


Thyme Leaf

Mecanique MVC & Templates

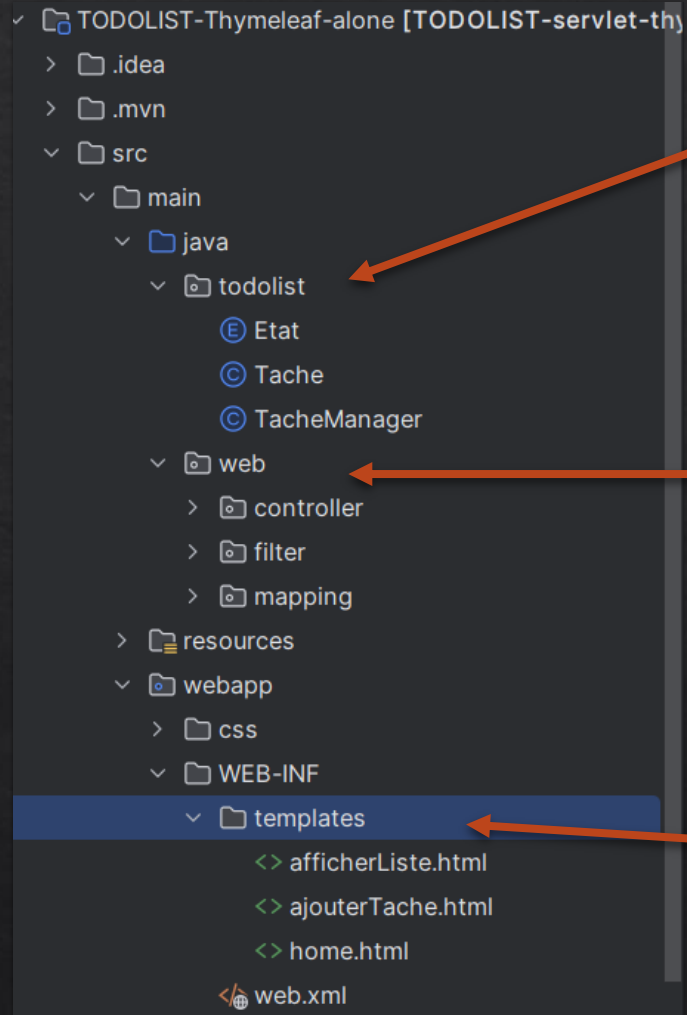
Modèle sans Spring

Dans un nouveau projet Jakarta, ajouter uniquement ces dépendances.



```
<dependencies>
  <dependency>
    <groupId>jakarta.servlet</groupId>
    <artifactId>jakarta.servlet-api</artifactId>
    <version>6.0.0</version>
    <scope>provided</scope>
  </dependency>
  <dependency>
    <groupId>org.thymeleaf</groupId>
    <artifactId>thymeleaf</artifactId>
    <version>3.1.0.M1</version>
  </dependency>
</dependencies>
```

Structure du projet



Todolist :

Package des objets métiers, « nos beans »

