

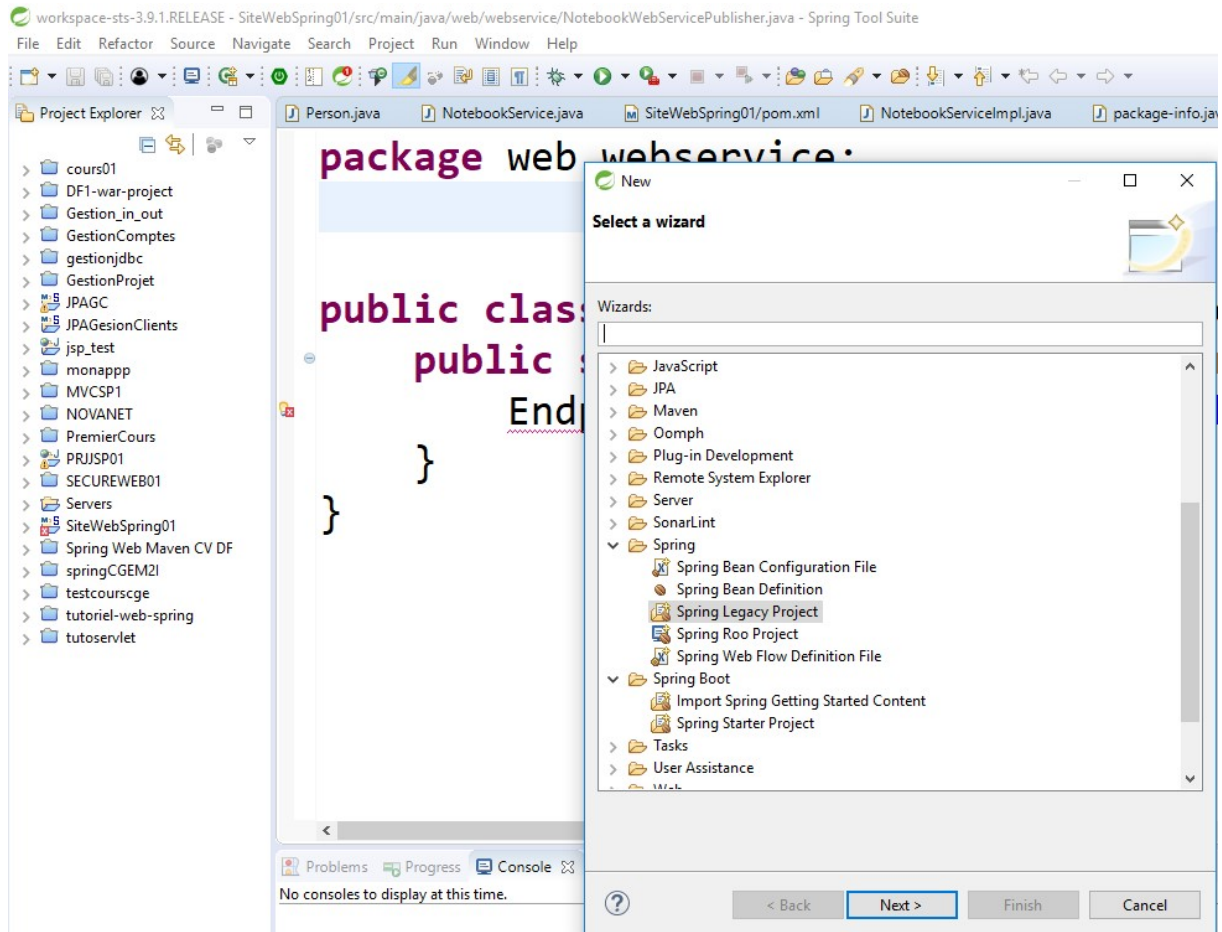
Tutoriel sur Spring sous STS

Sommaire

I.	Créer un nouveau projet :	3
II.	Création d'une JSP « bonjour.jsp »	5
III.	Utilisation de Spring avec la JSP « bonjour.jsp »	6
IV.	Passage de donnée à la JSP « bonjour.jsp »	11
V.	Récupération d'un paramètre de la requête http	15
VI.	Création de la base	20
VII.	Paramétrage du serveur Tomcat	28
VIII.	Modification du projet afin d'inclure l'affichage des données	29
IX.	Développement :	33
A.	Présentation :	33
B.	Création de la classe « d'entity » :	33
C.	Gestion des interfaces « métiers » :	35
D.	Création des classes « Métiers » implémentant les intf. « Métiers » : .	36
E.	Création des controllers :	38
F.	Gestion des « propriétés » :	39
G.	Gestions des pages JSP :	40
X.	Création de données en base (CRUD)	42
A.	Gestion des « propriétés » :	42
B.	Création de classes java utiles pour les controller :	43
C.	Mise à jour des fichiers java suivants :	44
D.	Création du Controller de mise à jour :	45
E.	Gestion de la page de formulaire (JSP) :	48
XI.	Suppression de données en base	52
A.	Objectifs :	52
B.	Gestion des fichiers « propriétés » :	52
C.	Modification des fichiers java suivants :	52

D.	Création du controller de gestion de suppression :.....	54
E.	Création la page JSP pour la suppression :	55
XII.	Modification des données en base :	58
A.	Objectifs :.....	58
B.	Gestion des « properties » :	58
C.	Mise à jour des fichiers java suivants :	58
D.	Gestion de classe utilitaire :	60
E.	Création du Controller de modification :.....	63
F.	Gestion de la page JSP pour la modification :.....	65
XIII.	Unification de l'application par un menu.....	70
A.	Gestion des fichiers « properties » :.....	70
B.	Mise à jour des fichiers XML :	71
C.	Gestion des pages JSP :	72

I. Créer un nouveau projet :



1. Créer un nouveau projet : Spring Legacy Project
2. Cliquer sur : Next

New Spring Legacy Project

Spring Legacy Project

Create a Spring project by selecting a template or simple project type.

Project name:

tutorial-web-spring-m2i

☒ Use default location

Location: C:\Users\Moi\Documents\workspace-sts-3.9.1.RELEASE\tuti

Browse...

Select Spring version:

Default

Templates:

Simple Projects

Simple Java

Simple Spring Maven

Simple Spring Web Maven

> Batch

> GemFire

> Integration

> Persistence

Simple Spring Utility Project

Spring MVC Project

requires downloading

[Configure templates...](#)

Refresh

Description:

Creates a simple Spring web project using Maven that contains a basic set of Spring web libraries.

Working sets

☐ Add project to working sets

New...

Working sets:

Select...

?

< Back

Next >

Finish

Cancel

4

II. Création d'une JSP « **bonjour.jsp** »

Nous allons créer une JSP simple sans utilisation de Spring.

Supprimer le fichier « **index.jsp** » contenu dans le dossier «**src/main/webapp**».

Cette étape consiste simplement à ajouter une JSP dans le projet Web.

Pour le moment, il n'y a aucun lien avec Spring. Créer un dossier « vues » dans le dossier « **src/main/webapp** ».

Créer un fichier « **bonjour.jsp** » (avec le contenu ci-dessous) dans ce nouveau dossier. **/vues/bonjour.jsp**

Bonjour le monde.

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1" pageEncoding="ISO-8859-1"%>
```

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
```

```
<html>
```

```
    <head>
```

```
        <title>Bonjour</title>
```

```
    </head>
```

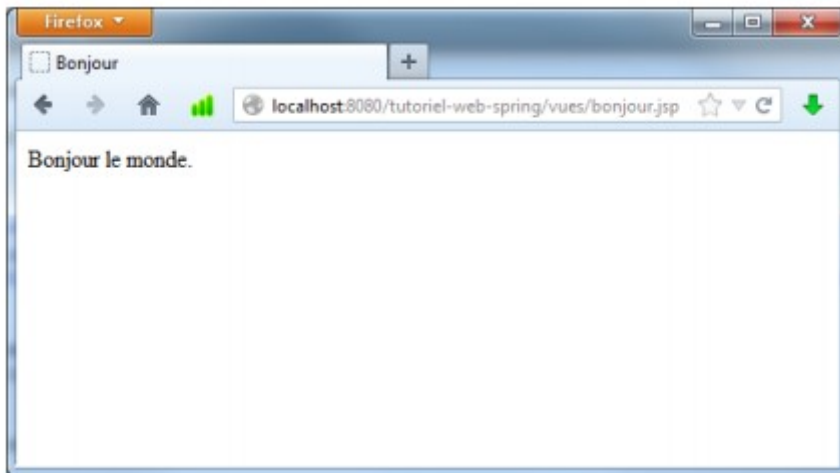
```
    <body>
```

```
        Bonjour le monde.
```

```
    </body>
```

```
</html>
```

Après avoir déployé le projet dans le serveur Tomcat, ouvrir un navigateur Web à l'adresse : **http://localhost:8080/ tutorial-web-spring/vues/bonjour.jsp**.



Nous avons su créer une JSP (correspondant à la partie vue du patron de conception MVC).

III. Utilisation de Spring avec la JSP « **bonjour.jsp** »

Dans ce paragraphe, nous allons modifier l'application afin d'intégrer Spring. Cela reste une utilisation très simple de Spring car elle se limite à l'utilisation des fichiers d'internationalisation. Toutefois, cela valide le déploiement correct des dépendances Maven ainsi que le chargement du fichier de configuration Spring.

1. Vérifier la dépendance vers «**spring-webmvc**» dans le fichier «**pom.xml**».

Dans le bloc : `<dependencies>`

```
<!-- Spring MVC -->
<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-webmvc</artifactId>
    <version>${spring-
        framework.version}</version>
</dependency>
```

2. Modifier le fichier « **web.xml** » comme ci-dessous. Le listener « **ContextLoaderListener** » charge la configuration Spring à partir de la variable de contexte « **contextConfigLocation** ».

/WEB-INF/web.xml

<!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>

<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>

</web-app>

1. Créer le fichier « **dispatcher-servlet.xml** » dans « WEB-INF » comme ci-dessous.
2. Le bean «messageSource» de classe «ReloadableResourceBundleMessageSource» (depuis Spring 1.0) est déclaré.
3. Il permettra de charger les messages internationalisés dans des fichiers « messages_xx.properties » contenus à la racine du classpath.

/WEB-INF/dispatcher-servlet.xml

```
<beans xmlns="http://www.springframework.org/schema/beans"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:context="http://
www.springframework.org/schema/context"
xmlns:mvc="http://www.springframework.org/schema/mvc"
xmlns:tx="http://www.springframework.org/schema/tx"
xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans-4.0.xsd
http://www.springframework.org/schema/context
http://www.springframework.org/schema/context/spring-context-
4.0.xsd
http://www.springframework.org/schema/mvc
http://www.springframework.org/schema/mvc/spring-mvc.xsd
http://www.springframework.org/schema/tx
http://www.springframework.org/schema/tx/springtx-3.0.xsd">

    <bean id="messageSource"

class="org.springframework.context.support.ReloadableResourc
eBundleMessageSource">

        <property name="basename" value="classpath:messages"
/>

        <property name="defaultEncoding" value="ISO-8859-1" />

    </bean>

</beans>
```


1. Créer le fichier suivant « **messages_fr.properties** » dans « **src/main/resources** » comme ci-dessous. Le dossier peut également contenir, par exemple, des fichiers « **messages_en.properties** » pour la langue anglaise ou « **messages.properties** » pour les messages par défaut (quand il n'y a pas de fichier correspondant à la langue).

messages_fr.properties

titre.bonjour=Bonjour avec Spring

libelle.bonjour.lemonde=Bonjour le monde avec Spring.

