Formation JAVA – Java Spring MVC

Pour	Formation
Date	10/10/2024
Objet	Java Spring MVC

I) Vocabulaire

- **API** : Signifie Application Programming Interface. Ce qui veut dire que c'est un ensemble de bibliothèques et librairies dédié pour implémenter une fonctionnalité donnée.

- **ORM** : Object-Relational Mapping (**MOR** : Mapping Objet-Relationnel en français) est une technique de programmation informatique qui crée l'illusion d'une base de données orientée objet à partir d'une base de données relationnelle en définissant des correspondances entre cette base de données et les objets du langage utilisé.

- **JPA** : Java Persistence API (abrégée en JPA), est une interface de programmation Java permettant aux développeurs d'organiser des données relationnelles dans des applications utilisant la plateforme Java.

- **JPQL** : Le langage JPQL (**Java Persistence Query Language**) est un langage de requête orienté objet, similaire à SQL, mais au lieu d'opérer sur les tables et colonnes, JPQL travaille avec des objets persistants et de leurs propriétés. Il est très proche du langage SQL dont il s'inspire fortement mais offre une approche objet. La grammaire de ce langage est définie par la spécification J.P.A.

- **HQL** : **Hibernate Query Language** est aussi un langage de requête orienté objet au même titre que JPQL. La principale différence avec le langage JQL est que le en HQL le « **Select** » sur l'objet n'est pas nécessaire. En fin de compte pour le JPQL on aura : **Select person from Personne person** alors que pour le HQL on aura : **from Personne**.

- **Bean** : le « **Bean** » (ou haricot en français) est une technologie de composants logiciels écrits en langage Java. Les **Beans** sont utilisés pour encapsuler plusieurs objets dans un seul objet. Le « **Bean** » regroupe alors tous les attributs des objets encapsulés. Ainsi, il représente une entité plus globale que les objets encapsulés de manière à répondre à un besoin métier.

- **Pattern IoC** : L'inversion de contrôle (inversion of control, IoC) est un patron d'architecture commun à tous les Frameworks (ou cadre de développement et d'exécution). Il fonctionne selon le principe que le flot d'exécution d'un logiciel n'est plus sous le contrôle direct de l'application elle-même mais du Framework ou de la couche logicielle sous-jacente. En effet selon un problème, il existe différentes formes, ou représentation d'IoC, le plus connu étant l'injection de dépendances (dependency injection) qui est un patron de conception permettant, en programmation orientée objet, de découpler les dépendances entre objets.

- **Pattern AOP** : L'AOP (Aspect Oriented Programming) ou POA (Programmation Orientée Aspect) est un paradigme de programmation ayant pour but de compléter la programmation orientée objet et permettre d'implémenter de façon plus propre les problématiques transverses à l'application. En effet, elle permet de factoriser du code dans des greffons et de les injecter en divers endroits sans pour autant modifier le code source des endroits en question.

- **GemFire** : GemFire est une infrastructure de gestion de données distribuée hautes performances qui se situe entre le cluster d'applications et les sources de données back-end. Avec GemFire, les données peuvent être gérées en mémoire, ce qui accélère l'accès.

II) Architecture Java Web et Java EE.

Java EE est la version "entreprise" de Java, elle a pour but de faciliter le développement d'applications distribuées.
Mais en fait, Java EE est avant tout une norme.

C'est un ensemble de standard décrivant des services techniques comme, par exemple, comment accéder à un annuaire, à une base de données, à des documents...

Important : Java EE définit ce qui doit être fournit mais ne dit pas comment cela doit être fournit.

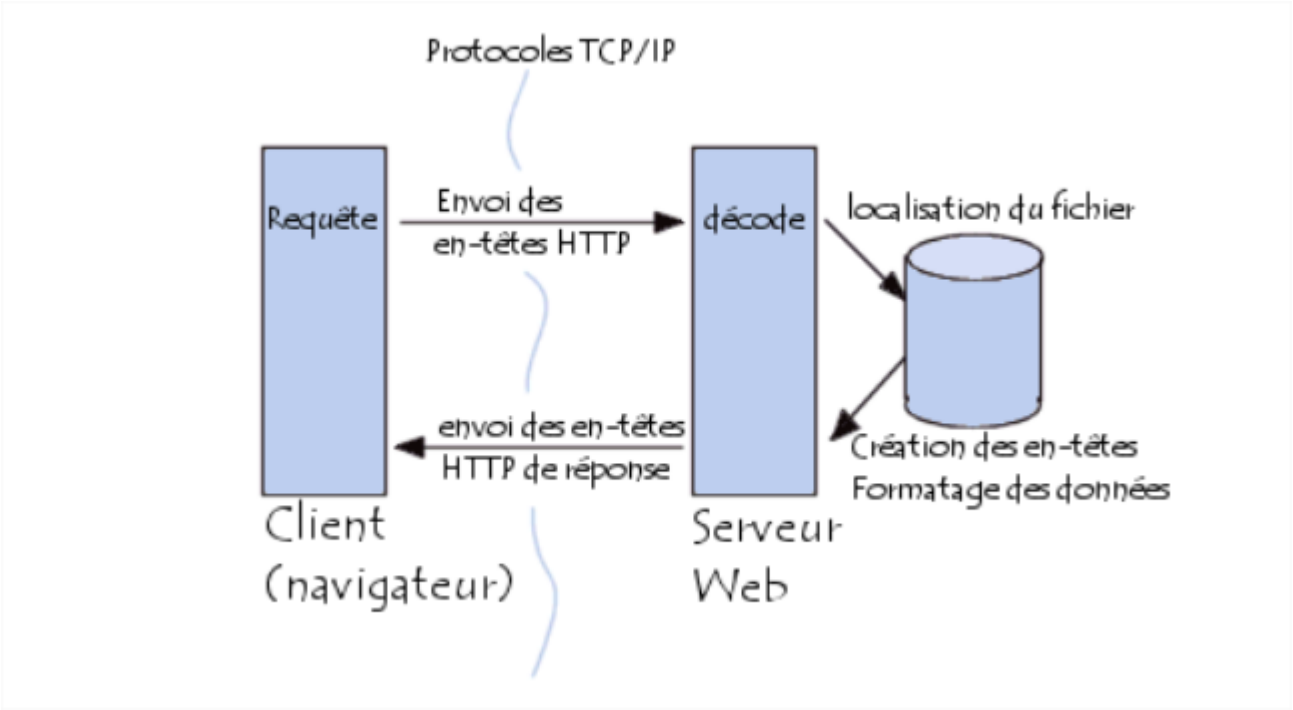
Exemple de services :

- JNDI (Java Naming and Directory Interface) est une API d'accès aux services de nommage et aux annuaires d'entreprises tels que DNS, NIS, LDAP…
- JTA (Java Transaction API) est une API définissant des interfaces standard avec un gestionnaire de transactions.

III) Rappels sur le protocole HTTP.

Le protocole HTTP (HyperText Transfer Protocol) est le protocole le plus utilisé sur Internet depuis 1990.
Le but ce protocole est de permettre un transfert de fichiers (essentiellement au format HTML) localisés grâce à une chaîne de caractères appelée URL entre un navigateur (le client) et un serveur Web.

La communication entre le navigateur et le serveur se fait en deux temps :



- Le navigateur effectue une **requête HTTP**
- Le serveur traite la requête puis envoie une **réponse HTTP**

Une requête HTTP est traitée à travers plusieurs méthodes :

GET : c'est la méthode la plus courante pour demander une ressource. Une requête GET est sans effet sur la ressource, il est possible de répéter la requête sans effet.

POST : cette méthode est utilisée pour transmettre des données en vue d'un traitement de ressource (le plus souvent depuis un formulaire HTML). L'URI fourni est l'URI d'une ressource à laquelle s'appliqueront les données envoyées. Le résultat peut être la création de nouvelles ressources ou la modification de ressources existantes.

HEAD : cette méthode ne demande que des informations sur la ressource, sans demander la ressource elle-même.

OPTIONS : cette méthode permet d'obtenir les options de communication d'une ressource ou du serveur en général.

CONNECT : cette méthode permet d'utiliser un proxy comme un tunnel de communication.

TRACE : cette méthode demande au serveur de retourner ce qu'il a reçu dans le but de tester et d'effectuer un diagnostic sur la connexion.

PUT: cette méthode permet de remplacer ou d'ajouter une ressource sur le serveur. L'URI fourni est celui de la ressource en question.

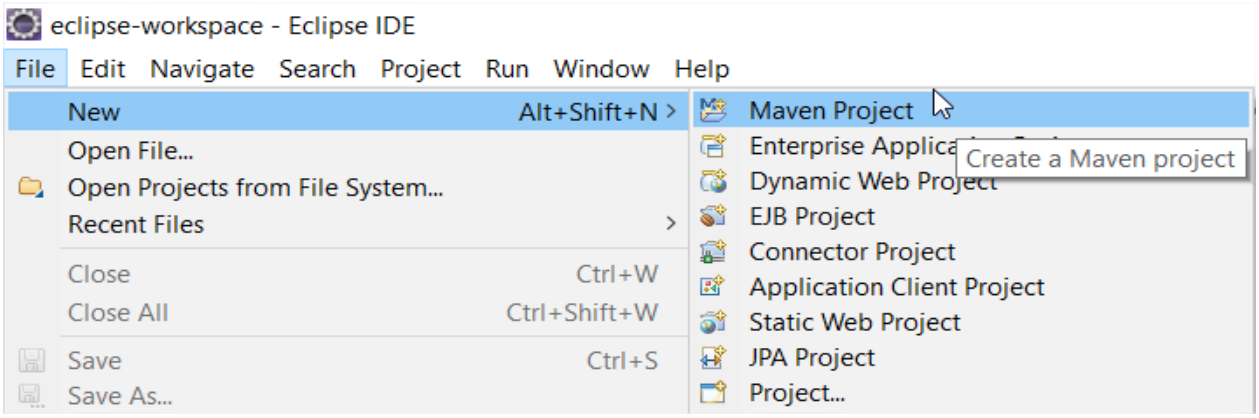
PATCH: cette méthode permet, contrairement à PUT, de faire une modification **partielle** d'une ressource.

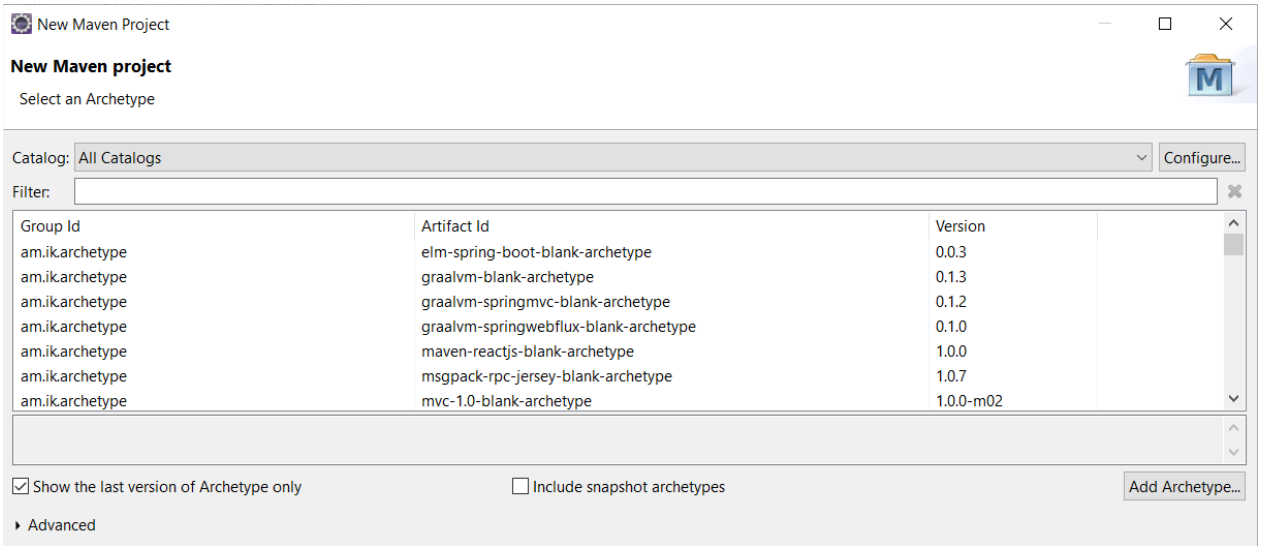
DELETE: cette méthode permet de supprimer une ressource du serveur.

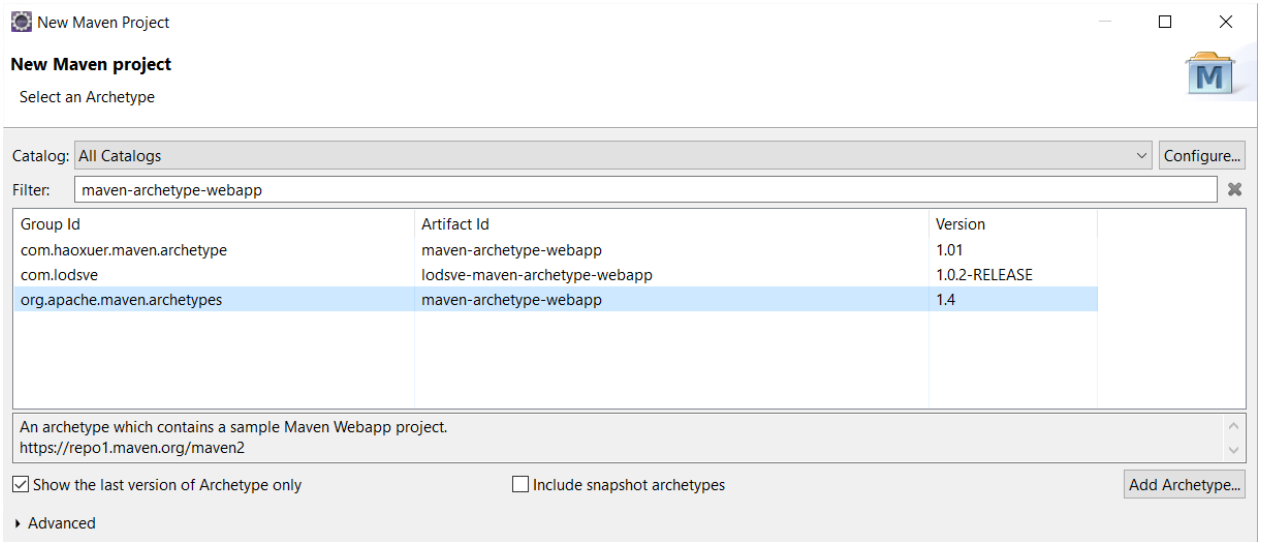
IV) Rappels sur les projets Maven Web HttpServlet

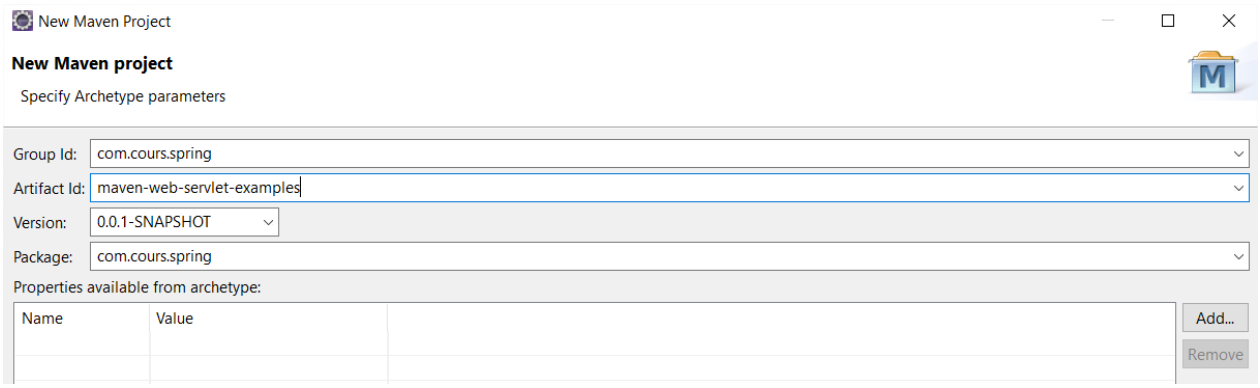
Créer le projet Maven **maven-web-servlet-examples** de type **war** (archetype=**maven-archetype-webapp**).

