

1 - SPRING & HTML

1.1 - Introduction

Plusieurs solutions sont envisageable pour créer des pages web au format HTML, dont voici quelques exemples, qui seront présentés ci-après :

1. Génération d'une page HTML par le code Java
2. Utilisations de fichiers ressource
3. JSP
4. Thymeleaf
5. AngularJS

1.2 - Génération de code HTML avec Java

Une page web peut être considérée comme une chaîne de caractères. Il est donc possible d'utiliser une (ou plusieurs) variables de format texte pour contenir le contenu de la page HTML :

```
StringBuilder pageweb = new StringBuilder()
pageweb.append(
    "<html>"
    + "<head>"
    + "  <title>Java & HTML</title>"
    + "</head>");

pageweb.append(
    "<body>"
    + "  <h1>Test Java & HTML : StringBuilder</h1>"
    + "  ...");
```

Avantage : Mise en œuvre très simpliste

Inconvénient : Utilisation fastidieuse

Nombreux risques d'erreurs ; Exemple : Caractère de délimitation → "

Ne convient pas pour des pages complexes

Ne sera réservé que pour des tests rapides et de portée limitée

1.3 - Utilisation de fichier ressources

Enregistrer dans le répertoire ressource src/main/resources/... les fichiers HTML préparés

Le code Java permet de récupérer le fichier et de le retourner en réponse

Solution simple à mettre en œuvre ne convient toutefois que pour des contenus figés

Les solutions proposées ci-après, sont à privilégier car elle correspondent à une véritable exploitation de la structure du code HTML / CSS / JS, en lien avec du code java pour gérer les partie dynamiques.

a) - Accès à un fichier ressource avec SPRING

Soit le fichier fichier.abc, enregistré dans le répertoire src/main/resources/repertoire :

```
Resource resource = new DefaultResourceLoader()
    .getResource("classpath:/repertoire/fichier.abc");

resource.getURL()      => Retourne le chemin vers le fichier
```

2 - JSP

Le format JSP permet de mixer la structure HTML avec le code Java, et reproduit un fonctionnement comparable à ce qui est obtenu avec le langage PHP (ou l'inverse?), et permet de manière plus élaborée de générer des pages web dynamique, en lien avec le code java du reste du projet.
Cette syntaxe permet également d'associer les feuilles de style CSS, le code JavaScript, ...

Exemple de mise en oeuvre : <https://o7planning.org/fr/11681/tutoriel-spring-boot-et-jsp>

a) - Dépendances

```
pom.xml
<dependency>
    <groupId>javax.servlet</groupId>
    <artifactId>jstl</artifactId>
</dependency>
```

b) - Gestion des chemins

```
application.properties
# =====
# VIEW RESOLVER
# =====

spring.mvc.view.prefix=/WEB-INF/jsp/
spring.mvc.view.suffix=.jsp
```

Créer le dossier : src/main/WEB-INF/jsp
Les pages JSP seront stockées dans ce dossier

Lien entre JSP et chemin contrôleur :

spring.mvc.view.prefix ⇒ Chemin vers le dossier contenant les pages JSP
spring.mvc.view.suffix ⇒ Extension des fichiers vues : fichiers .jsp

Contrôleur :

```
@RequestMapping(value = {"/personList" } ...
```

...

```
return "personList" ;       ⇒ Renvoi du fichier src/main/WEB-INF/JSP/personList.jsp
```

c) - Index.jsp

```
<!DOCTYPE HTML>
<html>
    <head>
        <meta charset="UTF-8" />
        <title>Welcome</title>
        <link rel="stylesheet" type="text/css"
            href="${pageContext.request.contextPath}/css/style.css"/>
    </head>
    <body>
        <h1>Welcome</h1>
        <h2>${message}</h2>
        <a href="${pageContext.request.contextPath}/personList">Person List</a>
    </body>
</html>
```

