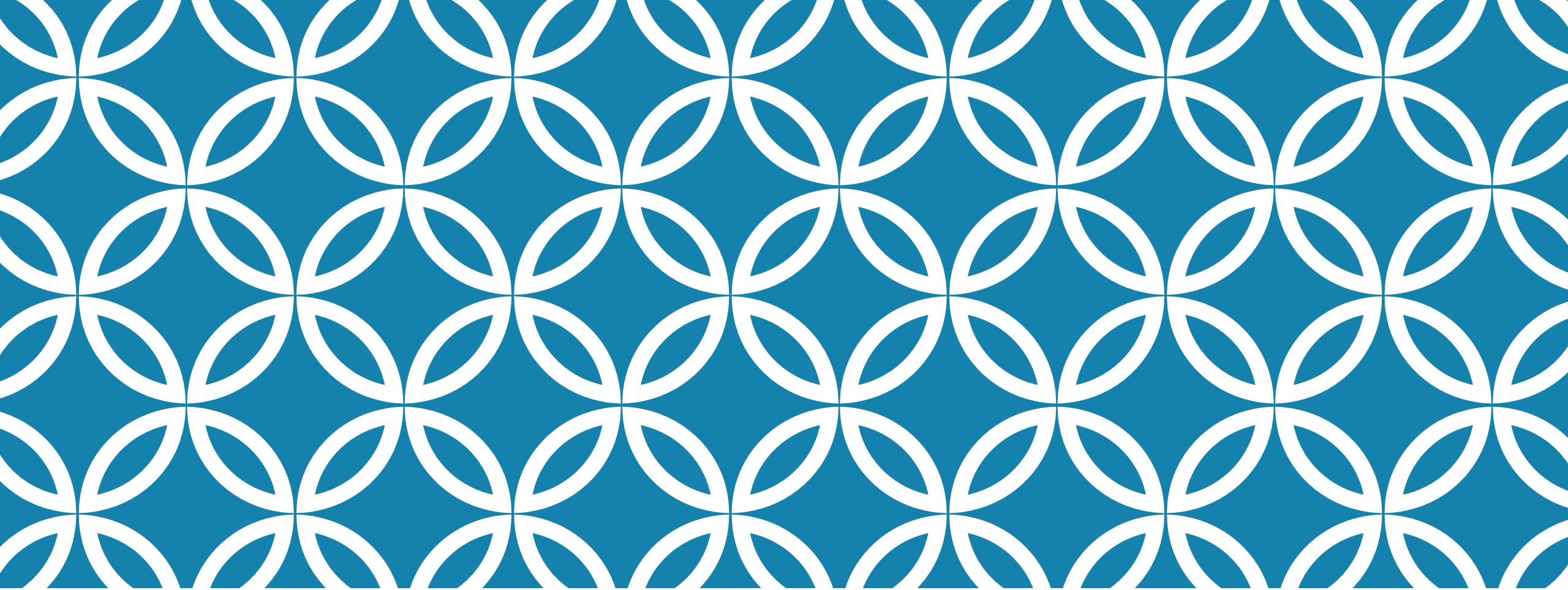
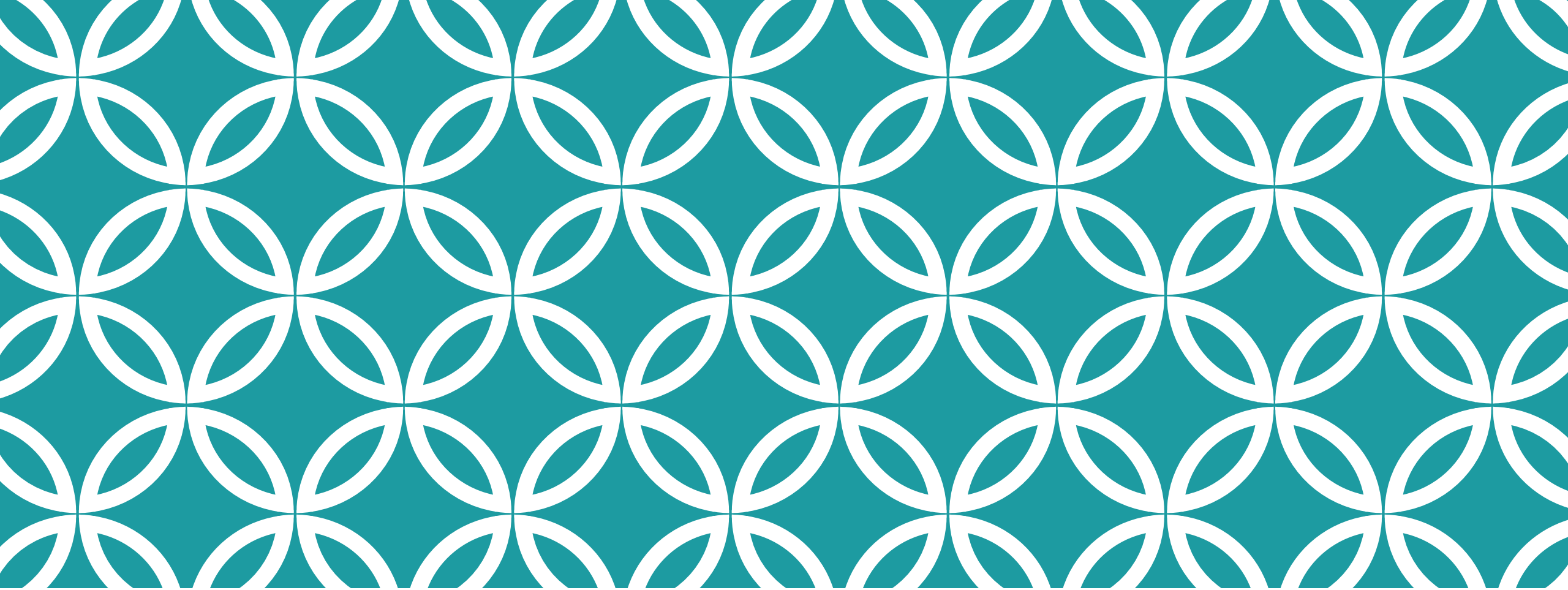