2. Modifier le fichier « **bonjour.jsp** » comme ci-dessous. On a ajouté la déclaration de la **taglib** « **spring** » et les utilisations du tag « **spring:message** ».

/vues/bonjour.jsp

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
pageEncoding="ISO-8859-1"%>

<%@taglib uri="http://www.springframework.org/tags" prefix="spring"%>

<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/
loose.dtd">

<html>

    <head>

        <title><spring:message code="titre.bonjour"/></title>

    </head>

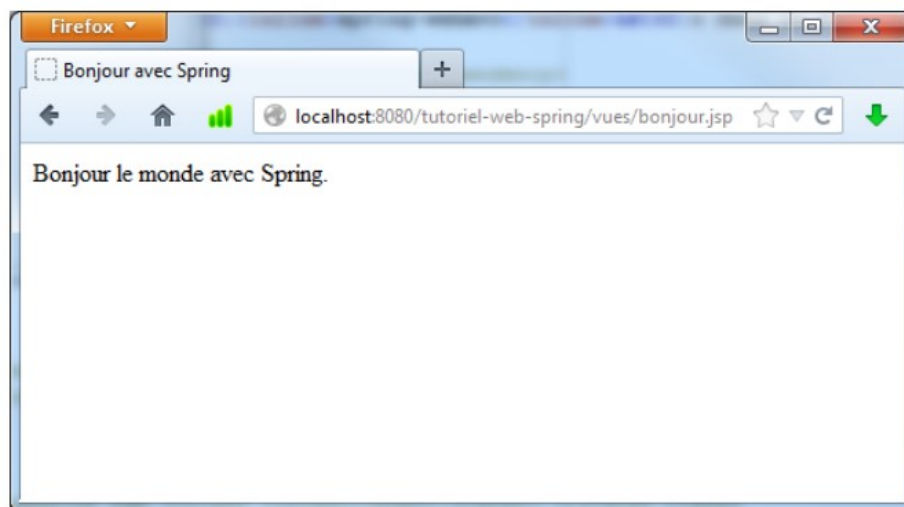
    <body>

        <spring:message code="libelle.bonjour.lemonde"/>

    </body>

</html>
```

Après déploiement, on obtient le résultat ci-dessous à l'adresse : **<http://localhost:8080/tutoriel-web-spring/vues/bonjour.jsp>**.



Nous avons utilisé le système d'internationalisation de Spring et vérifié ainsi que les dépendances s'étaient déployées correctement.

IV. Passage de donnée à la JSP « `bonjour.jsp` »

Nous allons maintenant modifier cela afin que la JSP affiche le texte en fonction d'une donnée. Pour cela, nous ajoutons un contrôleur qui transmettra une donnée à la JSP.

/WEB-INF/web.xml

```
<!-- Declaration de la servlet de Spring et de son mapping -->

<servlet>

<servlet-name>servlet-dispatcher</servlet-name>

<servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>

    <init-param>

        <param-name>contextConfigLocation</param-name>

        <param-value>/WEB-INF/dispatcher-servlet.xml</param-value>

    </init-param>

    <load-on-startup>1</load-on-startup>

</servlet>

<servlet-mapping>

    <servlet-name>servlet-dispatcher</servlet-name>

    <url-pattern>/</url-pattern>

</servlet-mapping>
```

Voici la classe « **BonjourController** ».

1. Elle est déclarée en tant que contrôleur grâce à l'annotation « **@Controller** » (elle existe depuis Spring 2.5).
2. L'annotation « **@RequestMapping** » (elle existe depuis Spring 2.5) indique que le contrôleur traite les requêtes GET dont l'URI est « **/bonjour** ».
3. On ne constate que la valeur « **Regis** » est associée à l'attribut « **personne** » grâce à la méthode « **addAttribute** » de « **ModelMap** ».
4. Ensuite, le contrôleur redirige vers la ressource « **bonjour** ».

Créer le package : **com.M2i.web.controller**

BonjourController.java

```
package com.M2i.web.controller;

import org.springframework.stereotype.Controller;
import org.springframework.ui.ModelMap;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;

@Controller
@RequestMapping("/bonjour")
public class BonjourController {

    @RequestMapping(method = RequestMethod.GET)
    public String afficherBonjour(final ModelMap pModel) {
        pModel.addAttribute("personne", "Regis");
        return "bonjour";
    }
}
```

Dans le fichier « **dispatcher-servlet.xml** » il faut ajouter les lignes ci-dessous. « **component-scan** » active la configuration par annotations. La déclaration du bean « **InternalResourceViewResolver** » (qui existe depuis Spring 1.0) permet d'indiquer où chercher les ressources (ici la ressource « **bonjour** » indiquée dans le contrôleur sera cherchée avec l'extension « **.jsp** » dans le dossier « **/vues/** »).

/WEB-INF/dispatcher-servlet.xml

```
<context:component-scan base-package="com.M2i.web" />

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

Modifier la valeur ci-dessous dans le fichier « **messages_fr.properties** » afin de prendre en compte un paramètre dans le texte.

messages_fr.properties

libelle.bonjour.lemonde=Bonjour {0} avec Spring.

Modifier le fichier « **bonjour.jsp** » comme ci-dessous. On a ajouté des Expressions Languages (EL) « **\${personne}** » afin de restituer la donnée.

/vues/bonjour.jsp

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
isELIgnored="false"

pageEncoding="ISO-8859-1"%>

<%@taglib uri="http://www.springframework.org/tags" prefix="spring"%>

<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/
loose.dtd">

<html>

    <head>

        <title><spring:message code="titre.bonjour"/> : ${personne}</title>

    </head>

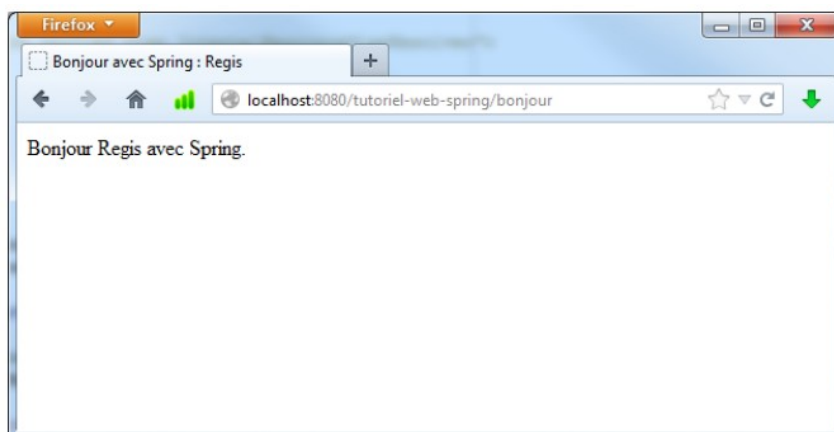
    <body>

        <spring:message code="libelle.bonjour.lemonde" arguments="${personne}"/>

    </body>

</html>
```

Après déploiement, on obtient le résultat ci-dessous à l'adresse : **http://localhost:8080/tutoriel-web-spring/ bonjour.**



Nous avons passé une valeur dans un attribut depuis le contrôleur que nous avons affiché dans le JSP grâce aux Expressions Languages (EL).

V. Récupération d'un paramètre de la requête http

Dans ce chapitre, nous allons récupérer un paramètre de la requête pour le passer à la place de la valeur de l'attribut.

Dans la classe « **BonjourController** », modifier la méthode « **afficherBonjour** » comme ci-dessous.

Le paramètre « **personne** » est récupéré de la requête grâce à l'annotation « **@RequestParam** » (l'annotation existe depuis Spring 2.5 mais l'élément « **defaultValue** » a été rajouté avec Spring 3.0)*

BonjourController.java

```
package com.M2i.web.controller; import
org.springframework.stereotype.Controller; import
org.springframework.ui.ModelMap; import
org.springframework.web.bind.annotation.RequestMapping; import
org.springframework.web.bind.annotation.RequestMethod; import
org.springframework.web.bind.annotation.RequestParam;

@Controller @RequestMapping("/bonjour")

public class BonjourController {

    @RequestMapping(method = RequestMethod.GET)

    public String afficherBonjour(final ModelMap pModel,
    @RequestParam(value="personne") final String pPersonne)
    {

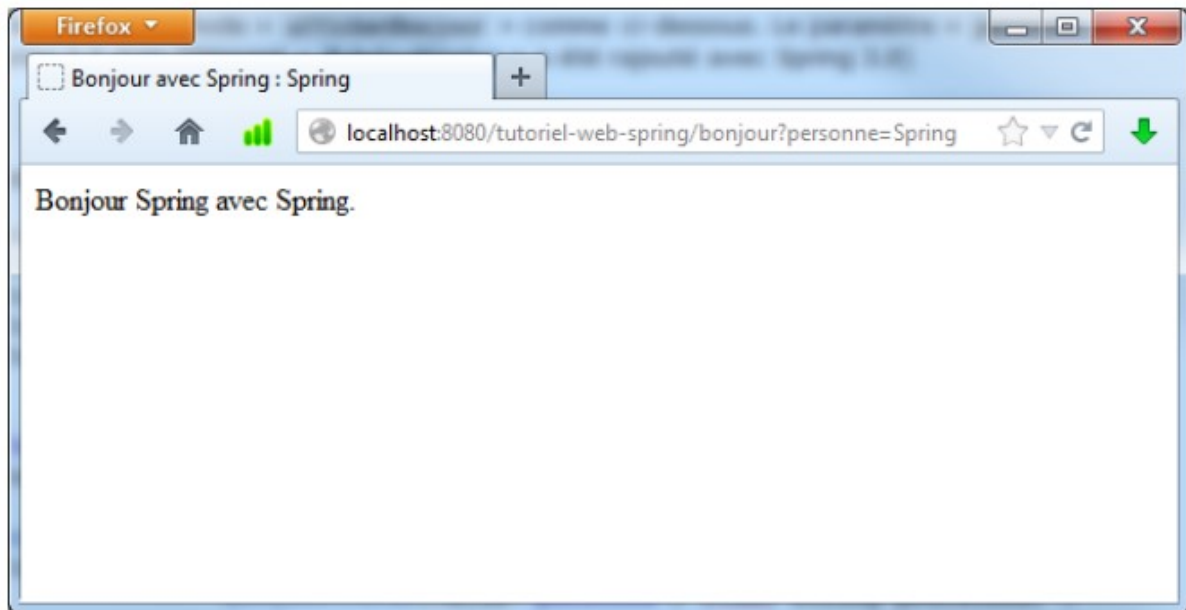
        pModel.addAttribute("personne", pPersonne);

        return "bonjour";

