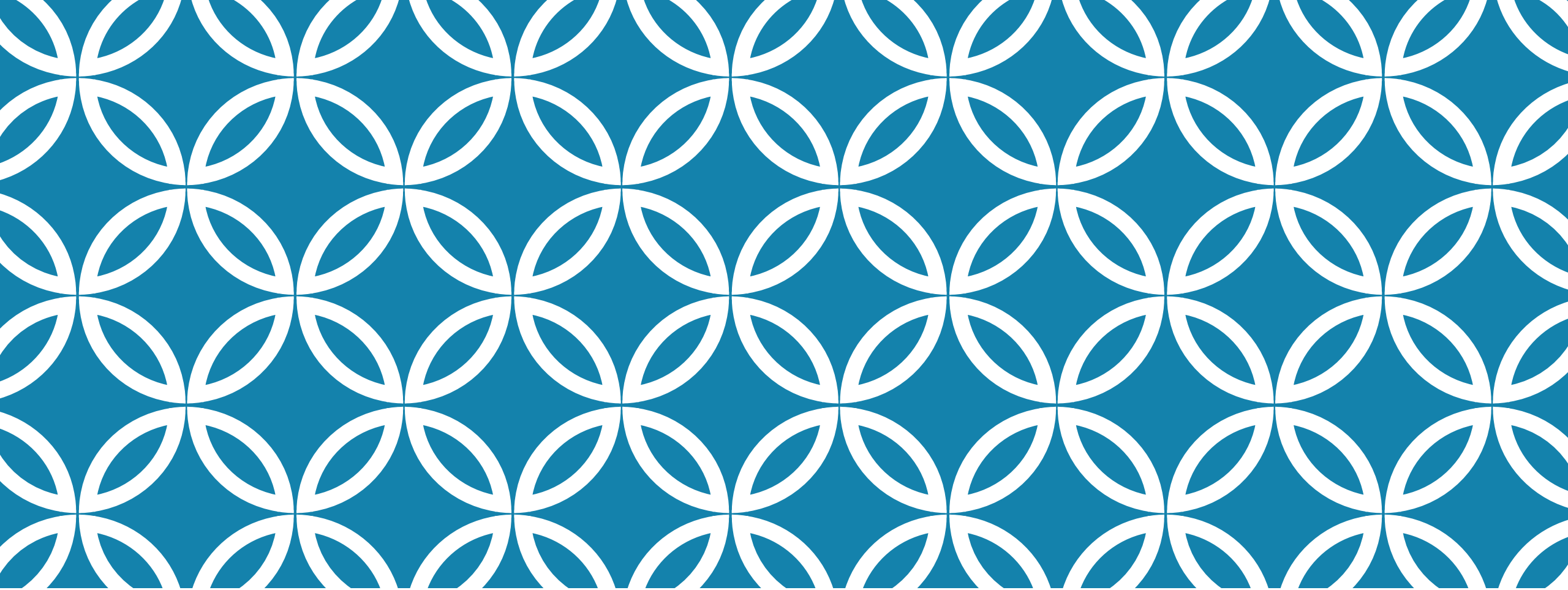
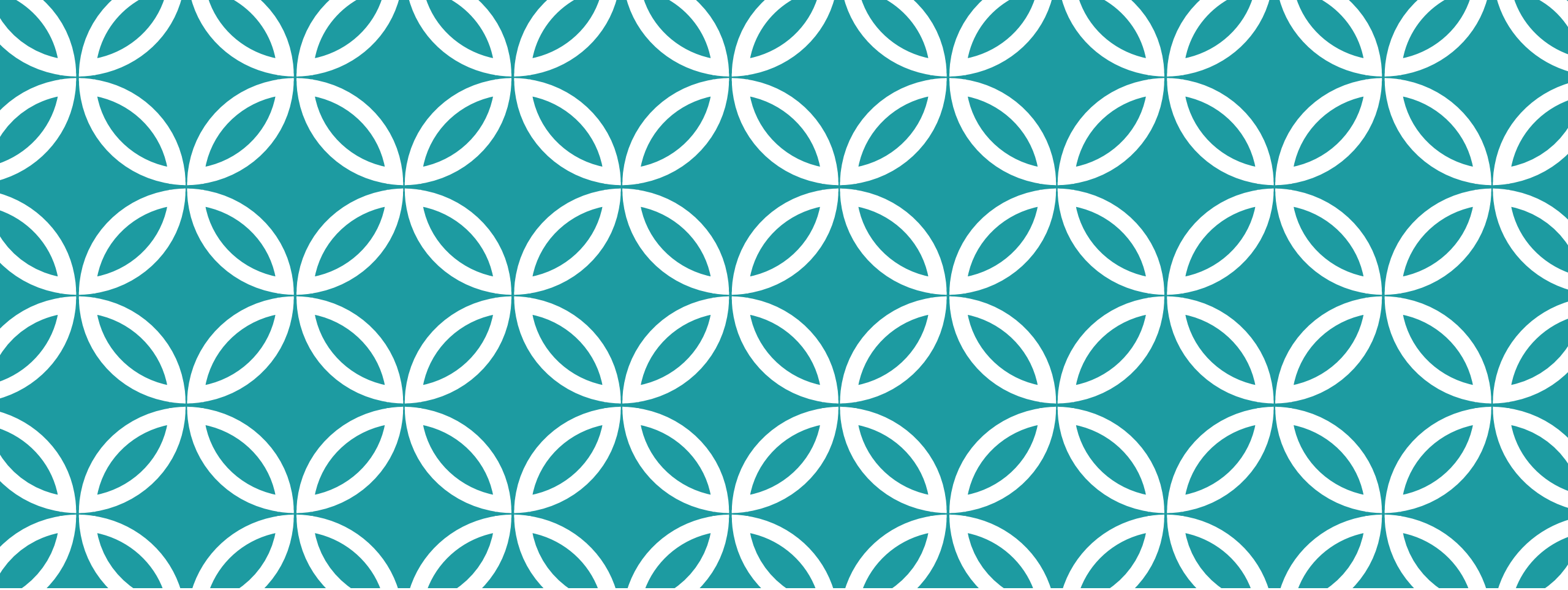