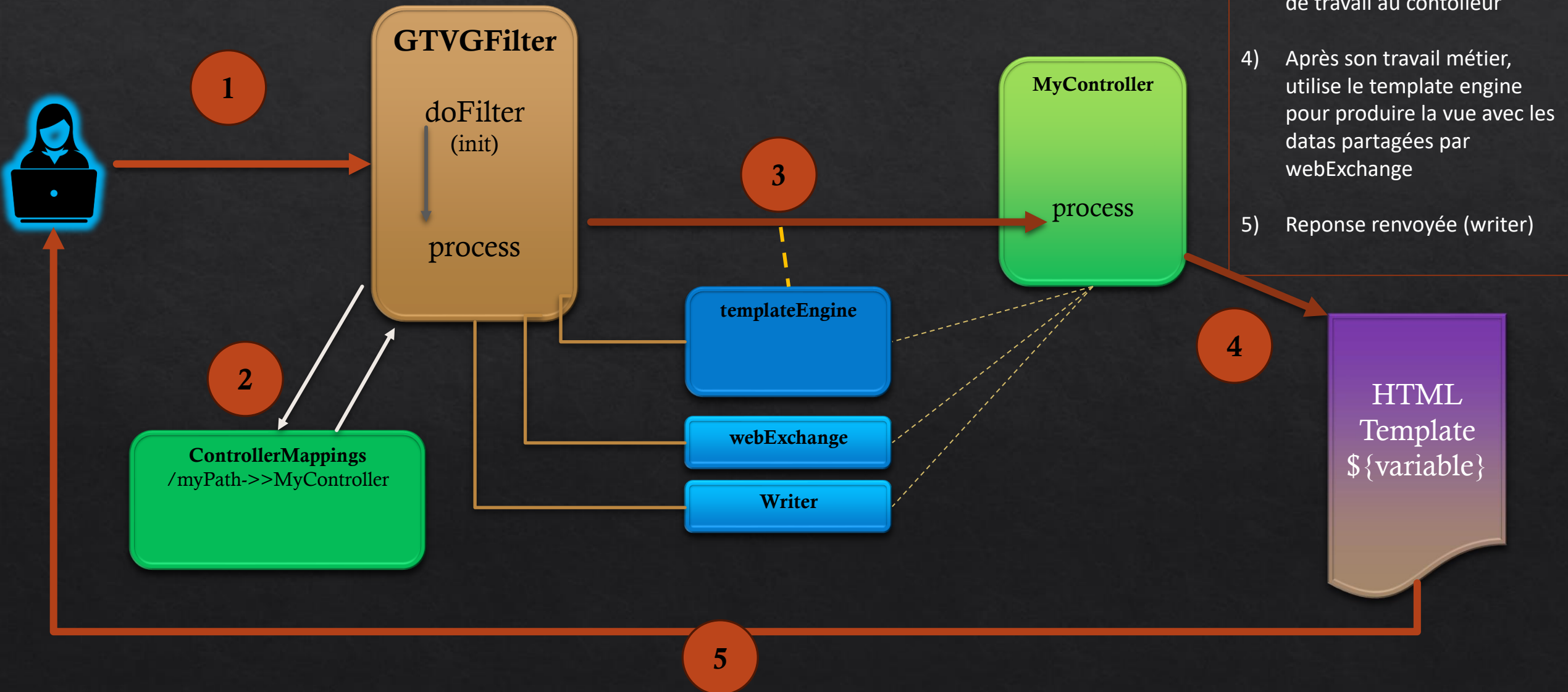
WEB : Organisation MVC et mécanique de thymeleaf.

Paramétrage de l'application

WEB-INF/templates

Nos Vues au format html, comprenant les expressions de thymeleaf pour les compléter

Mécanique



GTVGFilter

`@WebFilter("/*")`

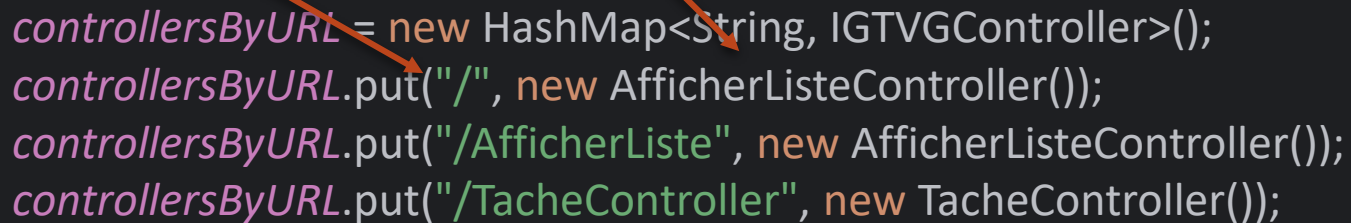
```
public class GTVGFilter implements Filter {
```

Classe générique :

- Contient le templateEngine de Thymeleaf.
- C'est le vrai controller de l'application, il met en œuvre le court-circuit des « jsp ».
- Permet de retrouver et solliciter les contrôleurs que nous définiront pour l'application.
- Permet de paramétrer l'application
 - Type de template (html ou autres)
 - Où les trouver ? WEB-INF/template ou ailleurs.
- On peut utiliser cette classe pour initialiser aussi notre application grâce à sa méthode `doFilter`, c'est un filtre classique.

