

SPRING MVC



PLAN DU COURS

- Le design Pattern **MVC**
- **Spring MVC**
- **TP** : Application Web (Gestion de Produits) avec Spring MVC, Développement en Pas à Pas.
- **TP** : Application Web (Gestion d'Employés) avec Spring MVC, Développement en mode autonome.

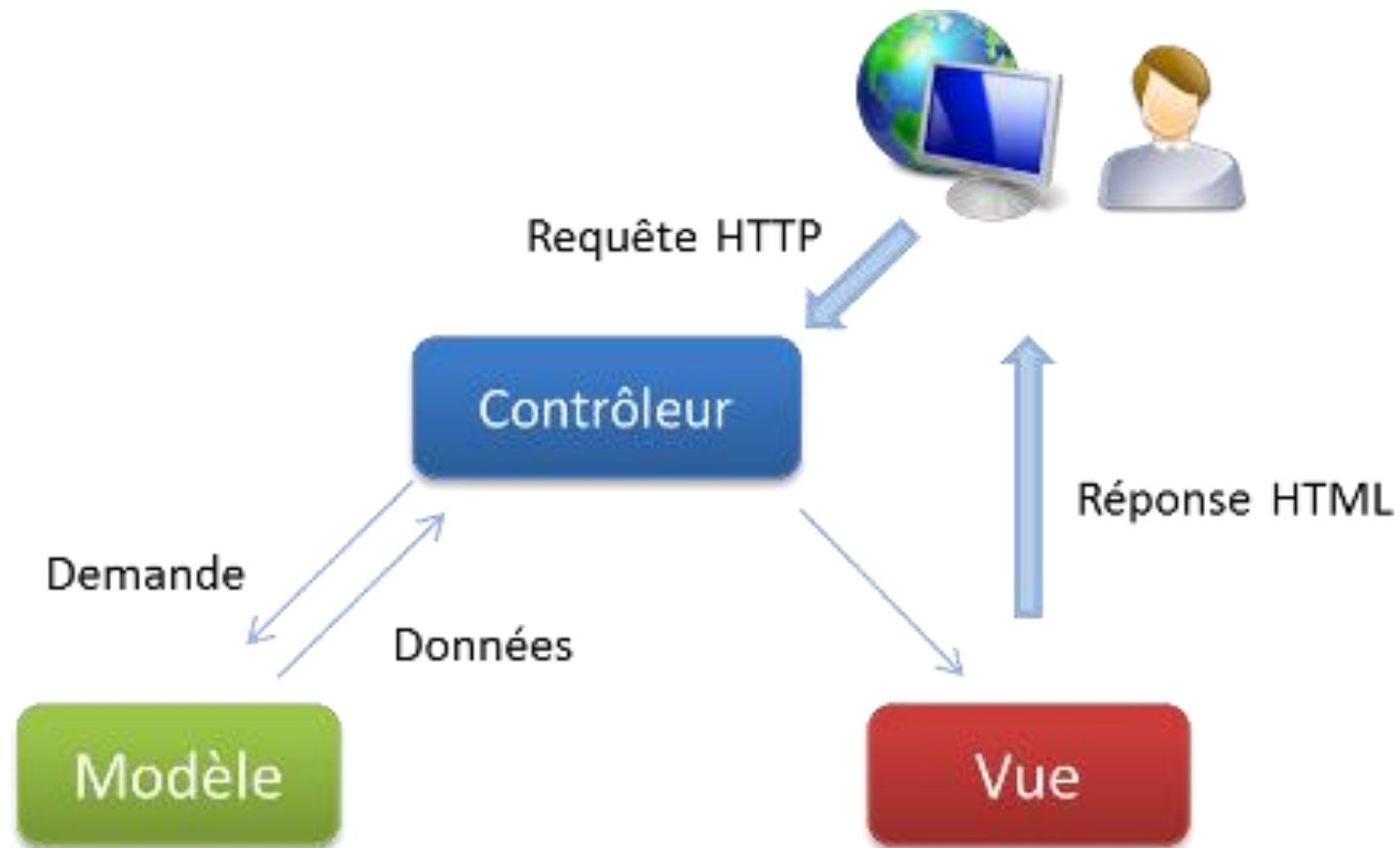
MVC ?

- **MVC** est un motif de conception (“Design Pattern”) : Model – View – Control.
- Un Design Pattern est une solution standard à une problématique récurrente, reconnue comme une bonne pratique (Singleton, DAO, ...).
- Le MVC mène à une organisation rigoureuse et logique du code
 - séparation/indépendance des couches
 - définition de règles permettant de savoir où ajouter une fonctionnalité
 - simplification de la maintenance et de l'évolutivité de chacune
 - clarté indispensable pour les gros projets : gain de temps

PRINCIPE GÉNÉRAL DU MVC

- Le MVC sépare :
 - Données : **Modèle** [**DAO + SERVICE**]
 - IHM : **Vue** [**WEB**]
 - Actions, coordonnées par des **Contrôleurs** [**PRESENTATION**]
- Une application Web JavaEE classique utilise les couches **Web – Presentation – Service – DAO**.
- Le Design Pattern **DAO** est venu pour épauler le Design Pattern **MVC**, et séparer clairement la couche accès à la base de données de la couche de traitement.

MVC



Modèle (MODEL)

- Le **Modèle** implémente les fonctionnalités de l'application (Couche Service maintenant).
- Le **Modèle** gère les données persistantes (Couche DAO maintenant).
- Les composants de la couche **Modèle** ne dépendent pas de la couche **Vue**. Ils doivent pouvoir être utilisés dans des contextes applicatifs différents (Web, Client lourd, ...). C'est l'un des avantages apportés par le MVC.

Vue (VIEW)

- La vue est responsable de l'interface (HTML, JSP, JSF,...)
- La vue assure la mise en forme des données (CSS)
- L'idéal est de n'avoir aucun traitement métier dans cette couche.
- La vue n'accède pas directement au modèle, elle obtient ses données de l'action (Présentation - Contrôleur).

CONTRÔLEUR (CONTROLLER)

- Le rôle des **Contrôleurs** est de :
 - Coordonner les séquences d'actions/réactions d'une application Web
 - Récupérer les données utilisateurs, les filtrer et de les contrôler,
 - Déclencher le traitement approprié (via le **Modèle**),
 - Déléguer la production du document de sortie à la **Vue** adaptée
- Ils permettent de :
 - Structurer hiérarchiquement l'application
 - Faciliter la compréhension du code et la maintenance
- Comme pour la Vue, on évitera de placer de la logique métier dans le contrôleur, car le code ne sera pas réutilisable.

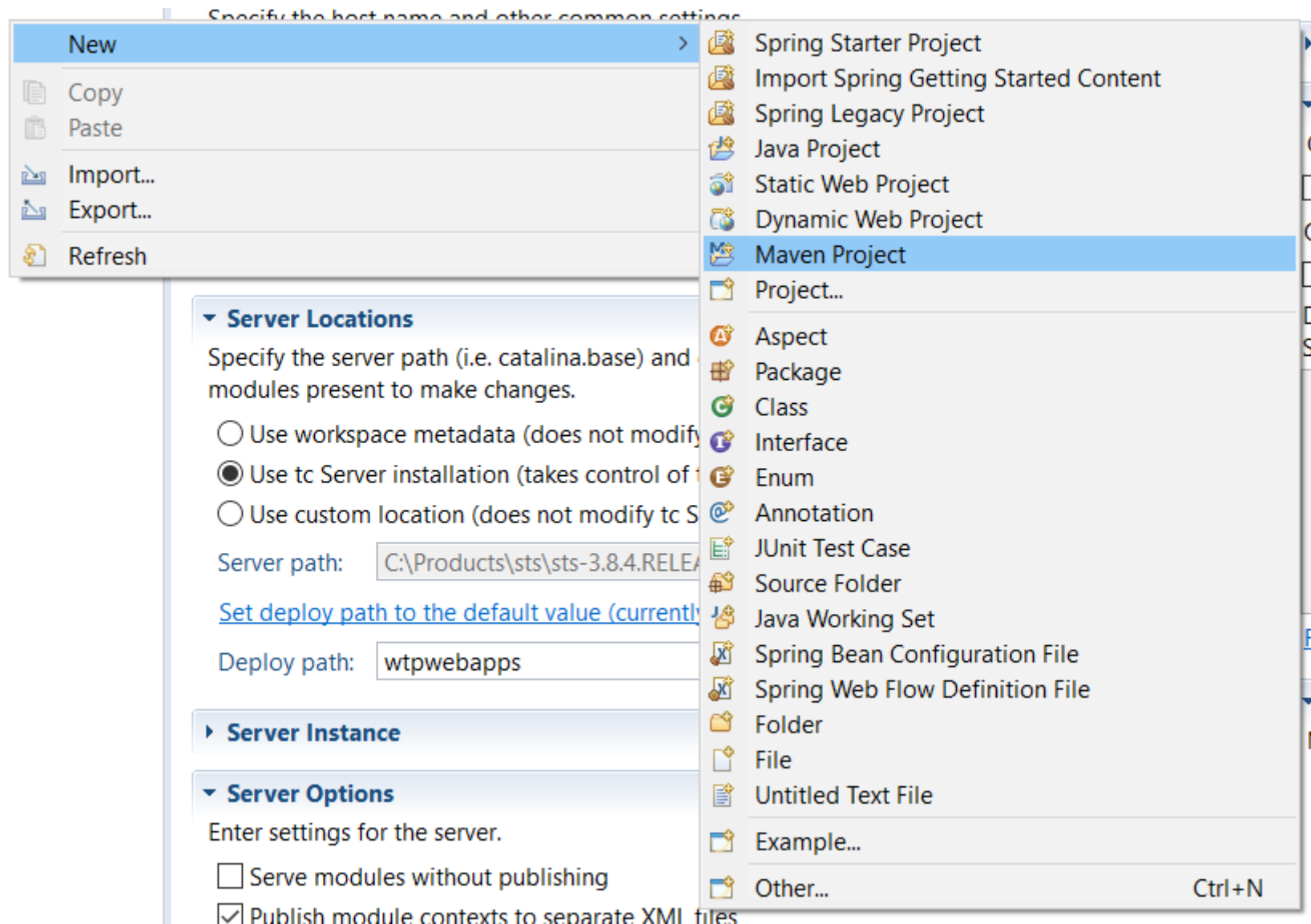
TP10 & TP11 : DEUX EXEMPLES MVC

- **MVC : deux exemples pratiques**

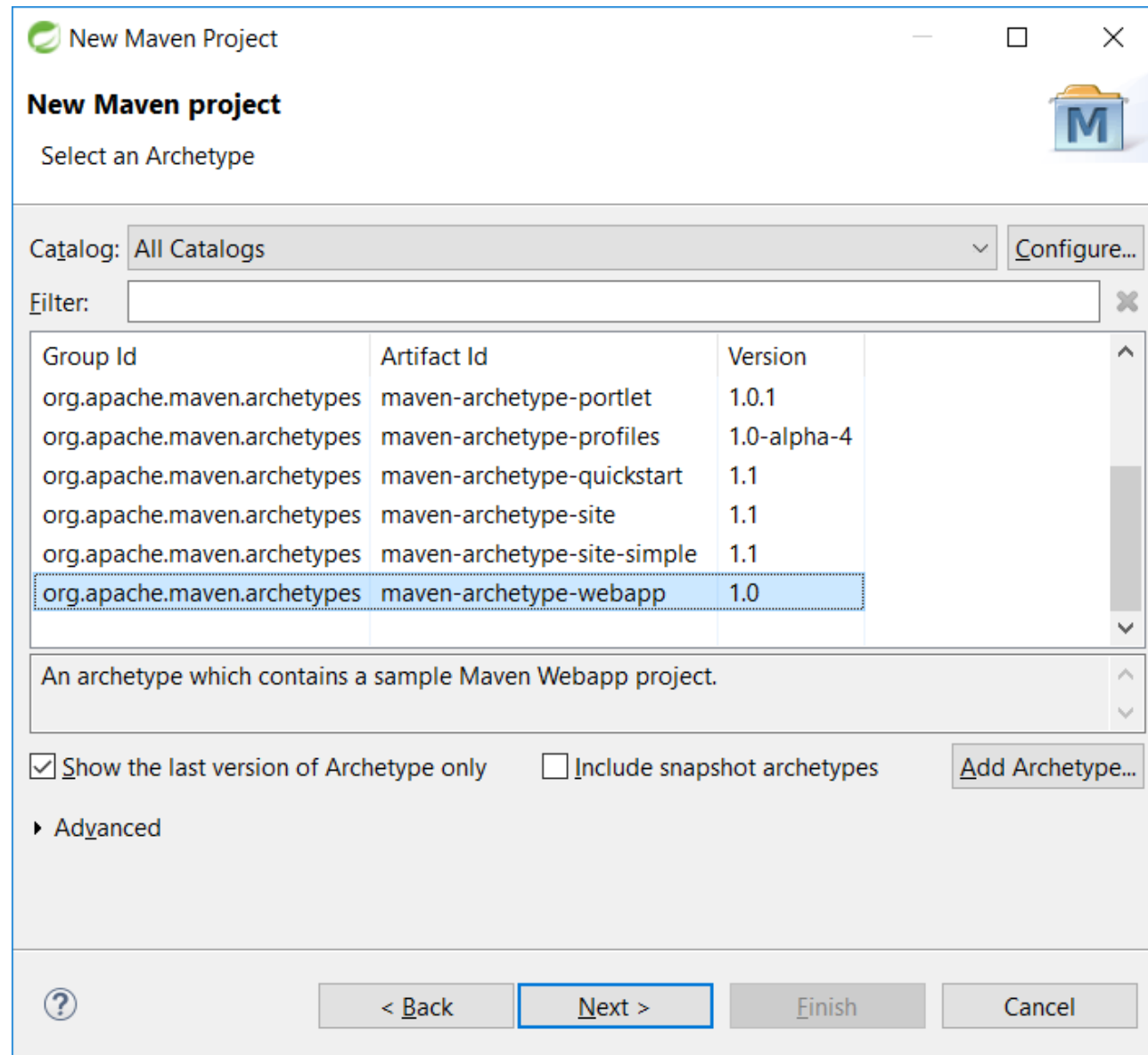
TP10 : MVC (POIDS IDÉAL)

- Calculateur du Poids idéal masculin :
- Poids idéal Masculin (en Kg) = Taille (en cm) - 100 - ((Taille (en cm) - 150) / 4).
- Un seul contrôleur (**Control**)
- Un seul objet métier (**Model**)
- Deux vues (**View**) : 2 JSP
 - Un formulaire permettant de saisir une taille donnée
 - Le résultat de montrant le poids idéal en fonction de la taille

