribute("helloMessage", helloMessage);

        return "home";
    }

    @RequestMapping("/user/{username}")
    public String userProfile(@PathVariable String username,
                              Model model, Locale locale) {
        try {
            User user = userService.findByUsername(username);
            model.addAttribute("user", user);
```

```
            String pageTitle = messageSource.getMessage("user.profile.title",
                                                        new Object[]{user.
getDisplayName()},
                                                        locale);
            model.addAttribute("pageTitle", pageTitle);

        } catch (UserNotFoundException e) {
            String errorMessage = messageSource.getMessage("error.user.notfound",
                                                           null, locale);
            model.addAttribute("errorMessage", errorMessage);
            return "error";
        }

        return "user/profile";
    }
}
```

4.11. Service d'internationalisation

```
@Service
public class MessageService {

    @Autowired
    private MessageSource messageSource;

    public String getMessage(String code, Locale locale) {
        return messageSource.getMessage(code, null, locale);
    }

    public String getMessage(String code, Object[] args, Locale locale) {
        return messageSource.getMessage(code, args, locale);
    }

    public String getMessageWithDefault(String code, String defaultMessage, Locale
locale) {
        return messageSource.getMessage(code, null, defaultMessage, locale);
    }

    // Méthodes utilitaires
    public String getValidationMessage(String fieldName, String validationType,
                                       Object[] args, Locale locale) {
        String code = "validation." + validationType;
        return messageSource.getMessage(code, args, locale);
    }

    public String getErrorMessage(String errorCode, Locale locale) {
        return messageSource.getMessage("error." + errorCode, null, locale);
    }
```

```
}
```

4.12. Utilisation dans les vues Thymeleaf

4.13. Messages simples

Utilisation des messages dans une page HTML Thymeleaf :

```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org">
<head>
    <title th:text="#{nav.home}">Accueil</title>
</head>
<body>
    <nav>
        <ul>
            <li><a href="/" th:text="#{nav.home}">Accueil</a></li>
            <li><a href="/products" th:text="#{nav.products}">Produits</a></li>
            <li><a href="/about" th:text="#{nav.about}">À propos</a></li>
            <li><a href="/contact" th:text="#{nav.contact}">Contact</a></li>
        </ul>
    </nav>

    <h1 th:text="#{welcome}">Bienvenue</h1>

    <!-- Message avec paramètres -->
    <p th:text="#{hello(${user.name})}">Bonjour, utilisateur</p>

    <!-- Message avec valeur par défaut -->
    <p th:text="#{custom.message} ?: 'Message par défaut'">Message par défaut</p>
</body>
</html>
```

4.14. Sélecteur de langue

Voici un exemple de sélecteur de langue simple, changeant la langue via une requête `GET` :

```
<div class="language-selector">
    <label th:text="#{language.select}">Choisir la langue :</label>
    <select onchange="changeLanguage(this.value)">
        <option value="fr" th:selected="${#locale.language == 'fr'}">Français</option>
        <option value="en" th:selected="${#locale.language == 'en'}">English</option>
        <option value="de" th:selected="${#locale.language == 'de'}">Deutsch</option>
    </select>
</div>
```

```
<script>
function changeLanguage(lang) {
    window.location.href = window.location.pathname + '?lang=' + lang;
}
</script>
```

4.15. Résolveurs de locale

4.16. CookieLocaleResolver

```
@Bean
public LocaleResolver cookieLocaleResolver() {
    CookieLocaleResolver resolver = new CookieLocaleResolver();
    resolver.setDefaultLocale(Locale.FRENCH);
    resolver.setCookieName("user-locale");
    resolver.setCookieMaxAge(3600 * 24 * 365); // 1 an
    resolver.setCookiePath("/");
    resolver.setCookieHttpOnly(true);
    return resolver;
}
```

4.17. SessionLocaleResolver

```
@Bean
public LocaleResolver sessionLocaleResolver() {
    SessionLocaleResolver resolver = new SessionLocaleResolver();
    resolver.setDefaultLocale(Locale.FRENCH);
    return resolver;
}
```

4.18. AcceptHeaderLocaleResolver

```
@Bean
public LocaleResolver headerLocaleResolver() {
    AcceptHeaderLocaleResolver resolver = new AcceptHeaderLocaleResolver();
    resolver.setDefaultLocale(Locale.FRENCH);
    resolver.setSupportedLocales(Arrays.asList(
        Locale.FRENCH, Locale.ENGLISH, Locale.GERMAN
    ));
    return resolver;
}
```

4.19. FixedLocaleResolver