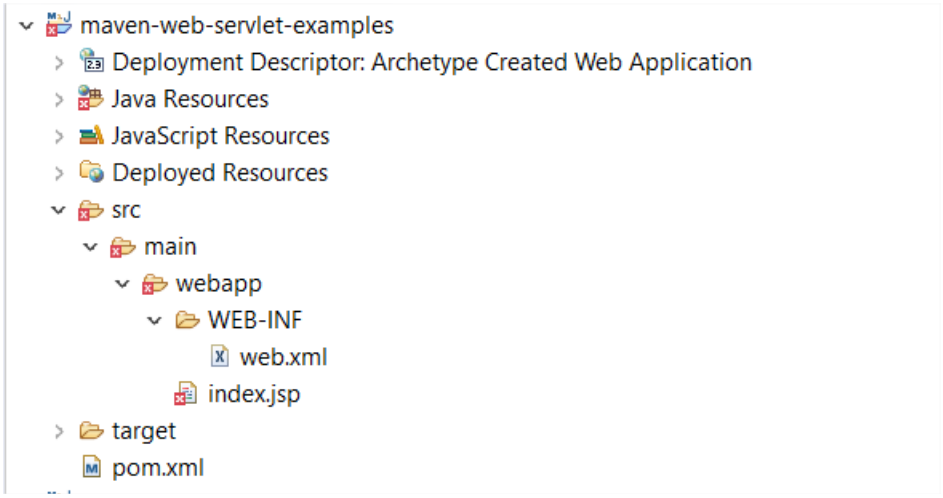
1. *Création du projet Maven*



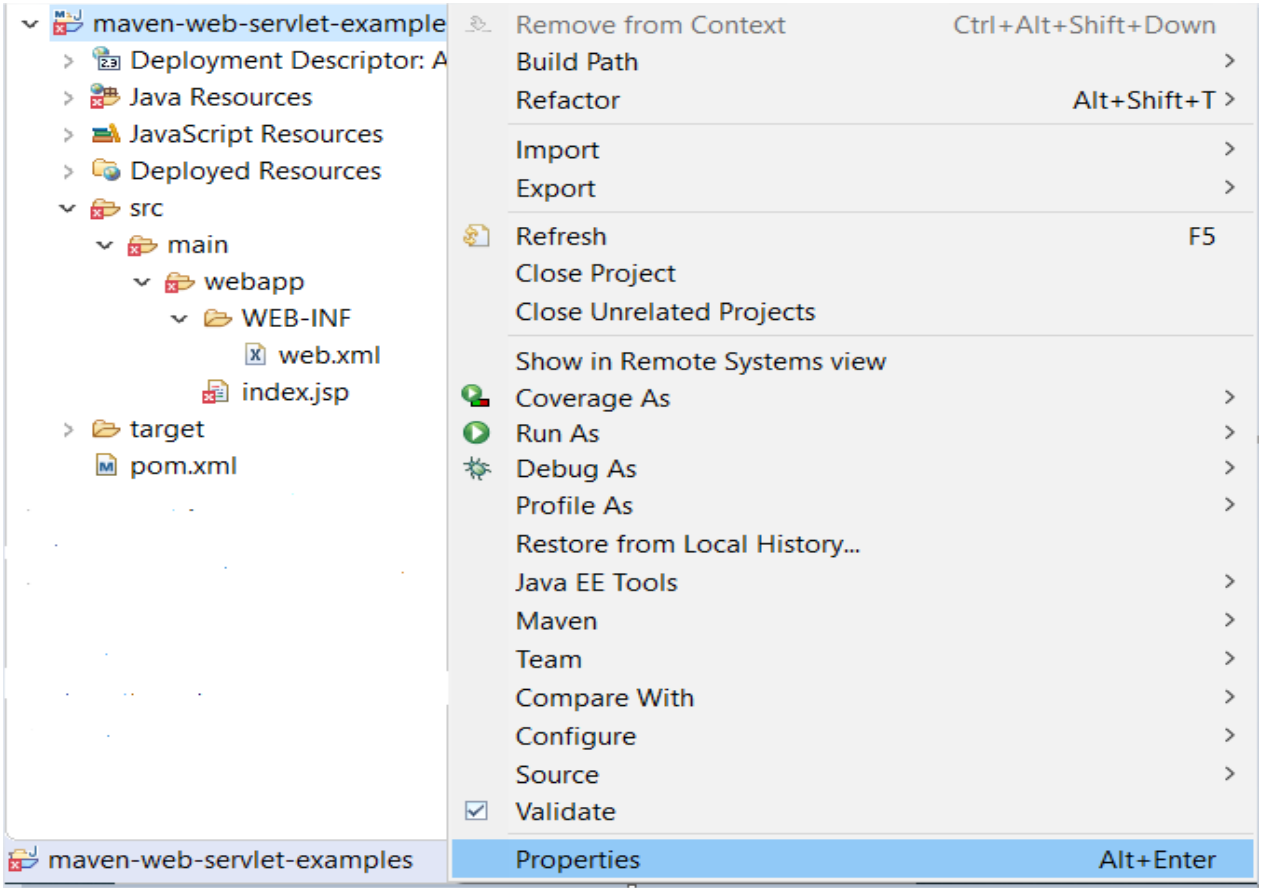


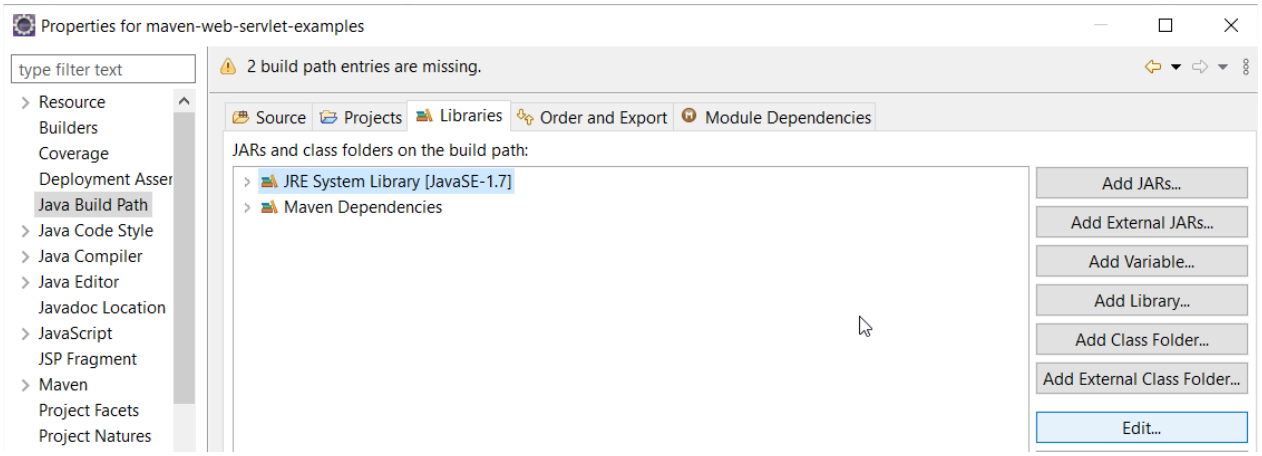

New Maven Project
New Maven project
Select an Archetype
Catalog: All Catalogs
Configure...
Filter: maven-archetype-webapp
Group Id
Artifact Id
Version
com.haoxuer.maven.archetype
maven-archetype-webapp
1.01
com.lodsve
lodsve-maven-archetype-webapp
1.0.2-RELEASE
org.apache.maven.archetypes
maven-archetype-webapp
1.4
An archetype which contains a sample Maven Webapp project.
https://repo1.maven.org/maven2
Show the last version of Archetype only
Include snapshot archetypes
Add Archetype...
Advanced


New Maven Project
New Maven project
Specify Archetype parameters
Group Id: com.cours.spring
Artifact Id: maven-web-servlet-examples
Version: 0.0.1-SNAPSHOT
Package: com.cours.spring
Properties available from archetype:
Name
Value
Add...
Remove

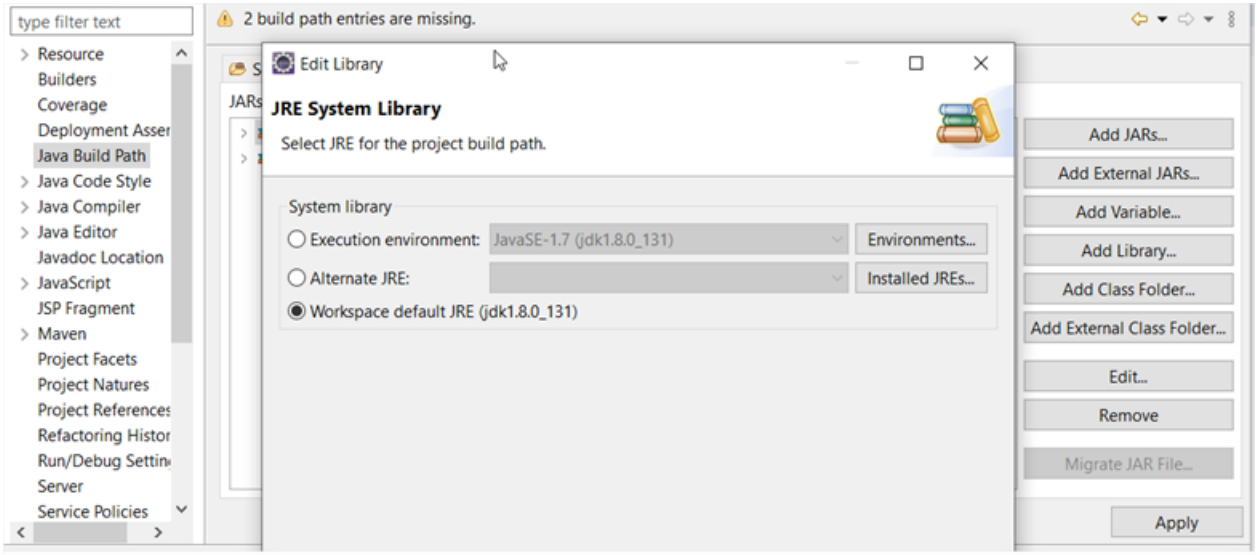

maven-web-servlet-examples
Deployment Descriptor: Archetype Created Web Application
Java Resources
JavaScript Resources
Deployed Resources
src
main
webapp
WEB-INF
web.xml
index.jsp
target
pom.xml

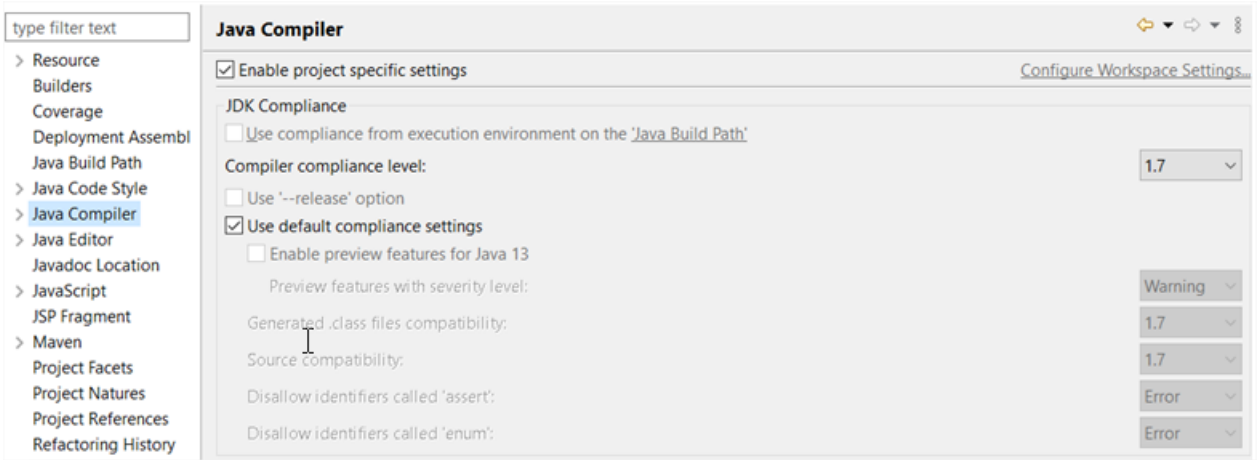
2. *Verification de la version de Java*

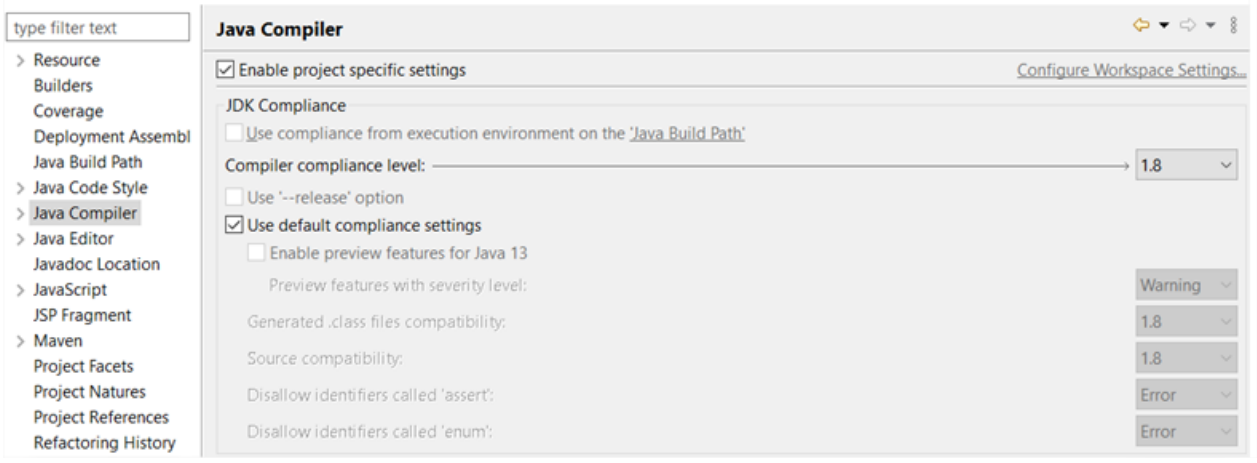




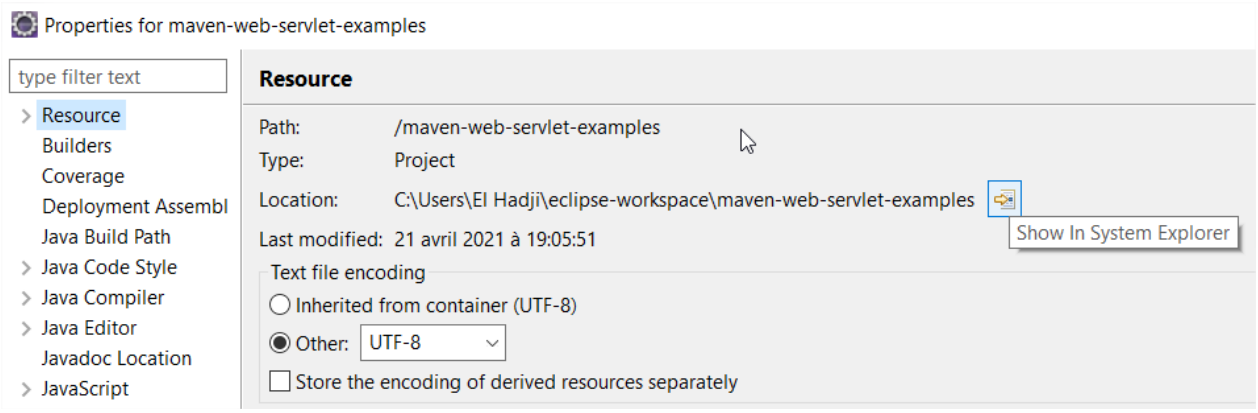
Changer la version du Build Path de **1.7 à 1.8**.

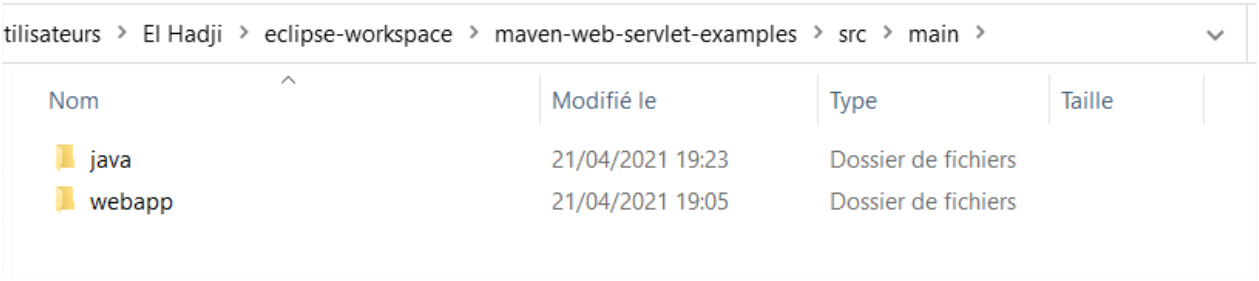




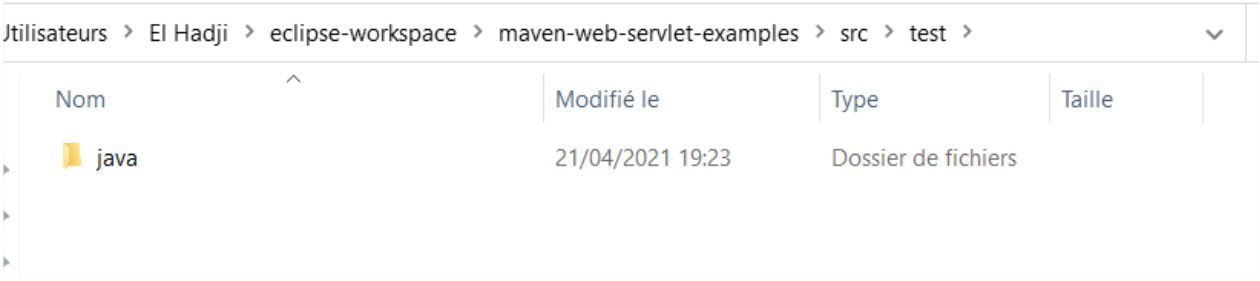


3. *Création des dossiers src/main/resources et src/test/resources*





Créer le dossier **src/main/resources** :



Créer le dossier **src/test/resources** :

Copier le fichier **log4j.properties** dans **/src/main/resources** dont le contenu sera :

```
# Set root logger level to DEBUG and its only appender to CONSOLE.
log4j.rootLogger=DEBUG, CONSOLE

# A1 is set to be a ConsoleAppender.
log4j.appender.CONSOLE=org.apache.log4j.ConsoleAppender
log4j.appender.CONSOLE.encoding=UTF-8

# A1 uses PatternLayout.
log4j.appender.CONSOLE.layout=org.apache.log4j.PatternLayout
log4j.appender.CONSOLE.layout.ConversionPattern=[%d{dd/MM/yyyy HH:mm:ss} %p %C{1}.%M] %m%n

# Change the level of messages for various packages.
log4j.logger.com.cours*=DEBUG
```

4. *Le pom.xml*

Le fichier **pom.xml** aura pour contenu initiale :

```
<?xml version="1.0" encoding="UTF-8"?>

<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance"
 xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
 <modelVersion>4.0.0</modelVersion>

 <groupId>com.cours.spring</groupId>
 <artifactId>maven-web-servlet-examples</artifactId>
 <version>0.0.1-SNAPSHOT</version>
 <packaging>war</packaging>

 <name>maven-web-servlet-examples Maven Webapp</name>
 <!-- FIXME change it to the project's website -->
 <url>http://www.example.com</url>

 <properties>
  <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
  <maven.compiler.source>1.7</maven.compiler.source>
  <maven.compiler.target>1.7</maven.compiler.target>
 </properties>

 <dependencies>
  <dependency>
   <groupId>junit</groupId>
   <artifactId>junit</artifactId>
   <version>4.11</version>
   <scope>test</scope>
  </dependency>
 </dependencies>

 <build>
  <finalName>maven-web-servlet-examples</finalName>
  <pluginManagement><!--
 lock down plugins versions to avoid using Maven defaults (may be moved to parent pom) -->
   <plugins>
    <plugin>
     <artifactId>maven-clean-plugin</artifactId>
     <version>3.1.0</version>
    </plugin>
    <!-- see http://maven.apache.org/ref/current/maven-core/default-
bindings.html#Plugin_bindings_for_war_packaging -->
    <plugin>
     <artifactId>maven-resources-plugin</artifactId>
     <version>3.0.2</version>
    </plugin>
```

```
      <plugin>
        <artifactId>maven-compiler-plugin</artifactId>
        <version>3.8.0</version>
      </plugin>
      <plugin>
        <artifactId>maven-surefire-plugin</artifactId>
        <version>2.22.1</version>
      </plugin>
      <plugin>
        <artifactId>maven-war-plugin</artifactId>
        <version>3.2.2</version>
      </plugin>
      <plugin>
        <artifactId>maven-install-plugin</artifactId>
        <version>2.5.2</version>
      </plugin>
      <plugin>
        <artifactId>maven-deploy-plugin</artifactId>
        <version>2.8.2</version>
      </plugin>
    </plugins>
  </pluginManagement>
 </build>
</project>
```

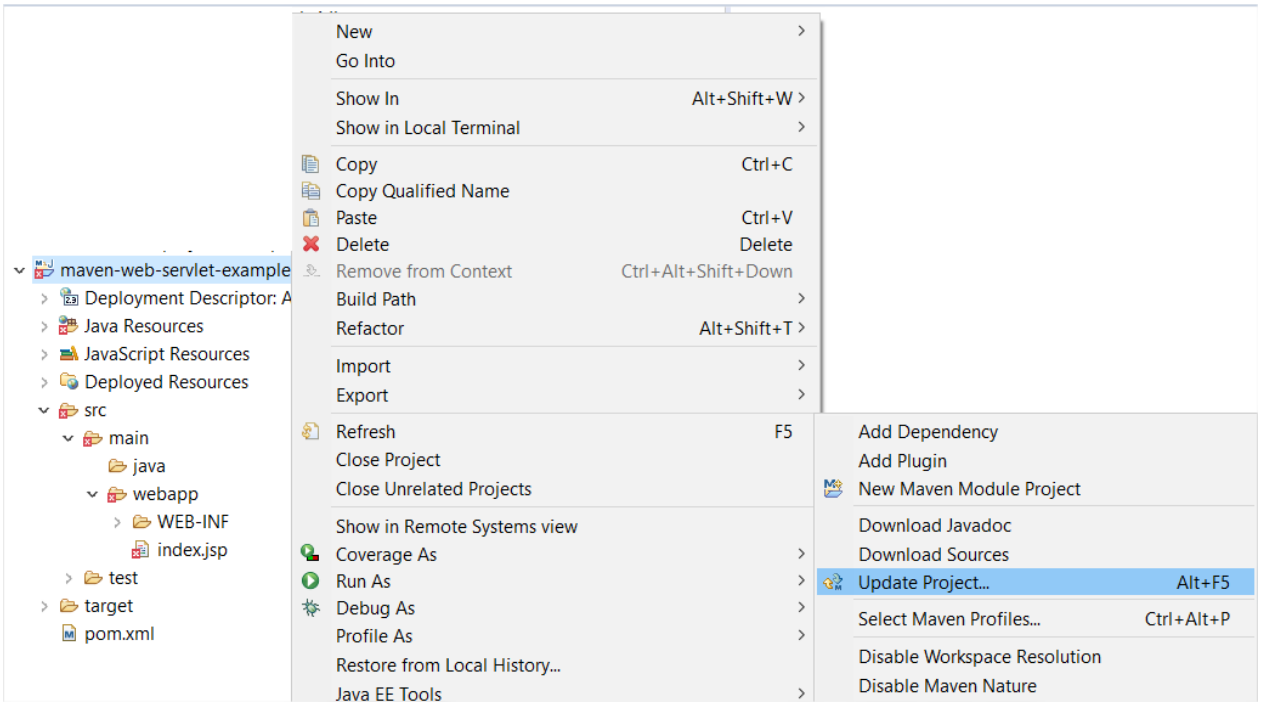
Le fichier **pom.xml** peut devenir :

```
<?xml version="1.0" encoding="UTF-8"?>

<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance"
 xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
 <modelVersion>4.0.0</modelVersion>
 <groupId>com.cours.spring</groupId>
 <artifactId>maven-web-servlet-examples</artifactId>
 <version>0.0.1-SNAPSHOT</version>
 <packaging>war</packaging>
 <name>maven-web-servlet-examples</name>
 <properties>
     <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
     <maven.compiler.source>1.8</maven.compiler.source>
     <maven.compiler.target>1.8</maven.compiler.target>
     <javax.servlet.version>4.0.0</javax.servlet.version>
     <javax.servlet.jsp.version>2.3.0</javax.servlet.jsp.version>
     <jstl.version>1.2</jstl.version>
     <log4j.version>1.2.17</log4j.version>
     <commons.logging.version>1.2</commons.logging.version>
  </properties>

  <dependencies>
     <!-- API Servlet : implémentation -->
     <dependency>
        <groupId>javax.servlet</groupId>
        <artifactId>javax.servlet-api</artifactId>
        <version>${javax.servlet.version}</version>
        <scope>provided</scope>
     </dependency>
     <dependency>
        <groupId>javax.servlet.jsp</groupId>
        <artifactId>javax.servlet.jsp-api</artifactId>
        <version>${javax.servlet.jsp.version}</version>
        <scope>provided</scope>
     </dependency>
     <dependency>
        <groupId>jstl</groupId>
        <artifactId>jstl</artifactId>
        <version>${jstl.version}</version>
     </dependency>
     <!-- Debut log4j dependencies -->
     <dependency>
        <groupId>log4j</groupId>
        <artifactId>log4j</artifactId>
        <version>${log4j.version}</version>
     </dependency>
     <dependency>
```

```
        <groupId>log4j</groupId>
        <artifactId>apache-log4j-extras</artifactId>
        <version>${log4j.version}</version>
      </dependency>
      <dependency>
        <groupId>commons-logging</groupId>
        <artifactId>commons-logging</artifactId>
        <version>${commons.logging.version}</version>
      </dependency>
      <!-- Fin log4j dependencies -->
    </dependencies>
    <build>
      <finalName>maven-web-servlet-examples</finalName>
    </build>
</project>
```

5. *Mise à jour du projet*

Faire un **Maven →Update Project** de votre projet.