ControllerMappings

Cette classe sera utilisée par le GTVGFilter pour préciser les liens entre
l'url demandée l'utilisateur et nos contrôleurs spécifiques de l'application qui prend en charge la requête



```
controllersByURL = new HashMap<String, IGTVGController>();  
controllersByURL.put("/", new AfficherListeController());  
controllersByURL.put("/AfficherListe", new AfficherListeController());  
controllersByURL.put("/TacheController", new TacheController());
```

IGTVGController

Une interface contenant une unique méthode pour fixer la structure de nos contrôleurs et ainsi leur passer en paramètres les objets outils qui nous permettront de définir les données à passer à nos templates.

```
public void process(final IWebExchange webExchange, final ITemplateEngine templateEngine, final Writer writer)  
    throws Exception;
```

AfficherListeController

Nos controller : ici on voit qu'il ne fait rien d'autre que définir la route à suivre vers le template « « afficherListe » ».

```
public class AfficherListeController implements IGTVGController {  
    public AfficherListeController() {  
        super();  
    }  
    public void process(final IWebExchange webExchange, final ITemplateEngine templateEngine, final Writer writer)  
        throws Exception {  
  
        WebContext ctx = new WebContext(webExchange, webExchange.getLocale());  
        //l'utilisateur est renvoyé vers la page d'affichage  
        templateEngine.process("afficherListe", ctx, writer);  
    }  
}
```


TacheController

```
public class TacheController implements IGTVGController {
    @Override
    public void process(IWebExchange webExchange, ITemplateEngine templateEngine, Writer writer)
    throws Exception {
        WebContext ctx = new WebContext(webExchange, webExchange.getLocale());

        //Quelle est l'action demandee
        String action = webExchange.getRequest().getParameterValue("action");
        TacheManager tacheManager = (TacheManager)
        ctx.getExchange().getSession().getAttributeValue("managerTache");

        ...
        <CODE MANQUANT DE GESTION ICI>
        ....
    }
}
ctx.setVariable("managerTache", tacheManager);
templateEngine.process("afficherListe", ctx, writer);
}
```

- Lire la les paramètres de la requête
- Lire/écrire des attributs dans les objets implicites et à disposition de nos templates
- Executer des opérations avec les objets métiers de notre application
- Selectionner la vue à afficher

Template : home.html

```
<html xmlns:th="http://www.thymeleaf.org">
<head>
  <title>Home</title>
</head>
<body>
<h1>
  Hello world
</h1>
<P> from <label th:text="${name}"></label>! </P>
</body>
</html>
```

HTML PUR

Balise Thymeleaf



Template : ajouterTache.html

```
<html xmlns:th="http://www.thymeleaf.org">
<div class="form" th:fragment="copy">
  <h3>Ajouter une Tâche</h3>
  <form action="TacheController" method="post">
    <label>Assignée à :</label>
    <input type="text" name="responsable">
    <label>Nom de la Tâche :</label>
    <input type="text" name="nom"></br>
    <input type="hidden" name="action" value="ajouter">
    <input type="submit" name="Créer" >
  </form>
</div>
```

Indique que ce template sera inclus
à d'autre, c'est un « fragment »