```
@Bean
public LocaleResolver fixedLocaleResolver() {
    FixedLocaleResolver resolver = new FixedLocaleResolver();
    resolver.setDefaultLocale(Locale.FRENCH);
    return resolver;
}
```

4.20. Contrôleur de changement de langue

```
@Controller
public class LanguageController {

    @Autowired
    private LocaleResolver localeResolver;

    @RequestMapping("/changeLanguage")
    public String changeLanguage(@RequestParam("lang") String language,
                                 HttpServletRequest request,
                                 HttpServletResponse response) {

        Locale locale = new Locale(language);
        localeResolver.setLocale(request, response, locale);

        // Redirection vers la page précédente
        String referer = request.getHeader("Referer");
        return "redirect:" + (referer != null ? referer : "/");
    }

    @RequestMapping("/api/currentLocale")
    @ResponseBody
    public Map<String, String> getCurrentLocale(Locale locale) {
        Map<String, String> response = new HashMap<>();
        response.put("language", locale.getLanguage());
        response.put("country", locale.getCountry());
        response.put("displayName", locale.getDisplayName());
        return response;
    }
}
```

4.21. Internationalisation des validations

4.22. Configuration des messages de validation

```java
@Configuration
public class ValidationConfig {

    @Bean
    public LocalValidatorFactoryBean validator(MessageSource messageSource) {
        LocalValidatorFactoryBean validator = new LocalValidatorFactoryBean();
        validator.setValidationMessageSource(messageSource);
        return validator;
    }
}
```

4.23. Messages de validation personnalisés

```
# Dans messages.properties
user.name.required=Le nom est obligatoire
user.email.invalid=Format d'email invalide
user.age.range=L'âge doit être entre {min} et {max} ans

# Dans messages_en.properties
user.name.required=Name is required
user.email.invalid=Invalid email format
user.age.range=Age must be between {min} and {max} years
```

4.24. Modèle avec validation

```java
public class User {

    @NotBlank(message = "{user.name.required}")
    @Size(min = 2, max = 50, message = "{validation.size}")
    private String name;

    @Email(message = "{user.email.invalid}")
    private String email;

    @Min(value = 18, message = "{user.age.min}")
    @Max(value = 99, message = "{user.age.max}")
    private Integer age;

    // Getters et setters
}
```

4.25. Formatage selon la locale

4.26. Dates et heures

Pour formater les dates en fonction de la locale courante, utilisez l'objet utilitaire `#temporals` de Thymeleaf (Spring 6 / Thymeleaf 3+) ou bien `#dates` (plus ancien).

```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org">
<body>

<!-- Formatage de date avec un pattern fixe -->
<p th:text="${#temporals.format(user.birthDate, 'dd/MM/yyyy')}">
    01/01/1990
</p>

<!-- Formatage d'heure et de date avec styles -->
<p th:text="${#temporals.format(order.createdAt, 'medium', 'short')}">
    27 juin 2025, 10:45
</p>

<!-- Utilisation d'un format défini dans messages.properties -->
<p th:text="${#temporals.format(event.date, #{date.format})}">
    27/06/2025
</p>

</body>
</html>
```

Le helper `#temporals` est disponible avec Thymeleaf 3+ et Spring 6+. Pour des versions plus anciennes, utilisez `#dates.format(…)`.

4.27. Nombres et devises

Thymeleaf fournit l'objet `#numbers` pour formater les nombres selon la locale active.

```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org">
<body>

<!-- Formatage en devise -->
<p th:text="${#numbers.formatCurrency(product.price)}">
    99,99 €
</p>

<!-- Formatage en pourcentage -->
<p th:text="${#numbers.formatPercent(discount)}">
    15 %
</p>
```

```
<!-- Formatage avec précision (2 chiffres après la virgule) -->
<p th:text="${#numbers.formatDecimal(average, 1, 2)}">
    4,37
</p>

</body>
</html>
```

4.28. Exemple dans messages.properties

```
date.format=dd/MM/yyyy
```

4.29. Remarque

Le formatage est automatiquement localisé en fonction de la langue courante (ex. `fr`, `en`, `de`) définie via le sélecteur ou la configuration du `LocaleResolver`.

4.30. Service de formatage côté serveur