Remarque : `${pageContext.request.contextPath}/` ⇒ Défini par `spring.mvc.view.prefix`

d) - personList.jsp

Contrôleur

```
import org.springframework.ui.Model;

@RequestMapping(value = { "/personList" }, method = RequestMethod.GET)
public String viewPersonList(Model model) {

    model.addAttribute("persons", persons);


    return "personList";
}
```

personList.jsp ⇒ Répertoire src/main/WEB-INF/jsp/

```
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c"%>

<!DOCTYPE HTML>
<html>
  <head>
    <meta charset="UTF-8" />
    <title>Person List</title>
    <link rel="stylesheet" type="text/css"
          href="${pageContext.request.contextPath}/css/style.css"/>
  </head>
  <body>
    <h1>Person List</h1>

    <br/><br/>
    <div>
      <table border="1">
        <tr>
          <th>First Name</th>
          <th>Last Name</th>
        </tr>
        <c:forEach items="${persons}" var="person">
          <tr>
            <td>${person.firstName}</td>
            <td>${person.lastName}</td>
          </tr>
        </c:forEach>
      </table>
    </div>
  </body>
</html>
```



remarque : feuille de style style.css

=> Répertoire src/main/Web-INF/css/

3 - Thymeleaf

Références :

<https://www.thymeleaf.org/doc/tutorials/2.1/thymeleafspring.html>

<https://o7planning.org/fr/11545/tutoriel-spring-boot-et-thymeleaf>

<https://www.thymeleaf.org/>

3.1 - Présentation

Non lié à SPRING, Thymeleaf peut être utilisé avec tout code Java pour la génération de page Web.

Chaque page web est défini par une portion de code définie en HTML, dans laquelle sont insérées des balises permettant de lier des contenus dynamique gérés en Java.

Ses caractéristiques sont les suivantes :

- Thymeleaf est proposé comme bibliothèque Java
- Gestion uniquement de la couche vue
- Intégration simplifiée avec les modules de test (JUnit, ...)
- Intégration avec SPRING Framework & SPRING Security
- Templates proposés : HTML, XML, TEXT, JavaScript, CSS, RAW

3.2 - Comparaison ente Thymeleaf et JSP

- La syntaxe de Thymeleaf est plus proche de HTML (peu de balises non HTML)
- Utilise SPRING Expression Language (SpEL) pour gérer les variables

a) - Standard Expression syntax

Simple expression

Variable	<code>\${...}</code>
Liste de variables	<code>*{...}</code>
Message	<code>#{...}</code>
Lien URL	<code>@{...}</code>
Fragments	<code>~ {...}</code>

Opérateurs

Conditionnel	<code>(if) ? (then)</code> <code>(if) ? (then) : (else)</code>
Default	<code>(value) ?: (defaultValue)</code>

3.3 - Configuration du projet

Ajouter la dépendance dans le fichier pom.xml

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

3.4 - Création d'une page Web avec Thymeleaf

Dans sa structure de base, le mode fonctionnement est très proche de celui utilisé précédemment avec JSP

Les modifications sont les suivantes :

- Les fichiers sont enregistrés avec l'extension .html et sont stockés dans le répertoire :
src/main/resources/templates/
- Les fichiers ressources CSS, JS, Images seront stockés dans src/main/resources/static/ (ou

l'un de ses sous-répertoires).

- Utiliser l'espace de nom `thymeleaf`. L'en-tête du fichier `html` devient donc :
`<!DOCTYPE HTML>`
`<html xmlns:th="http://www.thymeleaf.org">`
- Les références aux pages s'expriment avec :
 - `th:href="@{/css/style.css}"`
 -
- La gestion de boucle devient :
 - `<tr th:each="person : ${persons}">`
 `<td th:utext="${person.firstName}">...</td>`
 `<td th:utext="${person.lastName}">...</td>`
 `</tr>`
- Il est possible de définir des éléments (de type constante) directement dans le fichier `application.properties`

Exemple : `application.properties`

```
# =====  
# Thymeleaf  
# =====  
spring.thymeleaf.cache=false  
  
welcome.message=Hello Thymeleaf  
error.message=First Name & Last Name is required!
```

Remarque : Thymeleaf propose de nombreuses autres fonctionnalités non présentées ici.