JAVA EE

Jérémy PERROUAULT



THYMELEAF

Moteur de templates

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

Dans une *Servlet*, on modifie le comportement de *init* pour créer le **TemplateResolver**

- On précise le mode de template (HTML)
- L'emplacement des vues HTML
- L'extension de ces vues HTML

```
@WebServlet("/home")
public class HomeServlet extends HttpServlet {
  private ServletContextTemplateResolver resolver;

  public void init(ServletConfig config) throws ServletException {
    super.init(config);

    this.resolver = new ServletContextTemplateResolver(this.getServletContext());
    this.resolver.setTemplateMode(TemplateMode.HTML);
    this.resolver.setPrefix("/WEB-INF/templates/");
    this.resolver.setSuffix(".html");
    this.resolver.setCharacterEncoding("utf-8");
  }
}
```

THYMELEAF

Dans la *Servlet*, on crée le **TemplateEngine**, qui génèrera la vue « home »

- Depuis le fichier */WEB-INF/templates/home.html*, compte tenu de la configuration du **TemplateResolver**
- Une fois le résultat généré (**String**), on le retourne dans le flux de la réponse *HTTP*

```
protected void doGet(HttpServletRequest req, HttpServletResponse resp) throws ServletException, IOException {
  TemplateEngine myTemplateEngine = new TemplateEngine();
  WebContext myWebContext = new WebContext(req, resp, this.getServletContext(), req.getLocale());
  String myResult;

  resp.setCharacterEncoding(resolver.getCharacterEncoding());
  myTemplateEngine.setTemplateResolver(this.resolver);

  // Ici viennent les attributs de requêtes par exemple

  myResult = myTemplateEngine.process("home", myWebContext);
  resp.getWriter().println(myResult);
}
```

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)

Utilisation d'Expression Language

- ${el} Expression Language
- ${var} Valeur de var
- [[${var}]] Afficher la valeur de var
- #util Utilitaire spécifique 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>
```

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>
```