6. *Le fichier webapp/index.jsp*

Le fichier **index.jsp** aura pour contenu :

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
   pageEncoding="UTF-8"%>
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c"%>
<html>
   <head>
      <title>Page d'exemples pour la formation Java Web Servlet </title>
      <meta charset="UTF-8">
      <meta name="viewport" content="width=device-width, initial-scale=1.0">
   </head>
   <body>
      <h1>Page d'exemples pour la formation Java Web Servlet</h1>
   </body>
</html>
```

7. *Le web.xml*

Le fichier **web.xml** aura pour contenu :

```
<!DOCTYPE web-app PUBLIC
 "-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"
 "http://java.sun.com/dtd/web-app_2_3.dtd" >

<web-app>
  <display-name>Archetype Created Web Application</display-name>
</web-app>
```

Le fichier **web.xml** peut devenir :

```
<?xml version="1.0" encoding="UTF-8"?>
<web-
app version="3.1" xmlns="http://xmlns.jcp.org/xml/ns/javaee" xmlns:xsi="http://www.w3.org/2001/XMLSch
ema-
instance" xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee http://xmlns.jcp.org/xml/ns/javaee/web-
app_3_1.xsd">
   <session-config>
      <session-timeout>
            30
      </session-timeout>
   </session-config>
   <welcome-file-list>
      <welcome-file>index.jsp</welcome-file>
   </welcome-file-list>
</web-app>
```

8. *Le fichier webapp/pages/myPage.jsp*

Le fichier **webapp/pages/myPage.jsp** aura pour contenu :

```
<%@page contentType="text/html" pageEncoding="UTF-8"%>


<%

String myParam = "";

if (request.getParameter("myParam") != null) {
   myParam = request.getParameter("myParam");
   System.out.println("getParameter myParam : "+myParam);
}

if (request.getAttribute("myParam") != null && request.getAttribute("myParam") instanceof String) {
   myParam = (String) request.getAttribute("myParam");
   System.out.println("getAttribute myParam : "+myParam);
}

%>

<!DOCTYPE html>
<html>
   <head>
      <title>Servlet MyServlet</title>
   </head>
   <body>
      <h1>Servlet MyServlet at ${pageContext.request.contextPath}</h1>
      <div>myParam :  <% out.println(myParam);%></div>
   </body>
</html>
```

9. *Configuration d'une Servlet*

On peut configurer une servlet de deux façons :

- Par configuration XML avec le **web.xml**

```
<servlet>
  <servlet-name>MyServlet</servlet-name>
  <servlet-class>com.cours.servlets.MyServlet</servlet-class>
</servlet>
<servlet-mapping>
  <servlet-name>MyServlet</servlet-name>
  <url-pattern>/MyServlet</url-pattern>
</servlet-mapping>
```

- Par configuration Java avec l'annotation **@WebServlet**

```
package com.cours.servlets;

import java.io.IOException;
import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

@WebServlet(name = "MyServlet", urlPatterns = { "/MyServlet" })
public class MyServlet extends HttpServlet {

  private static final long serialVersionUID = 1L;

  @Override
  public void init() throws ServletException {

  }

  @Override
  protected void doGet(HttpServletRequest request, HttpServletResponse response)
      throws ServletException, IOException {
  }

  protected void doPost(HttpServletRequest request, HttpServletResponse response)
      throws ServletException, IOException {
  }

  @Override
  public void destroy() {
  }
}
```

10.La Servlet MyServlet

La classe **MyServlet** aura pour contenu :

```
package com.cours.servlets;

import java.io.IOException;
import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

@WebServlet(name = "MyServlet", urlPatterns = { "/MyServlet" })
public class MyServlet extends HttpServlet {

   private static final long serialVersionUID = 1L;

   @Override
   public void init() throws ServletException {
      System.out.println("******************** initialisation de la Servlet " + this.getClass().getSimpleName()
            + " ********************");

   }

   @Override
   protected void doGet(HttpServletRequest request, HttpServletResponse response)
         throws ServletException, IOException {
      System.out.println("******************** doGet de la Servlet " + this.getClass().getSimpleName()
            + " ********************");
      String myParam = request.getParameter("myParam");

      request.setAttribute("myParam", myParam);
      System.out.println("myParam : " + myParam);
      this.getServletContext().getRequestDispatcher("/pages/myPage.jsp").forward(request, response);
   }

   protected void doPost(HttpServletRequest request, HttpServletResponse response)
         throws ServletException, IOException {
      System.out.println("******************** doPost de la Servlet " + this.getClass().getSimpleName()
            + " ********************");
   }

   @Override
   public void destroy() {
      System.out.println("******************** Destruction de la Servlet " + this.getClass().getSimpleName()
            + " ********************");
   }
}
```

11.La Servlet AdminServlet

La classe **AdminServlet** aura pour contenu :

```
package com.cours.servlets;

import java.io.IOException;

import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

@WebServlet(name = "Admin", urlPatterns = { "/Admin" })
public class AdminServlet extends HttpServlet {

   /**
    *
    */
   private static final long serialVersionUID = 1L;

   @Override
   public void init() throws ServletException {
      System.out.println("******************** initialisation de la Servlet " + this.getClass().getSimpleName()
            + " ********************");

   }

   @Override
   protected void doGet(HttpServletRequest request, HttpServletResponse response)
         throws ServletException, IOException {
      System.out.println("******************** doGet de la Servlet " + this.getClass().getSimpleName()
            + " ********************");
      this.getServletContext().getRequestDispatcher("/pages/connectionAdminPage.jsp").forward(request, respon
se);
   }

   protected void doPost(HttpServletRequest request, HttpServletResponse response)
         throws ServletException, IOException {
      System.out.println("******************** doPost de la Servlet " + this.getClass().getSimpleName()
            + " ********************");
      String name = request.getParameter("name");
      String password = request.getParameter("password");
      request.setAttribute("name", name);
      request.setAttribute("password", password);
      System.out.println("name : " + name + " , password : " + password);
      if ("admin".equals(name) && "admin".equals(password)) {
         this.getServletContext().getRequestDispatcher("/pages/succesAdminPage.jsp").forward(request, response)
;
      } else {
```

```
            request.setAttribute("errorMessage", "Veuillez remplir les informations");
            this.getServletContext().getRequestDispatcher("/pages/connectionAdminPage.jsp").forward(request, resp
onse);
        }
    }

    @Override
    public void destroy() {
        System.out.println("******************** Destruction de la Servlet " + this.getClass().getSimpleName()
                + " ********************");

    }
}
```

La page **/webapp/pages/connectionAdminPage.jsp** aura pour contenu :

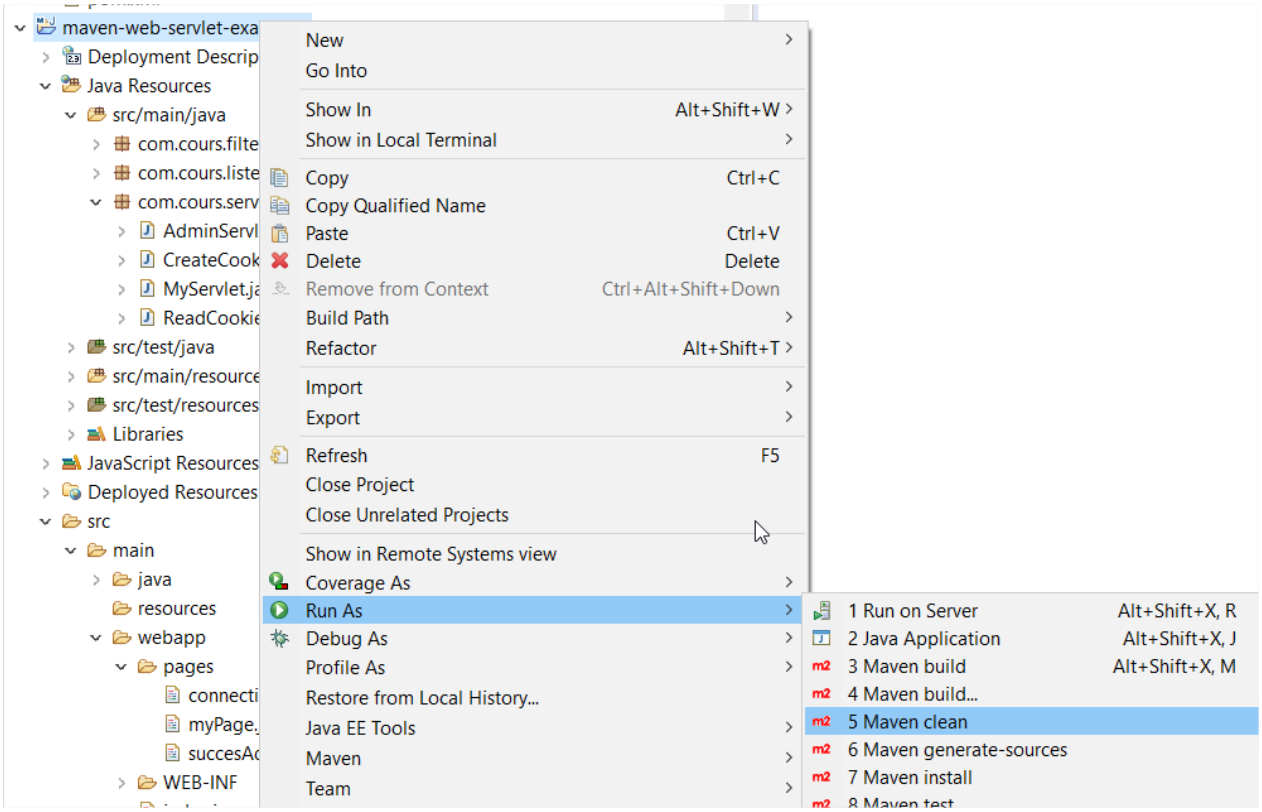
```
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c"%>
<!DOCTYPE html>
<html>
    <head>
        <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
        <title>Formulaire authentification pour la page Admin</title>
    </head>
    <body>
        <h1>Formulaire authentification pour la page Admin</h1>
        <%-- ${pageContext.request.contextPath} --%>
        <c:if test="${not empty errorMessage}">
            <div id="errorMessage" style="color: red">${errorMessage}</div>
             <%-- Equivalent à  <c:out value="${errorMessage}" />--%>
        </c:if>
            <form action="${pageContext.request.contextPath}/Admin" method="POST">
                Name : <input type="text" name="name" id="name"/><br/>
                Password : <input type="password" name="password" id="password"/><br/>
                <input type="submit" value="login">
            </form>
    </body>
</html>
```

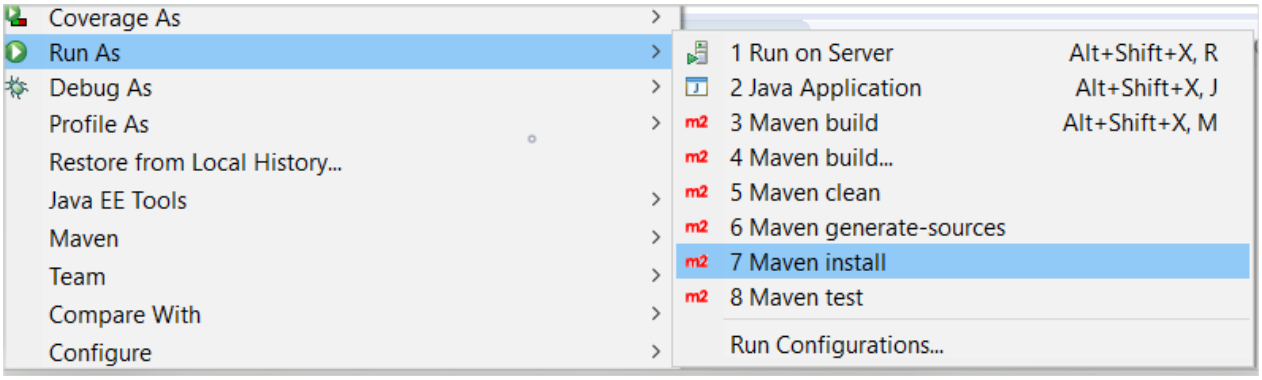
La page **/ webapp/pages/succesAdminPage.jsp** aura pour contenu :

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
   pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>Bravo Mr ${name} pour votre connexion à la page Admin</title>
</head>
<body>
   <h1>Bravo Mr ${name} pour votre connexion à la page Admin</h1>
</body>
</html>
```

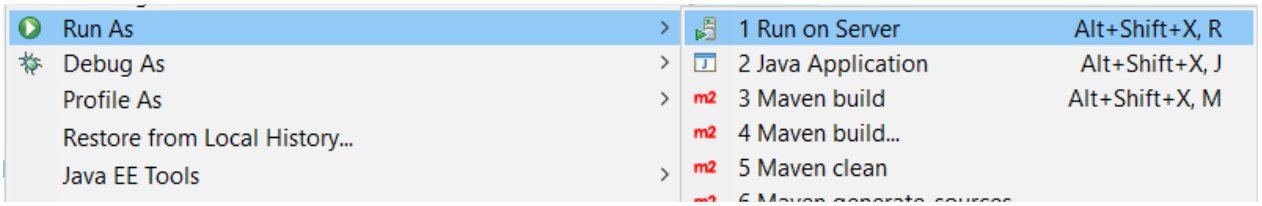
12.Lancement de l'application

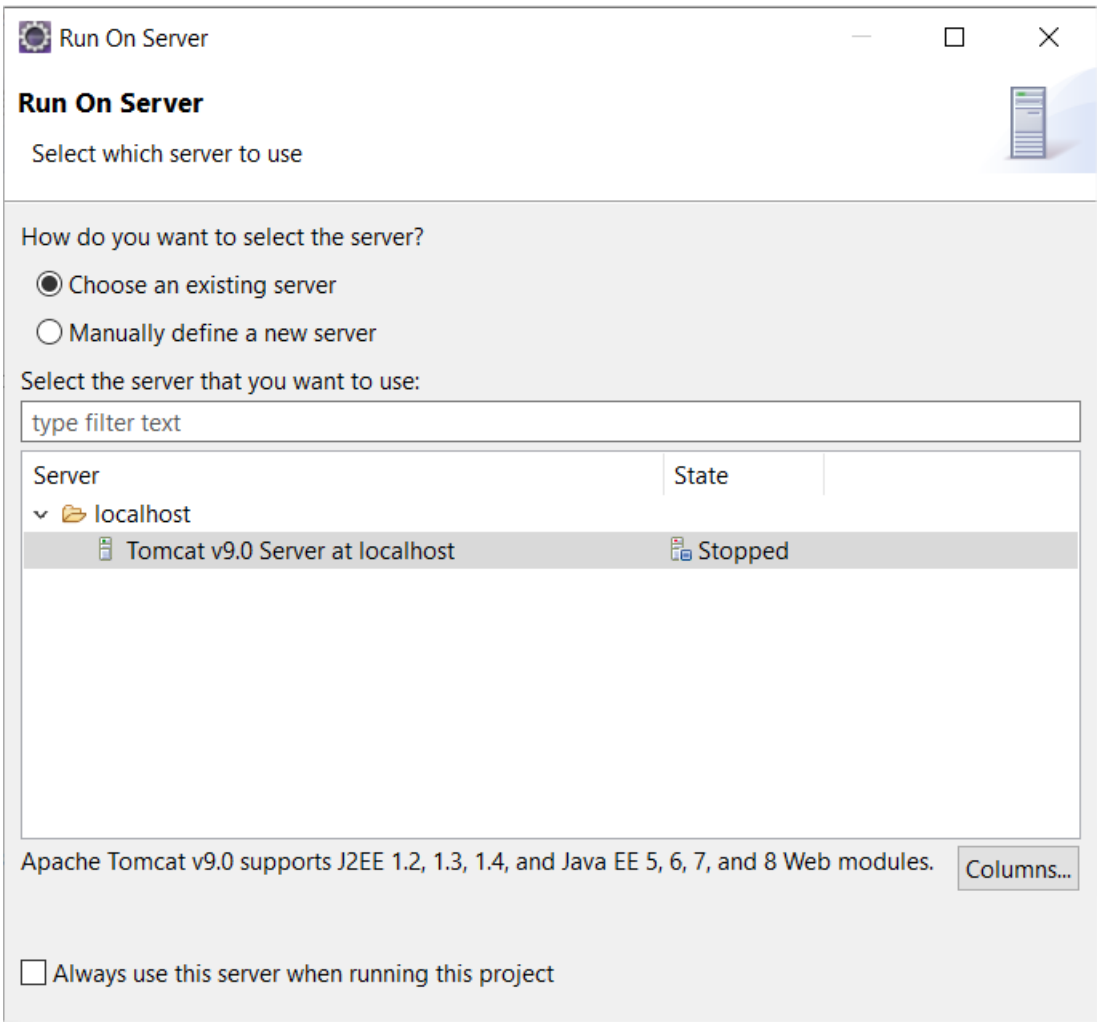
Lancez d'abord un **Maven clean** puis un **Maven install.**





Lancez un **Run on Server.**





On obtient sur http://localhost:8081/maven-web-servlet-examples/ :

On obtient aussi sur l'URL http://localhost:8081/maven-web-servlet-examples/MyServlet?myParam=123456

On obtient sur http://localhost:8081/maven-web-servlet-examples/Admin :

Taper : **admin** puis **admin** puis valider.