C'est le formulaire de saisie des tâches

Template : index.html

```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org" lang="en">
<head>
  <meta charset="UTF-8">
  <title>ToDoS</title>
  <link rel="stylesheet" type="text/css" href="/style.css" >
</head>
<body>
<h1 style="left: 0;"><span th:text="${session.message}">hop</span></h1><br/>
<div th:insert="~{ajouterTache::copy}">&copy; 2011 The Static Templates</div>
<hr>
<table class="tg bleu">
  <div th:each="tache : ${session.todoManager.getListeTache()}" >
    <tr th:class="((__${tache.etat}__ eq 'DONE')? 'done' : '')">
      <td th:text="${tache.nom}">Onions</td>
      <td th:text="${tache.responsable}">2.41</td>
      <td th:text="${tache.etat}">Done</td>
      <td><form method="post" action="/changeStatus">
        <input type="hidden" name="tacheid" th:value="${tache.id}"/>
        <input type="hidden" name="action" th:value="((__${tache.etat}__ eq 'DONE')? 'supp' : 'done')"/>
        <input type="submit" th:value="((__${tache.etat}__ eq 'DONE')? 'supp' : 'done')"/>
      </form>
      </td>
    </tr>
  </div>
</table>
<hr/>
<div th:text="${erreurMessage}">OH</div>
</body>
</html>
```

On utilise ici Thymeleaf pour récupérer nos objets fournis par les contrôleurs et aussi ajuster la manière dont ils seront affichés à l'écran.

ThymeLeaf = <% + EL + JSTL

Modèle avec SpringBoot

Dépendances Maven.



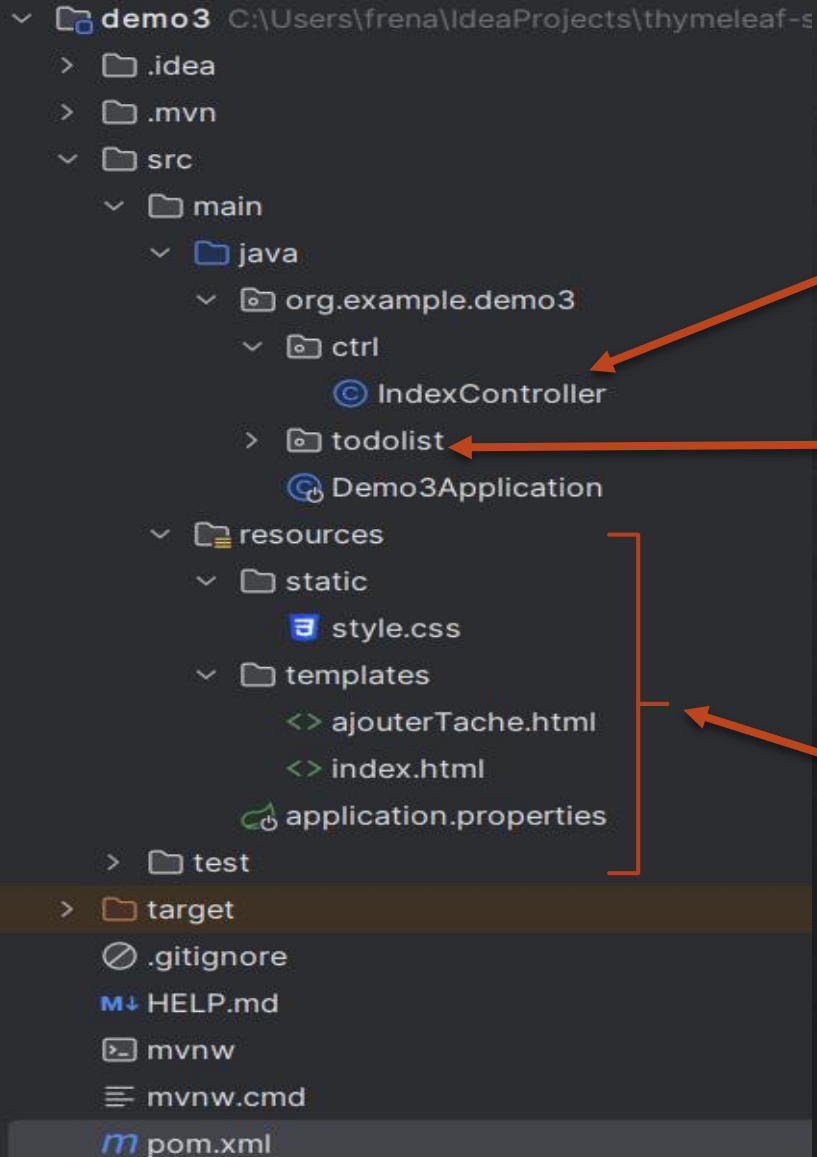
```
<dependencies>
  <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-thymeleaf</artifactId>
  </dependency>
  <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-web</artifactId>
  </dependency>

  <dependency>
    <groupId>org.projectlombok</groupId>
    <artifactId>lombok</artifactId>
    <optional>true</optional>
  </dependency>
  <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-test</artifactId>
    <scope>test</scope>
  </dependency>
</dependencies>
```