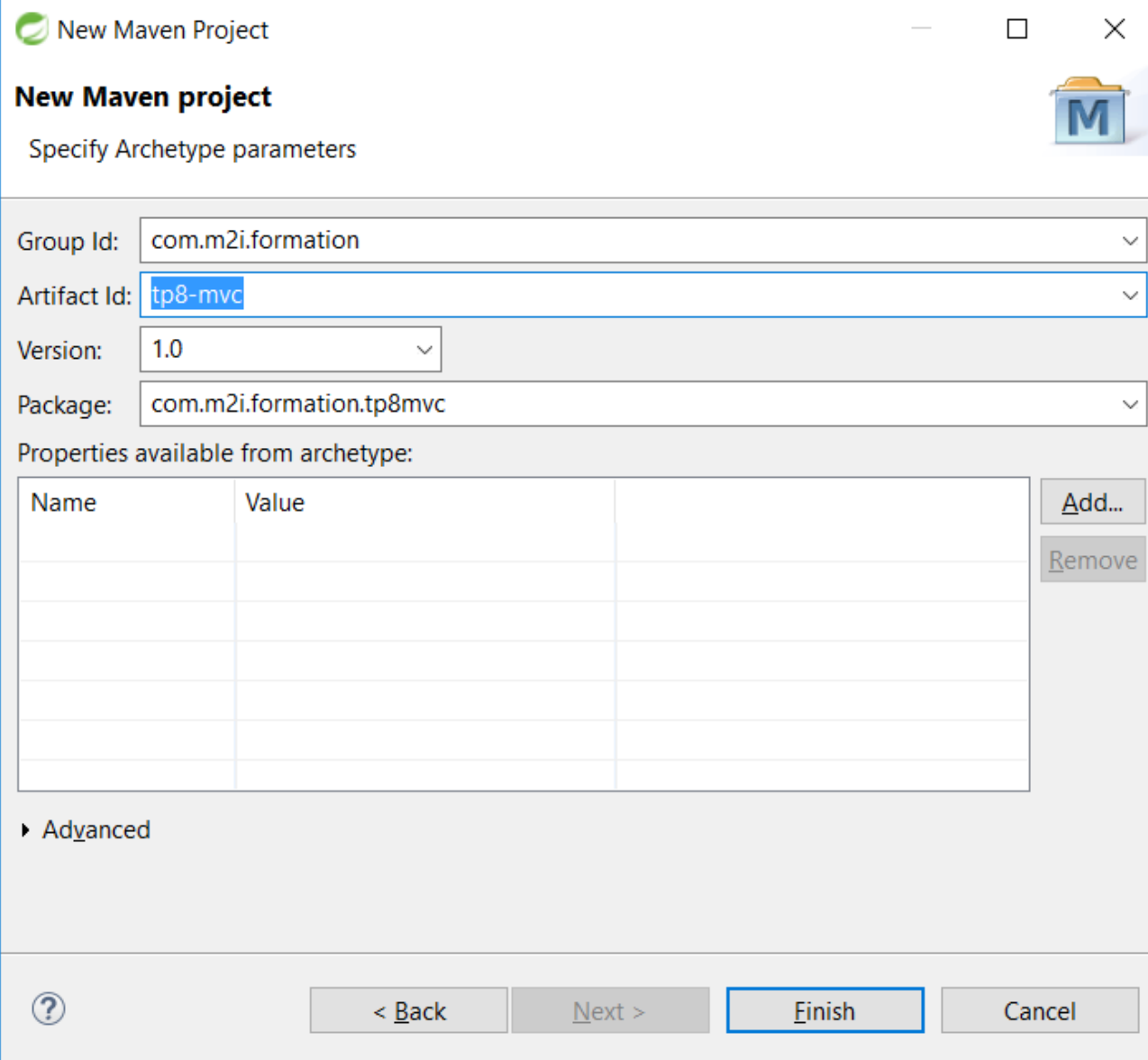
TP10 - MVC : SOLUTION PAS À PAS



TP10 - MVC



TP10 - MVC



The image shows a 'New Maven Project' dialog box. At the top, it says 'New Maven Project' with a green icon. Below that, 'New Maven project' is written in bold, followed by 'Specify Archetype parameters'. There is a small icon of a folder with an 'M' on it. The main section contains four input fields: 'Group Id' with 'com.m2i.formation', 'Artifact Id' with 'tp8-mvc', 'Version' with '1.0', and 'Package' with 'com.m2i.formation.tp8mvc'. Below these is a section titled 'Properties available from archetype:' which contains a table with two columns: 'Name' and 'Value'. The table is empty. To the right of the table are two buttons: 'Add...' and 'Remove'. At the bottom left, there is a link 'Advanced' with a right-pointing triangle. At the bottom right, there are four buttons: a help icon (?), '< Back', 'Next >', and 'Finish' (which is highlighted with a blue border), and 'Cancel'.

New Maven Project

New Maven project

Specify Archetype parameters

Group Id: com.m2i.formation

Artifact Id: tp8-mvc

Version: 1.0

Package: com.m2i.formation.tp8mvc

Properties available from archetype:

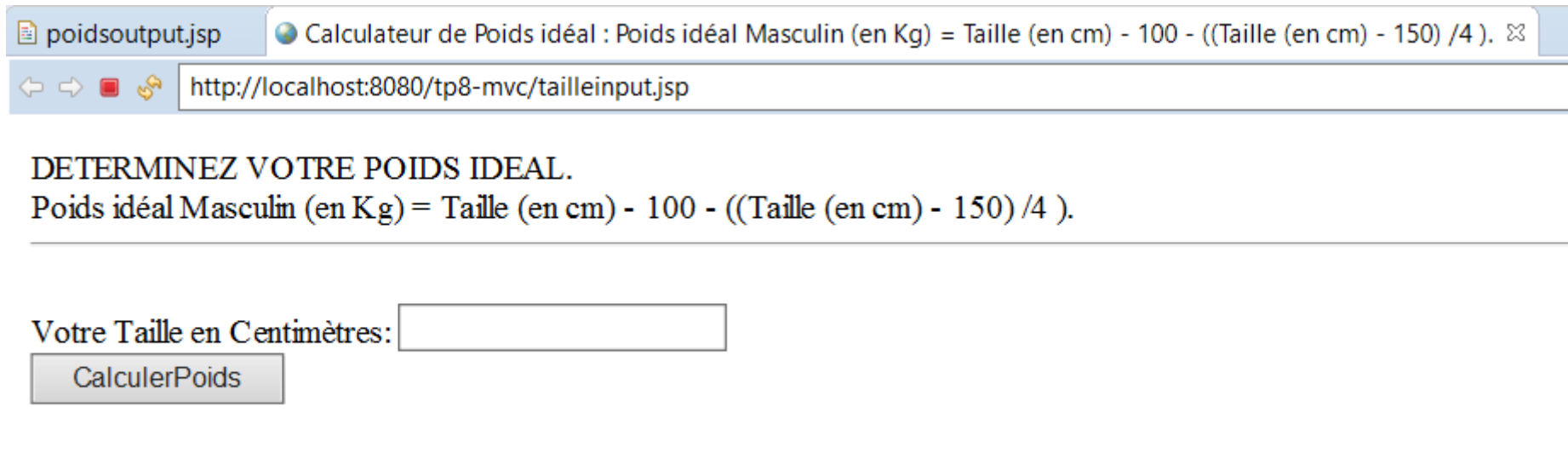
Name	Value

Advanced

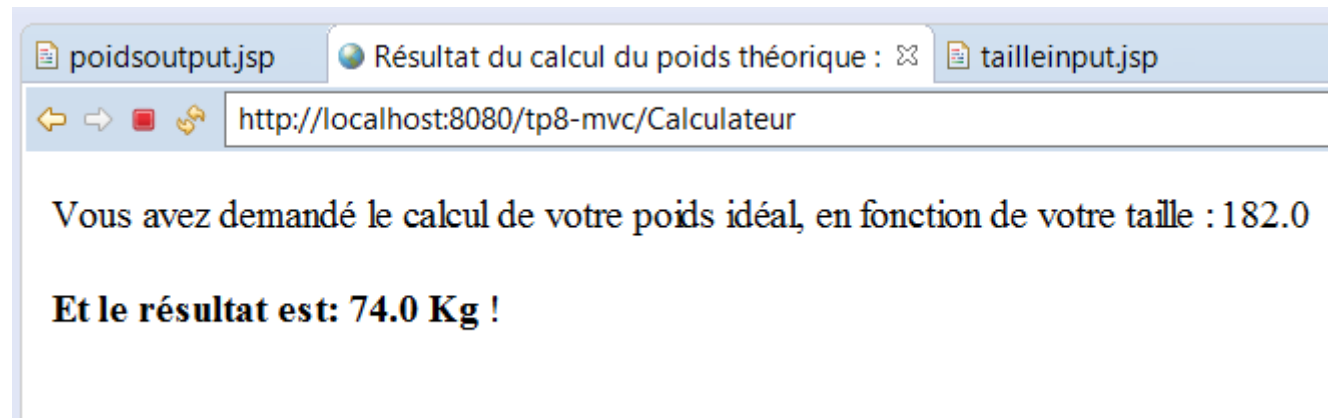
< Back Next > Finish Cancel

TP10 – MVC

- Voici le résultat attendu :



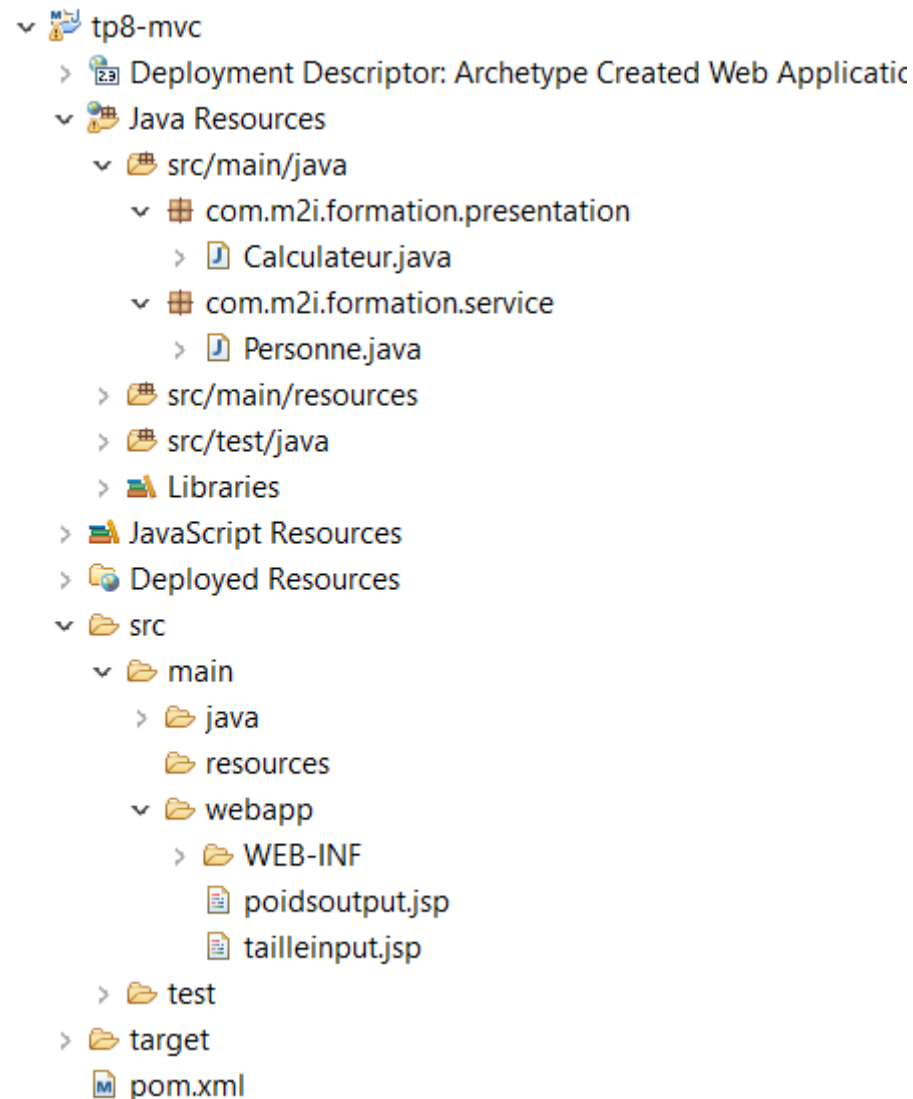
The screenshot shows a web browser window with two tabs: 'poidsoutput.jsp' and 'Calculateur de Poids idéal : Poids idéal Masculin (en Kg) = Taille (en cm) - 100 - ((Taille (en cm) - 150) / 4).'. The address bar shows 'http://localhost:8080/tp8-mvc/tailleinput.jsp'. The page content includes the text 'DETERMINEZ VOTRE POIDS IDEAL.' followed by the formula 'Poids idéal Masculin (en Kg) = Taille (en cm) - 100 - ((Taille (en cm) - 150) / 4).'. Below this is a label 'Votre Taille en Centimètres:' next to an empty text input field. A button labeled 'CalculerPoids' is positioned below the input field.



The screenshot shows a web browser window with three tabs: 'poidsoutput.jsp', 'Résultat du calcul du poids théorique :', and 'tailleinput.jsp'. The address bar shows 'http://localhost:8080/tp8-mvc/Calculateur'. The page content displays the text 'Vous avez demandé le calcul de votre poids idéal, en fonction de votre taille : 182.0' and 'Et le résultat est: 74.0 Kg !'.

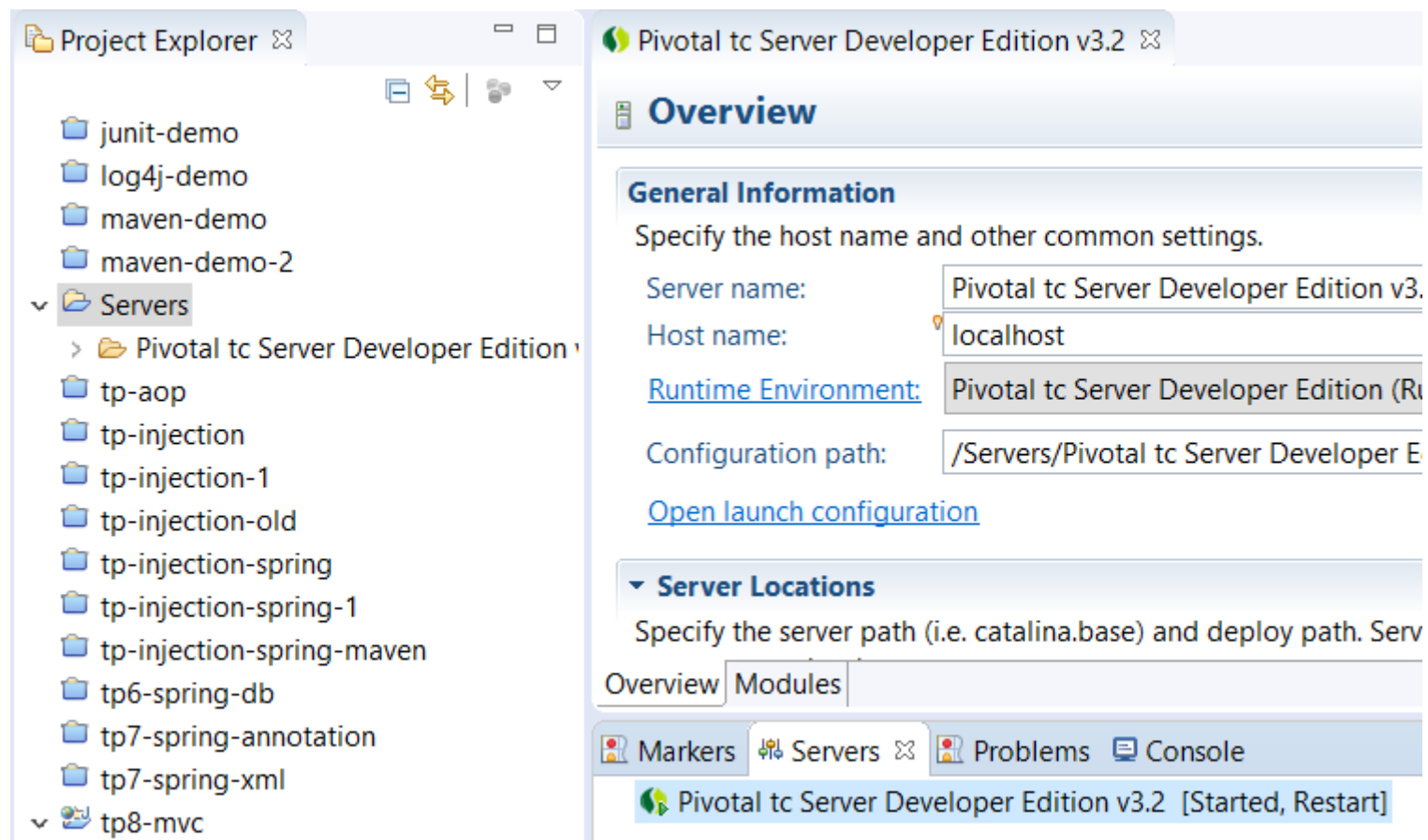
TP10 – MVC

- Voici le résultat attendu :



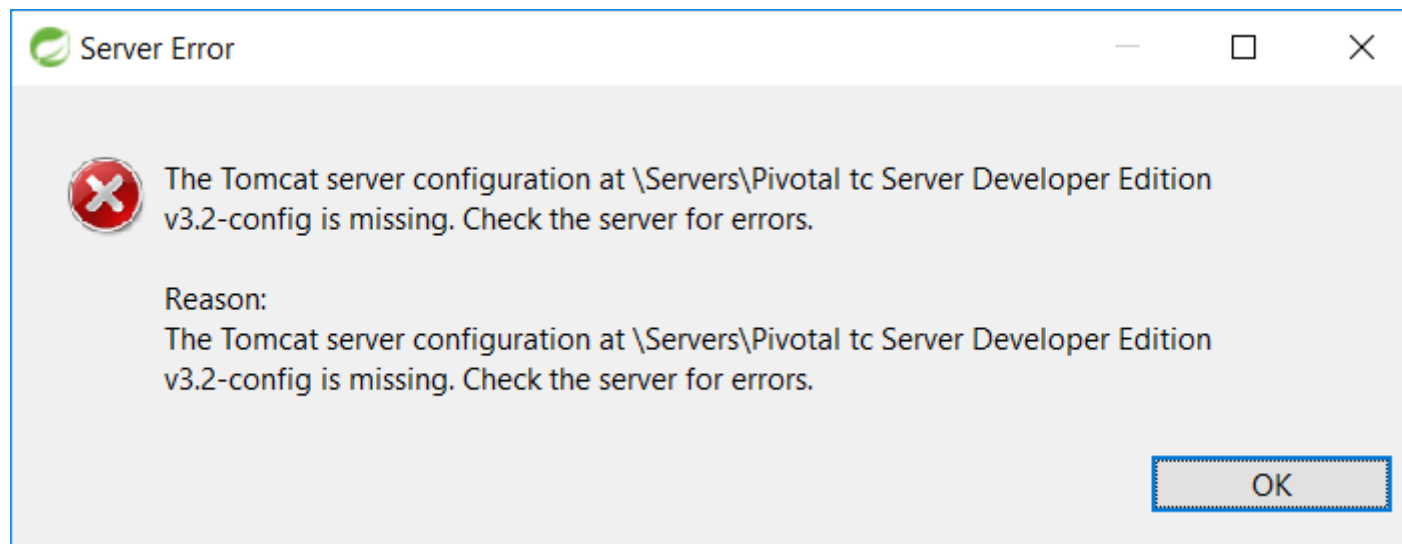
TP10 – MVC : TOMCAT STS

- Assurez-vous que le Serveur Tomcat fourni avec STS est bien configuré, et est lancé:

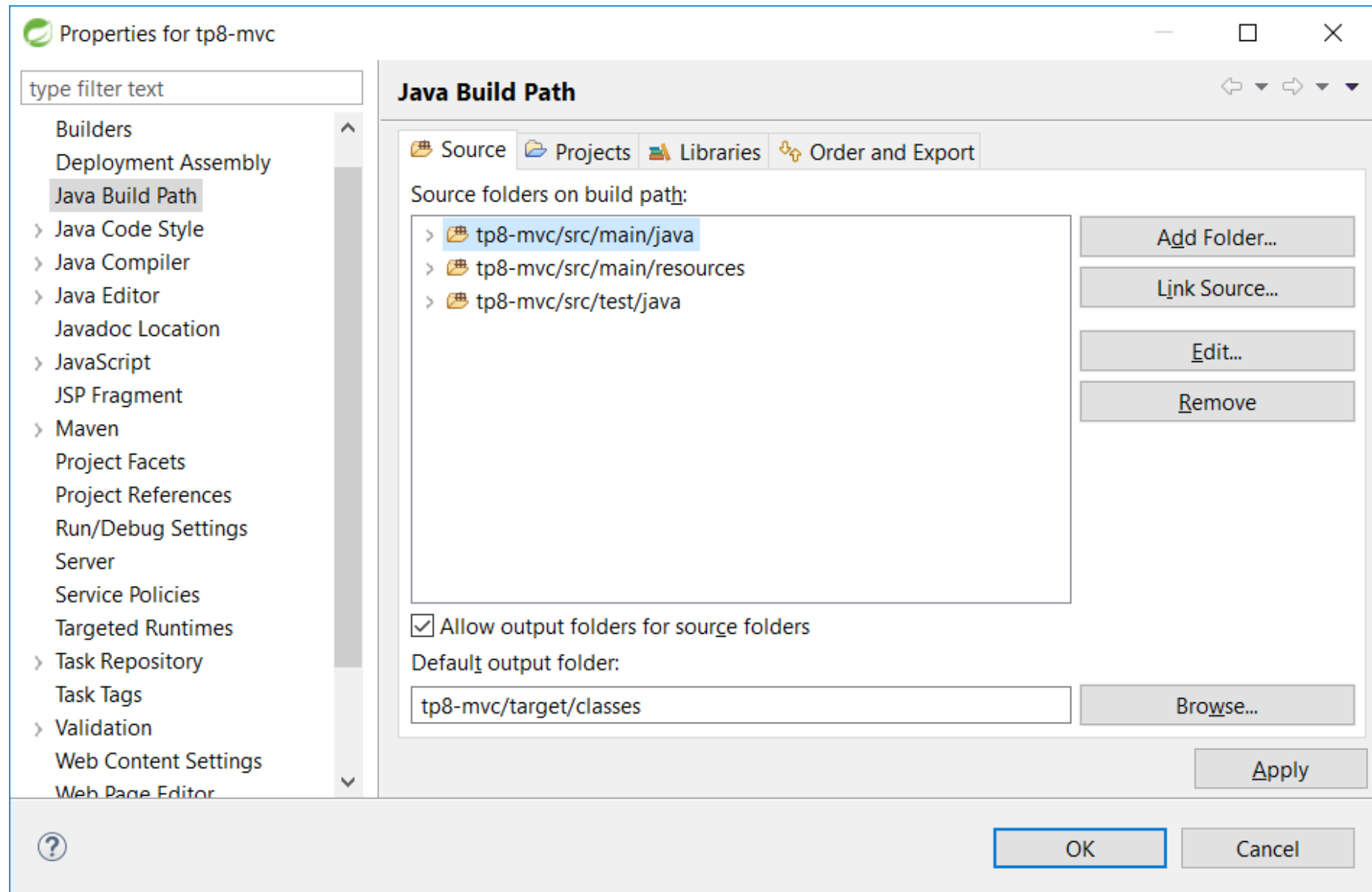


TP10 – MVC : TOMCAT STS

- Si l'erreur suivante, alors ouvrez le projet Eclipse « Servers » :



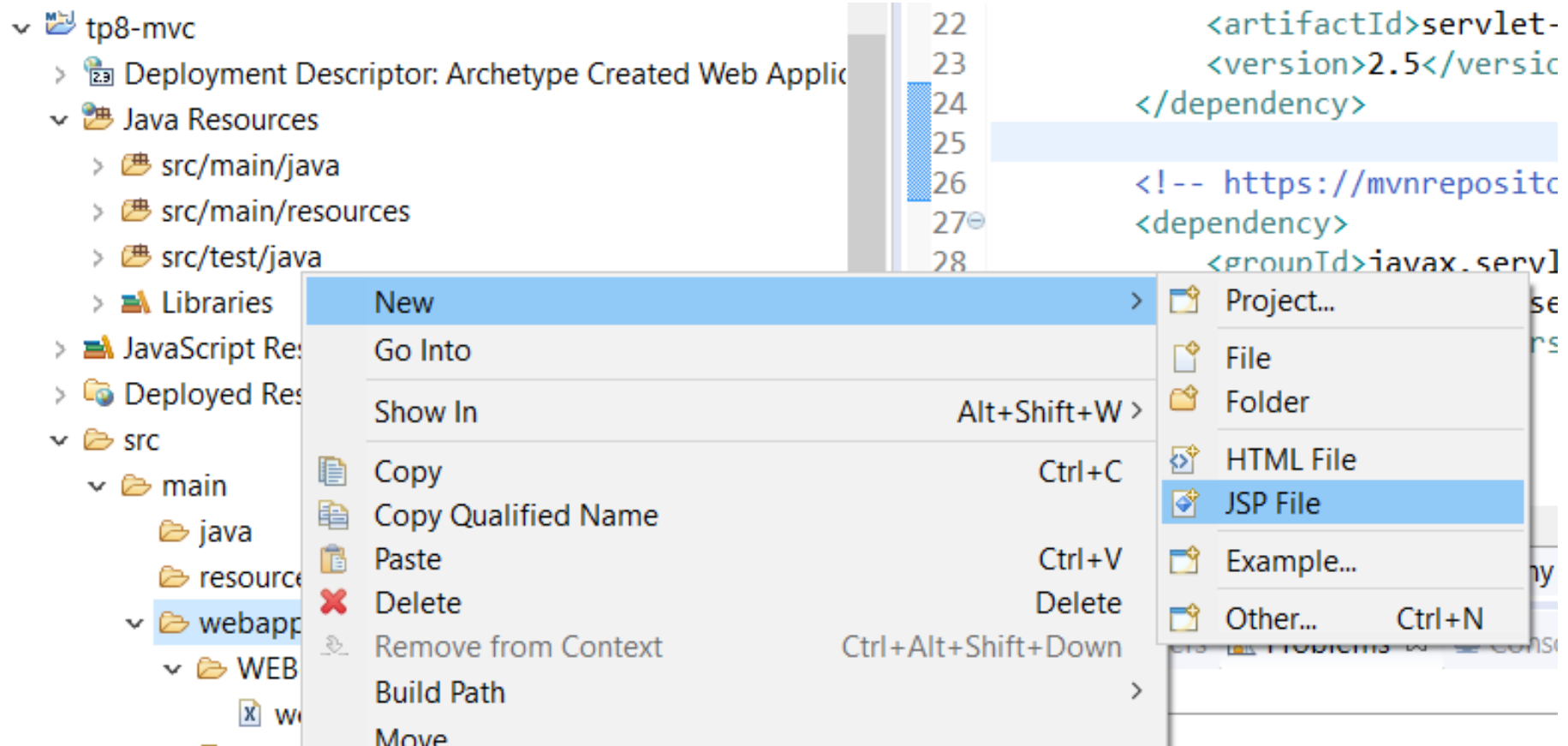
TP10 – MVC : BUILD PATH



TP10 – MVC : POM.XML

```
<!-- Add dependencies in pom.xml : -->
<!-- https://mvnrepository.com/artifact/javax.servlet/servlet-api -->
<dependency>
<groupId>javax.servlet</groupId>
<artifactId>servlet-api</artifactId>
<version>2.5</version>
</dependency>
<!-- https://mvnrepository.com/artifact/javax.servlet.jsp/javax.servlet.jsp-api -->
<dependency>
<groupId>javax.servlet.jsp</groupId>
<artifactId>javax.servlet.jsp-api</artifactId>
<version>2.3.1</version>
</dependency>
```

TP10 – MVC : JSP INPUT



TP10 – MVC : JSP INPUT

```
<!-- tailleinput.jsp -->
<%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8"%>
<%@ page isELIgnored="false"%>
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title>Calculateur de Poids idéal : Poids idéal Masculin (en Kg) = Taille (en cm) - 100
- ((Taille (en cm) - 150) /4 ). </title>
</head>
<body>
DETERMINEZ VOTRE POIDS IDEAL.<br>
Poids idéal Masculin (en Kg) = Taille (en cm) - 100 - ((Taille (en cm) - 150) /4 ).
<hr />
<form method="POST" action="${pageContext.request.contextPath}/Calculateur">
Votre Taille en Centimètres: <input type="text" size="20" name="taille" /> <br />
<input type="submit" value="CalculerPoids" />
</form>
<hr />
</body>
</html>
```

TP10 – MVC : JSP INPUT

Project Explorer

- tp8-mvc
 - Deployment Descriptor: Arc
 - Java Resources
 - src/main/java
 - src/main/resources
 - src/test/java
 - Libraries
 - JavaScript Resources
 - Deployed Resources
 - src
 - main
 - java
 - resources
 - webapp
 - WEB-INF
 - web.xml
 - index.jsp
 - tailleinput.jsp
 - test
 - target
 - pom.xml

tailleinput.jsp | web.xml | Calculateur de Poids idéal : Poids idéal Masculin (en Kg) =

http://localhost:8080/tp8-mvc/tailleinput.jsp

DETERMINEZ VOTRE POIDS IDEAL.
Poids idéal Masculin (en Kg) = Taille (en cm) - 100 - ((Taille (en cm) - 150) / 4).

Votre Taille en Centimètres:

CalculerPoids

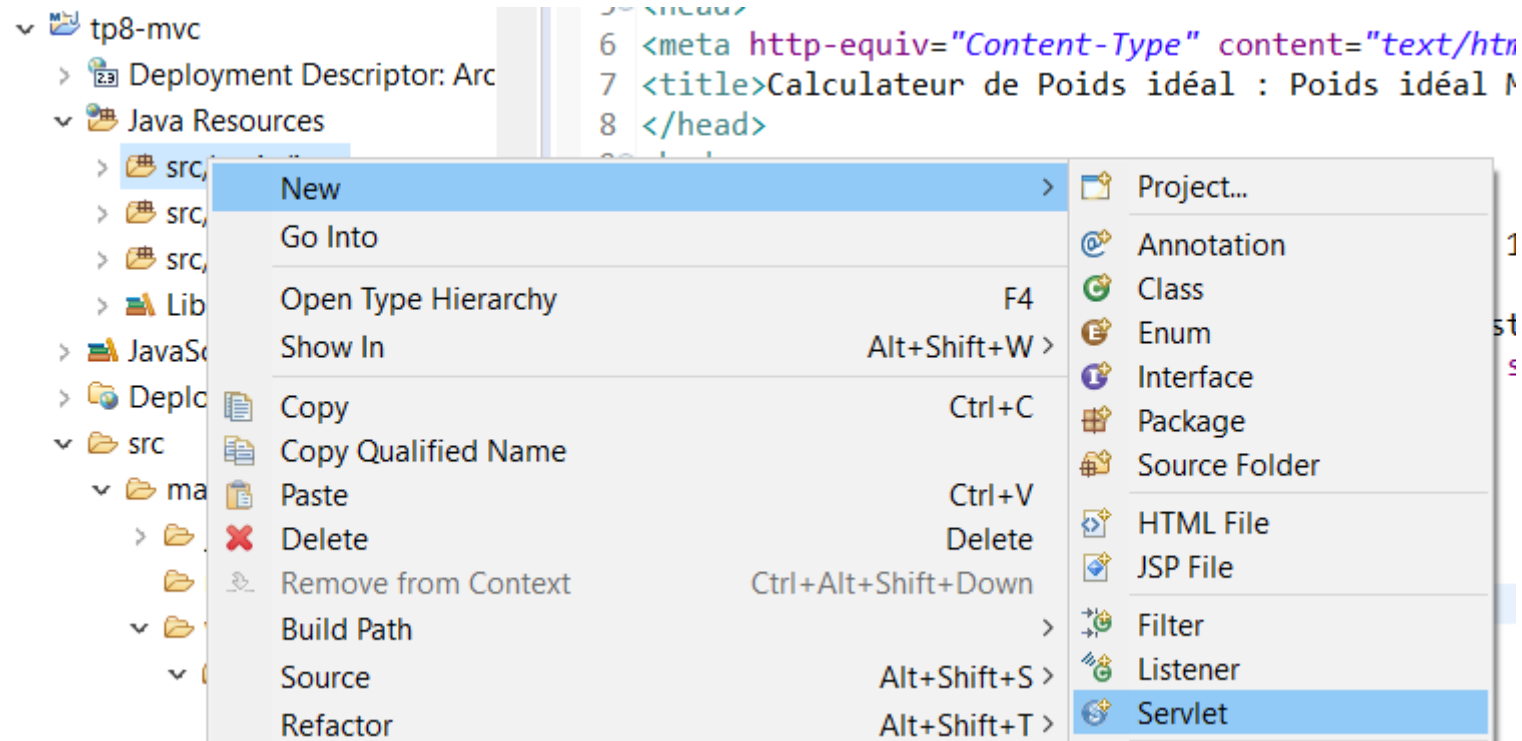
Markers | **Servers** | **Problems** | **Console**

- Pivotal tc Server Developer Edition v3.2 [Started, Synchronized]
 - tp8-mvc [Started, Synchronized]
 - tp8-mvc
J2EE Web module
Press 'F6' for Focus.