    }

}
```

Après déploiement, on obtient le résultat ci-dessous à l'adresse :
<http://localhost:8080/tutoriel-web-spring/ bonjour?personne=Spring>



Une alternative possible est d'extraire le paramètre depuis l'URI comme dans l'exemple ci-dessous.

Cela est possible en indiquant dans l'annotation « **@RequestMapping** » qu'il y a une variable (indiqué avec les accolades).

Ensuite, l'annotation « **@PathVariable** » (existant depuis Spring 3.0) permet d'indiquer l'utilisation de cette variable extraite de l'URI.

BonjourController.java

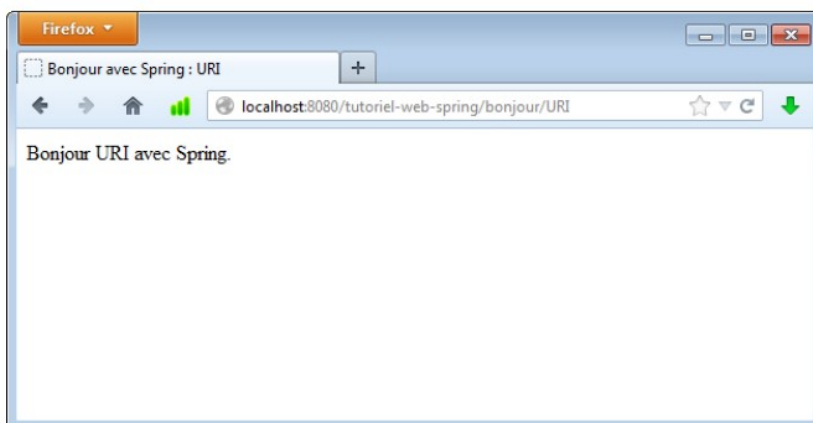
```
package com.M2i.web.controller;

import org.springframework.stereotype.Controller;
import org.springframework.ui.ModelMap;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;

@Controller
@RequestMapping("/bonjour/{personne}")
public class BonjourController {

    @RequestMapping(method = RequestMethod.GET)
    public String afficherBonjour(final ModelMap pModel,
        @PathVariable(value="personne") final String pPersonne) {
        pModel.addAttribute("personne", pPersonne);
        return "bonjour";
    }
}
```

Après déploiement, on obtient le résultat ci-dessous à l'adresse :
<http://localhost:8080/tutoriel-web-spring/ bonjour/URI>.



Nous avons vu comment récupérer un paramètre depuis la requête (dans cet exemple, dans une requête GET, donc dans l'URL, mais le principe est le même pour une requête POST) et comment parser une valeur dans l'URI.

VI. Création de la base

Pour les besoins de ce tutoriel, nous utiliserons la base de données **HSQLDB (HyperSQL DataBase)**.

(C'est la base de données par défaut qui existe dans Tomcat par exemple).

Dans ce chapitre, nous allons créer la base de données, une table et remplir cette table avec des valeurs de test.

L'organisation et le paramétrage des fichiers XML pour le Serveur de Bases de Données seront identiques au niveau logique pour SQL Serveur ou Postgress.

Il faut donc rajouter la dépendance Maven suivante :

Extrait du fichier « pom.xml » (entre les balises : <dependencies></dependencies>) :