EXERCICE

Reprendre le projet Web

- Désactiver le filtre …

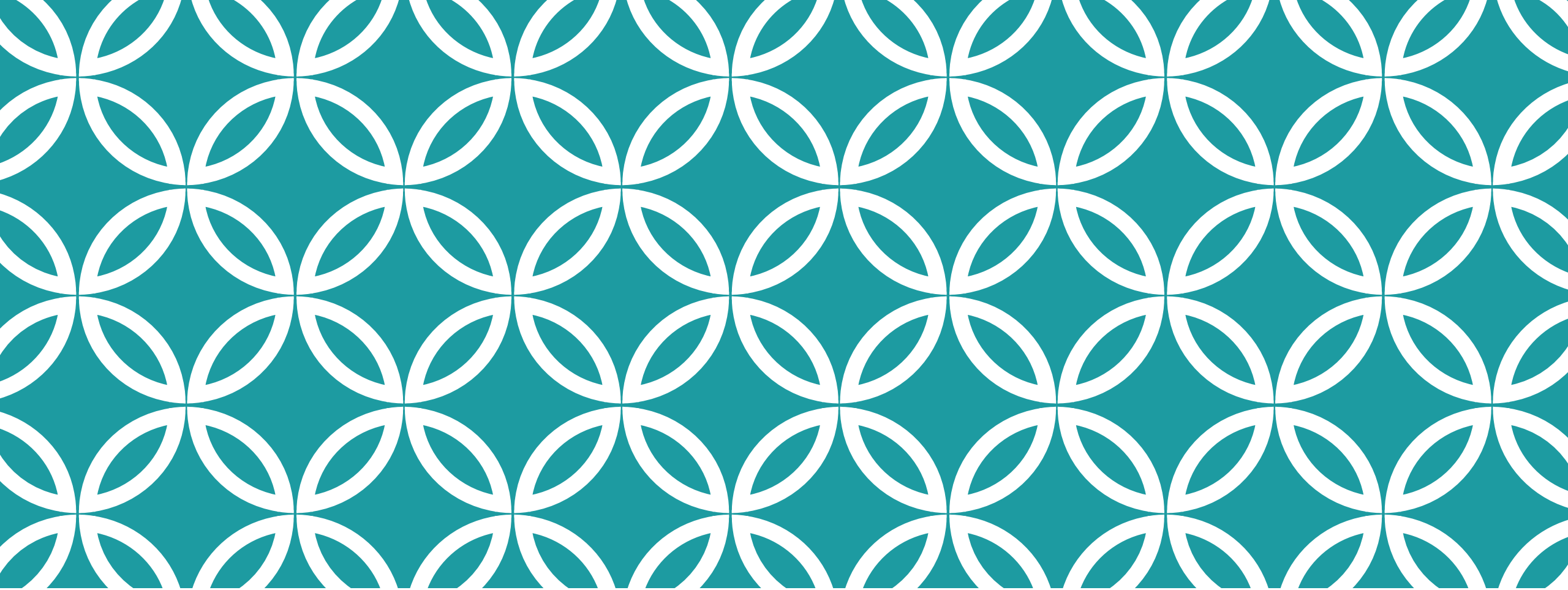
Mettre en place Thymeleaf

Créer une Servlet abstraite **ThymeleafServlet**

- Qui voit sa méthode init modifiée pour Thymeleaf
- Faire en sorte que le code d'impression du template soit factorisé au maximum