13.Le Listener SessionCounterListener

La classe **SessionCounterListener** aura pour contenu :

```
package com.cours.listeners;

import java.util.concurrent.atomic.AtomicInteger;

import javax.servlet.http.HttpSessionEvent;
import javax.servlet.http.HttpSessionListener;

import javax.servlet.annotation.WebListener;

@WebListener
public class SessionCounterListener implements HttpSessionListener {

   private static AtomicInteger atomicCounter = new AtomicInteger(0);

   public static int getTotalActiveSession() {
      return atomicCounter.get();
   }

   @Override
   public void sessionCreated(HttpSessionEvent arg0) {
      atomicCounter.incrementAndGet();
      System.out.println(
            "sessionCreated - add one session into counter, totalActiveSessions : " + getTotalActiveSession());
   }

   @Override
   public void sessionDestroyed(HttpSessionEvent arg0) {
      atomicCounter.decrementAndGet();
      System.out.println(
            "sessionDestroyed - deduct one session from counter, totalActiveSessions : " + getTotalActiveSession());
   }
}
```

14.Le Filtre à Servlet MyFilter

La classe **MyFilter** aura pour contenu :

```
package com.cours.filters;

import java.io.IOException;

import javax.servlet.Filter;
import javax.servlet.FilterChain;
import javax.servlet.RequestDispatcher;
import javax.servlet.ServletException;
import javax.servlet.ServletRequest;
import javax.servlet.ServletResponse;
import javax.servlet.annotation.WebFilter;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpSession;

@WebFilter(servletNames = "Admin")
public class MyFilter implements Filter {

   @Override
   public void doFilter(ServletRequest request, ServletResponse response, FilterChain chain)
         throws IOException, ServletException {
      String name = request.getParameter("name");
      String password = request.getParameter("password");
      if ("admin".equals(name) && "admin".equals(password)) {
         HttpServletRequest httpRequest = (HttpServletRequest) request;
         HttpSession session = httpRequest.getSession(); //sessionCreated() is executed
         session.setAttribute("name", name);
         session.setAttribute("password", password);
         //session.invalidate();  //sessionDestroyed() is executed
         chain.doFilter(request, response);// sends request to next resource
      } else {
         request.setAttribute("errorMessage", "Votre identifiant ou votre mot de passe est incorrecte");
         RequestDispatcher dispatcher = request.getRequestDispatcher("/pages/connectionAdminPage.jsp");
         dispatcher.include(request, response);
      }
   }
}
```

On obtient sur http://localhost:8081/maven-web-servlet-examples/Admin :

V) Présentation de Spring MVC

Le framework **Spring MVC** de **Spring** a une architecture de type MVC (**M**odel-**V**iew-**C**ontroller) et ses composants servent pour développer des applications Web flexibles et faiblement couplées. Le modèle MVC permet de séparer les différentes parties d'une application web à savoir la gestion de requêtes d'entrée envoyées par le client, la logique métier et logique UI (affichage résultats en réponses aux requêtes) tout en assurant un couplage moins fort (Lazy) entre les différentes classes de l'application.

- La partie **M**odèle encapsule les données de l'application; ces données sont en général définies et traitées par des simples POJO (simple classe java).
- La partie **V**ue est responsable du rendu des données du modèle et, en général, elle génère une sortie HTML que le navigateur du client peut interpréter.
- La partie **C**ontrôleur est responsable du traitement des demandes des utilisateurs et de la construction d'un modèle de vue approprié pour le rendu du résultat

Dans Spring-MVC le contenair (conteneur) sert à créer:

- Le **contexte** de l'application Web
- Les objets traitant les requêtes (**Controller**)
- Les objets créant les pages HTML (**View**)
- Les objets donnés des formulaires (les requêtes)
- Les liens avec les couches métiers et BD
- Le mapping des URL vers les contrôleurs
- Le mapping des vues, etc.

Le fait que dans Spring on ait un couplage moins fort (couplage lazy) entre les différentes classes est dû au concept de programmation appelé **IoC** (**Inversio of Control**); celui-ci offre un mécanisme qui facilite la mise en place des dépendances entre les classes par l'injection automatique des objets. Sous Spring, les objets traités sont des beans et l'injection automatique des objets se fait en se servant de fichiers XML, soit par des annotations, soit par les constructeurs ou les Setters des beans.
L'inversion de contrôle laisse au conteneur en l'occurrence ici celui de Spring la responsabilité de gérer le cycle de vie des objets. Avec l'IoC on référence des interfaces ou des classes plus génériques ce qui permet d'avoir un code clair, réutilisable et facile à tester. L'inversion de contrôle permet de changer le comportement de l'application, en modifiant la description xml du conteneur, sans changer les éléments programmés !
Sans **IoC** la création d'objets se ferait par l'opérateur ***new*** et cette création d'objets par instanciation augmente les dépendances (couplage fort) entre classes.

1. *Le modèle*

Dans le contexte de Spring MVC, un modèle représente généralement les données qui seront transmises vers la vue depuis une opération (définie dans un contrôleur Web) . Spring MVC prend en charge diverses options et modèles pour définir les données pouvant être échangées. Une application Spring MVC typique peut utiliser une combinaison de ces options. La flexibilité de Spring MVC permettant d'avoir un contrôle fin pour la définition des **modèles** est très puissante, mais elle s'accompagne d'une variabilité et d'une complexité considérables dans la mise en oeuvre.

Une méthode d'un contrôleur dans Spring peut prendre en charge ou pas des paramètres d'entrée, et les principaux mécanismes de Spring permettant de spécifier ces paramètres d'entrée sont les annotations **@RequestParam** et **@ModelAttribute**. L'annotation @RequestParam est utilisée pour lier des paramètres de requête individuels, tels que des chaînes et des entiers, aux paramètres de méthode du contrôleur. L'annotation @ModelAttribute est utilisée pour lier des objets complexes, tels que des objets de domaine, des objets de transfert de données et / ou des objets de sauvegarde de formulaire, aux paramètres de méthodes d'un contrôleur.

Une méthode d'un contrôleur dans Spring peut également retourner des données issues d'un modèle. Le principal mécanisme utilisé dans Spring pour spécifier les données de modèle de sortie est l'objet **ModelAndView**. Si la méthode ne renvoie pas des données, la méthode du contrôleur peut simplement renvoyer une chaîne représentant la vue à restituer. Si la méthode du contrôleur renvoie des données, un objet ModalAndView doit être instancié et chaque variable de sortie est ajoutée en tant qu'attribut de modèle à l'objet ModelAndView.

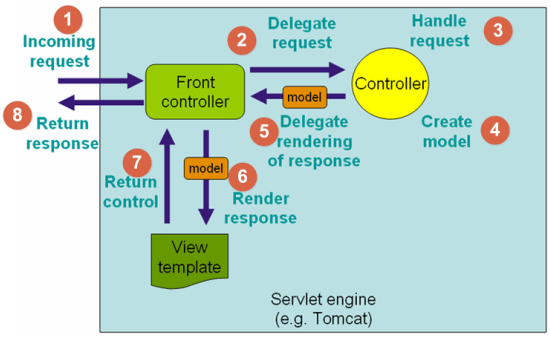
2. *La Vue*

Dans Spring MVC, une vue génère l'interface utilisateur en fonction des données issues du modèle. Il existe diverses technologies basées sur JAVA pour l'implémentation de vues, mais Java ServerPages (JSP) est la technologie prédominante pour la définition de vues. Les JSP sont conçus pour être implémentés de manière optimisée pour les concepteurs Web.

3. *Le Controleur*

Dans Spring MVC, une Le Front contrôleur traite les requêtes provenant de l'extérieur et de ce fait:

1. Il analyse l'URL de la requête,
2. Il appelle le contrôleur qui doit traiter la requête; ce dernier renvoie un objet de type Modèle et le nom logique de la page qui sert à construire la page à afficher,
3. Il appelle la page demandée et celle-ci construit la réponse
4. Il renvoie la réponse au client demandeur.



Le contrôleur du framework Spring MVC, la servlet **DispatcherServlet** gère donc toutes les requêtes et réponses HTTP de toutes les applications développées avec Spring MVC. Le *workflow* de traitement des requêtes de Spring MVC est illustré dans le diagramme présenté à coté. la servlet **DispatcherServlet** est en faite le tour de contrôle de Spring MVC, précisément c'est le **Contrôleur** dans le Design Pattern **MVC**. Chaque requête Web traitée par Spring MVC passe par cette servlet DispatcherServlet. C'est une implémentation du Pattern Front Controller; il fournit un point d'entrée unique dans toute application Spring MVC. C'est aussi le pont entre Java et Spring. La servlet DispatcherServlet est, comme tout autre Servlet d'une application web Java EE, déclarée dans le fichier **web.xml** avec un modèle d'URL, mais la seule particularité est que le modèle d'URL pour la servlet de répartition est suffisant pour mapper chaque requête Web à DispatcherServlert.

En résumé on a la servlet **org.springframework.web.servlet.DispatcherServlet** qui est le point d'entrée générique qui délègue les requêtes à des Controller
et en principe à un org.springframework.web.servlet.mvc.Controller qui prend en charge une requête, utilise la couche métier pour répondre en fabriquant un modèle sous la forme d'une java.util.Map qui contient les éléments de la réponse. C'est le Controller qui choisit une **org.springframework.web.servlet**.View qui sera
paramétrée par la Map pour donner la page qui sera affichée.

a. Implémentation du front controller avec une configuration XML

L'implémentation du front Controller peut se faire par configuration XML dans le fichier **WEB-INF/web.xml** en voici un exemple :

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns="http://xmlns.jcp.org/xml/ns/javaee"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee http://xmlns.jcp.org/xml/ns/javaee/web-app_3_1.xsd"
      version="3.1">
  <display-name>My Spring MVC App</display-name>
  <servlet>
    <servlet-name>dispatcher</servlet-name>
    <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
    <load-on-startup>1</load-on-startup>
  </servlet>
  <servlet-mapping>
    <servlet-name>dispatcher</servlet-name>
    <url-pattern>/</url-pattern>
  </servlet-mapping>
  <context-param>
    <param-name>contextConfigLocation</param-name>
    <param-value>/WEB-INF/dispatcher-servlet.xml</param-value>
  </context-param>
  <listener>
    <listener-class>org.springframework.web.context.ContextLoaderListener</listener-class>
  </listener>
  <session-config>
    <session-timeout>
      30
    </session-timeout>
  </session-config>
  <welcome-file-list>
    <welcome-file>index.jsp</welcome-file>
  </welcome-file-list>
</web-app>
```

Le fichier **WEB-INF/dispatcher-servlet.xml** aura pour contenu :

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

  <context:component-scan base-package="myPackageWithSpringCode" />
  <mvc:annotation-driven />
  <bean class="org.springframework.web.servlet.view.InternalResourceViewResolver">
    <property name="prefix">
      <value>/pages/</value>
    </property>
    <property name="suffix">
      <value>.jsp</value>
    </property>
  </bean>
</beans>
```

b. Implémentation du front controller avec une configuration Java

L'implémentation du front Controller peut se faire par configuration Java dans les classe **WebAppSpringMvcInitializer**, **AppConfig** et **WebMvcConfig**.

WebAppSpringMvcInitializer aura pour contenu :

```
package com.cours.spring.initializer;

import javax.servlet.ServletContext;
import javax.servlet.ServletException;
import javax.servlet.ServletRegistration;

import org.springframework.web.WebApplicationInitializer;
import org.springframework.web.context.support.AnnotationConfigWebApplicationContext;
import org.springframework.web.servlet.DispatcherServlet;

import com.cours.spring.config.AppConfig;

public class WebAppSpringMvcInitializer implements WebApplicationInitializer {
   @Override
   public void onStartup(ServletContext servletContext) throws ServletException {
      AnnotationConfigWebApplicationContext appContext = new AnnotationConfigWebApplicationContext();
      appContext.register(AppConfig.class);

      ServletRegistration.Dynamic dispatcher = servletContext.addServlet("dispatcher",
            new DispatcherServlet(appContext));
      dispatcher.setLoadOnStartup(1);
      dispatcher.addMapping("/");
   }
}
```

AppConfig aura pour contenu :

```
package com.cours.spring.config;

import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.ComponentScan;
import org.springframework.context.annotation.Configuration;
import org.springframework.transaction.annotation.EnableTransactionManagement;
import org.springframework.web.servlet.config.annotation.EnableWebMvc;



@Configuration
@EnableTransactionManagement
@EnableWebMvc
@ComponentScan({ "myPackageWithSpringCode*" })
public class AppConfig {

   @Bean
   public MyBean1 getMyBean1() {
      MyBean1 myBean1 = new MyBean1();
      return myBean1;
   }
   @Bean
   public MyBean2 getMyBean2() {
      MyBean2 myBean2 = new MyBean2();
      return myBean2;
   }
   @Bean
   public MyBean3 getMyBean3() {
      MyBean3 myBean3 = new MyBean3();
      return myBean3;
   }
}
```

WebMvcConfig aura pour contenu :

```
package com.cours.spring.config;

import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.ComponentScan;
import org.springframework.context.annotation.Configuration;
import org.springframework.web.servlet.view.InternalResourceViewResolver;
import org.springframework.web.servlet.view.JstlView;

@Configuration
@ComponentScan("myPackageWithSpringCode")
public class WebMvcConfig {
   @Bean(name = "viewResolver")
   public InternalResourceViewResolver viewResolver() {
      InternalResourceViewResolver viewResolver = new InternalResourceViewResolver();
      viewResolver.setViewClass(JstlView.class);
      viewResolver.setPrefix("/pages/");
      viewResolver.setSuffix(".jsp");
      return viewResolver;
   }
}
```

VI) La configuration dans Spring

1. Présentation

a. La configuration par fichier xml

La configuration de Spring se fait soit par programmation soit par fichiers XML. Dans la configuration par fichiers XML les classes représentées sont des beans. Si on utilise les annotations, on peut réduire et même éviter certains des fichiers XML de configuration. Cependant, les fichiers de configuration XML sont très populaires et faciles à utiliser. Un des principaux fichiers de configuration de Spring MVC est web.xml.

- **Le fichier web.xml**

Le fichier de configuration **web.xml** contient le point d'entrée de l'application à savoir la servlet **DispatcherServlet** le contrôleur qui traite les requêtes en entrée. Ce fichier "web.xml" est défini dans le répertoire **"WebContent/WEB-INF"** de l'application. Dans son lancement Spring initialise le contrôleur DispatcherServlet, puis charge le contexte de l'application à partir d'un fichier dont le nom est de la forme **[nom_servlet]-servlet.xml** situé lui aussi dans le répertoire "WebContent/WEB-INF". Voir la valeur de la partie entre-crochet ([nom_servlet]) dans la balise <param-value> dans le fichier web.xml.

Remarque:
A partir de Spring .3.0 on définit le fichier "[nom_servlet]-servlet.xml" dans le répertoire **"WebContent/WEB-INF/spring"** et le fichier web.xml est devenu optionnel.

Lorsque la servlet DispatcherServlet est initialisée, elle initialise un objet "**org.springframework.web.context.WebApplicationContext**" (qui dérive de **ApplicationContext**) dont une des implémentations donne l'objet "org.springframework.web.context.support.**XmlWebApplicationContext**". Cet objet XmlWebApplicationContext contient la configuration de tous les beans de l'application qui seront définis dans le conteneur Spring. Cet objet utilisera le fichier de configuration Spring [nom-servlet]-servlet.xml.

On peut définir plusieurs Servlets dans "web.xml". Chaque servlet peut être initialisé, mappé à différentes URL; on aura ainsi plusieurs conteneurs Spring correspondant à chaque servlet. Dans ce cas, certaines définitions de beans peuvent être répétées dans de nombreux conteneurs Spring; on a créée org.springframework.web.context.**ContextLoaderListener** pour résoudre ce problème; par conséquent on a la création d'un contexte org.springframework.web.context.**WebApplicationContext** racine pour toute l'application web; ce contexte racine est partagé par toutes les servlets. **ContextLoaderListener** utilisera le nom de fichier défini dans la balise <param-value> comme nom du contexte racine de l'application.

Dans le fichier **root-context.xml**, nous définirons le bean, les propriétés partagées entre les conteneurs Spring, chaque conteneur Spring dans chaque servlet utilisera le bean, ses propriétés et les attributs définis pour lui-même

Exemple de fichier **web.xml** :

```xml
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns="http://xmlns.jcp.org/xml/ns/javaee"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee http://xmlns.jcp.org/xml/ns/javaee/web-app_3_1.xsd"
         version="3.1">
    <display-name>My Spring MVC App</display-name>
    <servlet>
        <servlet-name>dispatcher</servlet-name>
        <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
        <load-on-startup>1</load-on-startup>
    </servlet>
    <servlet-mapping>
        <servlet-name>dispatcher</servlet-name>
        <url-pattern>/</url-pattern>
    </servlet-mapping>
    <context-param>
        <param-name>contextConfigLocation</param-name>
        <param-value>/WEB-INF/dispatcher-servlet.xml</param-value>
    </context-param>
    <listener>
        <listener-class>org.springframework.web.context.ContextLoaderListener</listener-class>
    </listener>
    <session-config>
        <session-timeout>
            30
        </session-timeout>
    </session-config>
    <welcome-file-list>
        <welcome-file>index.jsp</welcome-file>
    </welcome-file-list>
</web-app>
```

En version copiable :

```xml
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns="http://xmlns.jcp.org/xml/ns/javaee"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee http://xmlns.jcp.org/xml/ns/javaee/web-app_3_1.xsd"
      version="3.1">
  <display-name>My Spring MVC App</display-name>
  <servlet>
    <servlet-name>dispatcher</servlet-name>
    <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
    <load-on-startup>1</load-on-startup>
  </servlet>
  <servlet-mapping>
    <servlet-name>dispatcher</servlet-name>
    <url-pattern>/</url-pattern>
  </servlet-mapping>
  <context-param>
    <param-name>contextConfigLocation</param-name>
    <param-value>/WEB-INF/dispatcher-servlet.xml</param-value>
  </context-param>
  <listener>
    <listener-class>org.springframework.web.context.ContextLoaderListener</listener-class>
  </listener>
```

```
  <session-config>
    <session-timeout>
      30
    </session-timeout>
  </session-config>
  <welcome-file-list>
    <welcome-file>index.jsp</welcome-file>
  </welcome-file-list>
</web-app>
```

Ligne 7 : balise <servlet>

Sous la balise "<servlet>" on a déclaré une balise "<servlet-name>" qui définit le nom d'une instance de la servelt DispatcherServlet en l'occurrence ce nom est "dispatcher" ici dans notre exemple; cette instance sera initialisée avec un paramètre nommé "contextConfigLocation" qui contient le chemin d'accès au fichier de configuration "dispatcher-servlet.xml".
On a aussi sous cette balise "<servlet>" la balise "<load-on-startup>" qui est une valeur entière qui spécifie l'ordre de chargement si jamais on a dans web.xml déclaré plusieurs servlets. Ainsi, si vous devez déclarer plusieurs servlets, vous pouvez définir dans quel ordre elles seront initialisées. Les servlets marquées avec des entiers inférieurs sont chargés avant les servlets marquées avec des entiers plus élevés.
Avec cette balise "<servlet-mapping>", nous le lions la servlet par son nom à un modèle d'URL qui spécifie les requêtes HTTP qu'il traitera.

Sous la balise <servlet-name> est défini le nom d'une instance de la servlet DispatcherServlet

Ligne 10 : Balise <load-on-startup>
Sous cette balise est définie une valeur entière qui spécifie l'ordre de chargement des servlets si jamais on a dans web.xml déclaré plusieurs servlets

Ligne 16 : **Balise "<context-param>"**
L'existence de cette balise au sein de "web.xml" sert à définir le contexte général de l'application i.e "ApplicationContext"; en effet lorsque on aura une application web qui nécessitera plusieurs "[nom_servlet]-servlet.xml" donc plusieurs servlets, on y déclarera les différents beans qui sont utilisées dans le fichier de configuration dont le nom est sous la balise <param-value> de la balise <context-param>.

Ligne 20 : Balise "<listner>"

Un listener (écouteur) est un terme couramment utilisé en Java pour désigner un objet qui sera notifié lors d'une modification de son environnement. Dans les patrons de conception (Design Patterns) Objet, on l'appelle plus généralement Observateur.

Pour le conteneur Web Java EE, un listener designe une classe qui implémente une des interfaces définies dans l'API servlet.

Le principe d'utilisation est le même pour tous les types de listeners Web. Si l'application souhaite être avertie d'un événement particulier survenant pour un ServletContext (c'est-à-dire pour l'ensemble de l'application Web), une requête ou une session HTTP alors elle doit fournir une implémentation d'un listener. Une méthode de ce dernier sera appelée par le conteneur à chaque fois que l'événement concerné surviendra lors de la vie de l'application.