```
<dependency>
  <groupId>org.hsqldb</groupId>
  <artifactId>hsqldb</artifactId>
  <version>2.3.1</version>
</dependency>
```

Voici la table qui sera utilisée en exemple dans ce tutoriel :

LISTECOURSES	
*IDOBJET	INTEGER
°LIBELLE	VARCHAR
°QUANTITE	INTEGER

Voici son script SQL de création :

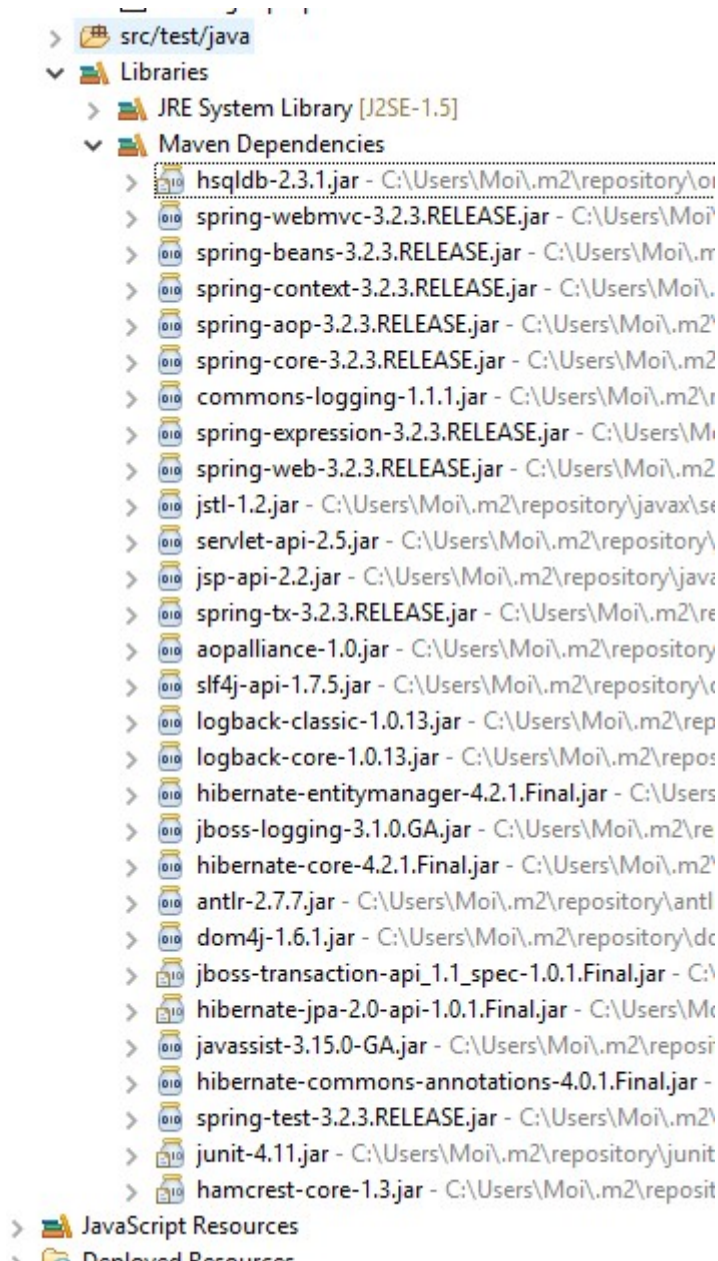
On utilisera le script suivant pour créer la table :

Script SQL de création de la table « LISTECOURSES »

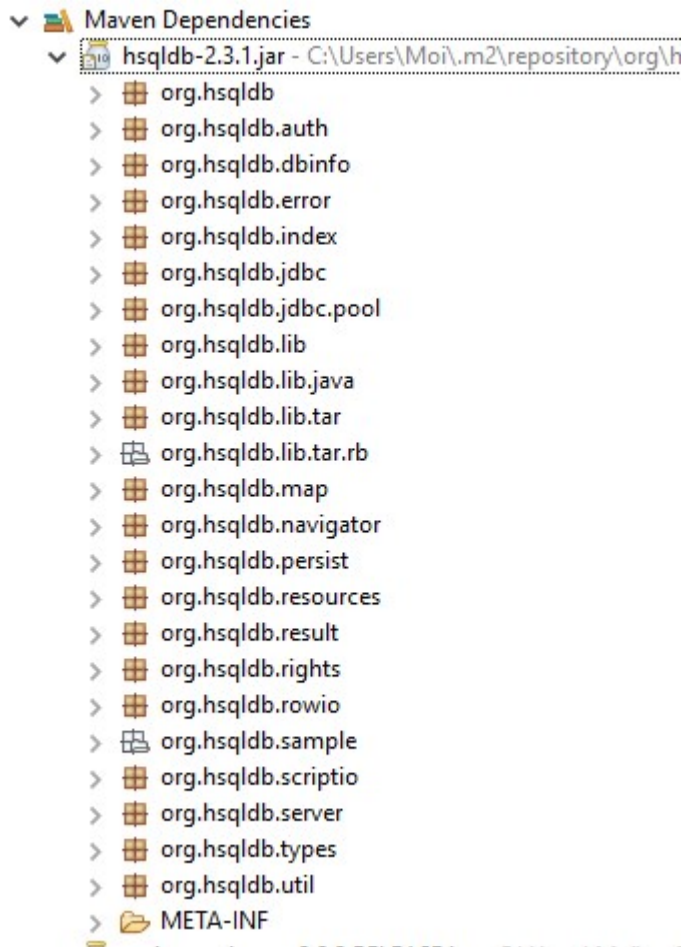
```
CREATE TABLE LISTECOURSES(
  IDOBJET INTEGER GENERATED BY DEFAULT AS IDENTITY(START WITH 1)
  PRIMARY KEY,
  LIBELLE VARCHAR(50) NOT NULL,
  QUANTITE INTEGER NOT NULL
);
```

Pour lancer le gestionnaire de base HSQLDB, exécuter la classe « org.hsqldb.util.DatabaseManager ». Voici la connexion à la base de données d'exemple (ici avec l'URL « jdbc:hsqldb:file:C:\hsqldb\data\maBase ») :

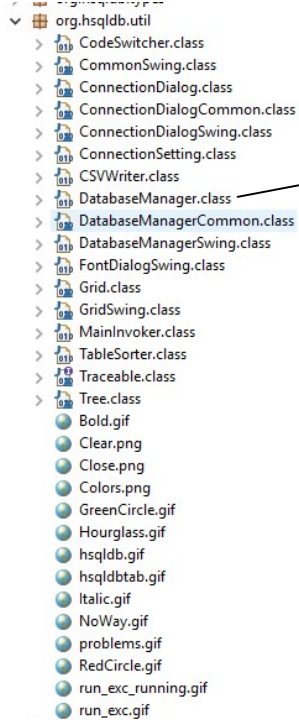
Vous avez après avoir sauvegardé votre fichier « pom.xml » le dossier suivant mis à jour :



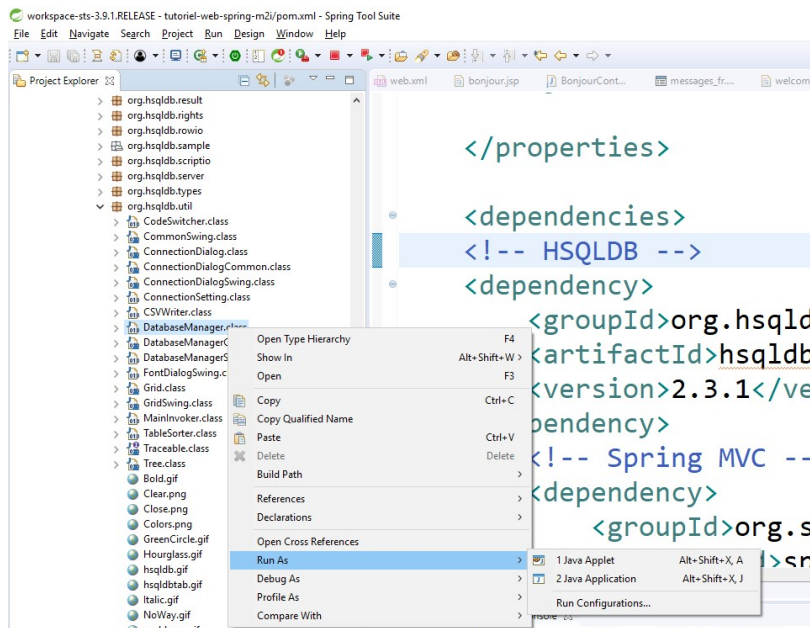
Vous allez « dépiler » le jar suivant :



Puis aller dans le package :

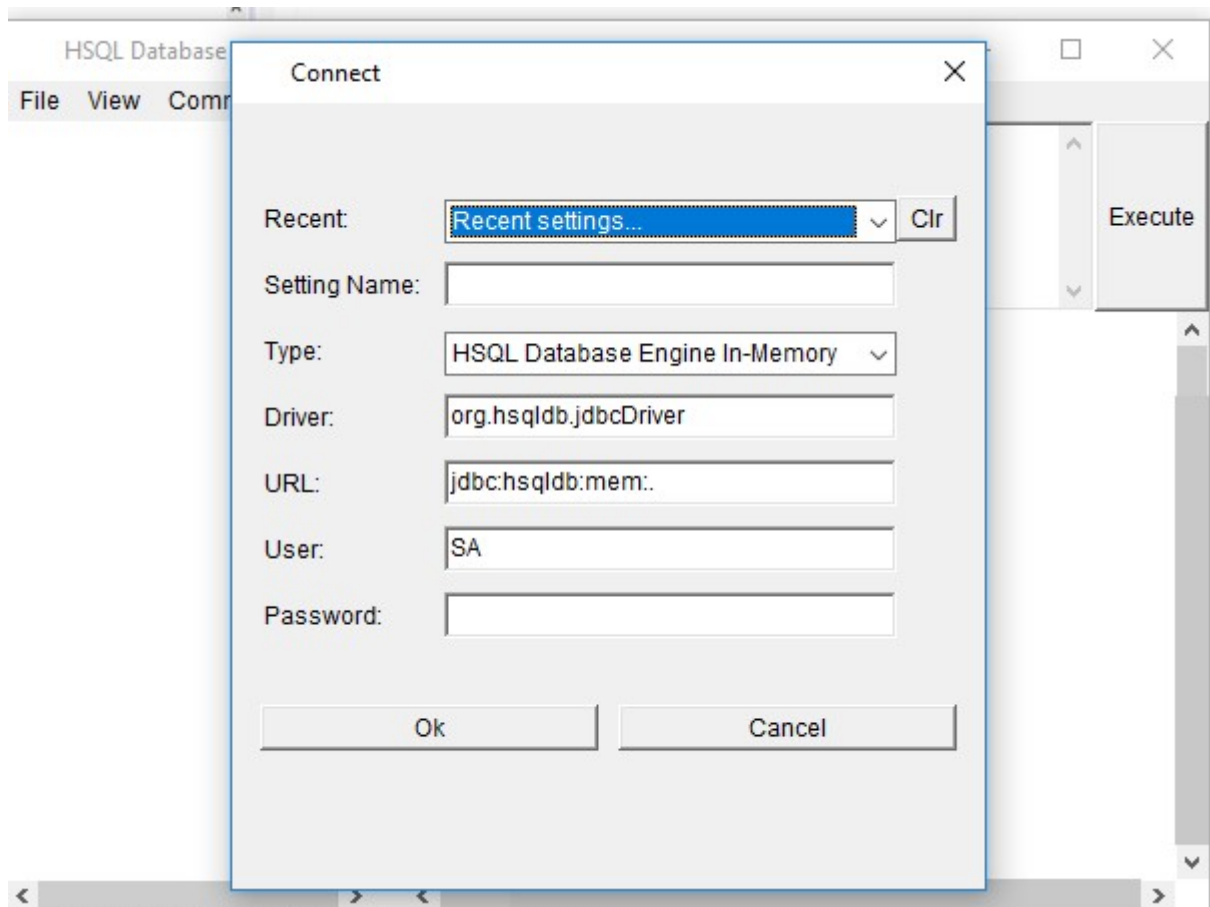


Cliquer sur la classe
« DatabaseManager.class »
avec le bouton droit de la
souris pour avoir le menu
suivant :

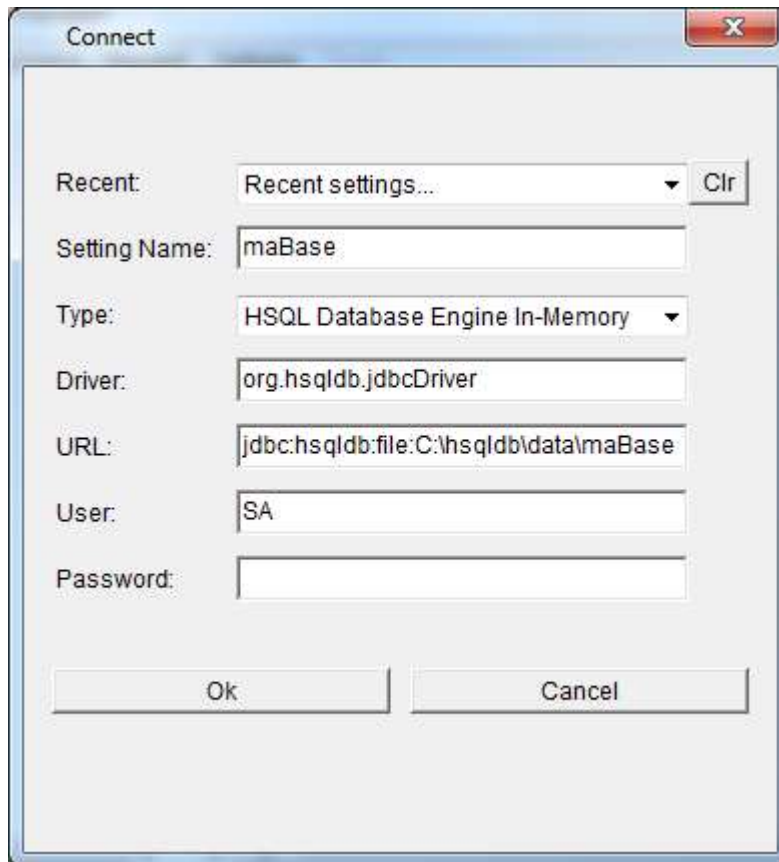


Choisir « Java Application »

On obtient l'écran suivant :



Mettre par le formulaire par défaut par les données de cet écran :

A screenshot of a 'Connect' dialog box. The dialog has a title bar with 'Connect' and a close button. It contains several input fields: 'Recent' with a dropdown menu showing 'Recent settings...' and a 'Clr' button; 'Setting Name' with a text field containing 'maBase'; 'Type' with a dropdown menu showing 'HSQL Database Engine In-Memory'; 'Driver' with a text field containing 'org.hsqldb.jdbcDriver'; 'URL' with a text field containing 'jdbc:hsqldb:file:C:\hsqldb\data\maBase'; 'User' with a text field containing 'SA'; and 'Password' with an empty text field. At the bottom are 'Ok' and 'Cancel' buttons.

Connect

Recent: Recent settings... Clr

Setting Name: maBase

Type: HSQL Database Engine In-Memory

Driver: org.hsqldb.jdbcDriver

URL: jdbc:hsqldb:file:C:\hsqldb\data\maBase

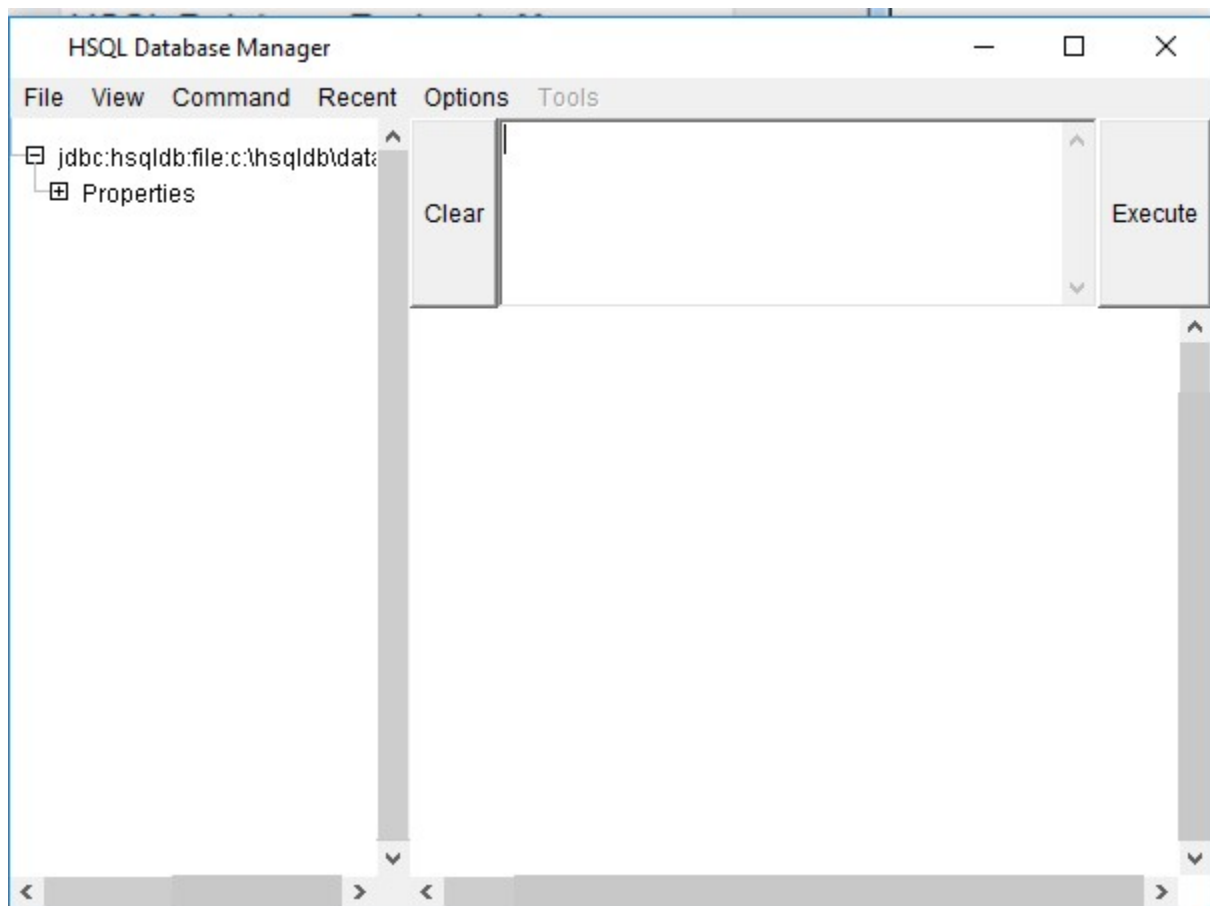
User: SA

Password:

Ok Cancel

Cliquer sur le bouton « ok »

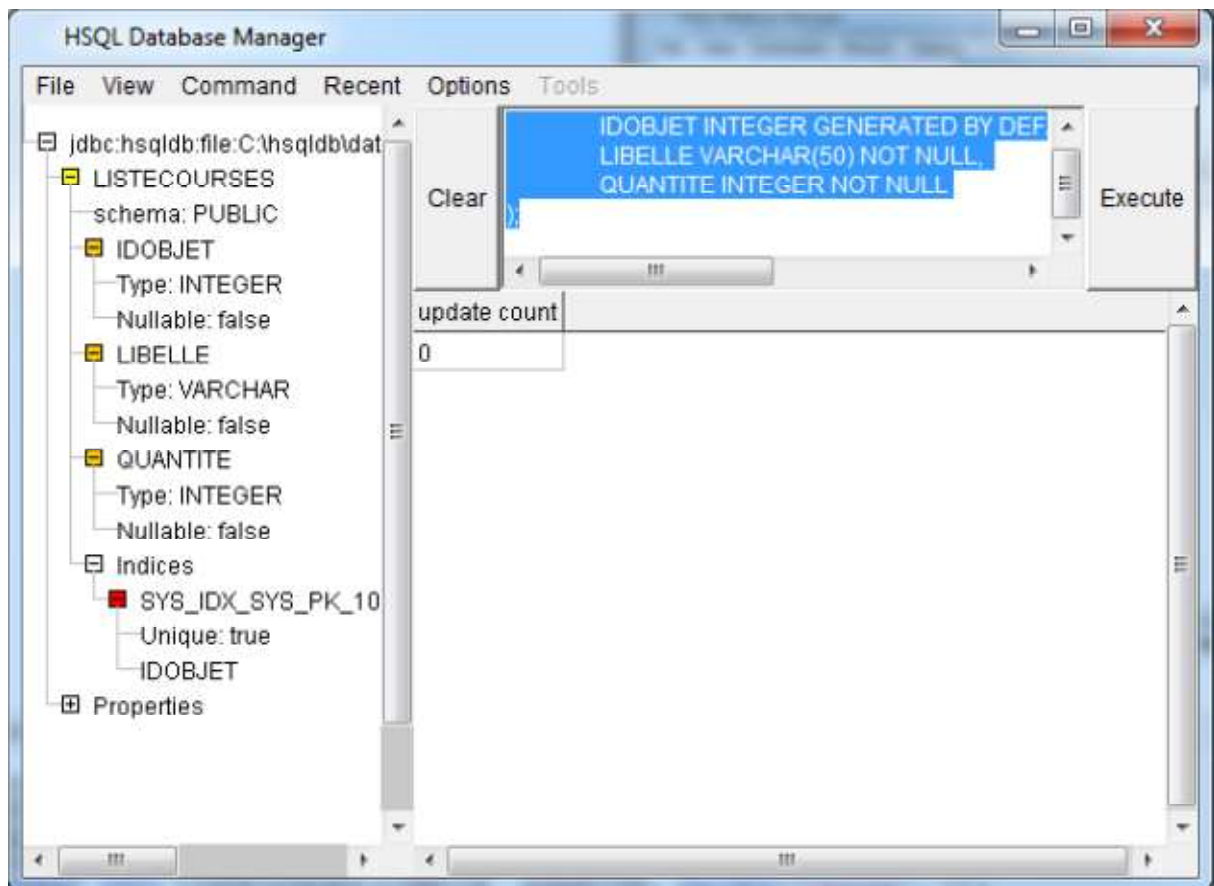
Vous avez l'écran suivant :



Dans cet écran, vous allez pouvoir copier *Script SQL de création de la table « LISTECOURSES »*

:

Voir l'écran suivant :



Quand la copie est faite, cliquer sur le bouton « Execute »
Pour avoir obtenu, l'écran complet ci-dessus (partie gauche).

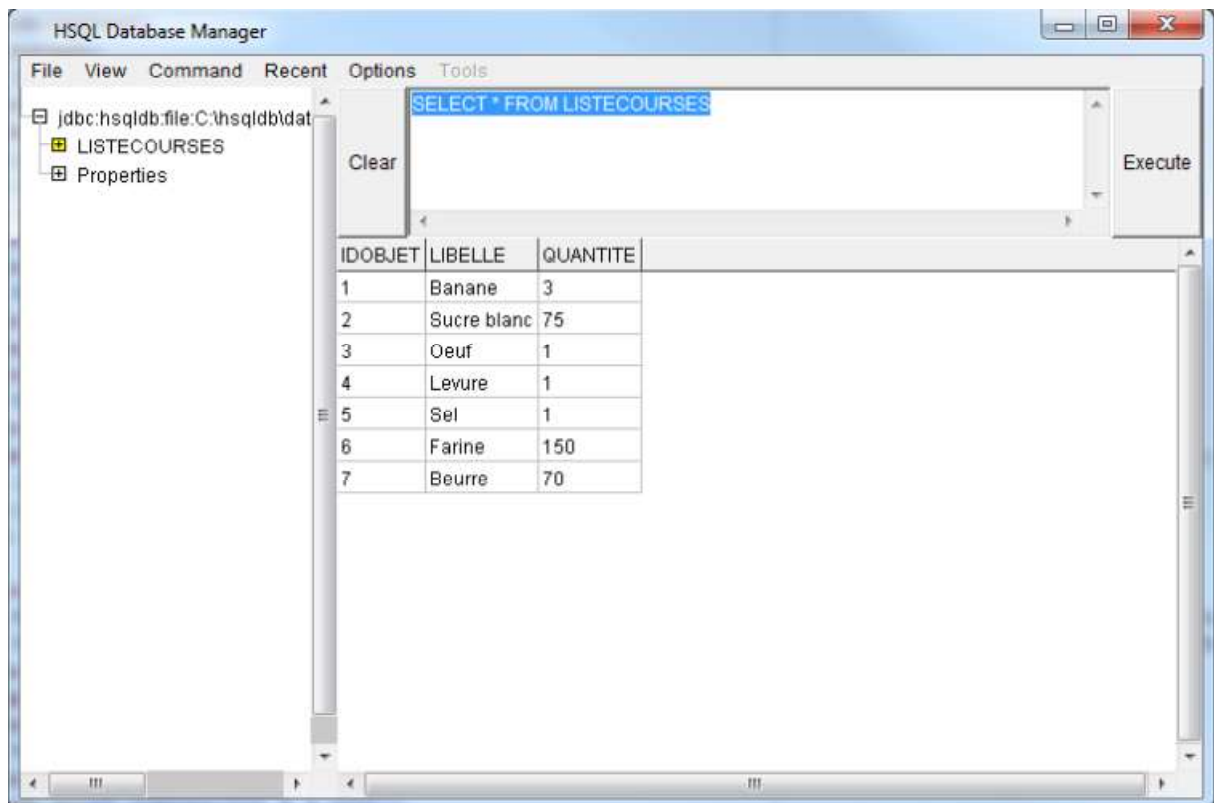
On va insérer des données pour la suite du TP :

(Appuyer sur le bouton « Clear » : avant d'insérer les lignes SQL suivantes)