3.5 - Exemple de code : Page Accueil

a) - `src/main/resources/templates/accueil.html`

Objectif : Affichage de la page d'accueil avec 2 éléments dynamique : message & date

```
<!DOCTYPE HTML>  
<html xmlns:th="http://www.thymeleaf.org">  
<head>  
  <title>ServeurIA</title>  
  <meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />  
  <link rel="stylesheet" type="text/css" th:href="@{/css/style.css}" />  
</head>  
  
<body>  
  <h1>Université de Rouen</h1>  
  <p>Département Informatique - L3 Info</p>  
  <h2>Projet : ServerIA</h2>  
  <p>Le : <span th:text="${today}">aaa</span></p>  
  <p th:text="${message}">aaa</p>  
</body>  
</html>
```

b) - `src/main/resources/static/css/style.css`

```
h1 {  
  color: blue;  
}
```

c) - `src/main/java/fr.univrouen.testws2.controllers.GETController.java`

Contrôleur : Associe la page web accueil avec le template, et gère les parties dynamiques

```
import java.text.SimpleDateFormat;
import java.util.Calendar;
import java.util.Date;

import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.GetMapping;

@Controller
public class GETController {
    @GetMapping("/accueil")
    public String getIndex(Model model) {
        // Initialisation du message
        String message = "Test Thymeleaf !";
        // Mise en forme de la date
        SimpleDateFormat format = new SimpleDateFormat("dd MMMM yyyy");
        String today = format.format(new Date());

        // Création des attributs pour insertion dans la page HTML via Thymeleaf
        model.addAttribute("message", message);
        model.addAttribute("today", today);

        // Création de la page HTML avec le template "accueil.html"
        return "accueil";
    }
}
```

Remarque : Respecter attentivement les annotations et bibliothèques utilisées

d) - Résultat obtenu



3.6 - Exemple de code : Liste de cartes

a) - src/main/resources/templates/listeCartes.html

Objectif : Afficher la liste des cartes sur la page web (utilisation d'une boucle)

```
<!DOCTYPE HTML>
<html xmlns:th="http://www.thymeleaf.org">
<head>
  <title>ServeurIA</title>
  <meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
</head>

<body>
  <h1>Université de Rouen</h1>
  <p>Département Informatique - L3 Info</p>
  <h2>Liste des cartes enregistrées dans la base</h2>
  <ul>
    <th:block th:each="carte:${cartes}">
      <ol>valeur : <span th:utext="${carte.valeur}">vvv</span>
        -> point : <span th:utext="${carte.point}">ppp</span>
      </ol>
    </th:block>
  </ul>
</body>
</html>
```

b) - src/main/java/fr.univrouen.testws2.controllers/GETController.java

Objectif : Afficher la liste des cartes sur la page web (utilisation d'une boucle)

```
@GetMapping("/Card")
public String getCards(Model model) {
    SessionFactory factory = HibernateUtil.getSessionFactory();
    Session session = factory.openSession();

    // Récupère la liste des cartes depuis la base de données
    session.clear();
    List<Card> cards = session.createQuery("from Card").list();

    // Transmission de l'objet List<Card>
    model.addAttribute("cartes", cards);

    // Création de la page HTML avec le template "listeCartes.html"
    return "listeCartes";
}
```

c) - Résultat obtenu



Université de Rouen

Département Informatique - L3 Info

Liste des cartes enregistrées dans la base

```
valeur : 1 -> point : 1
valeur : 2 -> point : 1
valeur : 3 -> point : 3
valeur : 4 -> point : 1
```