Exemples :

javax.servlet.ServletContextListener
Permet d'écouter les changements d'état du ServletContext. Le conteneur Web avertit l'application de la création du ServletContext grâce à la méthode contextInitialized et de la destruction du ServletContext grâce à la méthode contextDestroyed. Il est important de comprendre que le ServletContext représente l'application Web. Donc un ServletContextListener est un moyen de réaliser des traitements au moment du lancement de l'application Web et/ou au moment de son arrêt.

javax.servlet.ServletContextAttributeListener
Permet d'écouter les changements d'état des attributs stockés dans le ServletContext (les attributs de portée application). Le conteneur avertit l'application de l'ajout d'un attribut (appel à la méthode ServletContextAttributeListener.attributeAdded, de la suppression d'un attribut (appel à la méthode ServletContextAttributeListener.attributeRemoved et de la modification d'un attribut (appel à la méthode ServletContextAttributeListener.attributeReplaced.

javax.servlet.ServletRequestListener
Permet d'écouter l'entrée et/ou la sortie d'une requête de l'environnement de l'application Web. Le conteneur avertit l'application de l'entrée d'une requête à traiter grâce à la méthode requestInitialized et de la sortie de la requête grâce à la méthode requestDestroyed.

javax.servlet.ServletRequestAttributeListener
Permet d'écouter les changements d'état des attributs stockés dans la HttpServletRequest (les attributs de portée requête). Le conteneur avertit l'application de l'ajout d'un attribut (appel à la méthode ServletRequestAttributeListener.attributeAdded, de la suppression d'un attribut (appel à la méthode ServletRequestAttributeListener.attributeRemoved et de la modification d'un attribut (appel à la méthode ServletRequestAttributeListener.attributeReplaced.

javax.servlet.http.HttpSessionListener
Permet d'écouter la création et la suppression d'une HttpSession. Le conteneur avertit l'application de la création d'une session grâce à la méthode sessionCreated et de la suppression d'une session grâce à la méthode sessionDestroyed. La suppression d'une session signifie que soit elle a été invalidée par l'application elle-même grâce à la méthode HttpSession.invalidate soit elle est arrivée à expiration et le conteneur a décidé de l'invalider.

javax.servlet.http.HttpSessionAttributeListener
Permet d'écouter les changements d'état des attributs stockés dans la HttpSession (les attributs de portée session). Le conteneur avertit l'application de l'ajout d'un attribut (appel à la méthode HttpSessionAttributeListener.attributeAdded, de la suppression d'un attribut (appel à la méthode HttpSessionAttributeListener.attributeRemoved et de la modification d'un attribut (appel à la méthode HttpSessionAttributeListener.attributeReplaced.

Il existe également deux autres listeners Web : Le HttpSessionActivationListener et le HttpSessionBindingListener. Ils ne sont pas décrits ici car ils sont réservés à des usages plus avancés de l'API Servlet.

- **WebApplicationContext**

Dans toute application Java EE on dispose d'une interface **ApplicationContext** qui fournit toutes les informations de configuration. Le framework Spring propose plusieurs classes qui implémentent cette interface qui représente le contexte racine de l'application; tous les différents éléments de configuration de l'application sont définis dans le fichier de configuration **applicationContext.xml** qui est chargé par le ContextLoaderListener (voir le fichier web.xml).

Pour une application web, le contexte est spécifié par l'interface **WebApplicationContext** qui étend **ApplicationContext** et qui ajoute une méthode de récupération de l'API Servlet standard **ServletContext** pour l'application Web. Outre les standardscopes (portées) singleton et prototype des beans Spring, trois scopes supplémentaires sont disponibles dans un contexte d'application Web:

- **request** - étend une définition de bean unique au cycle de vie d'une requête HTTP unique; c'est-à-dire que chaque requête HTTP a sa propre instance d'un bean créé à l'arrière d'une définition de bean unique.
- **session** - étend une définition de bean unique au cycle de vie d'une session HTTP.
- **application** - étend une définition de bean unique au cycle de vie d'un ServletContext.

- **Le fichier dispatcher-servlet.xml**

Nous avons vu plus haut que spring charge **dispatcher-servlet.xml** ([nom du servlet] -servlet.xml) par défaut à partir du répertoire WEB-INF. On peut remplacer ce comportement par défaut en se servant du paramètre **contextConfigLocation** comme paramètre d'initialisation (voir dans la balise <init-param> de web.xml) de DispatcherServlet. Dans ce qui suit, nous pouvons charger dispatcher-servlet.xml au lieu de springMVC-servlet.xml en donnant comme valeur de la balise <init-param> la valeur "dispatcher".

Exemple de fichier dispatcher-servlet.xml

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


    <context:component-scan base-package="mesPackagesAvecClassesSpring" />
    <mvc:annotation-driven />
    <bean class="org.springframework.web.servlet.view.InternalResourceViewResolver">
        <property name="prefix">
            <value>/pages/</value>
        </property>
        <property name="suffix">
            <value>.jsp</value>
        </property>
    </bean>
</beans>
```

En version copiable :

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


  <context:component-scan base-package="mesPackagesAvecClassesSpring" />
  <mvc:annotation-driven />
```

```
  <bean class="org.springframework.web.servlet.view.InternalResourceViewResolver">
    <property name="prefix">
      <value>/pages/</value>
    </property>
    <property name="suffix">
      <value>.jsp</value>
    </property>
  </bean>
</beans>
```

Le fichier dispatcher-servlet.xml ([nom_servlet]-servlet.xml) s'appelle fichier de contexte pour l'application. Dans une application on peut avoir plusieurs fichiers de contexte; ce fichier a pour rôle l'analyse des composants de l'application, des annotations Web Spring (@Controller) mais aussi il configure le modèle. Le framework Spring charge le contexte de l'application à partir de ce fichier [nom_servlet]-servlet.xml.

Ligne 17 : La balise **<context: composant-scan ...>** permet d'activer l'analyse des annotations Spring MVC,

Ligne 19 : La balise **<bean** class="org.springframework.web.servlet.view.InternalResourceViewResolver**>** est utilisé pour définir les règles permettant de résoudre les noms de vue.

b. La configuration par programmation Java

Depuis la spécification Servlet 3.0, on peut développer une application Spring sans utiliser des fichiers de configuration "xml». Comme toutes les servlets, la servlet org.springframework.web.servlet.**DispatcherServlet** a nécessite une configuration pour que le conteneur Web puisse l'amorcer et le mapper afin qu'elle puisse traiter les requêtes; cette configuration de DispatcherServlet est un processus bidirectionnel. Tout d'abord, on doit préciser au conteneur de charger la servlet et de la mapper à un ou plusieurs Urlpatterns. Après l'amorçage, la servlet utilise le contexte de l'application (**org.springframework.web.context.WebApplicationContext**) pour se configurer. Ainsi la servlet va détecter les composants nécessaires à partir de ce contexte de l'application et si elle ne le trouve pas, elle utilisera les valeurs par défaut (dans la plupart des cas). Toutes ces différentes étapes vont se faire par programation en commençant par l'amorçage DispatcherServlet.

Comment amorçer DispatcherServlet

La spécification Servlet 3.0 a introduit plusieurs options pour configurer et enregistrer une servlet:

1. utilisation d'un fichier web.xml.
2. utilisation d'un web-fragment.xml.
3. utilisation de javax.servlet.**ServletContainerInitializer**.
4. On peut obtenir une quatrième option en implémentant l'interface **org.springframework.web.WebApplicationInitializer**

Ici c'est la troisième option qui nous intéresse i.e l'utilisation de "javax.servlet.**ServletContainerInitializer**". La servlet DispatcherServlet a besoin de WebApplicationContext qui doit contenir tous les beans concernées la configuration; toutefois, par défaut, DispatcherServlet créera un "org.springframework.web.context.support.XmlWebApplicationContext" et de ce faite Toute application Spring est à même charger la servlet "org.springframework.web.servlet.DispatcherServlet" et le mapper à toutes les requêtes arrivant sur "/ *".

La configuration de servlet par programmation est assurée par l'interface ServletContainerInitializer et Spring l'implémente via la classe **SpringServletContainerInitializer** qui gère d'autres interfaces comme **WebApplicationInitializer** qui permet la configuration du contexte (ServletContext) par programmation. La classe **AbstractAnnotationConfigDispatcherServletInitializer** implémente **WebMvcConfigurer** qui, à son tour implémente en interne WebApplicationInitializer, enregistre un ContextLoaderlistener (optionnelle) et un DispatcherServlet ce qui permet d'ajouter facilement des classes de configuration à charger pour les deux classes, d'appliquer des filtres au DispatcherServlet et de fournir le mappage de servlets.

Remarque:
Dans une configuration type, le contexte racine est chargé par ContextLoaderListener et le contexte de servlet par DispatcherServlet.

Exemple de configuration par programmation

La classe AppInitializer

Cette classe étend la classe abstraite AbstractAnnotationConfigDispatcherServletInitializer et on a vu que celle-ci gère "WebMvcConfigurer" et "WebApplicationInitializer" donc elle va amorcer DispatcherServlet et permettre de réaliser les différentes configurations attendues.

```
1   package com.cours.spring.config;
2
3   import org.springframework.web.servlet.support.AbstractAnnotationConfigDispatcherServletInitializer;
4
5   public class AppInitializer extends AbstractAnnotationConfigDispatcherServletInitializer {
6
7       @Override
8       protected Class<?>[] getRootConfigClasses() {
9           return new Class[] {PersistenceJPAConfig.class};
10      }
11      @Override
12      protected Class<?>[] getServletConfigClasses() {
13          return new Class[] {WebMvcConfig.class};
14      }
15      @Override
16      protected String[] getServletMappings() {
17          return new String[] {"/"};
18      }
19  }
```

En version copiable :

```
package com.cours.spring.config;

import org.springframework.web.servlet.support.AbstractAnnotationConfigDispatcherServletInitializer;

public class AppInitializer extends AbstractAnnotationConfigDispatcherServletInitializer {

   @Override
   protected Class<?>[] getRootConfigClasses() {
      return new Class[] {PersistenceJPAConfig.class};
   }
   @Override
   protected Class<?>[] getServletConfigClasses() {
      return new Class[] {WebMvcConfig.class};
   }
   @Override
   protected String[] getServletMappings() {
      return new String[] {"/"};
   }
}
```

Dans cette classe on implémente trois méthodes à savoir getRootConfigClasses (), getServletMappings () et getServletConfigClasses ():

Ligne 8 : getRootConfigClasses () retourne **"PersistenceJPAConfig.class**"; cet objet va permettre de définir et de configurer la base de donnée de l'application (Datasource), de préciser l'implémentation à utiliser pour la specification JPA (HibernateJpaVendorAdapter) et les propriétés additionnelles.

Il est possible d'utiliser une autre classe qui va permettre de faire la même chose que la classe **AppInitializer**. On appélera cette classe **WebAppSpringMvcInitializer.**

La classe WebAppSpringMvcInitializer

Cette classe implémente l'interface WebApplicationInitializer.
WebAppSpringMvcInitializer aura pour contenu :

```java
package com.cours.spring.initializer;

import javax.servlet.ServletContext;
import javax.servlet.ServletException;
import javax.servlet.ServletRegistration;

import org.springframework.web.WebApplicationInitializer;
import org.springframework.web.context.support.AnnotationConfigWebApplicationContext;
import org.springframework.web.servlet.DispatcherServlet;

import com.cours.spring.config.AppConfig;

public class WebAppSpringMvcInitializer implements WebApplicationInitializer {
    @Override
    public void onStartup(ServletContext servletContext) throws ServletException {
        AnnotationConfigWebApplicationContext appContext = new AnnotationConfigWebApplicationContext();
        appContext.register(PersistenceJPAConfig.class);
        appContext.scan("com.cours.spring");
        ServletRegistration.Dynamic dispatcher = servletContext.addServlet("dispatcher",
                new DispatcherServlet(appContext));
        dispatcher.setLoadOnStartup(1);
        dispatcher.addMapping("/");
    }
}
```

En version copiable :

```
package com.cours.spring.initializer;

import javax.servlet.ServletContext;
import javax.servlet.ServletException;
import javax.servlet.ServletRegistration;

import org.springframework.web.WebApplicationInitializer;
import org.springframework.web.context.support.AnnotationConfigWebApplicationContext;
import org.springframework.web.servlet.DispatcherServlet;

import com.cours.spring.config.AppConfig;

public class WebAppSpringMvcInitializer implements WebApplicationInitializer {
  @Override
  public void onStartup(ServletContext servletContext) throws ServletException {
    AnnotationConfigWebApplicationContext appContext = new AnnotationConfigWebApplicationContext();
    appContext.register(PersistenceJPAConfig.class);
    appContext.scan("com.cours.spring");
    ServletRegistration.Dynamic dispatcher = servletContext.addServlet("dispatcher",
        new DispatcherServlet(appContext));
    dispatcher.setLoadOnStartup(1);
    dispatcher.addMapping("/");
  }
}
```

PersistenceJPAConfig.java aura pour contenu :

```
package com.cours.spring.config;

import java.util.Properties;
import javax.persistence.EntityManagerFactory;
import javax.sql.DataSource;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.ComponentScan;
import org.springframework.context.annotation.Configuration;
import org.springframework.context.annotation.PropertySource;
import org.springframework.core.env.Environment;
import org.springframework.dao.annotation.PersistenceExceptionTranslationPostProcessor;
import org.springframework.data.jpa.repository.config.EnableJpaRepositories;
import org.springframework.jdbc.datasource.DriverManagerDataSource;
import org.springframework.orm.jpa.JpaTransactionManager;
import org.springframework.orm.jpa.LocalContainerEntityManagerFactoryBean;
import org.springframework.orm.jpa.vendor.HibernateJpaVendorAdapter;
import org.springframework.transaction.PlatformTransactionManager;
import org.springframework.transaction.annotation.EnableTransactionManagement;

@Configuration
@EnableTransactionManagement
@PropertySource({
  "classpath:database.properties"
})
@ComponentScan({
  "com.cours.spring"
})
@EnableJpaRepositories(basePackages = "com.cours.spring.repositories")
public class PersistenceJPAConfig {

  @Autowired
  private Environment env;

  public PersistenceJPAConfig() {
    super();
  }

  @Bean
  public LocalContainerEntityManagerFactoryBean entityManagerFactory() {
    final LocalContainerEntityManagerFactoryBean entityManagerFactoryBean = new LocalContainerEntityMan
agerFactoryBean();
    entityManagerFactoryBean.setDataSource(dataSource());
    entityManagerFactoryBean.setPackagesToScan(new String[] {
      "com.cours.entities"
    });

    final HibernateJpaVendorAdapter vendorAdapter = new HibernateJpaVendorAdapter();
    entityManagerFactoryBean.setJpaVendorAdapter(vendorAdapter);
```

```
        entityManagerFactoryBean.setJpaProperties(additionalProperties());

        return entityManagerFactoryBean;
    }

    final Properties additionalProperties() {
        final Properties hibernateProperties = new Properties();
        hibernateProperties.setProperty("hibernate.hbm2ddl.auto", env.getProperty("hibernate.hbm2ddl.auto"));
        hibernateProperties.setProperty("hibernate.dialect", env.getProperty("hibernate.dialect"));
        hibernateProperties.setProperty("hibernate.cache.use_second_level_cache", env.getProperty("hibernate.cache.
use_second_level_cache"));
        hibernateProperties.setProperty("hibernate.cache.use_query_cache", env.getProperty("hibernate.cache.use_qu
ery_cache"));
        // hibernateProperties.setProperty("hibernate.globally_quoted_identifiers", "true");
        return hibernateProperties;
    }

    @Bean
    public DataSource dataSource() {
        final DriverManagerDataSource dataSource = new DriverManagerDataSource();
        dataSource.setDriverClassName(env.getProperty("jdbc.driverClassName"));
        dataSource.setUrl(env.getProperty("jdbc.url"));
        dataSource.setUsername(env.getProperty("jdbc.user"));
        dataSource.setPassword(env.getProperty("jdbc.password"));
        return dataSource;
    }

    @Bean
    public PlatformTransactionManager transactionManager(final EntityManagerFactory emf) {
        final JpaTransactionManager transactionManager = new JpaTransactionManager();
        transactionManager.setEntityManagerFactory(emf);
        return transactionManager;
    }

    @Bean
    public PersistenceExceptionTranslationPostProcessor exceptionTranslation() {
        return new PersistenceExceptionTranslationPostProcessor();
    }
}
```

Le fichier database.properties aura pour contenu :

```
#MYSQL 5 : driverClassName=com.mysql.jdbc.Driver
#MYSQL 8 : driverClassName=com.mysql.cj.jdbc.Driver
jdbc.driverClassName=com.mysql.cj.jdbc.Driver
#MYSQL 5 : url=jdbc:mysql://localhost:3306/my_data_base?serverTimezone=UTC&useSSL=false
#MYSQL 8 : url=jdbc:mysql://localhost:3308/my_data_base?serverTimezone=UTC
jdbc.url=dbc:mysql://localhost:3308/my_data_base?serverTimezone=UTC
#Identifiant et mot de passe de votre base de donnes
jdbc.username=application
jdbc.password=passw0rd

hibernate.dialect= org.hibernate.dialect.MySQL5Dialect
hibernate.show_sql= true
hibernate.hbm2ddl.auto= update
hibernate.cache.use_second_level_cache = true
hibernate.cache.use_query_cache = true
```

Dans une application Spring type, il existe deux objets qui représentent deux instances de contexte pour l'application; l'une est ApplicationContext, le contexte de la racine de l'application et la seconde est le contexte de l'application en tant que servlet. Le contexte de la racine de l'application contient généralement des ressources partagées/générales comme DataSource, des services, des référentiels, etc. Le contexte de l'application en tant que servlet contient des beans spécifiques à ce contexte, généralement des choses comme le résolveur de vues, les mappages de gestionnaires, les contrôleurs, etc. Le contexte de servlet utilise le contexte racine comme un parent et peut ainsi voir les beans définis là-dedans (la racine ne connaît pas les contextes de servlet!).
Ainsi, **getRootConfigClasses ()** configurera ContextLoaderListener - c'est facultatif (on peut retourner null ou un tableau vide).

L'interface WebMvcConfigurer

La classe **WebMvcConfig** ci-dessous implément l'interface WebMvcConfigurer. Il aura comme contenu :

```java
package com.cours.spring.config;

import org.springframework.context.annotation.Bean;

import org.springframework.context.annotation.ComponentScan;
import org.springframework.context.annotation.Configuration;
import org.springframework.web.servlet.config.annotation.EnableWebMvc;
import org.springframework.web.servlet.config.annotation.ResourceHandlerRegistry;
import org.springframework.web.servlet.config.annotation.WebMvcConfigurer;
import org.springframework.web.servlet.view.InternalResourceViewResolver;
import org.springframework.web.servlet.view.JstlView;


@Configuration
@EnableWebMvc
@ComponentScan(basePackages = { "com.cours.spring"})
public class WebMvcConfig implements WebMvcConfigurer {
   @Bean
   public InternalResourceViewResolver resolver() {
      InternalResourceViewResolver resolver = new InternalResourceViewResolver();
      resolver.setViewClass(JstlView.class);
      resolver.setPrefix("/pages/");
      resolver.setSuffix(".jsp");
      return resolver;
   }

    @Override
    public void addResourceHandlers(ResourceHandlerRegistry registry) {
        registry.addResourceHandler("/resources/**").addResourceLocations("/resources/");
    }

    @Bean
    public MessageSource messageSource() {
      ResourceBundleMessageSource source = new ResourceBundleMessageSource();
      source.setBasename("messages");
      return source;
    }

    @Override
    public Validator getValidator() {
      LocalValidatorFactoryBean validator = new LocalValidatorFactoryBean();
      validator.setValidationMessageSource(messageSource());
      return validator;
    }
}
```

En version copiable :

```
package com.cours.spring.config;

import org.springframework.context.annotation.Bean;

import org.springframework.context.annotation.ComponentScan;
import org.springframework.context.annotation.Configuration;
import org.springframework.web.servlet.config.annotation.EnableWebMvc;
import org.springframework.web.servlet.config.annotation.ResourceHandlerRegistry;
import org.springframework.web.servlet.config.annotation.WebMvcConfigurer;
import org.springframework.web.servlet.view.InternalResourceViewResolver;
import org.springframework.web.servlet.view.JstlView;


@Configuration
@EnableWebMvc
@ComponentScan(basePackages = { "com.cours.spring"})
public class WebMvcConfig implements WebMvcConfigurer {
  @Bean
  public InternalResourceViewResolver resolver() {
    InternalResourceViewResolver resolver = new InternalResourceViewResolver();
    resolver.setViewClass(JstlView.class);
    resolver.setPrefix("/pages/");
    resolver.setSuffix(".jsp");
    return resolver;
  }

   @Override
   public void addResourceHandlers(ResourceHandlerRegistry registry) {
      registry.addResourceHandler("/resources/**").addResourceLocations("/resources/");
   }

  @Bean
  public MessageSource messageSource() {
    ResourceBundleMessageSource source = new ResourceBundleMessageSource();
    source.setBasename("messages");
    return source;
  }

  @Override
  public Validator getValidator() {
    LocalValidatorFactoryBean validator = new LocalValidatorFactoryBean();
    validator.setValidationMessageSource(messageSource());
    return validator;
  }
}
```

Ligne 14 : **@Configuration** indique qu'une classe déclare une ou plusieurs méthodes @Bean et peut être traitée par le conteneur Spring pour générer des définitions de bean et des demandes de service pour ces beans lors de l'exécution.

Ligne 15 : **@EnableWebMvc** active la configuration Spring MVC par défaut et enregistre les composants d'infrastructure Spring MVC attendus par DispatcherServlet.