```
INSERT INTO LISTECOURSES(LIBELLE, QUANTITE) VALUES('Banane', 3);
INSERT INTO LISTECOURSES(LIBELLE, QUANTITE) VALUES('Sucre blanc', 75);
INSERT INTO LISTECOURSES(LIBELLE, QUANTITE) VALUES('Oeuf', 1);
INSERT INTO LISTECOURSES(LIBELLE, QUANTITE) VALUES('Levure', 1);
INSERT INTO LISTECOURSES(LIBELLE, QUANTITE) VALUES('Sel', 1);
INSERT INTO LISTECOURSES(LIBELLE, QUANTITE) VALUES('Farine', 150);
INSERT INTO LISTECOURSES(LIBELLE, QUANTITE) VALUES('Beurre', 70);
```

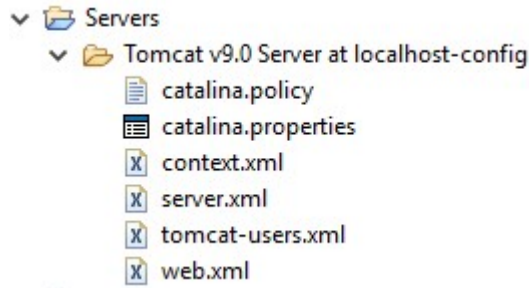
N'oublier pas de cliquer sur le bouton « Execute » pour que les insertions se fassent dans la table LISTECOURSES.

Voici le résultat de l'insertion de données dans la table :



VII. Paramétrage du serveur Tomcat

Nous allons paramétrer la ressource JDBC dans le serveur Tomcat (depuis Eclipse). C'est cette ressource qui est reliée à la base HSQLDB et qui sera utilisée dans l'application.



(Le serveur par défaut par STS : Pivotal tc Server)

Dans le fichier : **server.xml** :

Il faut ajouter la ressource ci-dessous dans la partie « GlobalNamingResources » pour dialoguer avec HSQLDB.

Entre les balises : <GlobalNamingResources>

```
...  
<Resource auth="Container" driverClassName="org.hsqldb.jdbcDriver"  
maxActive="100" maxIdle="30" maxWait="10000" name="jdbc/dsMaBase"  
password="" type="javax.sql.DataSource"  
url="jdbc:hsqldb:file:C:\hsqldb\data\maBase" username="sa" />  
...  
</GlobalNamingResources>
```

(Pensez à sauvegarder votre fichier !)

Dans le fichier « context.xml », il faut associer la ressource que l'on vient d'indiquer dans le fichier « server.xml » avec le nom « **jdbc/dsMonApplication** » que l'on utilisera dans l'application.

Dans le fichier : **context.xml** :

Ligne en dessous à insérer : entre les balises

```
<ResourceLink name="jdbc/dsMonApplication" global="jdbc/dsMaBase"  
type="javax.sql.DataSource" />
```

VIII. Modification du projet afin d'inclure l'affichage des données

Nous allons modifier le projet afin de réaliser l'affichage des données que l'on vient d'ajouter en base de données. Il faut rajouter la déclaration de la ressource JDBC de l'application dans le fichier « web.xml ».

Dans le fichier : /WEB-INF/web.xml

Ajouter à la fin du fichier (avant </web-app>)

```
<!-- Declaration de l'utilisation de la ressource JDBC -->
<resource-ref>
    <description>Ressource JDBC de l'application</description>
    <res-ref-name>jdbc/dsMonApplication</res-ref-name>
    <res-type>javax.sql.DataSource</res-type>
    <res-auth>Container</res-auth>
    <res-sharing-scope>Shareable</res-sharing-scope>
</resource-ref>
```

Il faut donc rajouter les dépendances Maven suivantes. Les dépendances « spring-orm » et « hibernate-entitymanager » sont pour l'accès aux données. Tandis que la dépendance « jstl » est pour l'utilisation de la JSTL (Java Standard Tag Library) dans la JSP d'affichage.

Mettre à jour votre fichier POM.XML avec les lignes suivantes :

(Ecraser les blocs « properties » et « dependencies »

```
<properties>

    <!-- Generic properties -->
    <java.version>1.6</java.version>
    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
    <project.reporting.outputEncoding>UTF-8</project.reporting.outputEncoding>
    <!-- Web -->
    <jsp.version>2.2</jsp.version>
    <jstl.version>1.2</jstl.version>
    <servlet.version>2.5</servlet.version>
    <!-- Spring -->
    <spring-framework.version>3.2.3.RELEASE</spring-framework.version>
    <!-- Hibernate / JPA -->
    <hibernate.version>4.3.4.Final</hibernate.version>
    <!-- Logging -->
    <logback.version>1.0.13</logback.version>
    <slf4j.version>1.7.5</slf4j.version>

    <!-- Test -->
    <junit.version>4.11</junit.version>

</properties>
<dependencies>

    <!-- Spring MVC -->
    <dependency>
        <groupId>org.springframework</groupId>
        <artifactId>spring-webmvc</artifactId>
        <version>${spring-framework.version}</version>
    </dependency>
    <dependency>
        <groupId>org.springframework</groupId>
        <artifactId>spring-orm</artifactId>
        <version>${spring-framework.version}</version>
    </dependency>
    <!-- Other Web dependencies -->
    <dependency>
        <groupId>javax.servlet</groupId>
        <artifactId>jstl</artifactId>
        <version>${jstl.version}</version>
    </dependency>
    <dependency>
        <groupId>javax.servlet</groupId>
        <artifactId>servlet-api</artifactId>
        <version>${servlet.version}</version>
        <scope>provided</scope>
    </dependency>
    <dependency>
        <groupId>javax.servlet.jsp</groupId>
        <artifactId>jsp-api</artifactId>
        <version>${jsp.version}</version>
        <scope>provided</scope>
    </dependency>
    <!-- Spring and Transactions -->
    <dependency>
        <groupId>org.springframework</groupId>
        <artifactId>spring-tx</artifactId>
        <version>${spring-framework.version}</version>
    </dependency>
    <!-- Logging with SLF4J & LogBack -->
    <dependency>
        <groupId>org.slf4j</groupId>
        <artifactId>slf4j-api</artifactId>
        <version>${slf4j.version}</version>
        <scope>compile</scope>
    </dependency>
    <dependency>
        <groupId>ch.qos.logback</groupId>
        <artifactId>logback-classic</artifactId>
        <version>${logback.version}</version>
        <scope>runtime</scope>
    </dependency>
    <!-- Hibernate -->
    <dependency>
        <groupId>org.hibernate</groupId>
        <artifactId>hibernate-entitymanager</artifactId>
        <version>${hibernate.version}</version>
    </dependency>