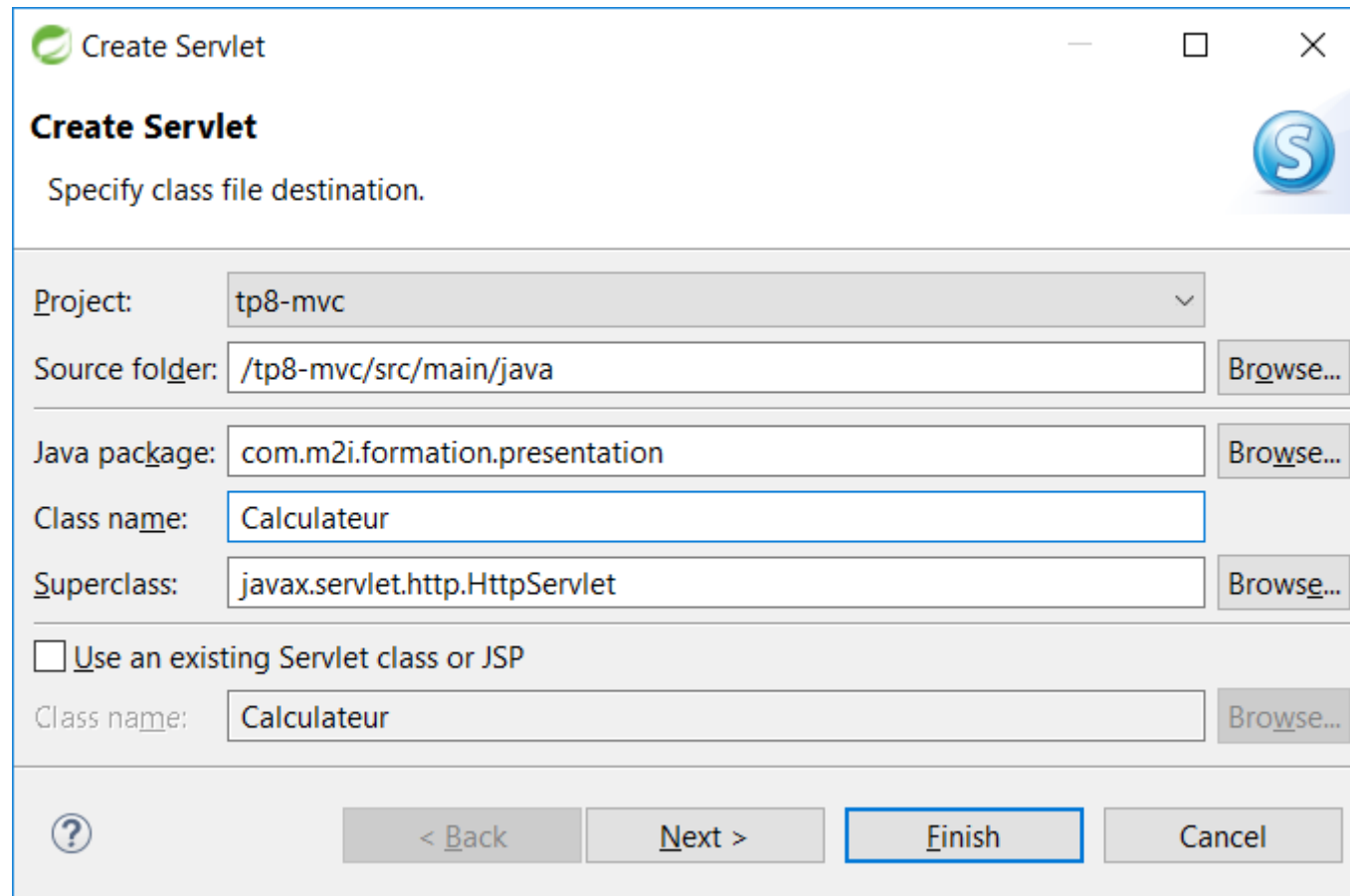
TP10 – MVC : CONTROLLER

- Une **Servlet** simple :
- À associer à l'URL “/Calculateur”
- Avec deux méthodes
- **doGet**, affichage du formulaire (taille)
- **doPost**, reçoit le paramètre du formulaire et demande le calcul du poids (action)

TP10 – MVC : CONTROLLER



TP10 – MVC : CONTROLLER



The image shows a 'Create Servlet' dialog box from an IDE. It has a title bar with a green icon and the text 'Create Servlet'. Below the title bar, there's a sub-header 'Create Servlet' and a description 'Specify class file destination.' with a blue 'S' icon. The dialog contains several input fields: 'Project' (a dropdown menu showing 'tp8-mvc'), 'Source folder' (a text field with '/tp8-mvc/src/main/java' and a 'Browse...' button), 'Java package' (a text field with 'com.m2i.formation.presentation' and a 'Browse...' button), 'Class name' (a text field with 'Calculateur'), and 'Superclass' (a text field with 'javax.servlet.http.HttpServlet' and a 'Browse...' button). There is a checkbox labeled 'Use an existing Servlet class or JSP' which is currently unchecked. Below this checkbox is another 'Class name' text field with 'Calculateur' and a 'Browse...' button. At the bottom, there are four buttons: a help button (question mark icon), '< Back', 'Next >', and 'Finish' (which is highlighted with a blue border), and a 'Cancel' button.

Create Servlet
Specify class file destination.

Project: tp8-mvc

Source folder: /tp8-mvc/src/main/java [Browse...](#)

Java package: com.m2i.formation.presentation [Browse...](#)

Class name: Calculateur

Superclass: javax.servlet.http.HttpServlet [Browse...](#)

☐ Use an existing Servlet class or JSP

Class name: Calculateur [Browse...](#)

[?](#) [< Back](#) [Next >](#) **Finish** [Cancel](#)

TP10 – MVC : CONTROLLER

```
/** On affiche le formulaire de saisie de la taille de la personne. Le controleur
    redirige l'utilisateur vers la JSP mentionnée : */
protected void doGet(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {
String maVue = "/tailleinput.jsp";
RequestDispatcher dispatcher = getServletContext().getRequestDispatcher(maVue);
dispatcher.forward(request,response);
}
```

TP10 – MVC : MODEL

Classe totalement indépendante du contexte Web et peut être utilisée dans d'autres applications Java :

```
package com.m2i.formation.service;
```

```
/** Une classe permettant d'obtenir le poids théorique en fonction de la taille d'une personne */
```

```
public class Personne {
```

```
/** La valeur, exprimée en centimètres */
```

```
private double taille;
```

```
/** Le constructeur, prend la taille en paramètre */
```

```
public personne(double valeurTaille) {taille = valeurTaille;}
```

```
/** Pour récupérer la taille de la personne */
```

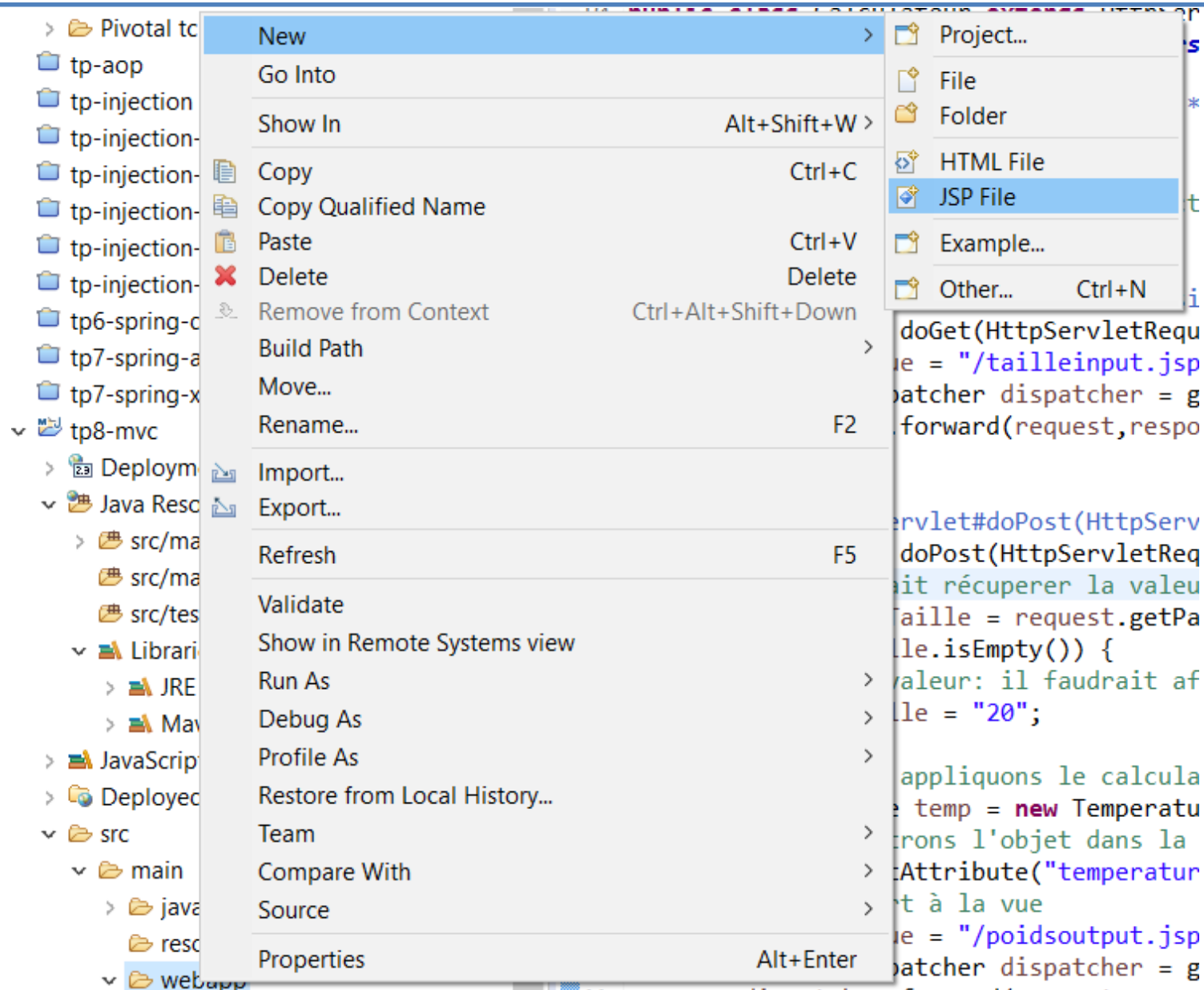
```
public double getTaille() {return taille;}
```

```
/** Pour obtenir le poids théorique : Poidsidéal(homme)=H-100-((H-15)/4) */
```

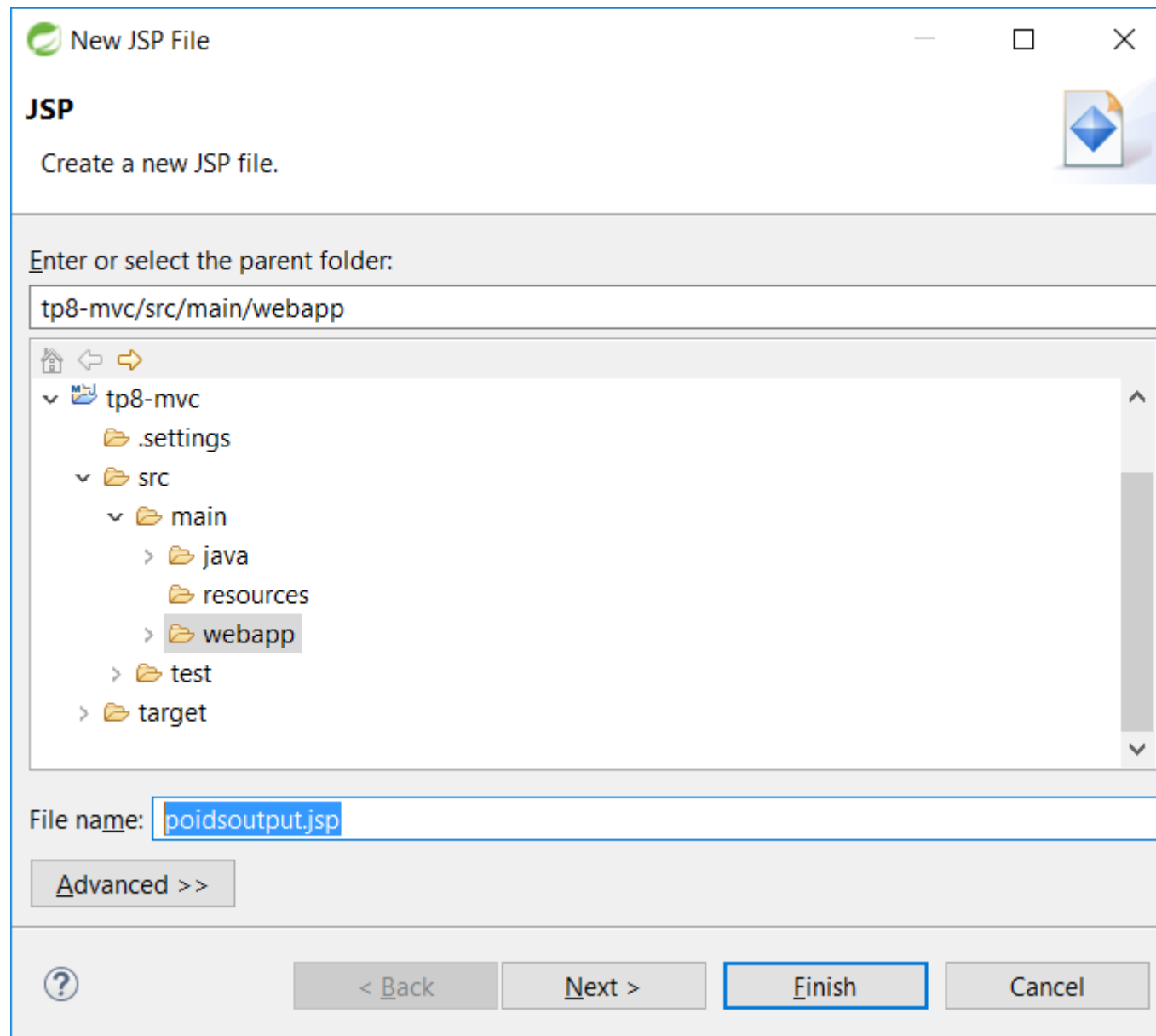
```
public double getPoids() {return taille - 100 - ((taille - 15)/4); }
```

```
}
```

TP10 – MVC : JSP DU RÉSULTAT



TP10 – MVC : JSP DU RÉSULTAT

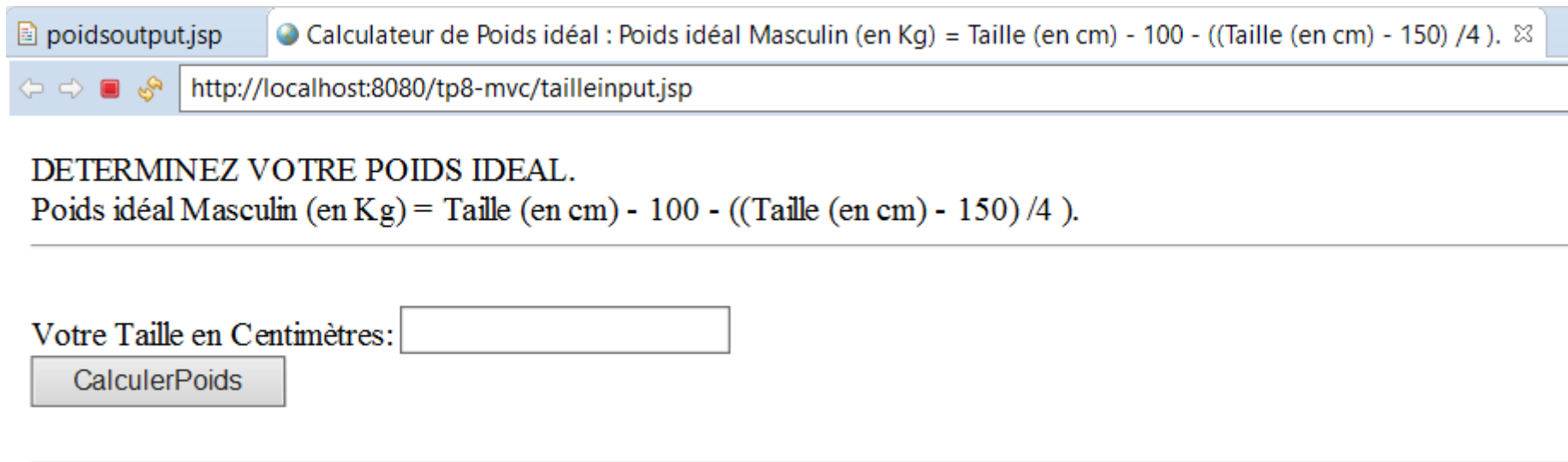


TP10 – MVC : JSP DU RÉSULTAT

```
<!-- poidsoutput.jsp -->
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<%@ page isELIgnored="false"%>
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title>Résultat du calcul du poids théorique : </title>
</head>
<body>

<p>Vous avez demandé le calcul de votre poids idéal, en fonction de
    votre taille : ${requestScope.personne.taille}</p>
<p>
<b>Et le résultat est: ${requestScope.personne.poids}   Kg </b>!
</p>
</body>
</html>
```

TP10 – MVC : RÉSULTAT

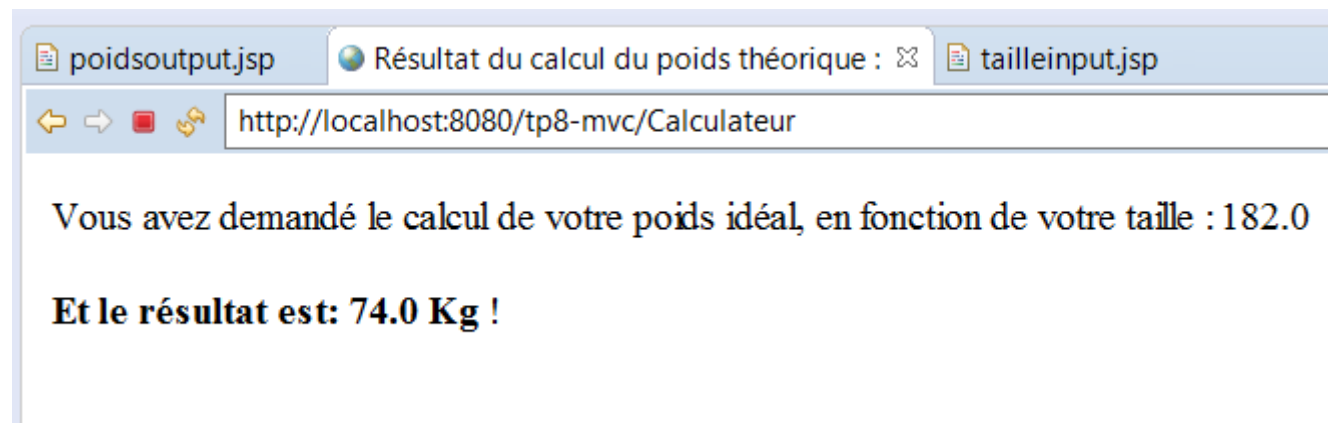


DETERMINEZ VOTRE POIDS IDEAL.