SPRING

Jérémy PERROUAULT



THYMELEAF

Configurer Thymeleaf

THYMELEAF

Un moteur de template permet de ne plus écrire de code directement dans la vue

- Un peu à l'image de la *JSTL*
- Responsabilise et sépare la couche vue du reste de l'application

Un avantage de Thymeleaf

- Extensible et communauté très active
- Pleinement compatible Spring MVC et Spring Boot

THYMELEAF

Thymeleaf permet non seulement de templéter une vue HTML

- Mais également du CSS et du JavaScript !

Il faut pour cela

- Utiliser et configurer un **TemplateResolver**
- Qui sera utilisé par la suite avec un **TemplateEngine**
- Utiliser la dépendance **thymeleaf**

THYMELEAF

Thymeleaf est un moteur de rendu pleinement compatible avec Spring

- Dépendance **thymeleaf-spring5**

Configuration

- D'un **ViewResolver**
- D'un **TemplateEngine**
- D'un **TemplateResolver**

THYMELEAF

Coniguration du Bean du **TemplateResolver**

```
<bean id="templateResolver" class="org.thymeleaf.spring5.templateresolver.SpringResourceTemplateResolver">
  <property name="prefix" value="/WEB-INF/templates/" />
  <property name="suffix" value=".html" />
  <property name="templateMode" value="HTML" />
</bean>
```

THYMELEAF

Bean du **TemplateEngine** à configurer

- Pour lequel on donnera la référence du bean **TemplateResolver**
- On lui ajoutera également le dialect **LayoutDialect**

```
<bean id="templateEngine" class="org.thymeleaf.spring5.SpringTemplateEngine">
  <property name="templateResolver" ref="templateResolver" />
  <property name="enableSpringELCompiler" value="true" />
</bean>
```

THYMELEAF

Bean du **ViewResolver** dans Spring

- Pour lequel on donnera la référence du bean **TemplateEngine**

```
<bean id="viewResolver" class="org.thymeleaf.spring5.view.ThymeleafViewResolver">
  <property name="templateEngine" ref="templateEngine" />
</bean>
```

THYMELEAF

Définition du @Bean **TemplateResolver**

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

  templateResolver.setPrefix("/WEB-INF/templates/");
  templateResolver.setSuffix(".html");

  return templateResolver;
}
```

THYMELEAF

Définition du @Bean **TemplateEngine**

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

  templateEngine.setTemplateResolver(templateResolver);
  templateEngine.setEnableSpringELCompiler(true);

  return templateEngine;
}
```

THYMELEAF

Définition du @Bean **ViewResolver**

```
@Bean
public ViewResolver viewResolver(SpringTemplateEngine templateEngine) {
  ThymeleafViewResolver viewResolver = new ThymeleafViewResolver();

  viewResolver.setTemplateEngine(templateEngine);
  return viewResolver;
}
```

THYMELEAF

Côté vues HTML

- Utilisation de l'espace de nom XML « th »

```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org">
  <head>
    <meta charset="UTF-8" />
    <title>Le titre de la page</title>
  </head>

  <body>
    <section>
      <h1>Le titre de ma page</h1>
    </section>

    <footer>
      <p>Le footer</p>
    </footer>
  </body>
</html>
```