Créer une Servlet **HomeServlet** qui hérite de **ThymeleafServlet**

- Afficher la vue *home.html*



VUES COMPOSITES

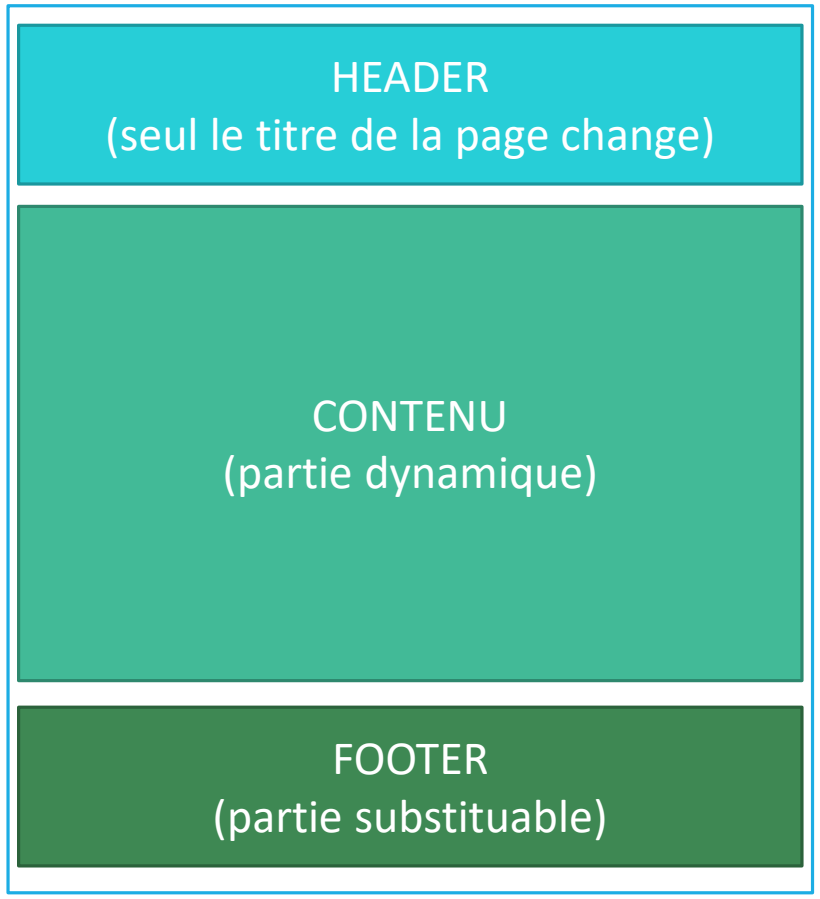
Les Layouts

VUES COMPOSITES

Les vues composites permettent d'avoir une structure générale (un layout)

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

VUES COMPOSITES



Possibilité d'avoir un contenu dynamique qui évolue en fonction de la vue

VUES COMPOSITES

Avec Thymeleaf, utilisation d'un *Dialect* **LayoutDialect** à ajouter au moteur de rendu

- Dépendance **thymeleaf-layout-dialect**

```
TemplateEngine templateEngine = new TemplateEngine();

templateEngine.setTemplateResolver(resolver);
templateEngine.addDialect(new LayoutDialect());
```

VUES COMPOSITES

Exemple de *layout.html*

- Définition de l'espace de nom XML « layout »
- Prévision de la surcharge du titre de la page
- Inclusion des feuilles de style générales à toutes les pages

```
<!DOCTYPE html>
<html xmlns:layout="http://www.ultraq.net.nz/thymeleaf/layout">
  <head>
    <meta charset="UTF-8" />
    <base href="/nom-du-projet/" />
    <title layout:title-pattern="$LAYOUT_TITLE - $CONTENT_TITLE">Formation Servlet</title>

    <link rel="stylesheet" href="resources/assets/css/le-style-general.css" />
  </head>
```

VUES COMPOSITES

Utilisation de « fragments » dans le template layout

- Prévision de la surcharge du H1
- Prévision de la surcharge de la section
- Prévision de la surcharge du footer

```
  <body>
    <header>
      <h1 layout:fragment="custom-title">Le titre par défaut</h1>
    </header>

    <section layout:fragment="custom-content">
      <p>Le contenu par défaut</p>
    </section>

    <footer layout:fragment="custom-footer">
      <p>Le footer par défaut</p>
    </footer>
  </body>
</html>
```

VUES COMPOSITES

Exemple de la vue *home.html*

- Définition des espaces de nom XML « layout » et « th »
- Indication du layout « décorateur » à utiliser, ici *layout.html*
- Ajout du titre de la page
- Ajout d'un style CSS spécifique à la page

```
<!DOCTYPE html>
<html
    xmlns:th="http://www.thymeleaf.org"
    xmlns:layout="http://www.ultraq.net.nz/thymeleaf/layout"
    layout:decorator="layout.html">

  <head>
    <title>Le titre de la page</title>
    <link rel="stylesheet" href="resources/assets/css/style-de-la-page.css" />
  </head>
```