    <!-- Test Artifacts -->
```

```

        <dependency>
            <groupId>org.springframework</groupId>
            <artifactId>spring-test</artifactId>
            <version>${spring-framework.version}</version>
            <scope>test</scope>
        </dependency>
        <dependency>
            <groupId>junit</groupId>
            <artifactId>junit</artifactId>
            <version>${junit.version}</version>
            <scope>test</scope>
        </dependency>
        <dependency>
            <groupId>org.hsqldb</groupId>
            <artifactId>hsqldb</artifactId>
            <version>2.3.1</version>
        </dependency>
    </dependencies>

```

Création du fichier « persistence.xml »

(Dans le dossier : src/main/resources/META-INF/persistence.xml)

Le fichier « persistence.xml » ci-dessous permet d'indiquer que la persistance est réalisée grâce à Hibernate.

Contenu du fichier :

```

<?xml version="1.0" encoding="UTF-8"?>
<persistence version="2.0" xmlns="http://java.sun.com/xml/ns/persistence"
xmlns:xsi="http://www.w3.org/2001/XMLSchemaInstance"
xsi:schemaLocation="http://java.sun.com/xml/ns/persistence
http://java.sun.com/xml/ns/persistence/persistence_2_0.xsd">

    <persistence-unit name="unit">
        <provider>org.hibernate.jpa.HibernatePersistenceProvider</provider>
    </persistence-unit>

</persistence>

```

Après avoir créé ce fichier et sauvegarder ...

Il faut rajouter les lignes ci-dessous dans le fichier « dispatcher-servlet.xml ».

Le bean « JndiObjectFactoryBean » permet de déclarer l'utilisation de la ressource JDBC.

Le bean « LocalContainerEntityManagerFactoryBean » utilise la ressource JDBC et le fichier « persistence.xml » pour aboutir à la création du « EntityManager » qui est utilisé dans la DAO.

Le bean « JpaTransactionManager » permet d'instancier le gestionnaire de transaction et lui associer la fabrique de « EntityManager ».

Mettre à jour le fichier : /WEB-INF/dispatcher-servlet.xml

Entête du fichier XML à écraser par les lignes suivantes :

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xmlns:mvc="http://www.springframework.org/schema/mvc"
       xmlns:context="http://www.springframework.org/schema/context"
       xmlns:tx="http://www.springframework.org/schema/tx"
       xsi:schemaLocation="http://www.springframework.org/schema/mvc
http://www.springframework.org/schema/mvc/spring-mvc.xsd
                           http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans.xsd
                           http://www.springframework.org/schema/tx
http://www.springframework.org/schema/tx/spring-tx-3.0.xsd
                           http://www.springframework.org/schema/context
http://www.springframework.org/schema/context/spring-context.xsd">
```

Puis ajouter entre ensuite ces lignes :

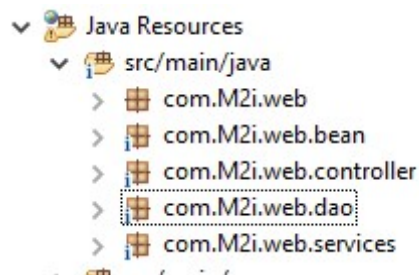
```
<tx:annotation-driven transaction-manager="transactionManager" />
<bean id="dataSource"
class="org.springframework.jndi.JndiObjectFactoryBean">
    <property name="jndiName"
value="java:comp/env/jdbc/dsMonApplication" />
</bean>
<bean id="entityManagerFactory"
class="org.springframework.orm.jpa.LocalContainerEntityManagerFactoryBe
an">
    <property name="dataSource" ref="dataSource" />
</bean>
<bean id="transactionManager"
class="org.springframework.orm.jpa.JpaTransactionManager">
    <property name="entityManagerFactory" ref="entityManagerFactory"
/>
</bean>
```


IX. Développement :

A. Présentation :

Nous allons avoir les sous packages suivants en plus du sous package controller :

- beans
- services
- dao



B. Création de la classe « d'entity » :

On va créer la classe « Course » qui est une entité correspondant à la table «LISTECOURSES».

Créer le package : `com.M2i.web.beans` :

```

package com.M2i.web.bean;

import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import javax.persistence.Table;

@Entity
@Table(name="LISTECOURSES")
public class Course {
    @Id
    @GeneratedValue(strategy=GenerationType.IDENTITY)
    @Column(name="IDOBJET")
    private Integer id;
    private String libelle;
    private Integer quantite;

    public Integer getId() {
        return id;
    }

    public void setId(final Integer pId) {
        id = pId;
    }

    public String getLibelle() {
        return libelle;
    }

    public void setLibelle(final String pLibelle) {
        libelle = pLibelle;
    }

    public Integer getQuantite() {
        return quantite;
    }

    public void setQuantite(final Integer pQuantite) {
        quantite = pQuantite;
    }
}

```

C. Gestion des interfaces « métiers » :

Voici les interfaces de la DAO et du service : « IListeCoursesDAO » et « IServiceListeCourses ».

Créer le package : com.M2i.web.dao :

Créer l'interface : IListeCoursesDAO.java

```
package com.M2i.web.dao;
import java.util.List;
import com.M2i.web.bean.Course;
    public interface IListeCoursesDAO {
        List<Course> rechercherCourses();
    }
```

Créer le package : com.M2i.web.services :

Créer l'interface : IServiceListeCourses.java

```
package com.M2i.web.services;
import java.util.List;
import com.M2i.web.bean.Course;
    public interface IServiceListeCourses {
        List<Course> rechercherCourses();
    }
```

Attention :

La DAO « ListeCoursesDAO » utilise le « EntityManager » pour lister les entités « Course » contenues dans la base de données.

D. Création des classes « Métiers » implémentant les intf. « Métiers » :

Créer la classe : ListeCoursesDAO.java

```
package com.M2i.web.dao;
import java.util.List;
import javax.persistence.EntityManager;
import javax.persistence.PersistenceContext;
import javax.persistence.TypedQuery;
import javax.persistence.criteria.CriteriaBuilder;
import javax.persistence.criteria.CriteriaQuery;
import javax.persistence.criteria.Root;
import org.springframework.stereotype.Repository;
import com.M2i.web.bean.Course;
@Repository
public class ListeCoursesDAO implements IListeCoursesDAO {
    private EntityManager entityManager;
    public List<Course> rechercherCourses() {
        final CriteriaBuilder ICriteriaBuilder =
            entityManager.getCriteriaBuilder();
        final CriteriaQuery<Course> ICriteriaQuery =
            ICriteriaBuilder.createQuery(Course.class);
        final Root<Course> IRoot = ICriteriaQuery.from(Course.class);
        ICriteriaQuery.select(IRoot);
        final TypedQuery<Course> ITypedQuery =
            entityManager.createQuery(ICriteriaQuery);
        return ITypedQuery.getResultList();
    }
}
```

Pour cette méthode, le service « ServiceListeCourses » sert surtout de passe-plat. Il faut noter toutefois que la transaction est indiquée en lecture seule.

Créer la classe : ServiceListeCourses.java

```
package com.M2i.web.services;  
import java.util.List;  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.stereotype.Service;  
import org.springframework.transaction.annotation.Transactional;  
import com.M2i.web.bean.Course;  
import com.M2i.web.dao.IListeCoursesDAO;  
@Service  
public class ServiceListeCourses implements IServiceListeCourses {  
@Autowired  
private IListeCoursesDAO dao;  
@Transactional(readOnly=true)  
    public List<Course> rechercherCourses() {  
        return dao.rechercherCourses();}  
}
```

E. Création des controllers :

Le contrôleur « AfficherListeCoursesController » appelle le service et place le résultat dans l'attribut « listeCourses ».

Dans le package : com.M2i.web.controller

Créer la classe : AfficherListeCoursesController.java

```
package com.M2i.web.controller;
import java.util.List;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Controller;
import org.springframework.ui.ModelMap;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;
import com.M2i.web.bean.Course;
import com.M2i.web.services.IServiceListeCourses;
@Controller
@RequestMapping(value="/afficherListeCourses")
public class AfficherListeCoursesController {
    @Autowired
    private IServiceListeCourses service;
    @RequestMapping(method = RequestMethod.GET)
    public String afficher(ModelMap pModel) {
        final List<Course> lListeCourses = service.rechercherCourses();
        pModel.addAttribute("listeCourses", lListeCourses);
        return "listeCourses";
    }
}
```

F. Gestion des « properties » :

On va mettre à jour les fichiers « properties ».

Il faut ajouter de nouveaux libellés dans le fichier « messages_fr.properties ».

titre.listecourses=Liste de courses

colonne.identifiant=IDOBJET

colonne.libelle=LIBELLE

colonne.quantite=QUANTITE

G. Gestions des pages JSP :

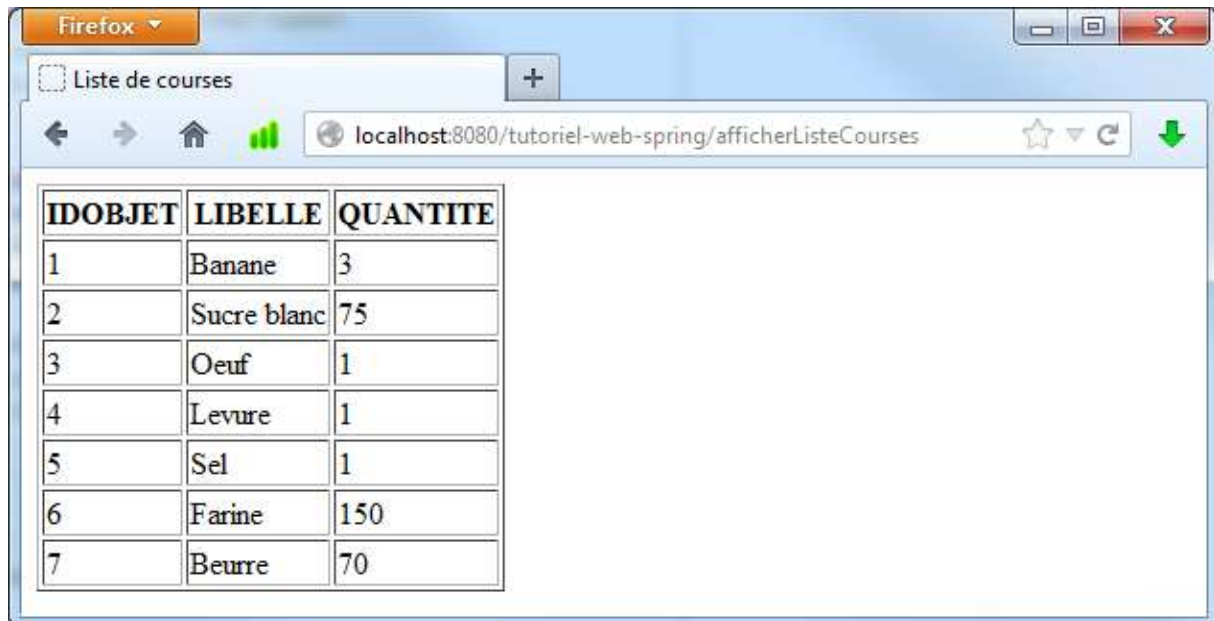
On va créer les fichiers JSP suivantes :

La JSP utilise la Java Standard Tag Library (JSTL) pour afficher le résultat.