THYMELEAF

L'espace de nom précédemment déclaré donne la possibilité d'utiliser

- L'affichage d'une variable (text)
- Les boucles (each)
- Les conditions (if)
- Les switch (case when)
- Les liens
- …

Utilisation d'Expression Language

- ${el} Expression Language
- ${var} Valeur de var
- [[${var}]] Afficher la valeur de var
- #util Utilitaire spécifique (fonctionnalités) à Thymeleaf
- #{message} Valeur de message dans un fichier *properties*

Documentation en ligne : https://www.thymeleaf.org/doc/tutorials/3.0/usingthymeleaf.html

THYMELEAF

Exemple de condition

```
<div th:if="${username != null}">
  Bonjour <span th:text="${username}"></span> !
</div>
```

```
<div th:if="${username != null}">
  Bonjour [[${username}]] !
</div>
```

THYMELEAF

Exemple de boucle

```
<p th:each="i : ${#numbers.sequence(0, 2)}">
  <span th:text="${i}"></span>
</p>
```

```
<ul th:each="produit : ${produits}">
  <li th:text="${produit.libelle}"></li>
</ul>
```

```
<ul th:each="produit,iterStat : ${produits}">
  <li th:text="${produit.libelle}"></li>
</ul>
```

THYMELEAF

Exemple de switch case

```
<div th:switch="${username}">
  <p th:case="null">Non connecté</p>
  <p th:case="*">Connecté !</p>
</div>
```

```
<th:block th:switch="${username}">
  <p th:case="null">Non connecté</p>
  <p th:case="*">Connecté !</p>
</th:block>
```

THYMELEAF

Exemple de liens

```
<a th:href="@{ accueil }">Accueil</a>
```

```
<a th:href="@{ produits/liste }">Liste des produits</a>
```

```
<a th:href="@{ produits/details(produitId=${ produit.id }) }">Un produit (lien paramètre requête)</a>
```

```
<a th:href="@{ produits/details/{produitId}(produitId=${ produit.id }) }">Un produit (lien PathVariable)</a>
```

THYMELEAF

Autres exemples

```
<span
  th:id="${ 'produit_' + produit.id }"
  th:text="${ #strings.toUpperCase(produit.libelle) }">
</span>
```

```
<input type="text" name="libelle" th:value="${ produit.libelle }" />
```

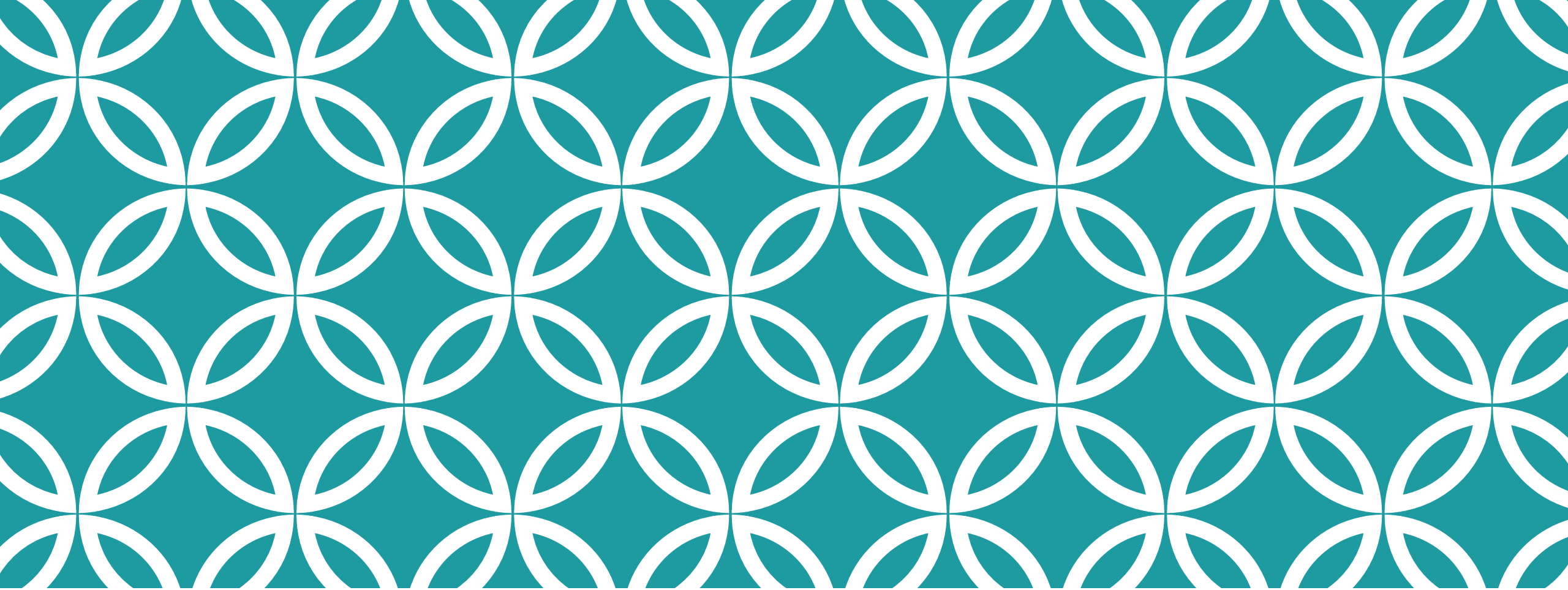
```
<input type="text" name="libelle" th:value="${ produit?.libelle }" />
```

```
<select name="fournisseur.id"
  <option
    th:each="fournisseur : ${ fournisseurs }"
    th:value="${ fournisseur.id }"
    th:text="${ fournisseur.nom }"
    th:selected="${ fournisseur.id == produit?.fournisseur?.id }" />
</select>
```