Poids idéal Masculin (en Kg) = Taille (en cm) - 100 - ((Taille (en cm) - 150) / 4).

Votre Taille en Centimètres:

CalculerPoids



Vous avez demandé le calcul de votre poids idéal, en fonction de votre taille : 182.0

Et le résultat est: 74.0 Kg !

TP11 : MVC (CONVERTISSEUR DE TEMPÉRATURE)

- Convertisseur de température de l'unité Celsius vers l'unité Fahrenheit :
- $^{\circ}\text{F} = (^{\circ}\text{C} \times 1,8) + 32$
- Un seul contrôleur (**Control**)
- Un seul objet métier (**Model**)
- Deux vues (**View**) : 2 JSP
 - Un formulaire permettant de saisir une valeur en Celsius
 - Le résultat de la conversion de la valeur entrée en Fahrenheit

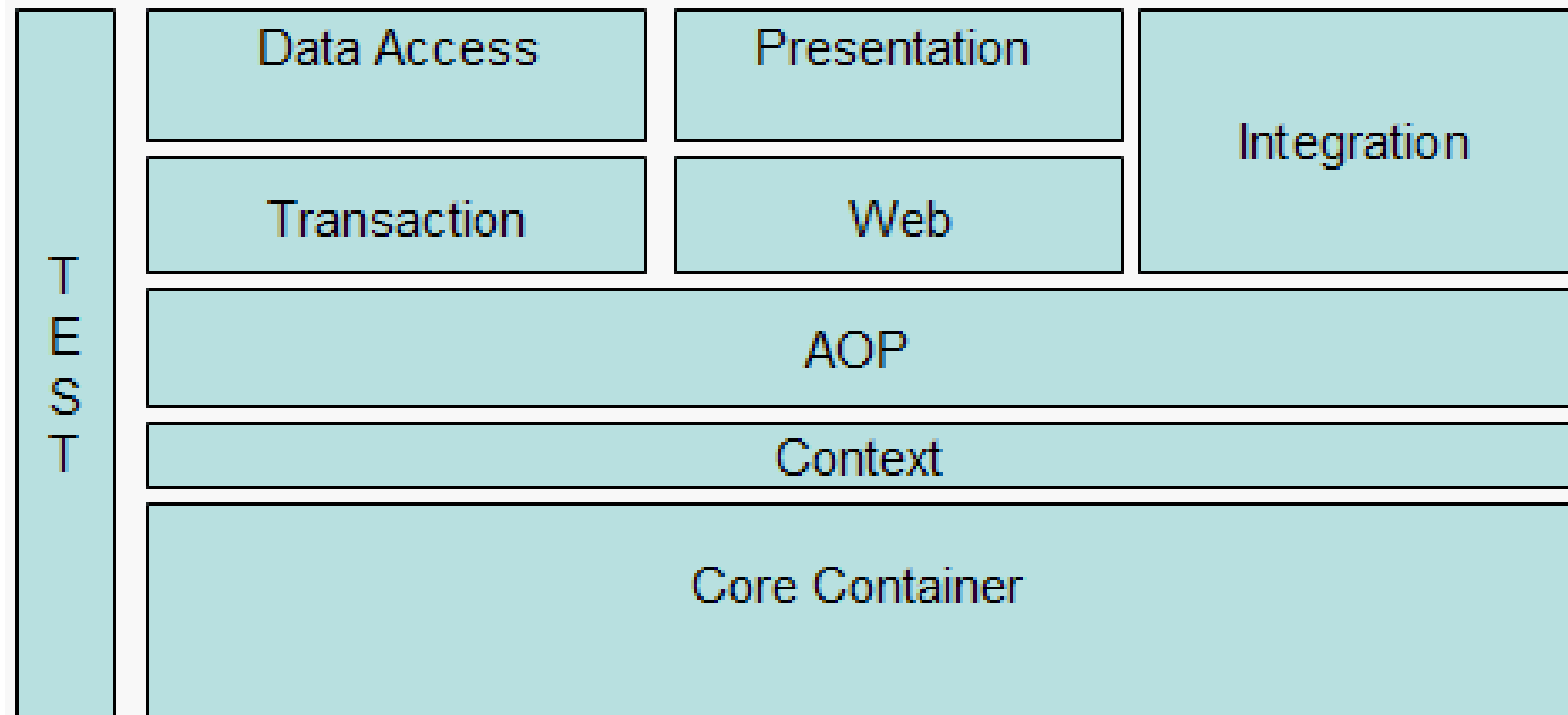
SPRING MVC



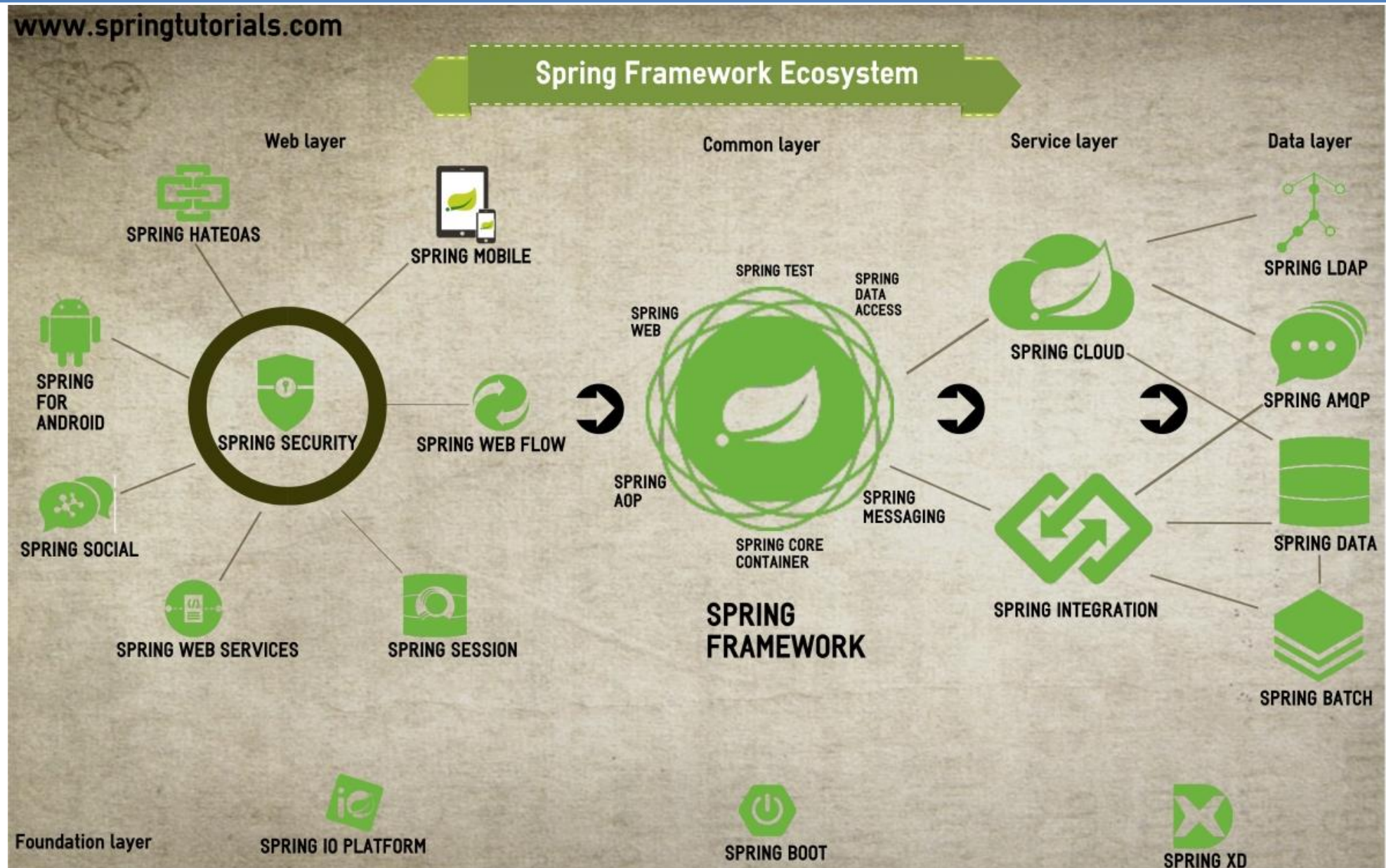
LES PROJETS SPRING

- **Spring MVC** est un module de Spring Framework.
- Ce n'est pas un projet Spring indépendant.
- Il implémente le Design Pattern Model-View-Controller
- Il permet de faire des applications Web d'une façon configurable et assez simple.
- Spring MVC est un concurrent direct de Struts.
- Une application qui utilise le Framework Spring, n'est pas obligée d'utiliser Spring MVC pour la couche Web. Elle peut utiliser Struts, Spring MVC ou tout autre Framework web.

SPRING FRAMEWORK (CORE)



LES PROJETS SPRING



LE CONTROLLER DE SPRING MVC

- Un Bean Spring, annoté **@Controller**
- Supporte donc l'injection de dépendance : c'est ainsi qu'il accède à la couche Service
- C'est lui qui intercepte les requêtes, appelle la couche Model (Service + DAO) pour faire les traitements nécessaires, et redirige vers la bonne Vue (View : Web).

LA VUE (VIEW) DE SPRING MVC

- C'est typiquement une JSP qui va être chargée d'afficher les données (Model).
- Spring MVC n'est pas limité à la technologie JSP, mais c'est elle qui est le plus couramment utilisée.

LE MODÈLE (MODEL) DE SPRING MVC

- Le modèle est l'ensemble des traitements métier (avec ou sans accès à la base de données).
- Il s'agit des traitements faits dans les couches Service et DAO.
- Le résultat du traitement permet au contrôleur de charger une vue de mettre en forme et d'afficher ces données.

SPRING MVC : CINÉMATIQUE

- 1-

Le client fait une demande au contrôleur.

Celui-ci voit passer toutes les demandes des clients.

C'est la porte d'entrée de l'application.

C'est le **C** de **MVC**.

Ici le contrôleur est assuré par une **Servlet** générique :
org.springframework.web.servlet.DispatcherServlet

SPRING MVC : CINÉMATIQUE

- 2 –

Le contrôleur principal [**DispatcherServlet**] fait exécuter l'action demandée par l'utilisateur par une classe implémentant l'interface : **org.springframework.web.servlet.mvc.Controller**

A cause du nom de l'interface, nous appellerons une telle classe un contrôleur secondaire pour le distinguer du contrôleur principal [DispatcherServlet]

SPRING MVC : CINÉMATIQUE

- 3-

Le contrôleur [**Controller**] traite une demande particulière de l'utilisateur.

Pour ce faire, il peut avoir besoin de l'aide de la couche métier.

Une fois la demande du client traitée, celle-ci peut appeler diverses réponses :

Une page d'erreurs si la demande n'a pu être traitée correctement.

Une page de confirmation...

SPRING MVC : CINÉMATIQUE

- 4-

Le contrôleur choisit la réponse (= VUE (VIEW)) à envoyer au client. Choisir la réponse nécessite plusieurs étapes :

Choisir l'objet qui va générer la réponse. C'est ce qu'on appelle la Vue (View). C'est le V de MVC.

Ce choix dépend en général du résultat de l'exécution de l'action demandée par l'utilisateur.

Lui fournir les données dont il a besoin pour générer cette réponse. En effet, celle-ci contient le plus souvent des informations calculées par la couche métier. Ce calcul se fait dans la couche Model (Service, DAO).

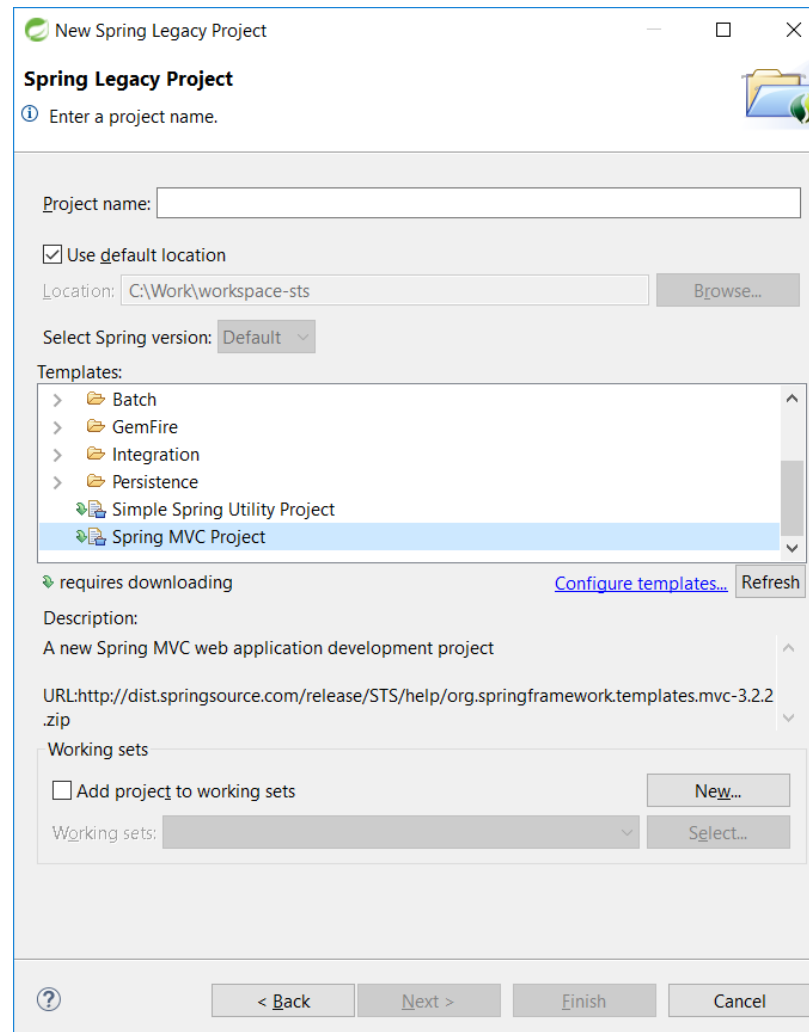
Cette étape consiste donc en le choix d'une vue V et la construction du modèle M nécessaire à celle-ci.

SPRING MVC : CINÉMATIQUE

- 5-
Le contrôleur **DispatcherServlet** demande à la vue choisie de s'afficher. Il s'agit d'une classe implémentant l'interface **org.springframework.web.servlet.View** : Spring MVC propose différentes implémentations de cette interface pour générer des flux HTML, Excel, PDF, ...
- 6.
Le générateur de vue (**View**) utilise les objets, renvoyés par le modèle (**Model**) au contrôleur (**Controller**), pour initialiser les parties dynamiques de la réponse qu'il doit envoyer au client.
- 7.
La réponse est envoyée au client. La forme exacte de celle-ci dépend du générateur de vue. Ce peut être un flux HTML, XML, PDF, Excel, ...