Maven : Plugins

```
<build>
  <plugins>
    <plugin>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-maven-plugin</artifactId>
      <configuration>
        <excludes>
          <exclude>
            <groupId>org.projectlombok</groupId>
            <artifactId>lombok</artifactId>
          </exclude>
        </excludes>
      </configuration>
    </plugin>
  </plugins>
</build>
```

Structure du projet



ctrl :

Controlleurs Spring MVC standards
associe PATH → Template Html
(« /index » vers index par exemple)

todolist :

Package des objets métiers, « nos
beans et autres services »

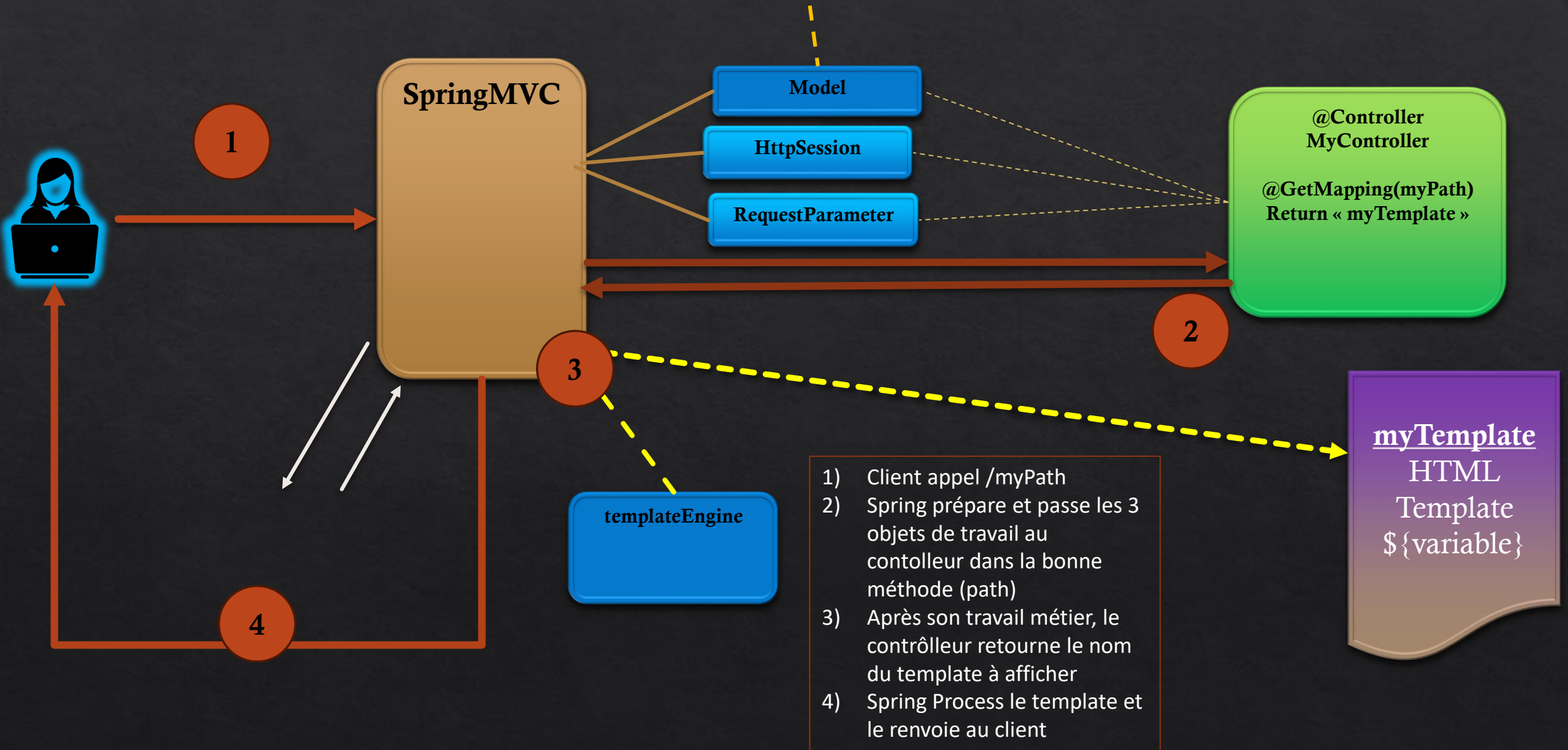
Ressources/

static : CSS, ScriptJS, Img, Sons...

templates :

Nos Vues au format html,
comprenant les expressions
thymeleaf pour les compléter

Mécanique



Contrôleur SpringMVC

```
@Controller
@Scope("request")
public class IndexController{

    @Autowired
    TodoManager todoManager;

    @GetMapping("/")
    public String gotoIndex(Model model, HttpSession session) {

        Todo t1 = new Todo("Base", "Default");
        todoManager.ajoute(t1);
        String a = "MyTodoList : Spring : ThymeLeaf";
        session.setAttribute("message", a);
        return "index";
    }

    @PostMapping("/changeStatus")
    public String changeStatus(@RequestParam Integer tacheid, @RequestParam String action) {

        if (action.equals("done"))
            todoManager.modifie(tacheid, Status.DONE);
        else
            todoManager.supprime(tacheid);

        return "index";
    }
}
```

Stockage en request du contrôleur
ses membres seraient par défaut
stockés aussi en request mais on
peut préciser dans les classes des
membres le scope de son choix

Traitements métiers

Partage de données

Vue (template à utiliser en réponse)