EXERCICE

Implémenter Thymeleaf

- Copier et adapter les vues



VUES FRAGMENTÉES

Les fragments

VUES FRAGMENTÉES

Les vues fragmentées permettent de composer sa vue avec différentes pages HTML

- Evite de réécrire un menu sur toutes les vues par exemple

Il est possible d'avoir une structure générale (un layout)

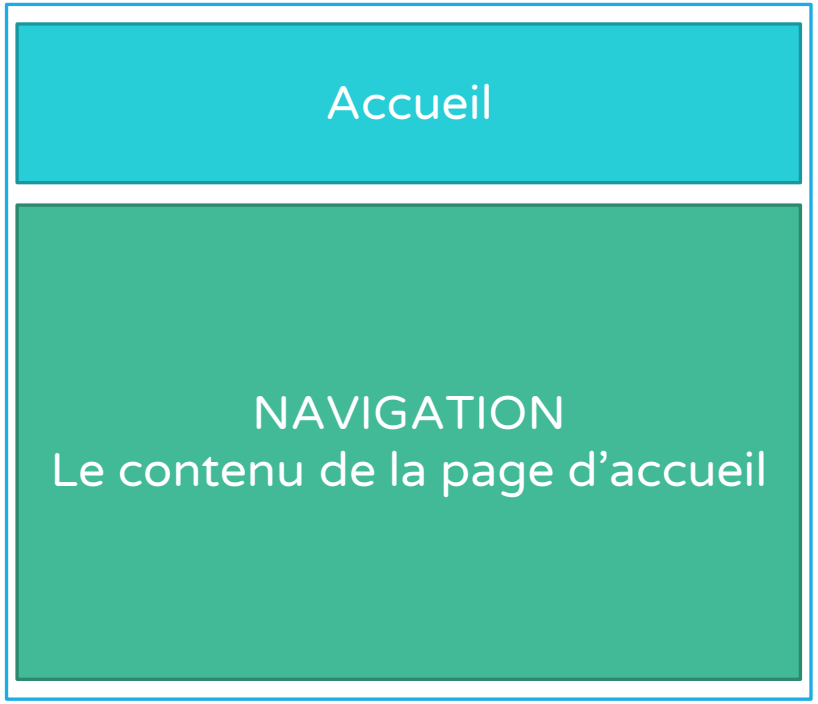
- Avec la possibilité de surcharger certaines zones définies dans ce layout

VUES FRAGMENTÉES

```
<!DOCTYPE html>
<html
    xmlns:th="http://www.thymeleaf.org"
    th:fragment="layout(title, content)">

  <head>
    <title th:replace="${title}">Le titre par défaut</title>
    <base href="/mon-projet/" />
    <link rel="stylesheet" href="assets/css/style.css" />
  </head>

  <body>
    <nav th:replace="navigation.html"></nav>
    <div th:include="${content}">
      Le contenu par défaut
    </div>
  </body>
</html>
```

VUES FRAGMENTÉES

```
<!DOCTYPE html>
<html
    xmlns:th="http://www.thymeleaf.org"
    th:replace="~{layout.html :: layout(~{::title}, ~{::section})}">

  <head>
    <title>Accueil</title>
  </head>

  <body>
    <section>
      Le contenu de la page d'accueil
    </section>
  </body>
</html>
```



VUES FRAGMENTÉES

Le résultat de la page d'accueil

- Utilise le layout « layout.html » comme base de structuration HTML
- Remplace la balise « title » du **layout** par le titre de home.html
- Insère dans la div du **layout** la section de home.html

EXERCICE

Créer le layout « layout.html »

Utiliser le layout dans les vues