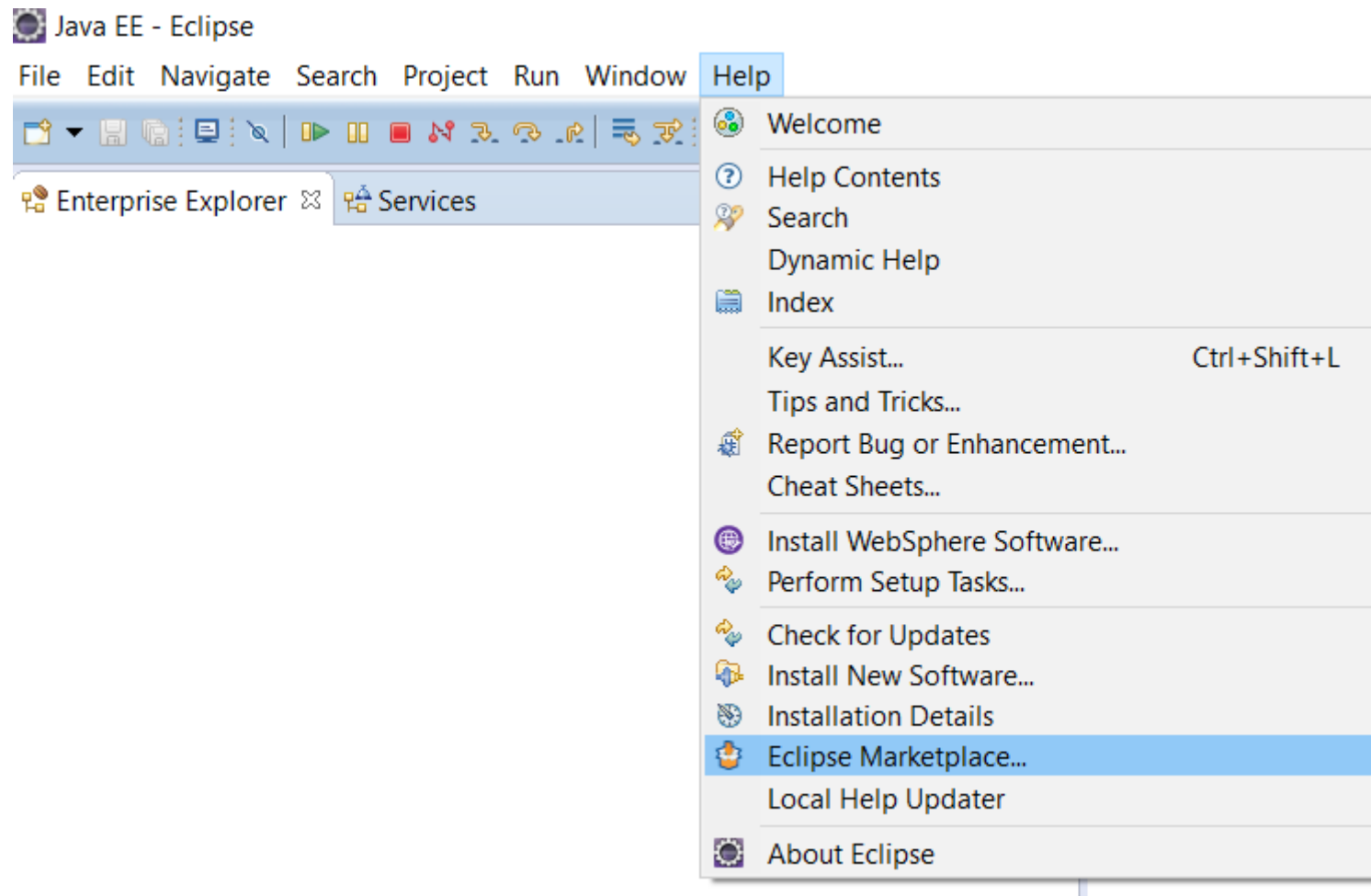
TP 12 : CRÉATION PROJET SPRING MVC - STS & ECLIPSE

- STS (Spring Tools Suite) nous permet de créer des projets de type « **Spring MVC** ».

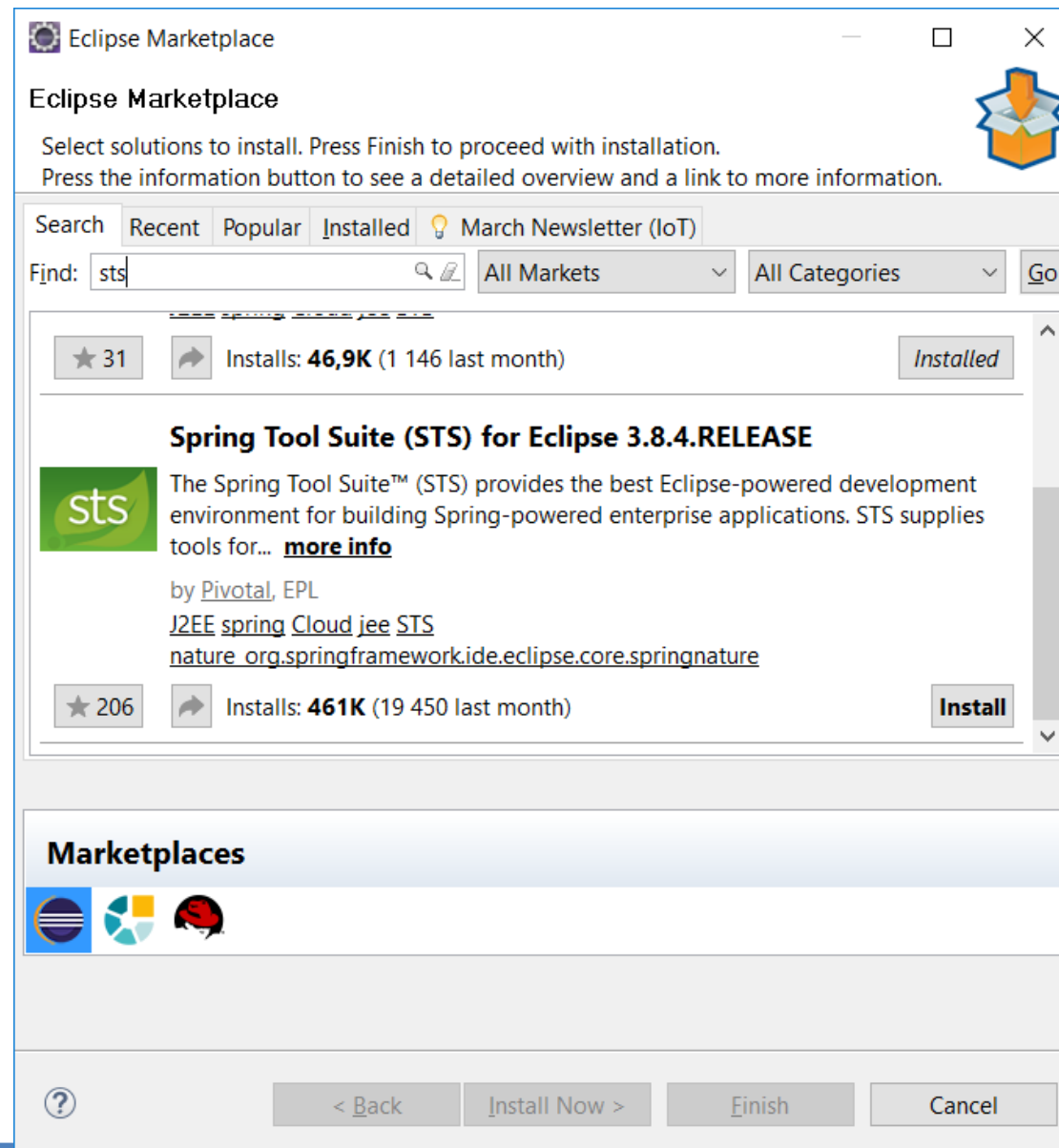


CRÉATION PROJET SPRING MVC - STS & ECLIPSE

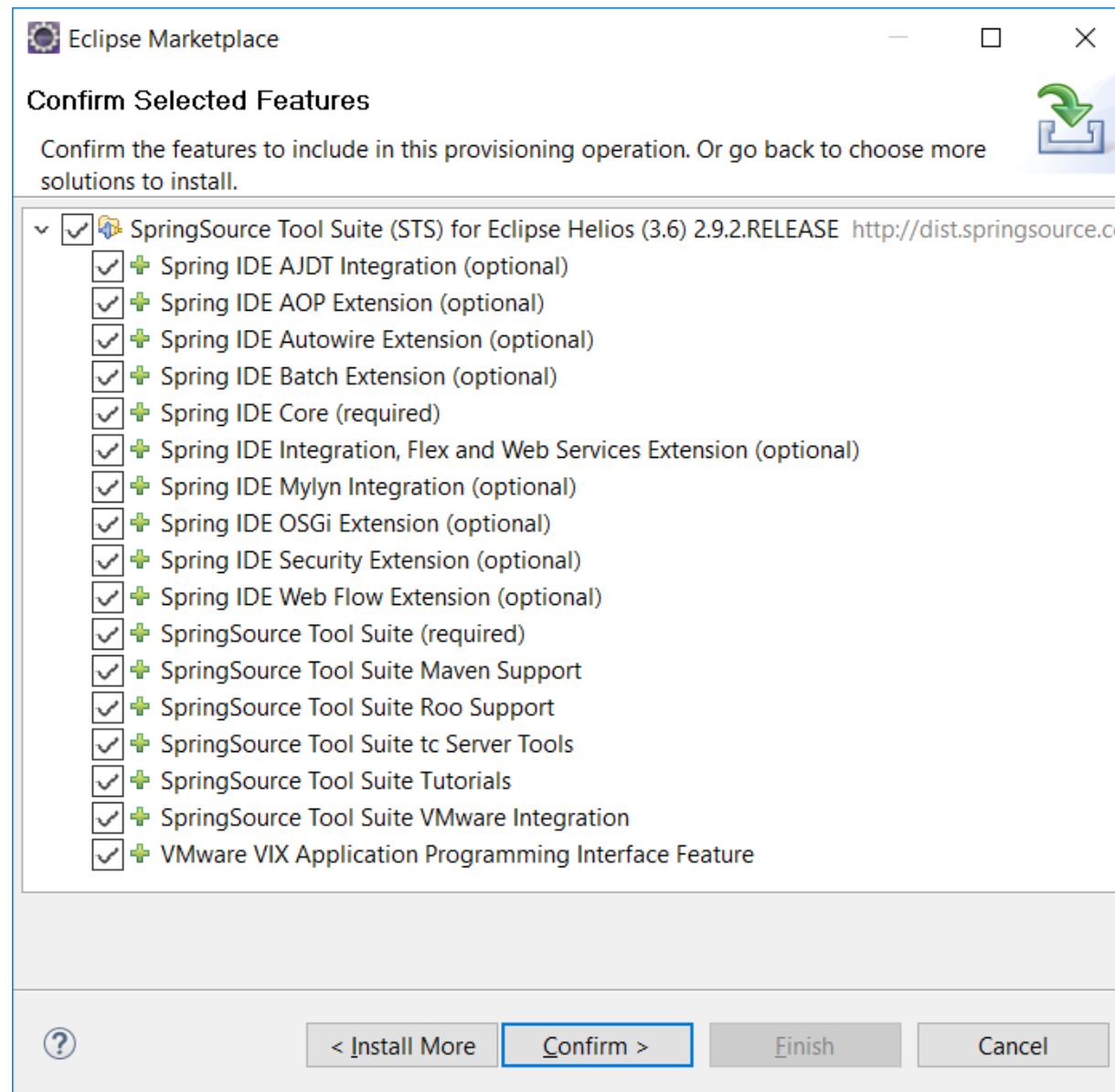
- Sous Eclipse, il vous faut ajouter le plugin STS :



PLUGIN STS POUR ECLIPSE



PLUGIN STS POUR ECLIPSE



PROJET SPRING MVC

New Spring Legacy Project

Spring Legacy Project

Click 'Next' to load the template contents.

Project name:

☒ Use default location

Location:

Select Spring version:

Templates:

- > Persistence
 - Simple Spring Utility Project
 - Spring MVC Project**

requires downloading [Configure templates...](#)

Description:
A new Spring MVC web application development project

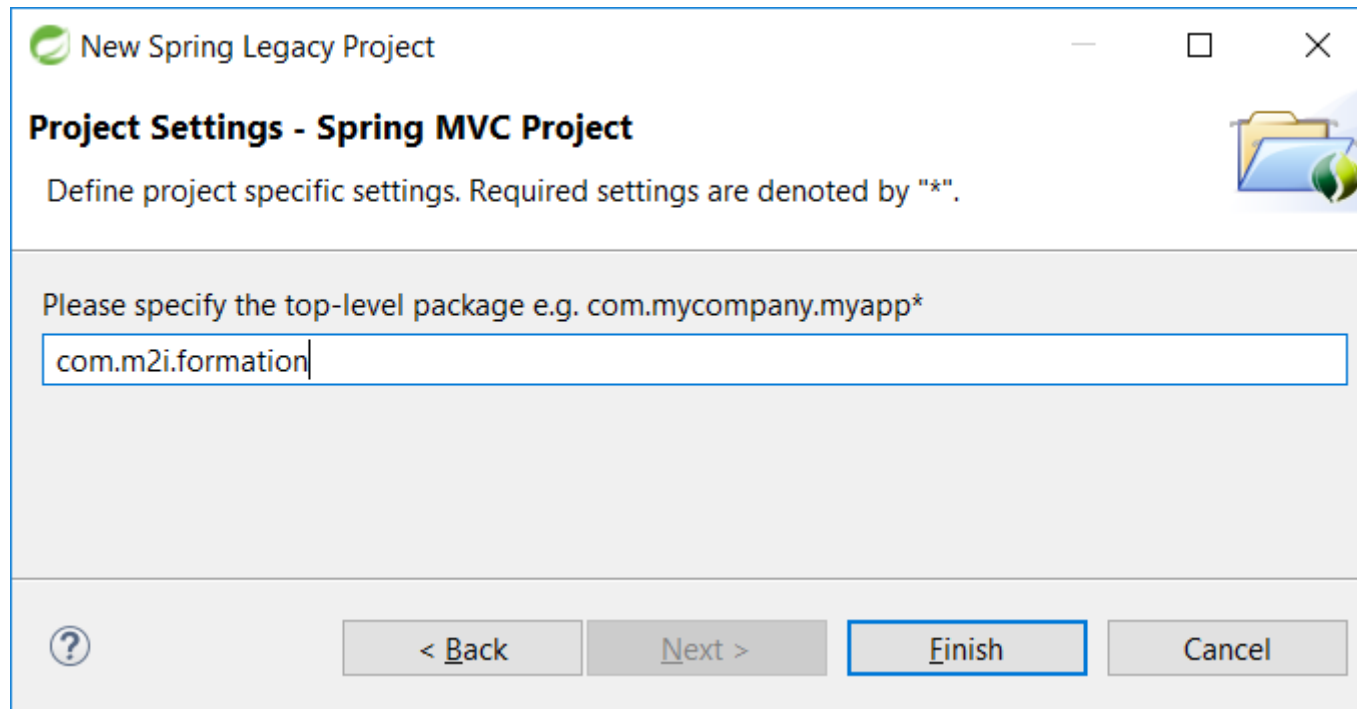
URL:

Working sets

☐ Add project to working sets

Working sets:

PROJET SPRING MVC



New Spring Legacy Project

Project Settings - Spring MVC Project

Define project specific settings. Required settings are denoted by "*".