Ligne 16 : L'annotation **@ComponentScan** est utilisée pour spécifier les packages de base à analyser. Toute classe annotée avec @Component et @Configuration sera analysée.

Ligne 17 : **WebMvcConfigurer** définit des options pour la personnalisation ou l'ajout à la configuration Spring MVC par défaut activée via @EnableWebMvc.

Ligne 19 : InternalResourceViewResolver aide à mapper les noms des vues logiques pour afficher directement les fichiers dans un certain répertoire préconfiguré.

Ligne 34 : ResourceBundleMessageSource accède aux ensembles de ressources à l'aide de noms de base spécifiés (ici, il s'agit de messages).

Ligne 41 : **LocalValidatorFactoryBean** démarre javax.validation.ValidationFactory et l'expose via l'interface Spring Validator, l'interface JSR-303 Validator et l'interface ValidatorFactory elle-même.

Les interfaces BeanFactory et ApplicationContext

L'interface BeanFactory (le container et l'interface porte le même nom) fournit un mécanisme avancé de configuration des beans managés, utilisant potentiellement n'importe quel type de stockage.
L'interface ApplicationContext (le container et l'interface porte le même nom) est construite au dessus de la classe BeanFactory; il y ajoute d'autres dispositifs comme l'intégration plus facile avec l'AOP, la manipulation des messages (pour l'internationalisation), la propagation des événements, les mécanismes déclaratifs pour créer l'ApplicationContext et les contextes parents optionnels, et des contextes spécifiques aux couches des applications comme **WebApplicationContext** qui dérive de l'interface ApplicationContext et qui est spécifique aux applications Web.

VII) Dépendances Maven pour Spring MVC

Spring MVC utilise les dépendances Maven suivantes :

```
<properties>
  <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
  <maven.compiler.source>1.8</maven.compiler.source>
  <maven.compiler.target>1.8</maven.compiler.target>
  <mysql.version>8.0.18</mysql.version>
  <hibernate.version>4.3.11.Final</hibernate.version>
  <jaxb.version>2.0</jaxb.version>
  <spring.version>5.1.5.RELEASE</spring.version>
  <spring.data.jpa.version>2.1.5.RELEASE</spring.data.jpa.version>
  <javax.servlet.version>4.0.0</javax.servlet.version>
  <javax.servlet.jsp.version>2.3.0</javax.servlet.jsp.version>
  <jstl.version>1.2</jstl.version>
  <validation.api.version>1.1.0.Final</validation.api.version>
  <hibernate.validator.version>5.3.1.Final</hibernate.validator.version>
</properties>

<dependencies>
  <!-- Debut Spring dependencies -->
  <dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-core</artifactId>
    <version>${spring.version}</version>
  </dependency>
  <dependency>
     <groupId>org.springframework</groupId>
     <artifactId>spring-context</artifactId>
     <version>${spring.version}</version>
  </dependency>
  <dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-web</artifactId>
    <version>${spring.version}</version>
  </dependency>
  <dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-orm</artifactId>
    <version>${spring.version}</version>
  </dependency>
  <dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-webmvc</artifactId>
    <version>${spring.version}</version>
  </dependency>
  <!-- Fin Spring dependencies -->
  <!-- Debut Spring Data JPA dependencies -->
  <dependency>
```

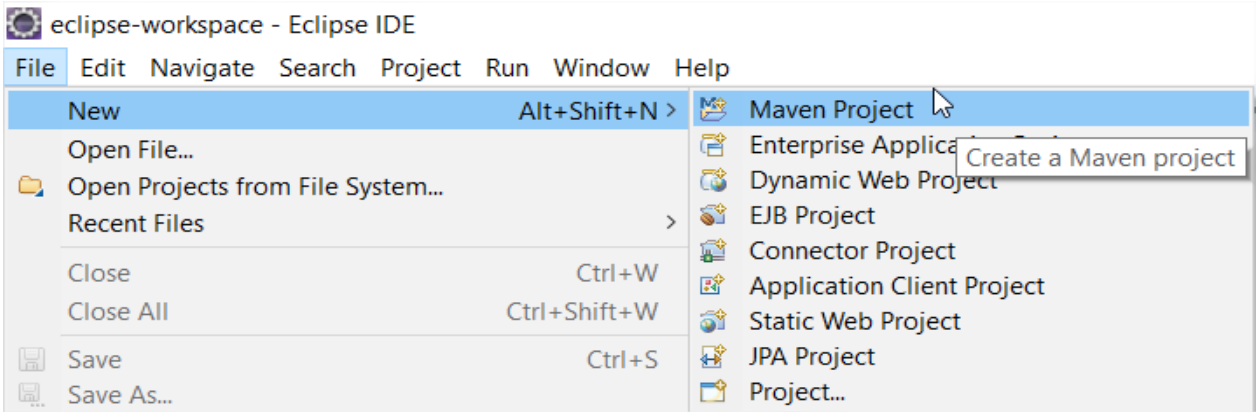
```
    <groupId>org.springframework.data</groupId>
    <artifactId>spring-data-jpa</artifactId>
    <version>${spring.data.jpa.version}</version>
  </dependency>
  <!-- Fin Spring Data JPA dependencies -->
  <!-- Hibernate dependencies -->
  <dependency>
    <groupId>org.hibernate</groupId>
    <artifactId>hibernate-core</artifactId>
    <version>${hibernate.version}</version>
  </dependency>
  <dependency>
    <groupId>org.hibernate</groupId>
    <artifactId>hibernate-entitymanager</artifactId>
    <version>${hibernate.version}</version>
  </dependency>
  <dependency>
    <groupId>javax.xml.bind</groupId>
    <artifactId>jaxb-api</artifactId>
    <version>${jaxb.version}</version>
  </dependency>
  <dependency>
    <groupId>com.sun.xml.bind</groupId>
    <artifactId>jaxb-impl</artifactId>
    <version>${jaxb.version}</version>
  </dependency>
  <!-- Fin Hibernate dependencies -->

  <!-- Debut mysql dependencies -->
   <dependency>
    <groupId>mysql</groupId>
    <artifactId>mysql-connector-java</artifactId>
    <version>${mysql.version}</version>
  </dependency>
  <!-- Fin mysql dependencies -->

  <!-- JSR 303 : Bean validation -->
  <dependency>
    <groupId>javax.validation</groupId>
    <artifactId>validation-api</artifactId>
    <version>${validation.api.version}</version>
  </dependency>

  <!-- Bean validation : implémentation -->
  <dependency>
    <groupId>org.hibernate</groupId>
    <artifactId>hibernate-validator</artifactId>
    <version>${hibernate.validator.version}</version>
  </dependency>
  <!-- Servlet dependencies -->
  <dependency>
```

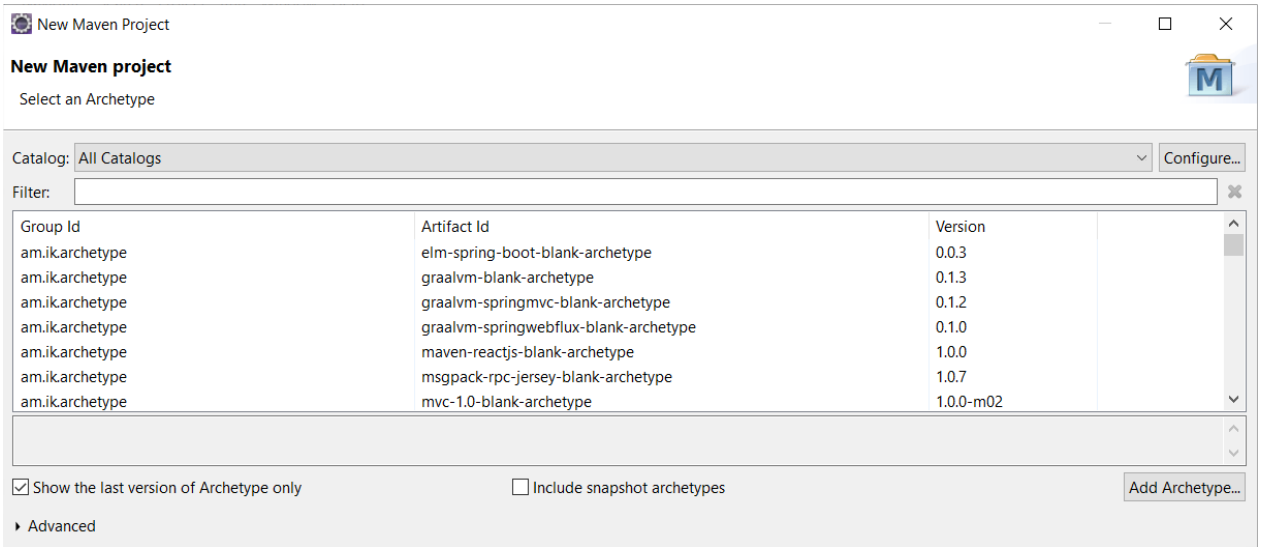
```
    <groupId>javax.servlet</groupId>
    <artifactId>javax.servlet-api</artifactId>
    <version>${javax.servlet.version}</version>
    <scope>provided</scope>
  </dependency>
  <dependency>
    <groupId>javax.servlet.jsp</groupId>
    <artifactId>javax.servlet.jsp-api</artifactId>
    <version>${javax.servlet.jsp.version}</version>
    <scope>provided</scope>
  </dependency>
  <dependency>
    <groupId>jstl</groupId>
    <artifactId>jstl</artifactId>
    <version>${jstl.version}</version>
  </dependency>
</dependencies>
```

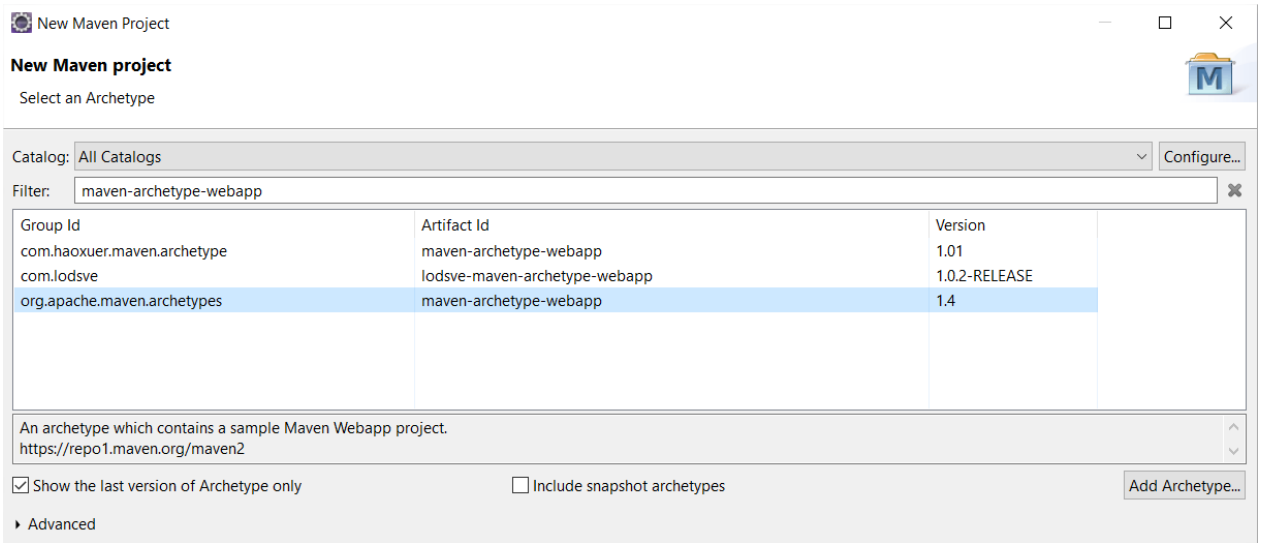
VIII) Comment créer un Projet Maven Spring MVC

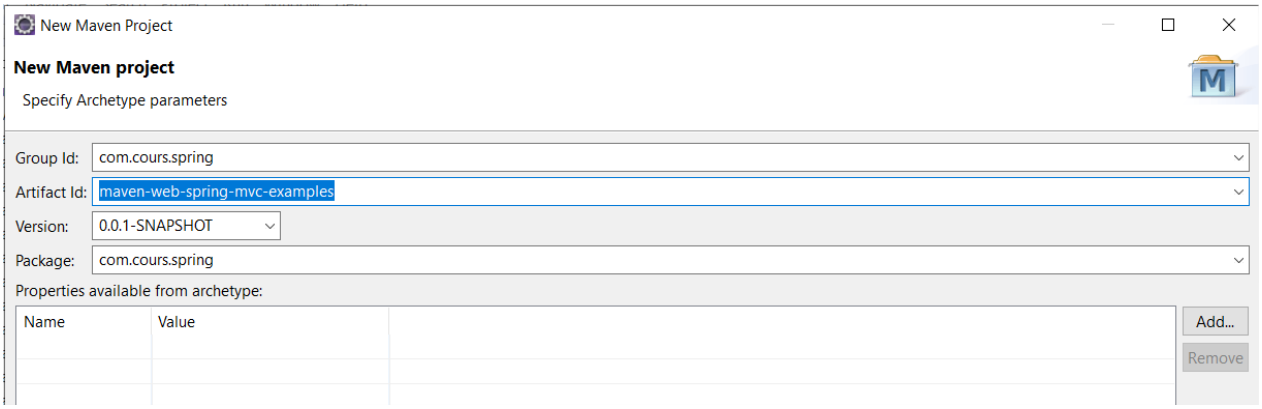
Créer le projet Maven **maven-web-spring-mvc-examples** de type **war** (archetype=maven-archetype-webapp).

1. *Création du projet Maven*



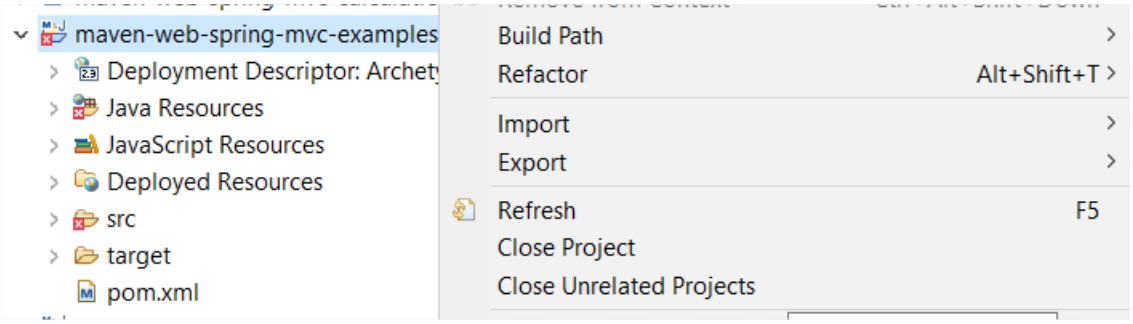


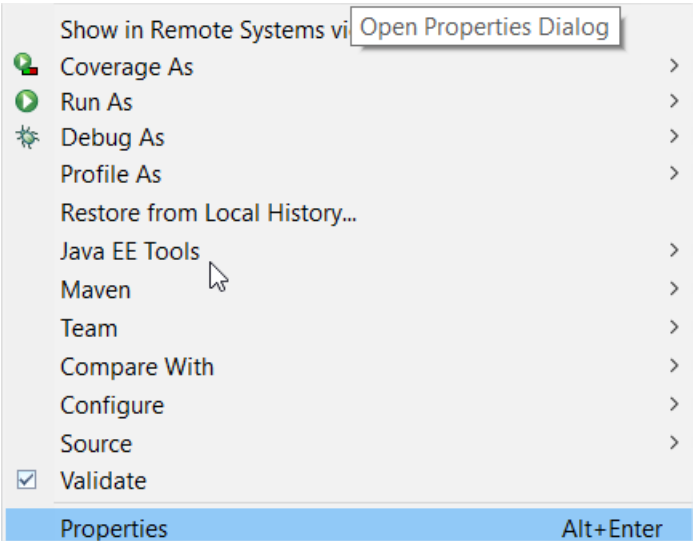

New Maven Project
New Maven project
Select an Archetype
Catalog: All Catalogs
Configure...
Filter: maven-archetype-webapp
Group Id
Artifact Id
Version
com.haoxuer.maven.archetype
maven-archetype-webapp
1.01
com.lodsve
lodsve-maven-archetype-webapp
1.0.2-RELEASE
org.apache.maven.archetypes
maven-archetype-webapp
1.4
An archetype which contains a sample Maven Webapp project.
https://repo1.maven.org/maven2
Show the last version of Archetype only
Include snapshot archetypes
Add Archetype...
Advanced

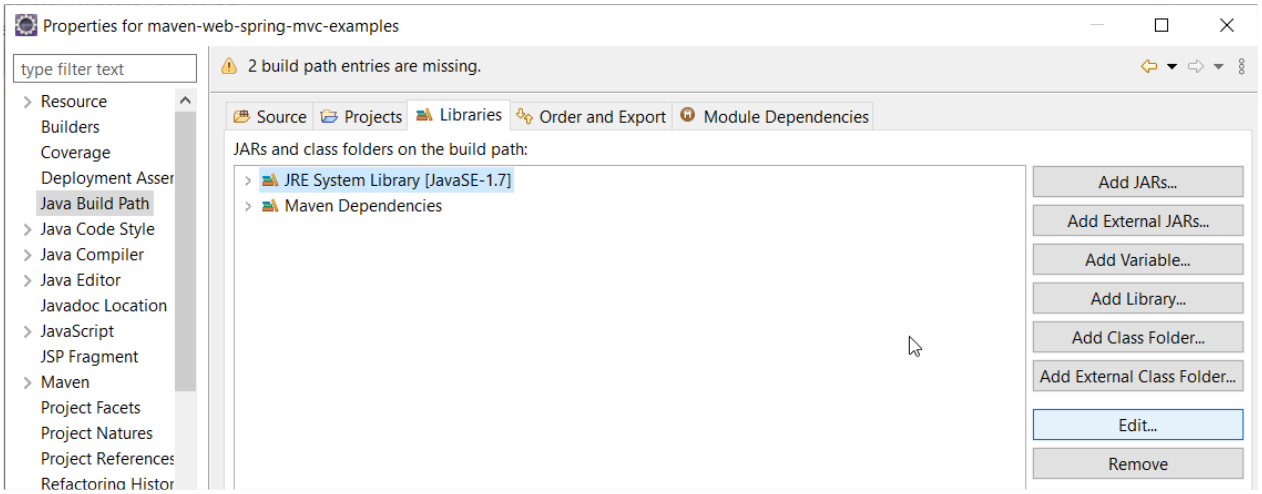

New Maven Project
New Maven project
Specify Archetype parameters
Group Id: com.cours.spring
Artifact Id: maven-web-spring-mvc-examples
Version: 0.0.1-SNAPSHOT
Package: com.cours.spring
Properties available from archetype:
Name
Value
Add...
Remove


maven-web-spring-mvc-examples
Deployment Descriptor: Archetype Created Web Application
Java Resources
JavaScript Resources
Deployed Resources
src
main
java
resources
webapp
test
target
pom.xml