```
@Service
public class LocaleFormattingService {

    public String formatCurrency(BigDecimal amount, Locale locale) {
        NumberFormat currencyFormat = NumberFormat.getCurrencyInstance(locale);
        return currencyFormat.format(amount);
    }

    public String formatDate(Date date, Locale locale) {
        DateFormat dateFormat = DateFormat.getDateInstance(DateFormat.MEDIUM, locale);
        return dateFormat.format(date);
    }

    public String formatDateTime(Date date, Locale locale) {
        DateFormat dateFormat = DateFormat.getDateTimeInstance(
            DateFormat.MEDIUM, DateFormat.SHORT, locale);
        return dateFormat.format(date);
    }

    public String formatNumber(Number number, Locale locale) {
        NumberFormat numberFormat = NumberFormat.getNumberInstance(locale);
        return numberFormat.format(number);
    }

    public String formatPercent(Double value, Locale locale) {
        NumberFormat percentFormat = NumberFormat.getPercentInstance(locale);
```

```
        return percentFormat.format(value);
    }
}
```

4.31. Bonnes pratiques

4.32. 1. Organisation des clés

```
# Organisez par catégories
# Navigation
nav.home=Accueil
nav.products=Produits

# Formulaires
form.save=Enregistrer
form.cancel=Annuler

# Messages d'erreur
error.validation.required=Champ obligatoire
error.business.insufficient.funds=Fonds insuffisants
```

4.33. 2. Gestion des pluriels

4.34. Utilisation de Thymeleaf avec ICU MessageFormat

Thymeleaf prend en charge les formats de message ICU (International Components for Unicode) via `ResourceBundleMessageSource` depuis Spring 6, ou avec une configuration personnalisée dans les versions antérieures. Ce support permet une gestion avancée du pluriel, des sélections conditionnelles, et bien plus, directement dans les fichiers `messages.properties`.

4.35. Configuration

Pour activer le support ICU, assurez-vous d'utiliser `ResourceBundleMessageSource` et que vos fichiers `.properties` soient encodés en UTF-8.

```
@Bean
public MessageSource messageSource() {
    ResourceBundleMessageSource messageSource = new ResourceBundleMessageSource();
    messageSource.setBasename("messages");
    messageSource.setDefaultEncoding("UTF-8");
    return messageSource;
}
```

4.36. Exemple de messages avec pluralisation

Voici un exemple de fichier `messages.properties` utilisant ICU MessageFormat pour gérer le pluriel :

```
item.count={0, plural, =0{Aucun élément} =1{1 élément} other{{0} éléments}}
notifications.count={0, plural, =0{Vous n'avez aucune notification} =1{Vous avez 1
notification} other{Vous avez {0} notifications}}
```

4.37. Utilisation dans une vue Thymeleaf

L'appel du message fonctionne comme d'habitude avec `#{…}` :

```
<p th:text="#{item.count(${itemCount})}">3 éléments</p>

<p th:text="#{notifications.count(${notifCount})}">
    Vous avez 3 notifications
</p>
```

4.38. Gestion du genre ou de sélections conditionnelles

ICU supporte aussi `select` pour les conditions. Exemple :

```
user.greeting={gender, select, male{Bienvenue M. {0}} female{Bienvenue Mme {0}}
other{Bienvenue {0}}}
```

```
<p th:text="#{user.greeting(${user.name}, gender=${user.gender})}">
    Bienvenue M. Jean
</p>
```

4.39. Remarques et références

- ICU MessageFormat est **très puissant** pour produire des messages **grammaticalement corrects** selon le **nombre**, le **genre**, ou d'autres critères.
- Il remplace avantageusement les approches classiques avec plusieurs clés (`item.count.0`, `item.count.1`, etc.).
- Disponible **nativement** avec Spring 6 / Thymeleaf 3.1+ et Java 17+, ou via des bibliothèques externes dans des versions plus anciennes.
- https://unicode-org.github.io/icu/userguide/format_parse/messages/

4.40. 3. Encodage des fichiers

Assurez-vous que tous les fichiers de propriétés sont en UTF-8 :

```
# Utilisez des caractères Unicode si nécessaire
welcome.chinese=\u6B22\u8FCE
welcome.arabic=\u0645\u0631\u062D\u0628\u0627
```

4.41. 4. Fallback intelligent