Please specify the top-level package e.g. com.mycompany.myapp*

com.m2i.formation

? < Back Next > Finish Cancel

STRUCTURE :

- ▼ tp12-spring-mvc
 - > Deployment Descriptor: tp12-spring-mvc
 - > Spring Elements
 - > JAX-WS Web Services
 - ▼ Java Resources
 - ▼ src/main/java
 - ▼ com.m2i.formation
 - > HomeController.java
 - ▼ src/main/resources
 - META-INF
 - log4j.xml
 - > src/test/java
 - > src/test/resources
 - > Libraries
 - > JavaScript Resources
 - > Deployed Resources
 - ▼ src
 - ▼ main
 - > java
 - > resources
 - ▼ webapp
 - resources
 - ▼ WEB-INF
 - classes
 - ▼ spring
 - ▼ appServlet
 - servlet-context.xml
 - root-context.xml
 - ▼ views
 - home.jsp
 - web.xml
 - > test
 - > target
 - pom.xml

WEB.XML

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app version="2.5" xmlns="http://java.sun.com/xml/ns/javaee"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
    http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd">

<!-- The definition of the Root Spring Container shared by all Servlets and Filters
-->

<context-param>
<param-name>contextConfigLocation</param-name>
<param-value>/WEB-INF/spring/root-context.xml</param-value>
</context-param>

<!-- Creates the Spring Container shared by all Servlets and Filters -->
<listener>
<listener-class>org.springframework.web.context.ContextLoaderListener</listener-
class>
</listener>

...
```

WEB.XML

```
<!-- Processes application requests -->
<servlet>
<servlet-name>appServlet</servlet-name>
<servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
<init-param>
<param-name>contextConfigLocation</param-name>
<param-value>/WEB-INF/spring/appServlet/servlet-context.xml</param-value>
</init-param>
<load-on-startup>1</load-on-startup>
</servlet>

<servlet-mapping>
<servlet-name>appServlet</servlet-name>
<url-pattern>/</url-pattern>
</servlet-mapping>

</web-app>
```

ROOT-CONTEXT.XML

- /WEB-INF/spring/**root-context.xml** :
- Ce fichier est lu par **ContextLoaderListener**, au démarrage du serveur
- C'est un fichier dans lequel contexte de l'application sera construit
- ContextLoaderListener représente Spring **IOC**
- C'est donc un fichier pour l'injection des dépendances
- Pour le moment, il est vide

ROOT-CONTEXT.XML

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans.xsd">

<!-- Root Context: defines shared resources visible to all other
web components -->

</beans>
```

SERVLET-CONTEXT.XML

- /WEB-INF/spring/appServlet/**servlet-context.xml** :
- Ce fichier est lu par DispatcherServlet qui représente le contrôleur web de l'application :

SERVLET-CONTEXT.XML

```
<?xml version="1.0" encoding="UTF-8"?>
<beans:beans xmlns="http://www.springframework.org/schema/mvc"
xmlns:beans="http://www.springframework.org/schema/beans" .....>

<!-- DispatcherServlet Context: defines this servlet's request-processing infrastructure -->
<!-- Enables the Spring MVC @Controller programming model -->
<annotation-driven />

<!-- Handles HTTP GET requests for /resources/** by efficiently serving up static resources in the
    ${webappRoot}/resources directory -->
<resources mapping="/resources/**" location="/resources/" />

<!-- Resolves views selected for rendering by @Controllers to .jsp resources in the /WEB-INF/views
    directory -->
<beans:bean class="org.springframework.web.servlet.view.InternalResourceViewResolver">
<beans:property name="prefix" value="/WEB-INF/views/" />
<beans:property name="suffix" value=".jsp" />
</beans:bean>
<context:component-scan base-package="com.m2i.formation" />

</beans:beans>
```

POM.XML

```
<properties>  
<java-version>1.6</java-version>  
<org.springframework-version>3.1.1.RELEASE</org.springframework-  
  version>  
<org.aspectj-version>1.6.10</org.aspectj-version>  
<org.slf4j-version>1.6.6</org.slf4j-version>  
</properties>
```

POM.XML

```
<!-- Spring -->
<dependency>
  <groupId>org.springframework</groupId>
  <artifactId>spring-context</artifactId>
  <version>${org.springframework-version}</version>
  <exclusions>
    <!-- Exclude Commons Logging in favor of SLF4j -->
    <exclusion>
      <groupId>commons-logging</groupId>
      <artifactId>commons-logging</artifactId>
    </exclusion>
  </exclusions>
</dependency>
<dependency>
  <groupId>org.springframework</groupId>
  <artifactId>spring-webmvc</artifactId>
  <version>${org.springframework-version}</version>
</dependency>
```

POM.XML

```
<!-- AspectJ -->
<dependency>
  <groupId>org.aspectj</groupId>
  <artifactId>aspectjrt</artifactId>
  <version>${org.aspectj-version}</version>
</dependency>
<!-- Logging -->
<dependency>
  <groupId>org.slf4j</groupId>
  <artifactId>slf4j-api</artifactId>
  <version>${org.slf4j-version}</version>
</dependency>
<dependency>
  <groupId>org.slf4j</groupId>
  <artifactId>jcl-over-slf4j</artifactId>
  <version>${org.slf4j-version}</version>
  <scope>runtime</scope>
</dependency>
<dependency>
  <groupId>org.slf4j</groupId>
  <artifactId>slf4j-log4j12</artifactId>
  <version>${org.slf4j-version}</version>
  <scope>runtime</scope>
</dependency>
```

POM.XML

```
<dependency>
  <groupId>log4j</groupId>
  <artifactId>log4j</artifactId>
  <version>1.2.15</version>
  <exclusions>
    <exclusion>
      <groupId>javax.mail</groupId>
      <artifactId>mail</artifactId>
    </exclusion>
  </exclusions>
</dependency>
<exclusion>
<groupId>javax.jms</groupId>
<artifactId>jms</artifactId>
</exclusion>
<exclusion>
<groupId>com.sun.jdmk</groupId>
<artifactId>jmxtools</artifactId>
</exclusion>
<exclusion>
<groupId>com.sun.jmx</groupId>
<artifactId>jmxri</artifactId>
</exclusion>
</exclusions>
<scope>runtime</scope>
</dependency>
```

POM.XML

```
<!-- SPRING JDBC : https://mvnrepository.com/artifact/org.springframework/spring-jdbc -->
<dependency>
<groupId>org.springframework</groupId>
<artifactId>spring-jdbc</artifactId>
<version>4.3.7.RELEASE</version>
</dependency>

<!-- https://mvnrepository.com/artifact/mysql/mysql-connector-java -->
<dependency>
<groupId>mysql</groupId>
<artifactId>mysql-connector-java</artifactId>
<version>5.1.41</version>
</dependency>

<!-- https://mvnrepository.com/artifact/commons-dbcp/commons-dbcp -->
<dependency>
<groupId>commons-dbcp</groupId>
<artifactId>commons-dbcp</artifactId>
<version>1.4</version>
</dependency>
```

UN EXEMPLE DE CONTRÔLEUR SPRING MVC

```
package com.m2i.formation;

import java.text.DateFormat; import java.util.Date; import java.util.Locale; import org.slf4j.Logger; import
    org.slf4j.LoggerFactory; import org.springframework.stereotype.Controller; import
    org.springframework.ui.Model; import org.springframework.web.bind.annotation.RequestMapping; import
    org.springframework.web.bind.annotation.RequestMethod;

/** * Handles requests for the application home page. */
@Controller
public class HomeController {

    private static final Logger logger = LoggerFactory.getLogger(HomeController.class);

    /** * Simply selects the home view to render by returning its name. */
    @RequestMapping(value = "/", method = RequestMethod.GET)
    public String home(Locale locale, Model model) {
        logger.info("Welcome home! The client locale is {}.", locale);
        Date date = new Date();
        DateFormat dateFormat = DateFormat.getDateInstance(DateFormat.LONG, DateFormat.LONG, locale);
        String formattedDate = dateFormat.format(date);
        model.addAttribute("serverTime", formattedDate );

        return "home";
    }

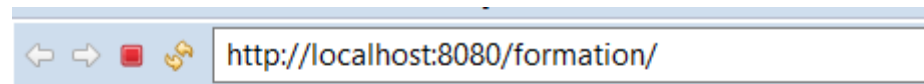
}
```

UN EXEMPLE DE VUE : JSP

```
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
<%@ page session="false" %>
<html>
<head>
<title>Home</title>
</head>
<body>
<h1>
Hello world!
</h1>

<P> The time on the server is ${serverTime}. </P>
</body>
</html>
```


UN EXEMPLE DE VUE : JSP



Hello world!

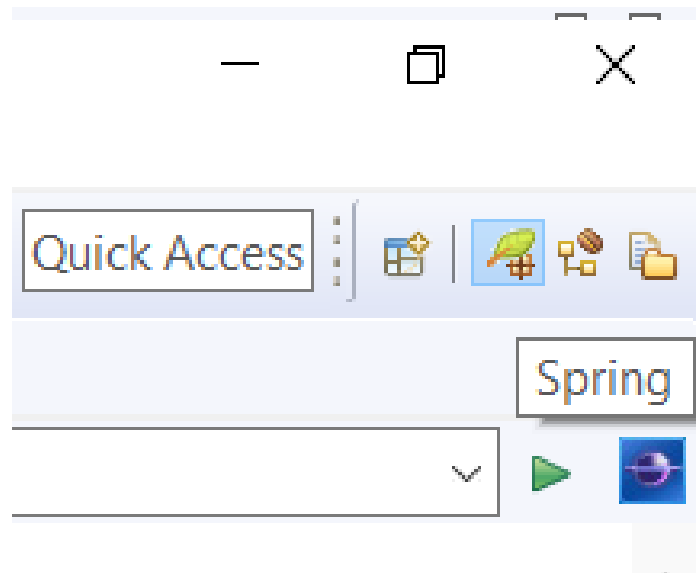
The time on the server is 17 avril 2017 14:35:03 CEST.

TP 13 : APPLICATION SPRING MVC PAS À PAS

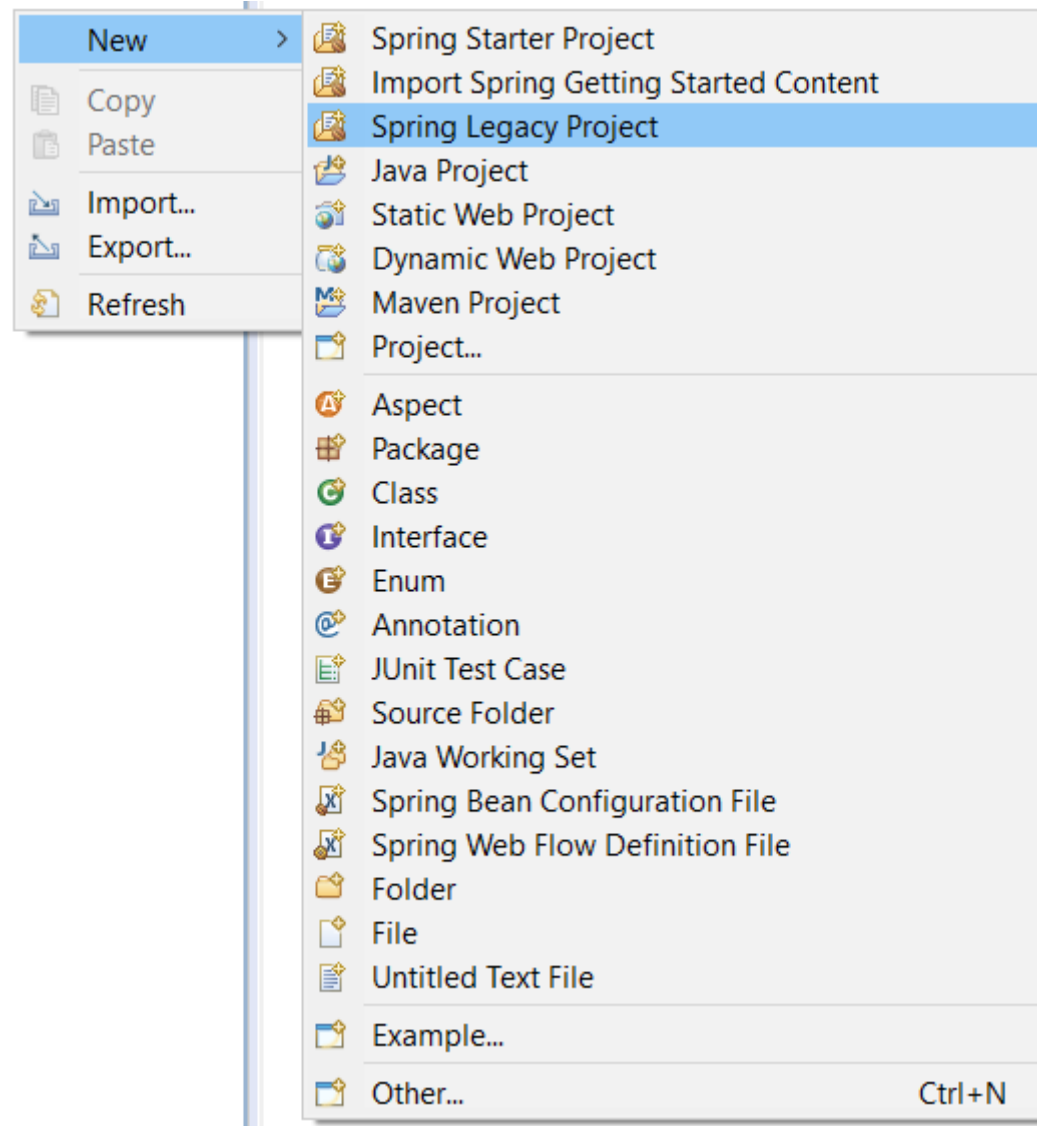
- On souhaite créer une application qui permet de gérer des **Produits**. Chaque produit est défini par sa référence (de type String), sa désignation, son prix et sa quantité.
- L'application doit permettre les opérations suivantes :
 - Ajouter un nouveau produit
 - Consulter tous les produits
 - Consulter les produits dont le nom contient un mot clé.
 - Consulter un produit
 - Supprimer un produit
 - Mettre à jour un produit.
- Cette application se compose de trois couches **DAO**, **Service** et **Présentation**.
- L'injection des dépendances sera effectuée en utilisant Spring **IOC**.
- La couche Web sera implémentée à travers **Spring MVC**.
- La couche DAO utilisera une Base de données de type **MySQL**.

TP 13 :

- Mettez vous dans la vue Spring :



TP 13 :



TP 13 :

The screenshot shows the 'New Spring Legacy Project' wizard. The title bar says 'New Spring Legacy Project'. The main title is 'Spring Legacy Project'. A tip says 'Click 'Next' to load the template contents.' The 'Project name' field contains 'tp13-spring-mvc'. The 'Use default location' checkbox is checked. The 'Location' field shows 'C:\Work\workspace-sts\tp13-spring-mvc' with a 'Browse...' button. The 'Select Spring version' dropdown is set to 'Default'. Under 'Templates', 'Simple Spring Utility Project' and 'Spring MVC Project' are listed, with 'Spring MVC Project' selected. A note indicates 'requires downloading' with a link to 'Configure templates...' and a 'Refresh' button. The 'Description' is 'A new Spring MVC web application development project'. The 'URL' is 'http://dist.springsource.com/release/STS/help/org.springframework.templates.mvc-3.2.2.zip'. In the 'Working sets' section, 'Add project to working sets' is unchecked, and there are 'New...' and 'Select...' buttons. At the bottom, there are navigation buttons: '?', '< Back', 'Next >', 'Finish', and 'Cancel'.

New Spring Legacy Project

Spring Legacy Project

Click 'Next' to load the template contents.

Project name: tp13-spring-mvc

☒ Use default location

Location: C:\Work\workspace-sts\tp13-spring-mvc [Browse...](#)

Select Spring version: Default

Templates:

- Simple Spring Utility Project
- Spring MVC Project**

requires downloading [Configure templates...](#) Refresh

Description:
A new Spring MVC web application development project

URL: <http://dist.springsource.com/release/STS/help/org.springframework.templates.mvc-3.2.2.zip>

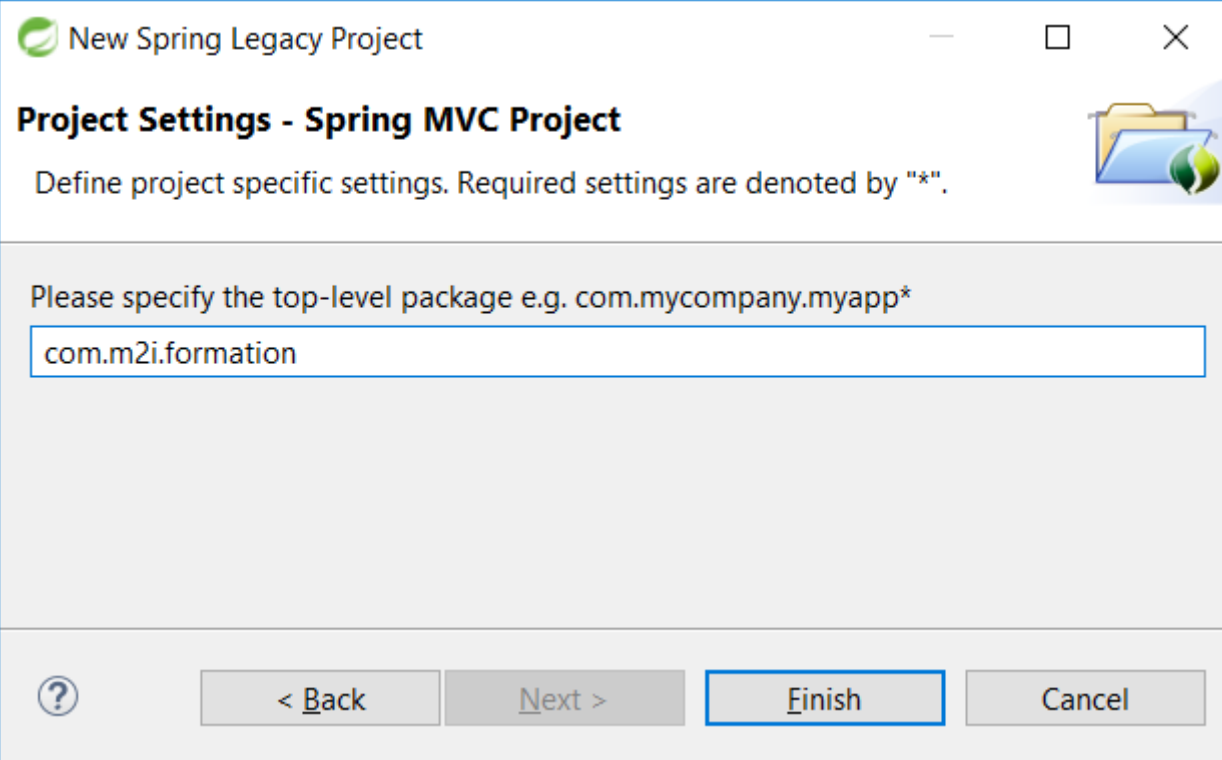
Working sets

☐ Add project to working sets [New...](#)

Working sets: [Select...](#)

? < Back **Next >** Finish Cancel

TP 13 :



New Spring Legacy Project

Project Settings - Spring MVC Project

Define project specific settings. Required settings are denoted by "*".