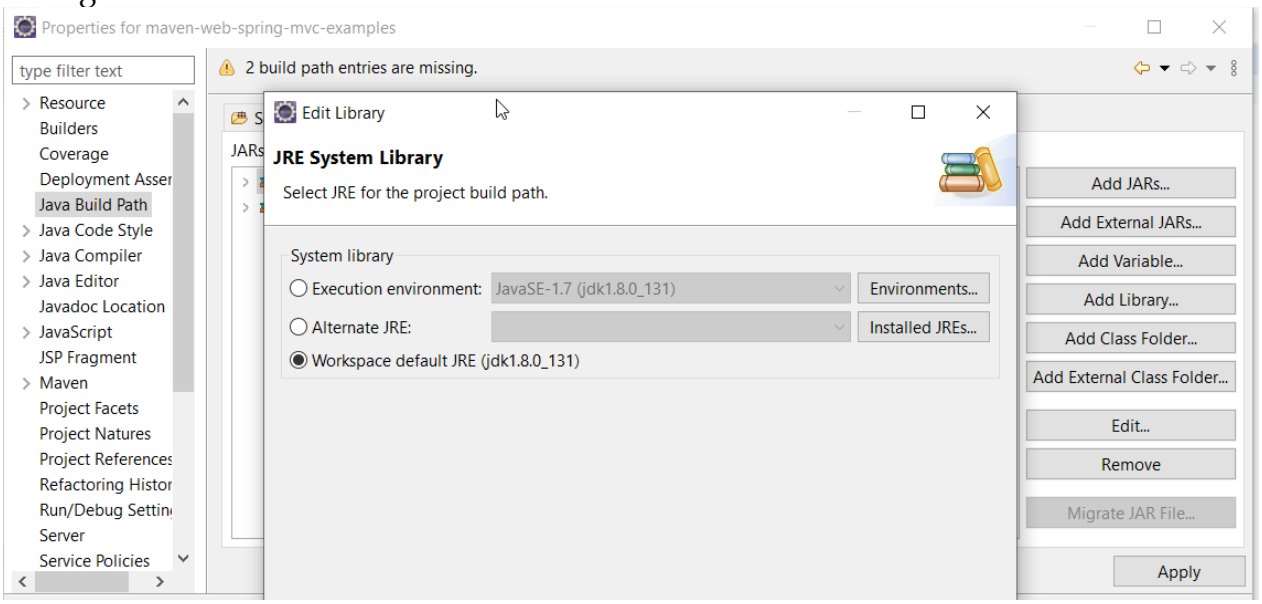
2. *Verification de la version de Java*

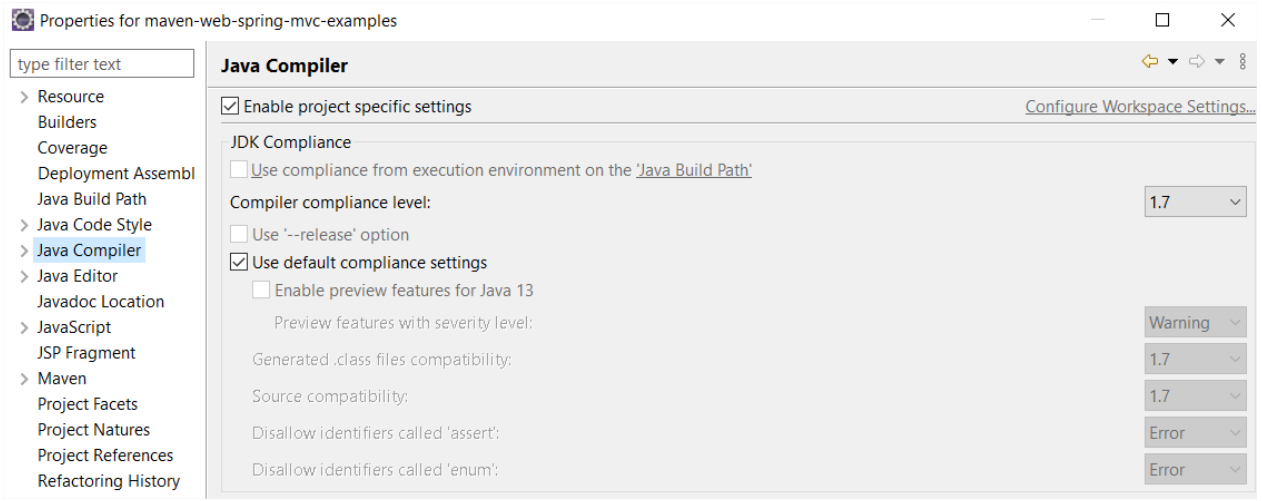


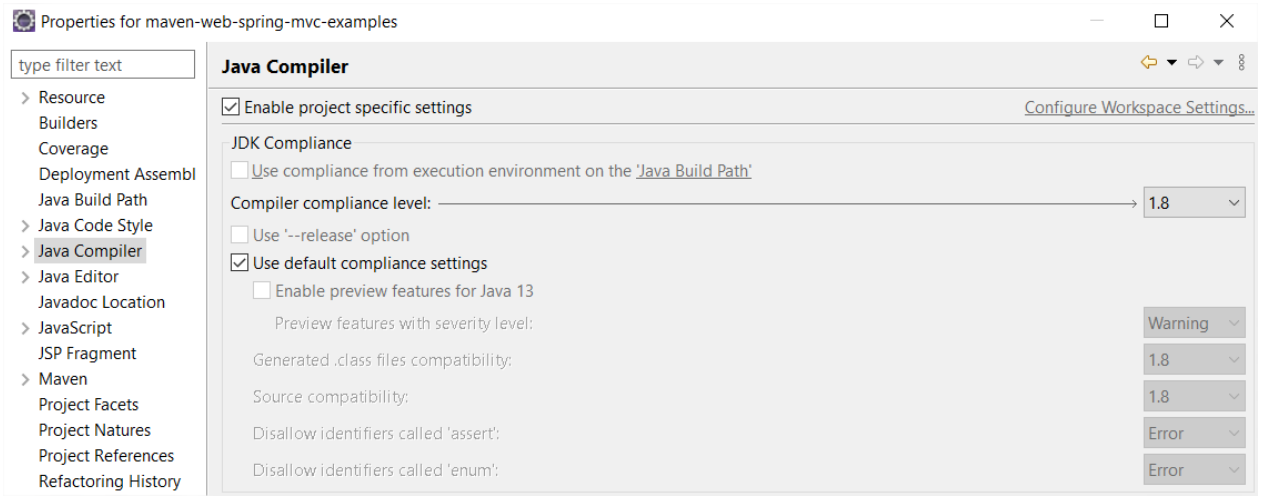




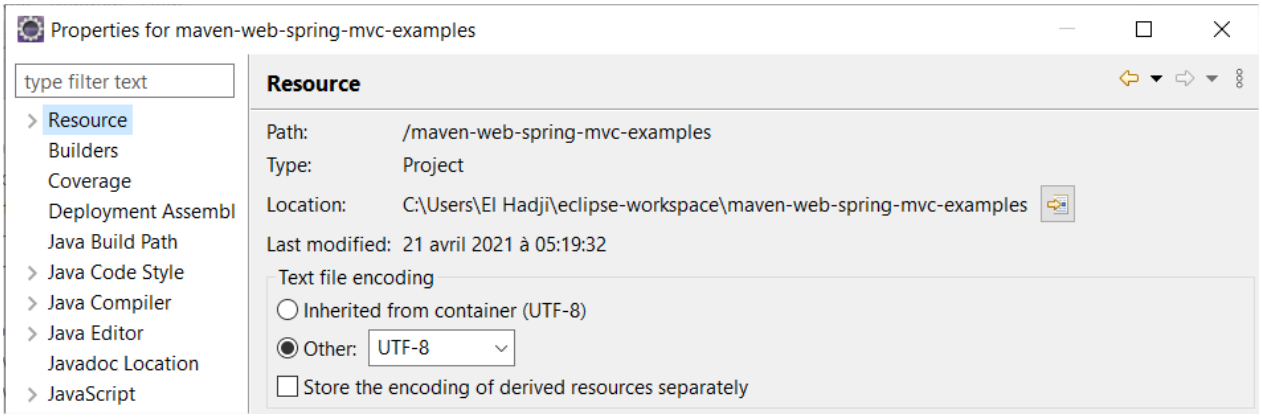
Changer la version du Build Path de **1.7 à 1.8**.



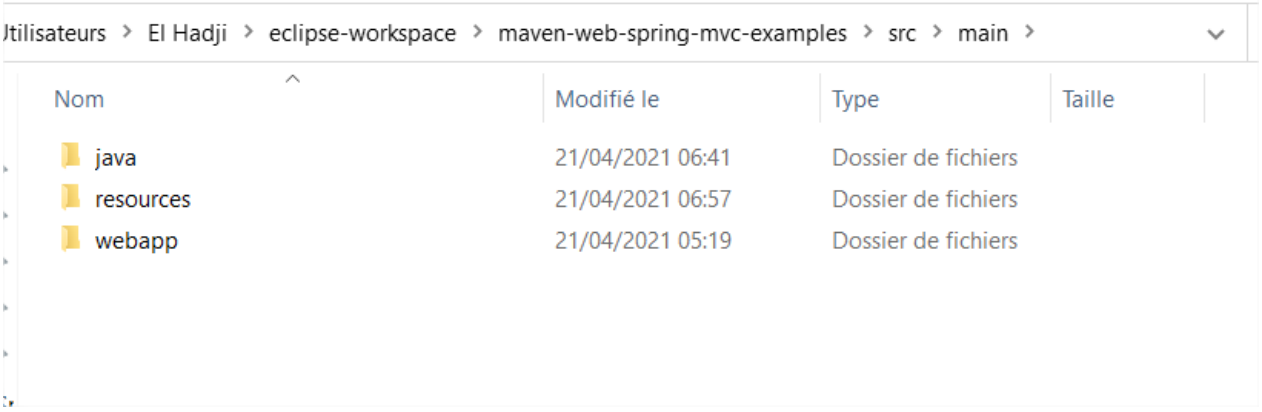


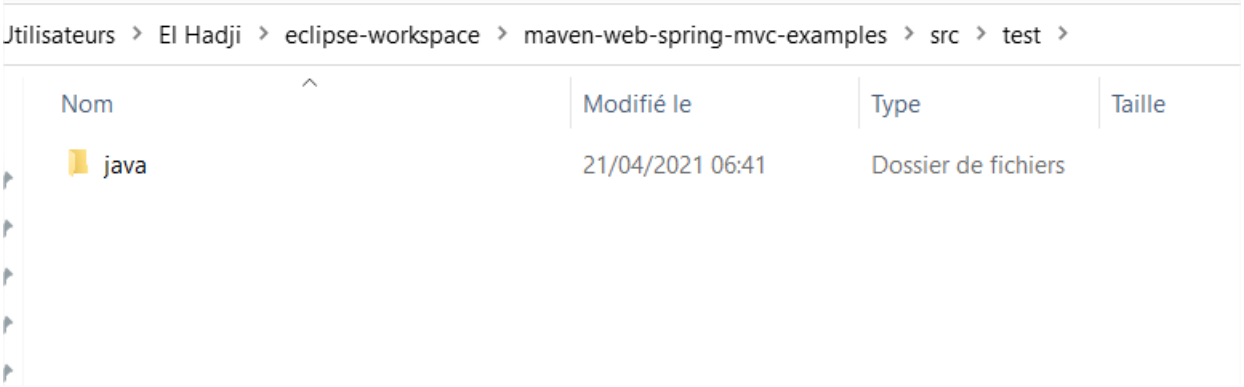


3. *Création des dossiers src/main/resources et src/test/resources*

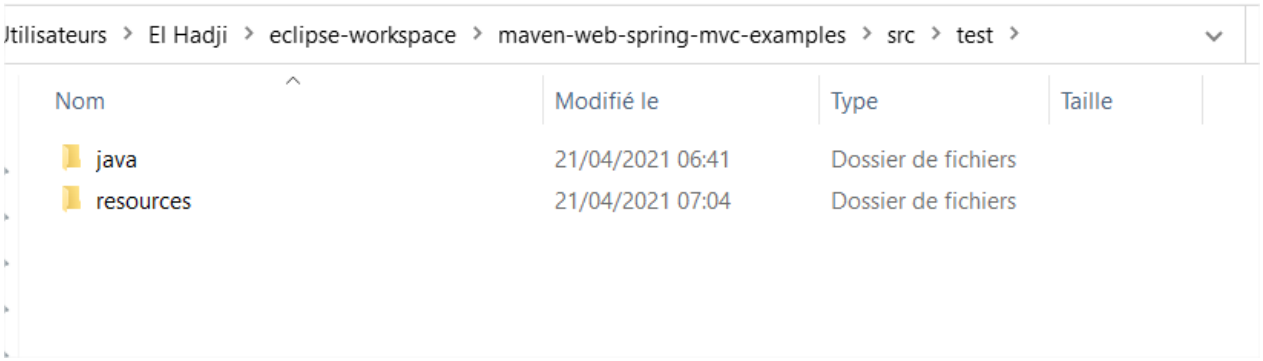


Créer le dossier **src/main/resources** :





Créer le dossier **src/test/resources** :



Copier le fichier **log4j.properties** dans **/src/main/resources** dont le contenu sera :

```
# Set root logger level to DEBUG and its only appender to CONSOLE.
log4j.rootLogger=DEBUG, CONSOLE

# A1 is set to be a ConsoleAppender.
log4j.appender.CONSOLE=org.apache.log4j.ConsoleAppender
log4j.appender.CONSOLE.encoding=UTF-8

# A1 uses PatternLayout.
log4j.appender.CONSOLE.layout=org.apache.log4j.PatternLayout
log4j.appender.CONSOLE.layout.ConversionPattern=[%d{dd/MM/yyyy HH:mm:ss} %p %C{1}.%M] %m%n

# Change the level of messages for various packages.
log4j.logger.com.cours*=DEBUG
```

4. *Le pom.xml*

Le fichier **pom.xml** aura pour contenu initiale :

```
<?xml version="1.0" encoding="UTF-8"?>

<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance"
 xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
 <modelVersion>4.0.0</modelVersion>

 <groupId>com.cours.spring</groupId>
 <artifactId>maven-web-spring-mvc-examples</artifactId>
 <version>0.0.1-SNAPSHOT</version>
 <packaging>war</packaging>

 <name>maven-web-spring-mvc-examples Maven Webapp</name>
 <!-- FIXME change it to the project's website -->
 <url>http://www.example.com</url>

 <properties>
  <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
  <maven.compiler.source>1.7</maven.compiler.source>
  <maven.compiler.target>1.7</maven.compiler.target>
 </properties>

 <dependencies>
  <dependency>
   <groupId>junit</groupId>
   <artifactId>junit</artifactId>
   <version>4.11</version>
   <scope>test</scope>
  </dependency>
 </dependencies>

 <build>
  <finalName>maven-web-spring-mvc-examples</finalName>
  <pluginManagement><!--
 lock down plugins versions to avoid using Maven defaults (may be moved to parent pom) -->
   <plugins>
    <plugin>
     <artifactId>maven-clean-plugin</artifactId>
     <version>3.1.0</version>
    </plugin>
    <!-- see http://maven.apache.org/ref/current/maven-core/default-
bindings.html#Plugin_bindings_for_war_packaging -->
    <plugin>
     <artifactId>maven-resources-plugin</artifactId>
     <version>3.0.2</version>
    </plugin>
```

```
      <plugin>
        <artifactId>maven-compiler-plugin</artifactId>
        <version>3.8.0</version>
      </plugin>
      <plugin>
        <artifactId>maven-surefire-plugin</artifactId>
        <version>2.22.1</version>
      </plugin>
      <plugin>
        <artifactId>maven-war-plugin</artifactId>
        <version>3.2.2</version>
      </plugin>
      <plugin>
        <artifactId>maven-install-plugin</artifactId>
        <version>2.5.2</version>
      </plugin>
      <plugin>
        <artifactId>maven-deploy-plugin</artifactId>
        <version>2.8.2</version>
      </plugin>
    </plugins>
  </pluginManagement>
 </build>
</project>
```

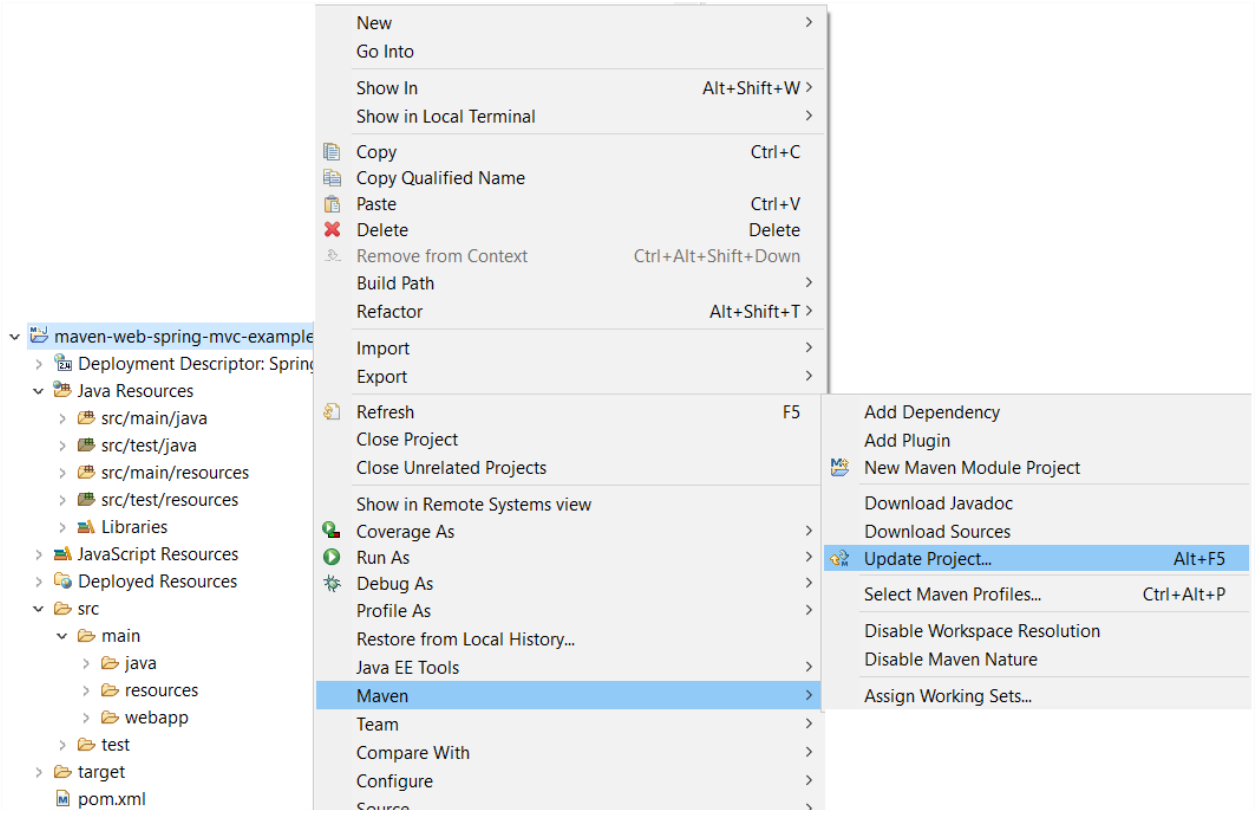
Le fichier **pom.xml** peut devenir :

```
<?xml version="1.0" encoding="UTF-8"?>

<project xmlns="http://maven.apache.org/POM/4.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <groupId>com.cours.spring</groupId>
  <artifactId>maven-web-spring-mvc-examples</artifactId>
  <version>0.0.1-SNAPSHOT</version>
  <packaging>war</packaging>

  <name>maven-web-spring-mvc-examples</name>
  <properties>
    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
    <maven.compiler.source>1.8</maven.compiler.source>
    <maven.compiler.target>1.8</maven.compiler.target>
    <spring.version>5.0.3.RELEASE</spring.version>
    <javax.servlet.version>4.0.0</javax.servlet.version>
    <javax.servlet.jsp.version>2.3.0</javax.servlet.jsp.version>
    <jstl.version>1.2</jstl.version>
    <log4j.version>1.2.17</log4j.version>
    <commons.logging.version>1.2</commons.logging.version>
    <validation.api.version>1.1.0.Final</validation.api.version>
    <hibernate.validator.version>5.3.1.Final</hibernate.validator.version>
  </properties>

  <dependencies>
    <!-- Spring -->
    <dependency>
      <groupId>org.springframework</groupId>
      <artifactId>spring-context</artifactId>
      <version>${spring.version}</version>
    </dependency>
    <dependency>
      <groupId>org.springframework</groupId>
      <artifactId>spring-webmvc</artifactId>
      <version>${spring.version}</version>
    </dependency>

    <!-- JSR 303 : Bean validation -->
    <dependency>
      <groupId>javax.validation</groupId>
      <artifactId>validation-api</artifactId>
      <version>${validation.api.version}</version>
    </dependency>

    <!-- Bean validation : implémentation -->
    <dependency>
```

```
      <groupId>org.hibernate</groupId>
      <artifactId>hibernate-validator</artifactId>
      <version>${hibernate.validator.version}</version>
    </dependency>
    <dependency>
      <groupId>javax.servlet</groupId>
      <artifactId>javax.servlet-api</artifactId>
      <version>${javax.servlet.version}</version>
      <scope>provided</scope>
    </dependency>
    <dependency>
      <groupId>javax.servlet.jsp</groupId>
      <artifactId>javax.servlet.jsp-api</artifactId>
      <version>${javax.servlet.jsp.version}</version>
      <scope>provided</scope>
    </dependency>
    <dependency>
      <groupId>jstl</groupId>
      <artifactId>jstl</artifactId>
      <version>${jstl.version}</version>
    </dependency>
    <!-- Debut log4j dependencies -->
    <dependency>
      <groupId>log4j</groupId>
      <artifactId>log4j</artifactId>
      <version>${log4j.version}</version>
    </dependency>
    <dependency>
      <groupId>log4j</groupId>
      <artifactId>apache-log4j-extras</artifactId>
      <version>${log4j.version}</version>
    </dependency>
    <dependency>
      <groupId>commons-logging</groupId>
      <artifactId>commons-logging</artifactId>
      <version>${commons.logging.version}</version>
    </dependency>
    <!-- Fin log4j dependencies -->
  </dependencies>
  <build>
    <finalName>maven-web-spring-mvc-examples</finalName>
  </build>
</project>
```

5. *Mise à jour du projet*

Faire un **Maven →Update Project** de votre projet.