```
@Bean
public MessageSource messageSource() {
    ReloadableResourceBundleMessageSource messageSource =
        new ReloadableResourceBundleMessageSource();
    messageSource.setBasename("classpath:messages");
    messageSource.setDefaultEncoding("UTF-8");
    messageSource.setFallbackToSystemLocale(false);
    messageSource.setUseCodeAsDefaultMessage(true); // Utilise la clé comme message
par défaut
    return messageSource;
}
```

4.42. Résumé

Dans ce chapitre, nous avons appris :

- La configuration de l'internationalisation dans Spring MVC
- La création et l'organisation des fichiers de messages
- L'utilisation des différents résolveurs de locale
- L'implémentation du changement de langue dynamique
- L'internationalisation des validations
- Le formatage selon la locale
- Les bonnes pratiques pour une i18n efficace

Chapter 5. Gestion des paramètres

5.1. Processus de validation d'un formulaire en Spring MVC

La validation d'un formulaire avec Spring MVC suit un enchaînement clair de responsabilités entre le contrôleur, le modèle, le moteur de templates (Thymeleaf) et le système de validation intégré.

Étape	Description
1. Affichage du formulaire	Le contrôleur initialise un objet vide (ex. : `new Utilisateur()`) et le place dans le modèle avec `model.addAttribute(…)`. Thymeleaf utilise `th:object` pour lier cet objet aux champs du formulaire HTML.
2. Saisie utilisateur	L'utilisateur remplit les champs du formulaire et le soumet. Les données sont envoyées en `POST` vers le contrôleur.
3. Réception et validation	Le contrôleur reçoit l'objet via `@ModelAttribute`, et déclenche automatiquement les validations grâce à `@Valid` (ou `@Validated`).

5.2. Processus de validation d'un formulaire en Spring MVC

Étape	Description
4. Gestion des erreurs	Si des erreurs sont détectées (`BindingResult.hasErrors()`), le contrôleur doit conduire à la page du formulaire. Les erreurs sont rendues dans Thymeleaf avec `th:errors` et `#fields.hasErrors(…)`.
5. Succès	Si aucune erreur n'est présente, le contrôleur peut poursuivre le traitement (enregistrement, redirection, etc.), et afficher une page de confirmation ou un message de succès.
6. Messages personnalisés (optionnel)	Les messages des annotations de validation peuvent être personnalisés via un fichier `messages.properties`, configuré dans `application.properties`.

5.3. Informations importantes

Ce mécanisme repose sur :

- Les annotations de `jakarta.validation` (`@NotBlank`, `@Email`, etc.)
- L'injection automatique des erreurs dans `BindingResult`
- La capacité de Thymeleaf à accéder aux erreurs de champ

5.4. Dépendance nécessaire (Maven)

- Pour les annotations de validation

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-validation</artifactId>
</dependency>
```

5.5. Le modèle avec validation

- On crée une classe `Utilisateur` annotée avec des contraintes de validation.

```
import jakarta.validation.constraints.*;

public class Utilisateur {

    @NotBlank(message = "Le nom est obligatoire.")
    private String nom;

    @NotBlank(message = "L'email est obligatoire.")
    @Email(message = "Email invalide.")
    private String email;

    @Min(value = 18, message = "Vous devez avoir au moins 18 ans.")
    private int age;

    // Getters et setters
}
```

5.6. Le contrôleur Spring MVC

- On crée un contrôleur avec un formulaire GET et un POST avec validation.

```
@Controller
@RequestMapping("/utilisateur")
public class UtilisateurController {

    @GetMapping("/formulaire")
    public String afficherFormulaire(Model model) {
        model.addAttribute("utilisateur", new Utilisateur());
        return "formulaire";
    }

    @PostMapping("/formulaire")
    public String soumettreFormulaire(
```

```
        @Valid @ModelAttribute("utilisateur") Utilisateur utilisateur,
        BindingResult result
    ) {
        if (result.hasErrors()) {
            return "formulaire";
        }
        return "confirmation";
    }
}
```

5.7. Le formulaire HTML (Thymeleaf)

Le fichier `formulaire.html` situé dans `src/main/resources/templates`.

```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org">
<head>
    <title>Formulaire Utilisateur</title>
</head>
<body>
<h2>Formulaire</h2>

<form th:action="@{/utilisateur/formulaire}" th:object="${utilisateur}" method="post">
    <div>
        <label>Nom :</label>
        <input type="text" th:field="*{nom}" />
        <span th:if="${#fields.hasErrors('nom')}" th:errors="*{nom}"></span>
    </div>

    <div>
        <label>Email :</label>
        <input type="email" th:field="*{email}" />
        <span th:if="${#fields.hasErrors('email')}" th:errors="*{email}"></span>
    </div>

    <div>
        <label>Âge :</label>
        <input type="number" th:field="*{age}" />
        <span th:if="${#fields.hasErrors('age')}" th:errors="*{age}"></span>
    </div>

    <button type="submit">Envoyer</button>
</form>

</body>
</html>
```