VUES COMPOSITES

Surcharge des fragments

- On choisi de ne pas modifier le H1
- On modifie la section
- On modifie le footer

```
  <body>
    <section layout:fragment="custom-content">
      <ul th:each="produit : ${produits}">
        <li th:text="${produit.libelle}"></li>
      </ul>
    </section>

    <footer layout:fragment="custom-footer">
      <p>Nouveau footer pour cette page</p>
    </footer>
  </body>
</html>
```

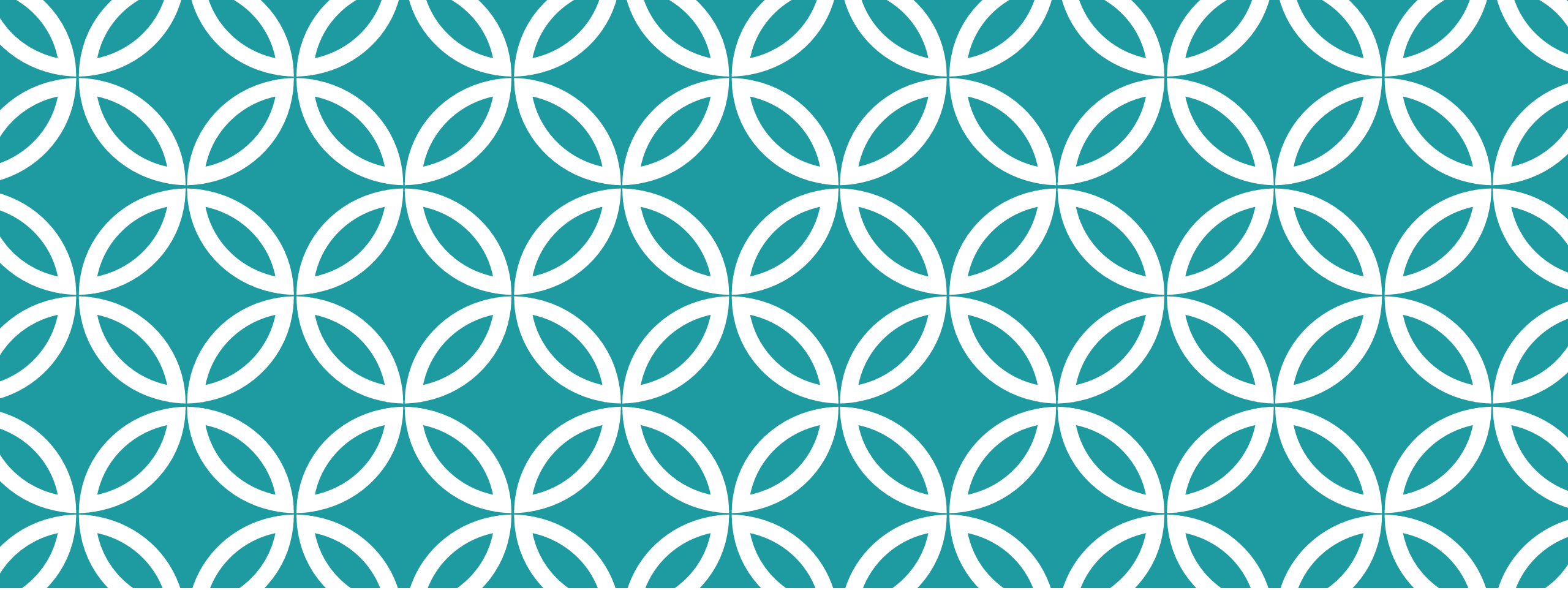
VUES COMPOSITES

Le résultat final de la vue *home* sera

- Pour le head, le head du template *layout.html* combiné au head de la vue *home.html*
- Pour le body
  - Chaque fragment précisé dans *home.html* viendra écraser le fragment défini dans *layout.html* du même nom

VUES COMPOSITES

Résultat de la vue *home.html*



EXERCICE SERVLET & THYMELEAF

Manipulations

EXERCICE

Remplacer les JSP du projet Web par Thymeleaf

- Avec un layout
- Réactiver le filtre !