Remarquez en bleu les types d'objets
passés en paramètres. Ce sont les
outils de communication entre
l'infrastructure Web et les contrôleurs
où nous programmons le métier de
l'application

Application.properties

Paramétrage de Thymeleaf :

Path vers les templates

Type de template (HTML)

`spring.thymeleaf.prefix=classpath:/templates/`

#Prefix that gets prepended to view names when building a URL.

`spring.thymeleaf.suffix=.html`

#Suffix that gets appended to view names when building a URL.

ThymeLeaf

Template Modes:

- XML
- Valid XML
- XHTML
- Valid XHTML
- HTML5
- Legacy HTML5

Standard Dialect :

Expression Language :

```
<form:inputText name="userName" value="${user.name}" />
```

ThymeLeaf :

```
<input type="text" name="userName" value="James Carrot" th:value="${user.name}" />
```

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

Basic

```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org">
  <head>
    <title>Thymeleaf Example</title>
  </head>
  <body>
    <h1>Hello, <span th:text="${name}">User</span>!</h1>
  </body>
</html>
```

Iterations

```
<ul>  
  <li th:each="item : ${items}" th:text="${item}">Item</li>  
</ul>
```

Conditions

```
<div th:if="{condition}">Condition is true</div>  
<div th:unless="{!condition}">Condition is false</div>
```

Autre syntaxe & cas plus complexe

```
((${variableEtat} eq 'DONE')? 'done' : 'todo')
```

Elements pré-processés :

```
<input type="submit" th:value="((__${tache.etat}__ eq 'DONE')? 'supp' : 'done')"/>
```

Les doubles underscores permettent à la variable `tache.etat` d'être évaluée avant l'expression complète.

La valeur de la condition d'abord établie, le choix conditionnel peut se faire

Rendu Conditionnel

```
<div th:if="{condition}">Condition is true</div>
```