5.8. La page de confirmation

Le fichier `confirmation.html` :

```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org">
<head>
    <title>Confirmation</title>
</head>
<body>
    <h2>Formulaire soumis avec succès !</h2>
</body>
</html>
```

5.9. Configuration des messages (optionnel)

Pour centraliser les messages dans un fichier `messages.properties` :

```
NotBlank.utilisateur.nom=Le nom est obligatoire.
Email.utilisateur.email=Veuillez saisir un email valide.
Min.utilisateur.age=Vous devez avoir au moins 18 ans.
```

Et dans `application.properties` :

```
spring.messages.basename=messages
```

5.10. Résultat attendu

- Si les champs ne sont pas valides, les erreurs sont affichées à côté des champs.
- Si tous les champs sont valides, l'utilisateur est redirigé vers la page de confirmation.

Chapter 6. Gestion des Sessions dans Spring MVC

6.1. Objectifs du chapitre

À la fin de ce chapitre, vous serez capable de :

- Comprendre les mécanismes de session dans Spring MVC
- Utiliser `@SessionAttributes` pour gérer les données de session
- Manipuler directement l'objet `HttpSession`
- Implémenter des objets de session personnalisés
- Gérer le cycle de vie des sessions

6.2. Introduction aux sessions

- Une session HTTP permet de maintenir l'état entre plusieurs requêtes d'un même utilisateur.
- Spring MVC offre plusieurs mécanismes pour gérer les sessions de manière élégante et efficace.

6.3. Concepts clés

- **Session** : Conteneur de données associées à un utilisateur spécifique
- **Session ID** : Identifiant unique généralement stocké dans un cookie
- **Timeout** : Durée de vie maximale d'une session inactive
- **Scope** : Portée des données de session

6.4. Méthodes de gestion des sessions

6.5. Utilisation de @SessionAttributes

L'annotation `@SessionAttributes` permet de stocker automatiquement certains attributs du modèle dans la session.

```
@Controller
@SessionAttributes("user")
public class UserController {

     @RequestMapping(value = "/login", method = RequestMethod.GET)
    public String showLoginForm(Model model) {
        model.addAttribute("user", new User());
        return "login";
    }
```

```
    @RequestMapping(value = "/login", method = RequestMethod.POST)
    public String processLogin(@ModelAttribute("user") User user,
                               BindingResult result, Model model) {
        if (isValidUser(user)) {
            // L'objet user sera automatiquement stocké en session
            // grâce à @SessionAttributes("user")
            model.addAttribute("user", user);
            return "redirect:/dashboard";
        } else {
            result.rejectValue("username", "error.invalid.credentials");
            return "login";
        }
    }

    @RequestMapping("/dashboard")
    public String dashboard(@ModelAttribute("user") User user, Model model) {
        // L'objet user est récupéré automatiquement depuis la session
        model.addAttribute("welcomeMessage", "Bienvenue " + user.getUsername());
        return "dashboard";
    }

    @RequestMapping("/logout")
    public String logout(SessionStatus sessionStatus) {
        // Marque la session comme complète et nettoie les @SessionAttributes
        sessionStatus.setComplete();
        return "redirect:/";
    }

    private boolean isValidUser(User user) {
        // Logique de validation
        return user.getUsername() != null && user.getPassword() != null;
    }
}
```

6.6. Manipulation directe de HttpSession

Pour un contrôle plus fin, vous pouvez manipuler directement l'objet `HttpSession`.

```
@Controller
public class ShoppingCartController {

    @RequestMapping("/addToCart")
    public String addToCart(@RequestParam("productId") Long productId,
                            @RequestParam("quantity") int quantity,
                            HttpSession session) {

        // Récupération du panier depuis la session
        ShoppingCart cart = (ShoppingCart) session.getAttribute("cart");
        if (cart == null) {
```

```
            cart = new ShoppingCart();
            session.setAttribute("cart", cart);
        }

        // Ajout du produit au panier
        cart.addProduct(productId, quantity);

        // Mise à jour du timestamp de dernière activité
        session.setAttribute("lastActivity", new Date());

        return "redirect:/cart";
    }

    @RequestMapping("/cart")
    public String viewCart(HttpSession session, Model model) {
        ShoppingCart cart = (ShoppingCart) session.getAttribute("cart");
        if (cart == null) {
            cart = new ShoppingCart();
        }

        model.addAttribute("cart", cart);
        model.addAttribute("totalItems", cart.getTotalItems());
        model.addAttribute("totalPrice", cart.getTotalPrice());

        return "cart";
    }

    @RequestMapping("/clearCart")
    public String clearCart(HttpSession session) {
        session.removeAttribute("cart");
        return "redirect:/cart";
    }
}
```