Créer la JSP : /vues/listeCourses.jsp

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
isELIgnored="false"
pageEncoding="ISO-8859-1"%>
<%@taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c"%>
<%@taglib uri="http://www.springframework.org/tags" prefix="spring"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/
loose.dtd">
<html>
    <head>
        <title><spring:message code="titre.listecourses"/></title>
    </head>
<body>
    <table border="1">
        <thead>
            <tr>
                <th><spring:message code="colonne.identifiant"/></th>
                <th><spring:message code="colonne.libelle"/></th>
                <th><spring:message code="colonne.quantite"/></th>
            </tr>
        </thead>
        <tbody>
            <c:forEach items="${listeCourses}" var="course">
                <tr>
                    <td><c:out value="${course.id}"/></td>
                    <td><c:out value="${course.libelle}"/></td>
                    <td><c:out value="${course.quantite}"/></td>
                </tr>
            </c:forEach>
        </tbody>
    </table>
</body>
</html>
```


Après déploiement, on obtient le résultat ci-dessous à l'adresse :
<http://localhost:8080/tutoriel-web-spring-M2i/afficherListeCourses>.



Firefox

Liste de courses

localhost:8080/tutoriel-web-spring/afficherListeCourses

IDOBJET	LIBELLE	QUANTITE
1	Banane	3
2	Sucre blanc	75
3	Oeuf	1
4	Levure	1
5	Sel	1
6	Farine	150
7	Beurre	70

X. Création de données en base (CRUD)

Dans ce paragraphe, nous allons créer de nouvelles données dans la base de données. Pour cela, nous aurons besoin d'un formulaire que sera validé pour vérifier les données le constituant.

Il faut donc rajouter les dépendances Maven suivantes. Les dépendances « **validation-api** » et « **hibernate-validator** » permettent la validation du formulaire.

(Ces dépendances sont déjà existantes dans le fichier « **pom.xml** » donné.)

Il faut rajouter la ligne ci-dessous dans le fichier « **dispatcher-servlet.xml** ». Elle permet d'activer les annotations de validation de formulaire.

<mvc:annotation-driven /> (entre les balises **<beans ...></beans>**)

A. Gestion des « properties » :

Il faut ajouter de nouveaux libellés dans le fichier « **messages_fr.properties** ».

titre.creation.elementcourses=Création d'élément de la liste de courses
creation.elementcourses.libelle.libelle=Libellé
creation.elementcourses.libelle.quantite=Quantité
NotEmpty.creation.libelle=Le libellé est nécessaire.
NotEmpty.creation.quantite=La quantité est nécessaire.
Pattern.creation.quantite=La quantité doit être numérique et positive.

B. Création de classes java utiles pour les controller :

Le formulaire « **CreationForm** » utilise les annotations « **NotEmpty** » et « **Pattern** » pour indiquer les contraintes de validation.

Créer la classe : **CreationForm.java**

```
package com.M2i.web.controller;
import javax.validation.constraints.Pattern;
import org.hibernate.validator.constraints.NotEmpty;
public class CreationForm {
    @NotEmpty
    private String libelle;
    @NotEmpty
    @Pattern(regexp="\\d*")
    private String quantite;
    public String getLibelle() {
        return libelle;
    }
    public void setLibelle(final String pLibelle) {
        libelle = pLibelle;
    }

    public String getQuantite() {
        return quantite;
    }
    public void setQuantite(final String pQuantite) {
        quantite = pQuantite;
    }
}
```

C. Mise à jour des fichiers java suivants :

- Dans l'interface « IListeCoursesDAO » de la DAO, nous ajoutons la méthode recevant une entité en paramètre.

Fichier : IListeCoursesDAO.java

```
void creerCourse(final Course pCourse);
```

- Tandis que dans l'interface « IServiceListeCourses » du service, nous ajoutons la méthode recevant en paramètres les valeurs de l'entité.

Fichier : IServiceListeCourses.java

```
void creerCourse(final String pLibelle, final Integer pQuantite);
```

- L'implémentation de la méthode de la DAO « ListeCoursesDAO » sauve la nouvelle entité en base.

Fichier : ListeCoursesDAO.java

```
public void creerCourse(final Course pCourse) {  
    entityManager.persist(pCourse);  
}
```

- L'implémentation de la méthode du service « ServiceListeCourses » constitue l'entité et l'envoie en paramètre à la DAO. Cette fois-ci la transaction n'est pas en lecture seule.

Fichier : ServiceListeCourses.java

@Transactional

```
public void creerCourse(final String pLibelle, final Integer pQuantite) {  
    final Course lCourse = new Course();  
    lCourse.setLibelle(pLibelle);  
    lCourse.setQuantite(pQuantite);  
    dao.creerCourse(lCourse);  
}
```

D. Création du Controller de mise à jour :

Synoptique :

Le contrôleur « CreerListeCoursesController » comporte deux méthodes « afficher » et « creer ». La méthode « afficher » place la liste des courses dans l'attribut « listeCourses » et initialise le formulaire « creation » s'il n'est pas déjà présent dans l'attribut « creation ».

L'annotation « @ModelAttribute » (existant depuis Spring 2.5) de la méthode « creer » indique que le paramètre « pCreation » est constitué à partir de l'attribut « creation ».

L'annotation « @Valid » indique que le formulaire doit être validé grâce aux annotations contenues dans la classe de formulaire « CreationForm ».

Ensuite, la méthode « creer » appelle la méthode de création en base de données s'il n'y a pas d'erreurs dans la validation, puis appelle simplement la méthode « afficher » pour l'affichage de la page.

Les messages d'erreur sont présents dans le fichier « messages_fr.properties ».

On peut remarquer que dans ce cas les clés des messages sont de la forme « contrainteValidation.nomAttribut.nomChamp » (par exemple « NotEmpty.creation.libelle »).

Créer la classe : CreerListeCoursesController.java

```
package com.M2i.web.controller;
import java.util.List;
import javax.validation.Valid;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Controller;
import org.springframework.ui.ModelMap;
import org.springframework.validation.BindingResult;
import org.springframework.web.bind.annotation.ModelAttribute;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;
import com.M2i.web.bean.Course;
import com.M2i.web.services.IServiceListeCourses;

@Controller
public class CreerListeCoursesController {
    @Autowired
    private IServiceListeCourses service;
    @RequestMapping(value="/afficherCreationListeCourses", method =
RequestMethod.GET)
    public String afficher(final ModelMap pModel) {
        final List<Course> IListeCourses = service.rechercherCourses();
        pModel.addAttribute("listeCourses", IListeCourses);
        if (pModel.get("creation") == null) {
            pModel.addAttribute("creation", new CreationForm());
        }
        return "creation";
    }
}
```

```

@RequestMapping(value="/creerCreationListeCourses", method =
RequestMethod.POST)
    public String creer(@Valid @ModelAttribute(value="creation") final
CreationForm pCreation,
    final BindingResult pBindingResult, final ModelMap pModel) {
        if (!pBindingResult.hasErrors()) {
            final Integer lIntQuantite =
                Integer.valueOf(pCreation.getQuantite());
            service.creerCourse(pCreation.getLibelle(), lIntQuantite);
        }
        return afficher(pModel);
    }
}

```

E. Gestion de la page de formulaire (JSP) :

La JSP comporte le formulaire de création d'une nouvelle « course ». Ce formulaire contient également l'affichage des messages d'erreur.

Création de la page : /vues/creation.jsp

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
isElIgnored="false" 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"%>
<%@taglib uri="http://www.springframework.org/tags" prefix="spring"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<title><spring:message code="titre.creation.elementcourses"/></title>
</head>
<body>
    <form:form method="post" modelAttribute="creation"
    action="creerCreationListeCourses">
        <spring:message code="creation.elementcourses.libelle.libelle" />
        <form:input path="libelle"/>
        <b><i><form:errors path="libelle" cssclass="error"/></i></b><br>
        <spring:message code="creation.elementcourses.libelle.quantite"/>
        <form:input path="quantite"/>
        <b><i><form:errors path="quantite" cssclass="error"/></i></b><br>
        <input type="submit"/>
    </form:form>
```

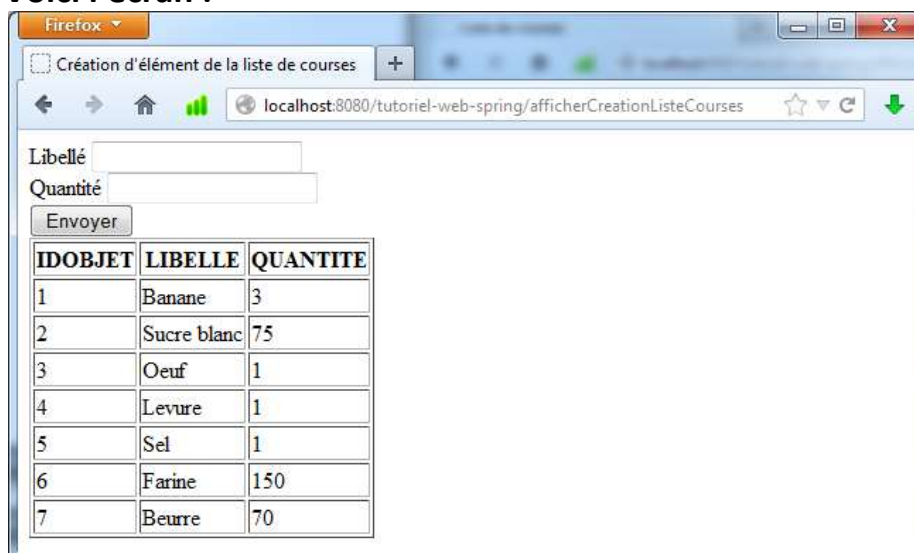


```

<table border="1">
<thead>
<tr>
<th><spring:message code="colonne.identifiant"/></th>
<th><spring:message code="colonne.libelle"/></th>
<th><spring:message code="colonne.quantite"/></th>
</tr>
</thead>
<tbody>
<c:forEach items="${listeCourses}" var="course">
<tr>
<td><c:out value="${course.id}"/></td>
<td><c:out value="${course.libelle}"/></td>
<td><c:out value="${course.quantite}"/></td>
</tr>
</c:forEach>
</tbody>
</table>
</body>
</html>