Please specify the top-level package e.g. com.mycompany.myapp*

com.m2i.formation

? < Back Next > **Finish** Cancel

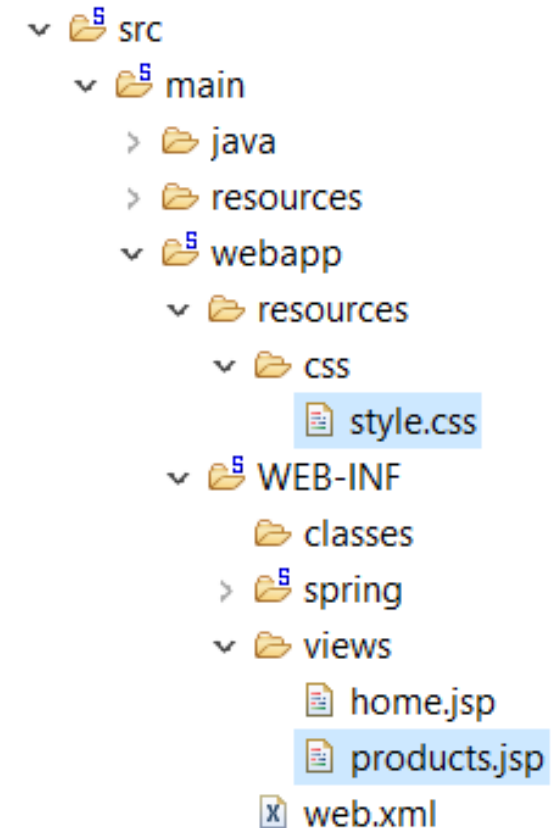
TP 13 :

- Créer :

Une page Web JSP **products.jsp**

Un fichier CSS **style.css**

Le contenu ressemble à la capture d'écran de la page suivante :



TP 13 : Couche Vue Web

Catalogue de produits

localhost:8080/formation/ Rechercher

Référence:

ABC

Désignation:

Livre Spring

Prix:

25

Quantité:

10

Save

REF	DESIGNATION	PRIX	QUANTITE		
ABC1	Livre1 Spring par la pratique	25	10	Supprimer	Modifier
ABC2	Livre2 Spring par la pratique	20	20	Supprimer	Modifier
ABC3	Livre3 Spring par la pratique	30	05	Supprimer	Modifier

TP 13 : VIEW : products.jsp (statique)

C'est une JSP statique, juste l'affichage nous intéresse à cette étape :

```
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c"%>
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>Catalogue de produits</title>
<link rel="stylesheet" type="text/css"
      href="<%=request.getContextPath()%>/resources/css/style.css">
</head>
<body>
<div id="formProducts">
<form method="post" action="saveProduct">
<table class="table1">
<tr><th>Référence:</th><td></td><td><input value = "ABC"/></td></tr>
<tr><th>Désignation:</th><td></td><td><input value = "Livre Spring"/></td></tr>
<tr><th>Prix:</th><td></td><td><input value = "25"/></td></tr>
<tr><th>Quantité:</th><td></td><td><input value = "10"/></td></tr>
<tr><td><input type="submit" value="Save"></td></tr>
```

TP 13 : VIEW : products.jsp (statique)

```
...
</table>
</form>
</div>
<div id="ListProducts">
<table class="table1">
<tr><th>REF</th><th>DESIGNATION</th><th>PRIX</th><th>QUANTITE</th></tr>
<!-- c:forEach items=""-->
<tr><td>ABC1</td><td>Livre1 Spring par la pratique</td><td>25</td><td>10</td><td><a
    href="deleteProduct?ref=">Supprimer</a></td><td><a
    href="editProduct?ref=">Modifier</a></td></tr>
<tr><td>ABC2</td><td>Livre2 Spring par la pratique</td><td>20</td><td>20</td><td><a
    href="deleteProduct?ref=">Supprimer</a></td><td><a
    href="editProduct?ref=">Modifier</a></td></tr>
<tr><td>ABC3</td><td>Livre3 Spring par la pratique</td><td>30</td><td>05</td><td><a
    href="deleteProduct?ref=">Supprimer</a></td><td><a
    href="editProduct?ref=">Modifier</a></td></tr>
<!-- /c:forEach-->
</table>
</div>
</body>
</html>
```

TP 13 : Script SQL

- Script SQL de création de la table product (entity : product.java).
- Table : product
- Champs : ID, REFERENCE, DESIGNATION, PRICE, AMOUNT, CREATED, UPDATED

TP 13 : Script SQL

-- product.sql

-- Base de données: `formation`

-- Structure de la table `product`

```
CREATE TABLE IF NOT EXISTS `product` (  
  `ID` integer UNSIGNED NOT NULL auto_increment,  
  `REFERENCE` VARCHAR(25) UNIQUE NOT NULL ,  
  `DESIGNATION` VARCHAR(10) NOT NULL ,  
  `PRICE` DECIMAL(6,2) NOT NULL ,  
  `AMOUNT` VARCHAR(30) NOT NULL ,  
  `CREATED` DATETIME,  
  `UPDATED` DATETIME,  
  PRIMARY KEY (`ID`)  
) AUTO_INCREMENT=10 ;
```

-- Contenu de la table `product`

```
INSERT INTO `product` (REFERENCE, DESIGNATION, PRICE, AMOUNT, CREATED, UPDATED)  
  VALUES ('ABC', 'Livre Spring par la pratique', 25, 50, sysdate(), sysdate());  
INSERT INTO `product` (REFERENCE, DESIGNATION, PRICE, AMOUNT, CREATED, UPDATED)  
  VALUES ('ABD', 'Livre Comprendre Hibernate', 12, 100, sysdate(), sysdate());  
INSERT INTO `product` (REFERENCE, DESIGNATION, PRICE, AMOUNT, CREATED, UPDATED)  
  VALUES ('ABE', 'Livre Maitriser MySQL', 11, 200, sysdate(), sysdate());
```

TP 13 : Script SQL

```
Invite de commandes - mysql -u root -p
C:\Work\workspace-sts\tp13-spring-mvc\src\main\webapp\resources\sql>mysql -u root -p
Enter password:
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 292
Server version: 5.6.17 MySQL Community Server (GPL)

Copyright (c) 2000, 2014, Oracle and/or its affiliates. All rights reserved.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> use formation;
Database changed
mysql> source product.sql
Query OK, 0 rows affected (0.02 sec)








Query OK, 1 row affected, 1 warning (0.00 sec)

Query OK, 1 row affected, 1 warning (0.00 sec)

Query OK, 1 row affected, 1 warning (0.01 sec)

mysql>
```

TP 13 : Packages

- ▼  tp13-spring-mvc
 - ▼  src/main/java
 - >  com.m2i.formation
 - >  com.m2i.formation.dao
 - >  com.m2i.formation.dao.entity
 - >  com.m2i.formation.presentation
 - >  com.m2i.formation.service

TP 13 : DAO – entity

```
package com.m2i.formation.dao.entity;
import java.util.Date;
public class Product {
private int id; private String reference; private String designation; private double
    price;
private int amount; /* quantité*/ private Date created; private Date updated;
public String getReference() {return reference;}
public void setReference(String reference) {this.reference = reference;}
public int getId() {return id;}
public void setId(int id) {this.id = id;}
public Date getCreated() {return created;}
public void setCreated(Date created) {this.created = created;}
public Date getUpdated() {return updated;}
public void setUpdated(Date updated) {this.updated = updated;}
public String getDesignation() {return designation;}
public void setDesignation(String designation) {this.designation = designation;}
public double getPrice() {return price;}
public void setPrice(double price) {this.price = price;}
public int getAmount() {return amount;}
public void setAmount(int amount) {this.amount = amount;}
}
```

TP 13 : DAO Interface

```
package com.m2i.formation.dao;

import java.util.List;

import com.m2i.formation.dao.entity.Product;

public interface ICatalogDAO {

    public void addProduct(Product p);
    public boolean removeProduct(Product p);
    public void updateProduct(Product p);
    public Product findProductByRef(String ref);
    public List<Product> findProductByCriteria(String criteria);
    public List<Product> findAllProducts();

}
```


TP 13 : DAO Implémentation – Spring JDBC

```
package com.m2i.formation.dao;

import java.util.List; import org.springframework.jdbc.core.JdbcTemplate; import
    org.springframework.stereotype.Repository; import com.m2i.formation.dao.entity.Product;

@Repository("catalogDAO")
public class CatalogDAOImpl implements ICatalogDAO {

    JdbcTemplate jdbcTemplate;

    public CatalogDAOImpl(JdbcTemplate jdbcTemplate) { super(); this.jdbcTemplate = jdbcTemplate; }
    public JdbcTemplate getJdbcTemplate() { return jdbcTemplate; }
    public void setJdbcTemplate(JdbcTemplate jdbcTemplate) { this.jdbcTemplate = jdbcTemplate; }
    public void addProduct(Product p) { final String INSERT_QUERY = "INSERT INTO product (REFERENCE,
        DESIGNATION, PRICE, AMOUNT, CREATED, UPDATED) VALUES (?, ?, ?, ?, ?, ?)";
        jdbcTemplate.update(INSERT_QUERY, new Object[] {p.getReference(), p.getDesignation(), p.getPrice(),
            p.getAmount(), p.getCreated(), p.getUpdated()}); }
    public void updateProduct(Product p) { /* TODO Auto-generated method stub */ }
    public boolean removeProduct(Product p) { /* TODO Auto-generated method stub */ return false; }
    public Product findProductByRef(String ref) { /* TODO Auto-generated method stub */ return null; }
    public List<Product> findProductByCriteria(String criteria) { /* TODO Auto-generated */ return null; }
    public List<Product> findAllProducts() { /* TODO Auto-generated method stub */ return null; }

}
```

TP 13 : DAO Implémentation : root-context.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans" xmlns:xsi = "http://www.w3.org/2001/XMLSchema-
instance" xmlns:context = "http://www.springframework.org/schema/context" xsi:schemaLocation =
"http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-
beans.xsd http://www.springframework.org/schema/context
http://www.springframework.org/schema/context/spring-context-4.3.xsd">
  <!-- Root Context: defines shared resources visible to all other web components -->
  <!-- Déclaration du PropertyPlaceholderConfigurer -->
  <bean class="org.springframework.beans.factory.config.PropertyPlaceholderConfigurer">
    <property name="locations" ><list><value>classpath:db.properties</value></list></property>
  </bean>
  <context:component-scan base-package="com.m2i.formation" />
  <bean id="jdbcTemplate" class="org.springframework.jdbc.core.JdbcTemplate">
    <property name="dataSource" ref="datasource" />
  </bean>
  <!-- Déclaration de la DATASOURCES -->
  <bean id="datasource" destroy-method="close" class="org.apache.commons.dbcp.BasicDataSource">
    <property name="driverClassName" value="${db.driver}" /><property name="url" value="${db.url}"
    /><property name="username" value="${db.login}" /> <property name="password" value="${db.password}"
    />
  </bean>
</beans>
```

TP 13 : Service :

```
package com.m2i.formation.service;

import org.springframework.stereotype.Service; import com.m2i.formation.dao.ICatalogDAO;
import com.m2i.formation.dao.entity.Product;

@Service("catalogService")
public class CatalogService {
    ICatalogDAO catalogDAO;

    public CatalogService(ICatalogDAO catalogDAO) { super(); this.catalogDAO = catalogDAO; }

    public ICatalogDAO getCatalogDAO() {return catalogDAO;}

    public void setCatalogDAO(ICatalogDAO catalogDAO) {this.catalogDAO = catalogDAO;}

    public Boolean addEmployee(Product p) {
        Boolean productAdded = true;
        try {catalogDAO.addProduct(p);} catch (Exception e) {productAdded = false; }
        return productAdded;
    }
}
```

TP 13 : VIEW : products.jsp

```
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c"%><%@ taglib
    uri="http://www.springframework.org/tags/form" prefix="f"%><!DOCTYPE html>
<html>
<head><meta charset="UTF-8" ><title>Catalogue de produits</title><link rel="stylesheet"
    type="text/css" href="<%=request.getContextPath()%>/resources/css/style.css"></head>
<body>
<div id="formProducts">
<f:form modelAttribute="product" method="post" action="saveProduct">
<table>
<tr><th>Référence:</th><td><f:input path="reference" /></td></tr>
<tr><th>Désignation:</th><td><f:input path="designation" /></td></tr>
<tr><th>Prix:</th><td><f:input path="price" /></td></tr>
<tr><th>Quantité:</th><td><f:input path="amount" /></td></tr>
<f:hidden path="id"/>
<tr><td><input type="submit" value="Save"></td></tr>
</table>
</f:form>
</div>
...
```

TP 13 : VIEW : products.jsp

```
...
<div id="listProducts">
<table class="table1">
<tr><th>REF</th><th>DESIGNATION</th><th>PRIX</th><th>QUANTITE</th></tr>
<c:forEach items="${products}" var="p">
<tr>
<td>${p.reference}</td><td>${p.designation}</td><td>${p.price}</td><td>${p.amount}</td>
<td><a href="deleteProduct?ref=${p.reference}">Supprimer</a></td><td><a
    href="editProduct?ref=${p.reference}">Modifier</a></td>
</tr>
</c:forEach>
</table>
</div>
</body>
</html>
```

TP 13 : Contrôleur

@Controller

```
public class CatalogController {  
    @Resource(name="catalogService")  
    private ICatalogService catalogService;  
    @RequestMapping(value = "/", method = RequestMethod.GET)  
    public String products(Locale locale, Model model) {  
        model.addAttribute("product", new Product());  
        model.addAttribute("products", catalogService.findAllProducts());  
        return "products";  
    }  
    @RequestMapping(value="/saveProduct")  
    public String save(Product p, Model model){  
        if(p.getId() == 0)  
            catalogService.addProduct(p);  
        else catalogService.updateProduct(p);  
        model.addAttribute("product", new Product());  
        model.addAttribute("products", catalogService.findAllProducts());  
        return "products";  
    }  
}
```

.....

TP 13 : Contrôleur

```
@RequestMapping(value="/deleteProduct")  
public String delete(String ref, Model model){  
.....  
}
```

```
@RequestMapping(value="/editProduct")  
public String edit(String ref, Model model){  
.....  
}
```

TP 13 : Fin

Catalogue de produits

http://localhost:8080/formation/editProduct?ref=ABE

Référence:

ABE

Désignation:

Livre3 Maitriser Maven

Prix:

11.0

Quantité:

200

Save

TP 14 : APPLICATION SPRING MVC

- On souhaite créer une application qui permet de gérer des **Employées** d'une entreprise. Chaque employé est défini par son id, son nom, son prénom, le nombre d'années d'expérience dans l'entreprise, et son poste.
- L'application doit permettre les opérations suivantes :
 - Ajouter un nouvel employé
 - Consulter tous les employés
 - Consulter les employés par poste.
 - Consulter un employé
 - Supprimer un employé
 - Mettre à jour un employé.
- Ce sera projet **Maven** de type Projet **Spring MVC**.
- Cette application se compose de trois couches **dao (entity)**, **Service** et **Présentation**.
- L'injection des dépendances sera effectuée en utilisant **Spring IOC**.
- La couche Web sera implémentée à travers **Spring MVC**.
- La couche DAO utilisera une Base de données de type **MySQL**, et implémentera **SPRING JDBC**.
- **LOG4J** sera utilisé dans la plupart des méthodes (debug, info, error).
- **JUNIT** sera implémenté.

RÉVISION

- Quelques questions sur les notions abordées autour de Spring MVC.



SI DES QUESTIONS :