6.7. Objet de session personnalisé

Création d'une classe dédiée pour gérer les données de session.

```
@Component
@Scope("session")
public class UserSession {

    private User currentUser;
    private String selectedLanguage = "fr";
    private List<String> visitedPages = new ArrayList<>();
    private Date loginTime;

    public void login(User user) {
        this.currentUser = user;
```

```
        this.loginTime = new Date();
    }

    public void logout() {
        this.currentUser = null;
        this.loginTime = null;
        this.visitedPages.clear();
    }

    public boolean isLoggedIn() {
        return currentUser != null;
    }

    public void addVisitedPage(String page) {
        visitedPages.add(page);
        // Limite à 10 dernières pages
        if (visitedPages.size() > 10) {
            visitedPages.remove(0);
        }
    }

    // Getters et setters
    public User getCurrentUser() { return currentUser; }
    public void setCurrentUser(User currentUser) { this.currentUser = currentUser; }

    public String getSelectedLanguage() { return selectedLanguage; }
    public void setSelectedLanguage(String selectedLanguage) { this.selectedLanguage =
selectedLanguage; }

    public List<String> getVisitedPages() { return new ArrayList<>(visitedPages); }

    public Date getLoginTime() { return loginTime; }
}
```

```
@Controller
public class SessionController {

    @Autowired
    private UserSession userSession;

    @RequestMapping("/profile")
    public String profile(Model model, HttpServletRequest request) {
        if (!userSession.isLoggedIn()) {
            return "redirect:/login";
        }

        userSession.addVisitedPage(request.getRequestURI());

        model.addAttribute("user", userSession.getCurrentUser());
        model.addAttribute("visitedPages", userSession.getVisitedPages());
```

```
        model.addAttribute("sessionDuration",
            System.currentTimeMillis() - userSession.getLoginTime().getTime());

        return "profile";
    }
}
```

6.8. Configuration des sessions

6.9. Configuration du timeout

```
<!-- Dans web.xml -->
<session-config>
    <session-timeout>30</session-timeout> <!-- 30 minutes -->
</session-config>
```

```
// Configuration programmatique
@Configuration
public class SessionConfig {

    @Bean
    public ServletContextInitializer servletContextInitializer() {
        return servletContext -> {
            servletContext.getSessionCookieConfig().setMaxAge(1800); // 30 minutes
            servletContext.getSessionCookieConfig().setHttpOnly(true);
            servletContext.getSessionCookieConfig().setSecure(true); // HTTPS
uniquement
        };
    }
}
```

6.10. Intercepteur pour la gestion des sessions

```
@Component
public class SessionInterceptor implements HandlerInterceptor {

    @Override
    public boolean preHandle(HttpServletRequest request,
                             HttpServletResponse response,
                             Object handler) throws Exception {

        HttpSession session = request.getSession(false);
        String requestURI = request.getRequestURI();

        // Pages publiques qui ne nécessitent pas de session
```

```
        List<String> publicPages = Arrays.asList("/", "/login", "/register", "/public
");

        if (publicPages.contains(requestURI)) {
            return true;
        }

        // Vérification de la session pour les pages protégées
        if (session == null || session.getAttribute("user") == null) {
            response.sendRedirect("/login");
            return false;
        }

        // Mise à jour de l'activité
        session.setAttribute("lastAccess", new Date());

        return true;
    }

    @Override
    public void postHandle(HttpServletRequest request,
                           HttpServletResponse response,
                           Object handler,
                           ModelAndView modelAndView) throws Exception {

        if (modelAndView != null) {
            HttpSession session = request.getSession(false);
            if (session != null) {
                // Ajout d'informations de session au modèle
                User user = (User) session.getAttribute("user");
                if (user != null) {
                    modelAndView.addObject("sessionUser", user);
                    modelAndView.addObject("sessionId", session.getId());
                }
            }
        }
    }
}
```

Configuration de l'intercepteur :

```
@Configuration
@EnableWebMvc
public class WebConfig implements WebMvcConfigurer {

    @Autowired
    private SessionInterceptor sessionInterceptor;

    @Override
    public void addInterceptors(InterceptorRegistry registry) {
```

```
        registry.addInterceptor(sessionInterceptor);
    }
}
```