6. *Le fichier webapp/index.jsp*

Le fichier **index.jsp** aura pour contenu :

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
   pageEncoding="UTF-8"%>
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c"%>
<html>
   <head>
      <title>Page d'exemples pour la formation Spring MVC</title>
      <meta charset="UTF-8">
      <meta name="viewport" content="width=device-width, initial-scale=1.0">
   </head>
   <body>
      <h1>Page d'exemples pour la formation Spring MVC</h1>
   </body>
</html>
```

7. *Le web.xml*

Le fichier **web.xml** aura pour contenu :

```
<!DOCTYPE web-app PUBLIC
 "-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"
 "http://java.sun.com/dtd/web-app_2_3.dtd" >

<web-app>
  <display-name>Archetype Created Web Application</display-name>
</web-app>
```

Le fichier **web.xml** peut devenir :

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns="http://xmlns.jcp.org/xml/ns/javaee"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee http://xmlns.jcp.org/xml/ns/javaee/web-
app_3_1.xsd"
        version="3.1">
    <display-name>Spring MVC Examples</display-name>
    <servlet>
        <servlet-name>dispatcher</servlet-name>
        <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
        <load-on-startup>1</load-on-startup>
    </servlet>
    <servlet-mapping>
        <servlet-name>dispatcher</servlet-name>
        <url-pattern>/</url-pattern>
    </servlet-mapping>
    <context-param>
        <param-name>contextConfigLocation</param-name>
        <param-value>/WEB-INF/dispatcher-servlet.xml</param-value>
    </context-param>
    <listener>
        <listener-class>org.springframework.web.context.ContextLoaderListener</listener-class>
    </listener>
    <session-config>
        <session-timeout>
            30
        </session-timeout>
    </session-config>
    <welcome-file-list>
        <welcome-file>index.jsp</welcome-file>
    </welcome-file-list>
</web-app>
```

Le fichier **dispatcher-servlet.xml** aura comme contenu :

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

  <!-- -->
  <context:component-scan base-package="com.cours.spring" />
  <mvc:resources mapping="/resources/**" location="/resources/"/>
  <mvc:annotation-driven validator="validator"/>
  <bean class="org.springframework.web.servlet.view.InternalResourceViewResolver">
    <property name="prefix">
      <value>/pages/</value>
    </property>
    <property name="suffix">
      <value>.jsp</value>
    </property>
  </bean>


  <bean id="messageSource" class="org.springframework.context.support.ResourceBundleMessageSource">
    <property name="basenames">
      <list>
        <value>messages</value>
      </list>
    </property>
    <property name="defaultEncoding" value="UTF-8" />
    <property name="alwaysUseMessageFormat" value="true" />
  </bean>

  <bean id="validator" class="org.springframework.validation.beanvalidation.LocalValidatorFactoryBean">
    <property name="validationMessageSource" ref="messageSource"/>
  </bean>
</beans>
```

8. *Le fichier webapp/pages/myPage.jsp*

Le fichier **webapp/pages/myPage.jsp** aura pour contenu :

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
   pageEncoding="ISO-8859-1"%>
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c"%>
<%@ taglib uri="http://www.springframework.org/tags/form" prefix="form"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
   "http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
  <link href="${pageContext.request.contextPath}/resources/assets/css/style.css" rel="stylesheet" >
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<title>Spring MVC Form</title>
</head>
<body>
   <div align="center">
      <h2>Spring MVC Form</h2>
       <c:if test="${not empty errorMessage}">
         <div id="errorMessage" style="color: red">${errorMessage}</div>
       </c:if>
      <form:form action="${pageContext.request.contextPath}/myPath/save" method="post" modelAttribute="my
Model">
         <table border="0" cellpadding="5">
            <tr>
               <td>Value1: </td>
               <td><form:input path="value1" /></td>
               <td><form:errors cssClass="error" path="value1" /></td>
            </tr>
            <tr>
               <td>value2: </td>
               <td><form:input path="value2" /></td>
               <td><form:errors cssClass="error" path="value2" /></td>
            </tr>
            <tr>
               <td colspan="2"><input type="submit" value="Save"></td>
            </tr>
         </table>
      </form:form>
   </div>
</body>
</html>
```

9. *La classe MyModel*

La classe **MyModel** aura pour contenu :

```
package com.cours.entities;

//import javax.validation.constraints.Size;

public class MyModel {

   //@Size(min = 3, max = 10)
   private String value1;
   //@Size(min = 3, max = 10)
   private String value2;

   public MyModel() {

   }

   @Override
   public String toString() {
      return "MyModel [value1=" + value1 + ", value2=" + value2 + "]";
   }

   public MyModel(String value1, String value2) {
      this.value1 = value1;
      this.value2 = value2;
   }

   public String getValue1() {
      return value1;
   }

   public void setValue1(String value1) {
      this.value1 = value1;
   }

   public String getValue2() {
      return value2;
   }

   public void setValue2(String value2) {
      this.value2 = value2;
   }
}
```

10.La classe MyModelValidator

La classe **MyModelValidator** aura pour contenu :

```
package com.cours.spring.validators;

import org.springframework.stereotype.Component;
import org.springframework.validation.Errors;
import org.springframework.validation.Validator;

import com.cours.entities.MyModel;

@Component
public class MyModelValidator implements Validator {

   @Override
   public boolean supports(Class<?> clazz) {
      return MyModel.class.isAssignableFrom(clazz);
   }

   @Override
   public void validate(Object target, Errors errors) {
      if(target instanceof MyModel) {
         MyModel myModel = (MyModel) target;
         if ((myModel.getValue1() == null || "".equals(myModel.getValue1()))) {
            errors.rejectValue("value1", "data.error.value1");
         }
         if ((myModel.getValue2() == null || "".equals(myModel.getValue2()))) {
            errors.rejectValue("value2", "data.error.value2");
         }
      }
   }
}
```

11.Le fichier messages.properties

Le fichier **messages.properties** aura pour contenu :

```
data.error.value1=Vos donn\u00e9es dans value1 sont incorrectes
data.error.value2=Vos donn\u00e9es dans value2 sont incorrectes
```

12.La classe MyController

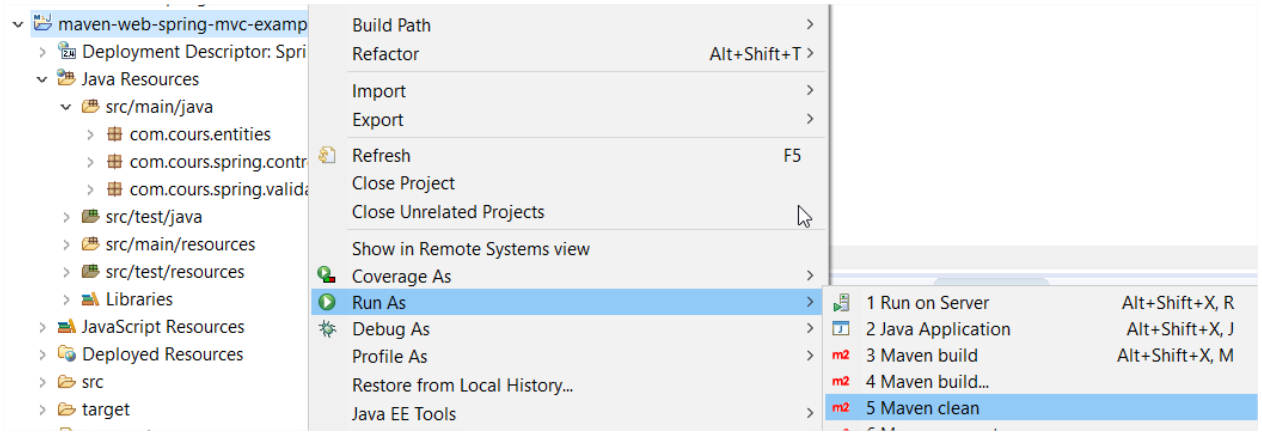
La classe **MyController** aura pour contenu :

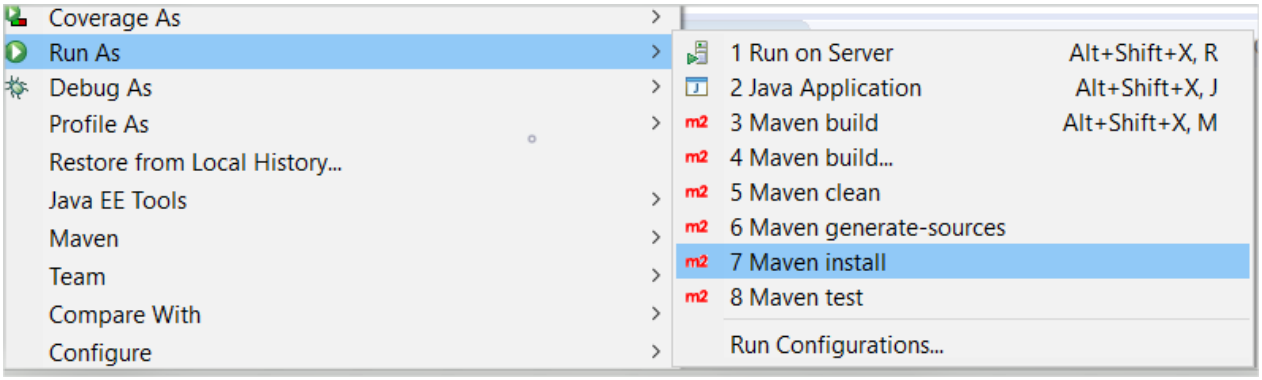
```
package com.cours.spring.controller;

import javax.validation.Valid;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.validation.BindingResult;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.ModelAttribute;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;
import org.springframework.web.bind.annotation.RequestParam;

import com.cours.entities.MyModel;
import com.cours.spring.validators.MyModelValidator;

@Controller
@RequestMapping("/myPath")
public class MyController {


   @Autowired
   private MyModelValidator myModelValidator;

   // @RequestMapping("/myMethod1")
   // @RequestMapping(value = "/myMethod1", method = RequestMethod.POST)
   // @RequestMapping(value = "/myMethod1", params = {
   // "param1=valueParam1","param2=valueParam2" })
   // @RequestMapping(value = "/myMethod1", headers = "Accept=text/html")
   @GetMapping("/myMethod1")
   public String myMethod1(Model model) {
      model.addAttribute("myKey", "myValue");
      return "myPage";
   }

   @RequestMapping("/myMethod2")
   public String myMethod2(@RequestParam("myParam") String myParam, Model model) {
      model.addAttribute("myKey", myParam);
      return "myPage";
   }

   @RequestMapping("/myMethod3/{id}")
   public String myMethod3(@PathVariable("id") Long id, Model model) {
      model.addAttribute("myKey", id);
      return "myPage";
```

```
    }

    @RequestMapping("/myMethod4/{id}")
    // Melange de PathVariable et RequestParam
    public String myMethod4(@RequestParam("myParam") String myParam, @PathVariable("id") Long id, Model
model) {
        model.addAttribute("myKey", "id=" + id + " et myParam=" + myParam);
        return "myPage";
    }

    @RequestMapping("/create")
    public String create(Model model) {
        MyModel myModel = new MyModel("myValue1","myValue2");
        model.addAttribute("myModel", myModel);
        return "myFormPage";
    }


    @RequestMapping(value = "/save", method = RequestMethod.POST)
    public String save(@Valid @ModelAttribute("myModel") MyModel myModel,BindingResult result,Model mode
l) {
        System.out.println("myModel : " + myModel);
        myModelValidator.validate(myModel, result);
        if (result.hasErrors()) {
            return "myFormPage";
        }
        return "redirect:/";
    }

    @RequestMapping(value = "/saveV2", method = RequestMethod.POST)
    public String saveV2(@Valid @ModelAttribute("myModel") MyModel myModel,BindingResult result,Model m
odel) {
        System.out.println("myModel : " + myModel);
        if (result.hasErrors()) {
            model.addAttribute("errorMessage", "DONNEES INCORECTS");
            return "myFormPage";
        }
        return "redirect:/";
    }

    @RequestMapping(value = "/saveV1", method = RequestMethod.POST)
    public String saveV1(@ModelAttribute("myModel") MyModel myModel) {
        System.out.println("myModel : " + myModel);
        return "redirect:/";
    }
}
```

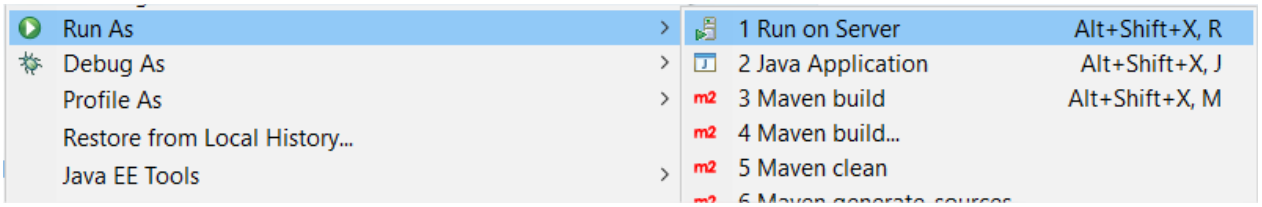
13.Lancement de l'application

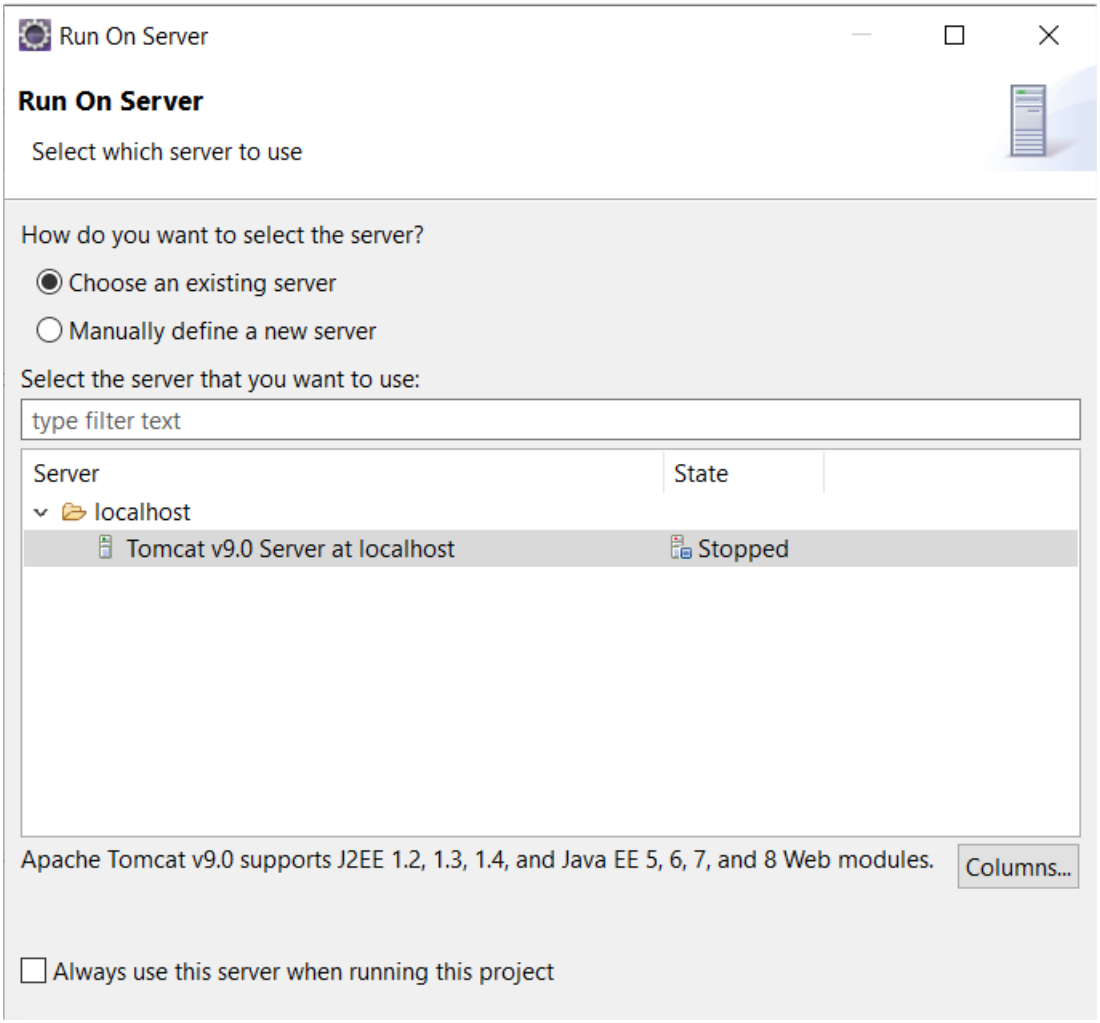
Lancez d'abord un **Maven clean** puis un **Maven install.**

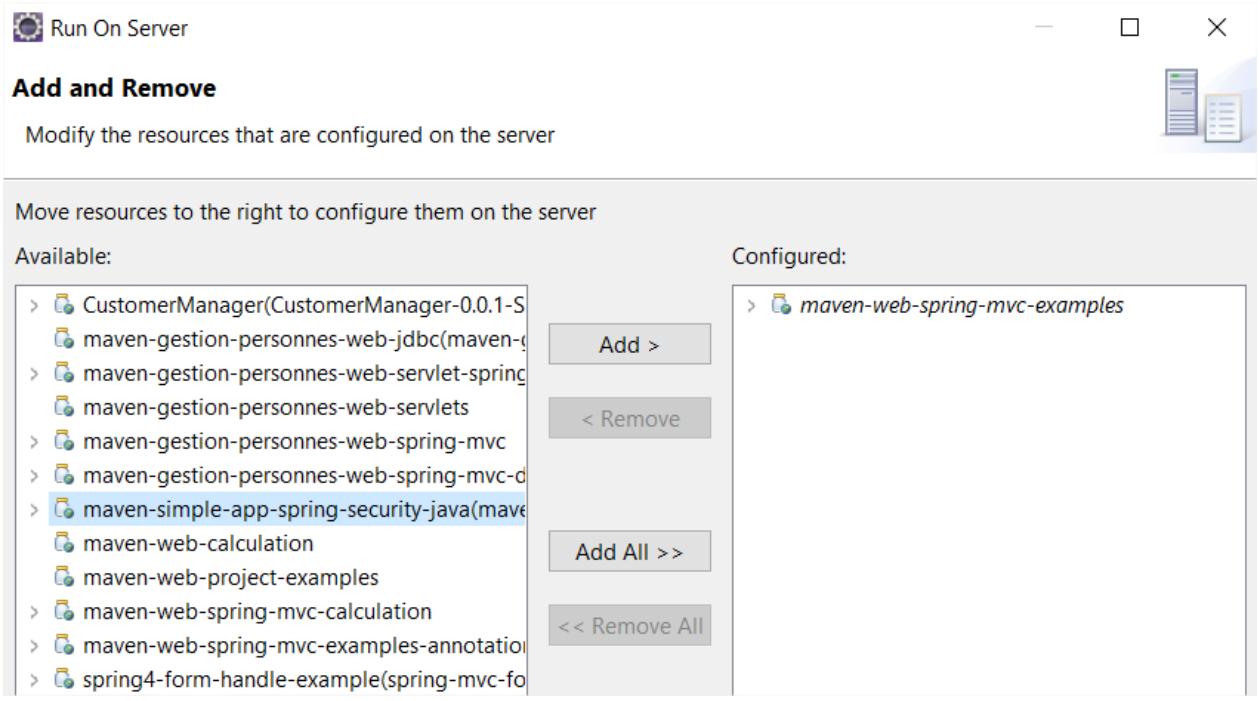




Lancez un **Run on Server.**







On obtient sur http://localhost:8081/maven-web-spring-mvc-examples/:

Tester aussi l'URL http://localhost:8081/maven-web-spring-mvc-examples/myPath/myMethod1