```

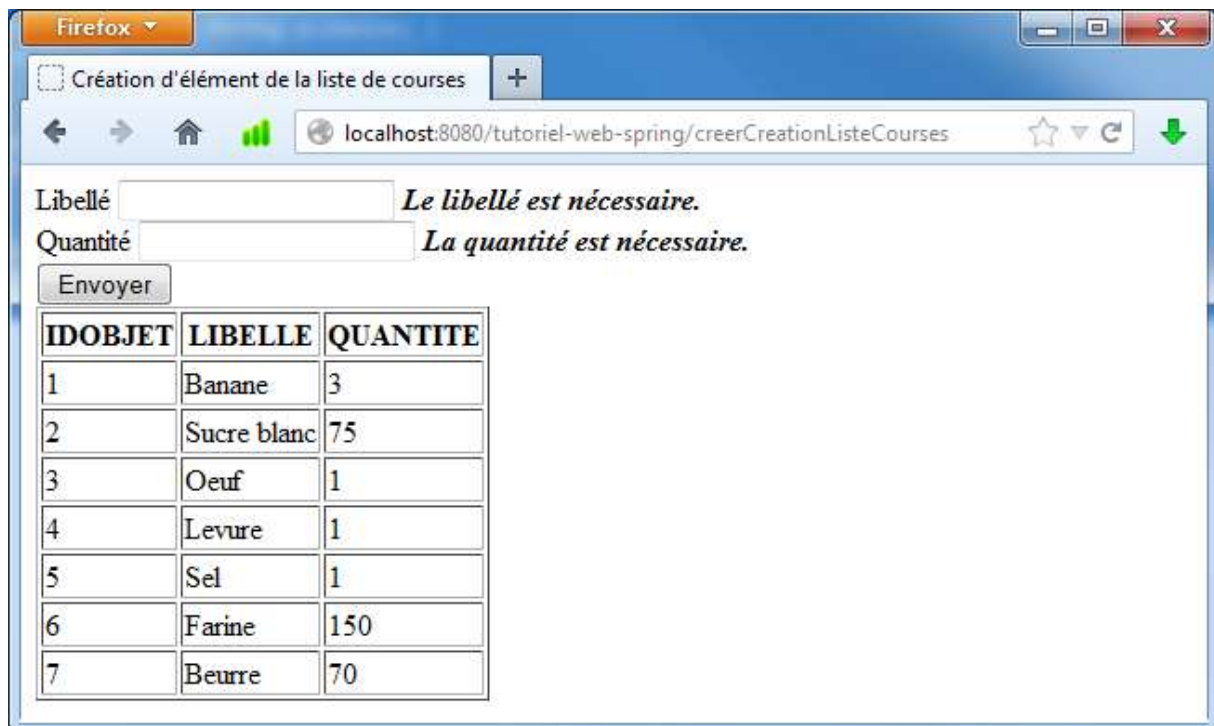
Après déploiement, on obtient le résultat ci-dessous à l'adresse :
<http://localhost:8080/tutoriel-web-spring-M2i/afficherCreationListeCourses>.
 Voici l'écran :



The screenshot shows a web browser window with the address bar displaying `localhost:8080/tutoriel-web-spring/afficherCreationListeCourses`. The page contains a form with two input fields labeled "Libellé" and "Quantité", and an "Envoyer" button. Below the form is a table with three columns: "IDOBJET", "LIBELLE", and "QUANTITE". The table contains seven rows of data:

IDOBJET	LIBELLE	QUANTITE
1	Banane	3
2	Sucre blanc	75
3	Oeuf	1
4	Levure	1
5	Sel	1
6	Farine	150
7	Beurre	70

Voici le résultat de la validation lorsque les champs ne sont pas renseignés.



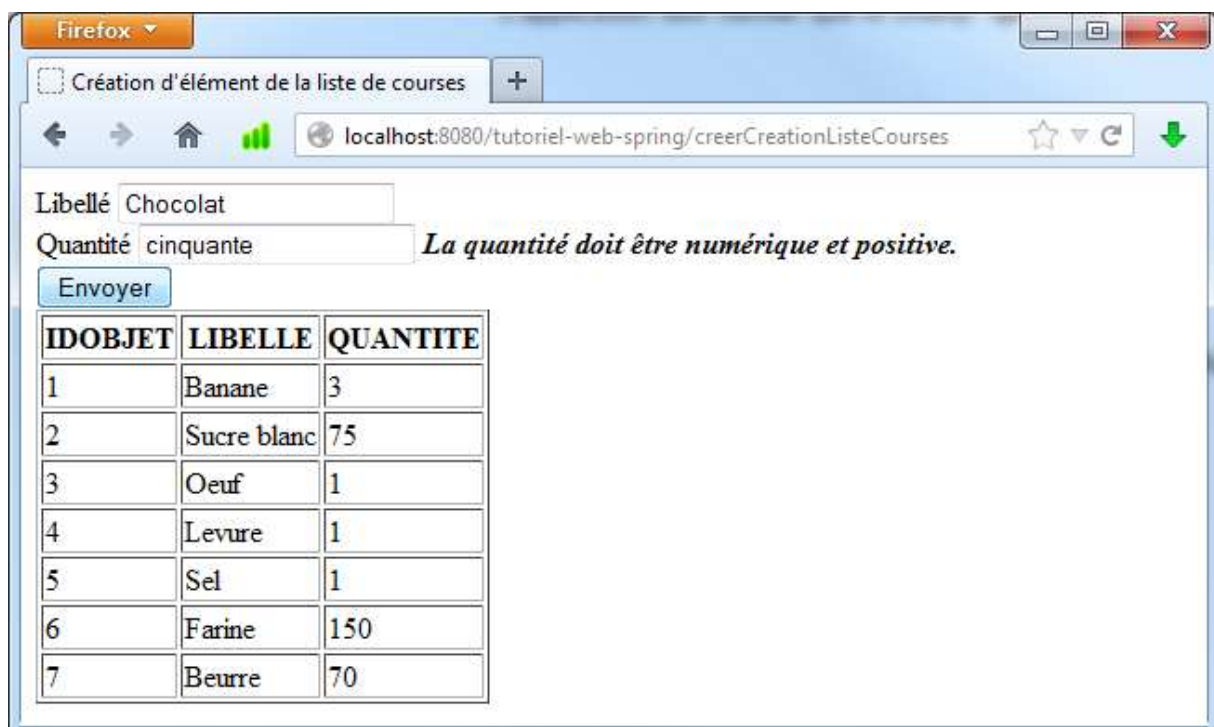
Création d'élément de la liste de courses

Libellé *Le libellé est nécessaire.*

Quantité *La quantité est nécessaire.*

IDOBJET	LIBELLE	QUANTITE
1	Banane	3
2	Sucre blanc	75
3	Oeuf	1
4	Levure	1
5	Sel	1
6	Farine	150
7	Beurre	70

Et le résultat de la validation lorsque le champ « quantité » n'est pas numérique.



Création d'élément de la liste de courses

Libellé

Quantité *La quantité doit être numérique et positive.*

IDOBJET	LIBELLE	QUANTITE
1	Banane	3
2	Sucre blanc	75
3	Oeuf	1
4	Levure	1
5	Sel	1
6	Farine	150
7	Beurre	70

Lorsque le formulaire respecte les contraintes de la validation, l'occurrence est créée en base puis est visible au rafraîchissement de la page.

The screenshot shows a Firefox browser window with the address bar displaying `localhost:8080/tutoriel-web-spring/creerCreationListeCourses`. The page title is "Création d'élément de la liste de courses". The form contains two input fields: "Libellé" with the value "Chocolat" and "Quantité" with the value "50". Below these fields is an "Envoyer" button. Underneath the form is a table with three columns: "IDOBJET", "LIBELLE", and "QUANTITE". The table contains eight rows of data.

IDOBJET	LIBELLE	QUANTITE
1	Banane	3
2	Sucre blanc	75
3	Oeuf	1
4	Levure	1
5	Sel	1
6	Farine	150
7	Beurre	70
8	Chocolat	50

XI. Suppression de données en base

A. Objectifs :

Dans ce paragraphe, nous allons supprimer des données dans la base de données. Pour cela, nous allons créer des liens avec paramètre qui permettront de déterminer l'occurrence à supprimer.

B. Gestion des fichiers « properties » :

Il faut ajouter de nouveaux libellés dans le fichier « messages_fr.properties ».

titre.suppression.elementcourses=Suppression d'élément de la liste de
courses

suppression.supprimer.libelle=Supprime

C. Modification des fichiers java suivants :

- Dans l'interface « IListeCoursesDAO » de la DAO, nous ajoutons la méthode de suppression recevant une entité en paramètre.

Dans le fichier : IListeCoursesDAO.java

```
void supprimerCourse(final Course pCourse);
```

- Tandis que dans l'interface « IServiceListeCourses » du service, nous ajoutons la méthode recevant en paramètre l'identifiant de l'entité.

Dans le fichier : IServiceListeCourses.java

```
void supprimerCourse(final Integer pIdCourse);
```

- L'implémentation de la méthode de la DAO « ListeCoursesDAO » supprime l'entité en base.

Dans le fichier : ListeCoursesDAO.java

```
public void supprimerCourse(final Course pCourse) {  
    final Course lCourse = entityManager.getReference(Course.class,  
    pCourse.getId());  
    entityManager.remove(lCourse);  
}
```

- L'implémentation de la méthode du service « ServiceListeCourses » instancie une entité avec l'identifiant et l'envoie en paramètre à la DAO.

Dans le fichier : ServiceListeCourses.java

@Transactional

```
public void supprimerCourse(final Integer pIdCourse) {  
    final Course lCourse = new Course();  
    lCourse.setId(pIdCourse);  
    dao.supprimerCourse(lCourse);  
}
```

D. Création du controller de gestion de suppression :

Synoptique :

Le contrôleur « SupprimerListeCoursesController » comporte deux méthodes « afficher » et « supprimer ».

La méthode « afficher » place la liste des courses dans l'attribut « listeCourses ». Ensuite, la méthode « supprimer » utilise le paramètre « idCourse » de la requête pour appeler la méthode « supprimerCourse » du service, puis elle relance l'affichage de la page.

Création de la classe : SupprimerListeCoursesController.java

```
package com.M2i.web.controller;
import java.util.List;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Controller;
import org.springframework.ui.ModelMap;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;
import org.springframework.web.bind.annotation.RequestParam;
import com.M2i.web.bean.Course;
import com.M2i.web.services.IServiceListeCourses;
@Controller
public class SupprimerListeCoursesController {
    @Autowired
    private IServiceListeCourses service;
    @RequestMapping(value="/afficherSuppressionListeCourses", method
    = RequestMethod.GET)
    public String afficher(final ModelMap pModel) {
        final List<Course> lListeCourses = service.rechercherCourses();
        pModel.addAttribute("listeCourses", lListeCourses);
        return "suppression";
    }
}
```

```

    @RequestMapping(value="/supprimerSuppressionListeCourses",
    method = RequestMethod.GET)
    public String supprimer(@RequestParam(value="idCourse") final
    Integer pIdCourse, final
    ModelMap pModel) {
        service.supprimerCourse(pIdCourse);
        return afficher(pModel);
    }
}

```

E. Création la page JSP pour la suppression :

La JSP affiche un tableau des différents éléments de la liste de courses. Ce tableau comporte une colonne avec un lien pour supprimer.

Création de la page : /vues/suppression.jsp

```

<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
isELIgnored="false"
pageEncoding="ISO-8859-1"%>
<%@taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c"%>
<%@taglib uri="http://www.springframework.org/tags" prefix="spring"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/
loose.dtd">
<html>
    <head>
        <title><spring:message
code="titre.suppression.elementcourses"/></title>
    </head>

```

```

<body>
<table border="1">
<thead>
  <tr>
    <th><spring:message code="colonne.identifiant"/></th>
    <th><spring:message code="colonne.libelle"/></th>
    <th><spring:message code="colonne.quantite"/></th>
    <th>&nbsp;</th>
  </tr>
</thead>
<tbody>
  <c:forEach items="{listeCourses}" var="course">
    <tr>
      <td><c:out value="{course.id}"/></td>
      <td><c:out value="{course.libelle}"/></td>
      <td><c:out value="{course.quantite}"/></td>
      <td>
        <c:url value="/supprimerSuppressionListeCourses" var="url">
          <c:param name="idCourse" value="{course.id}"/>
        </c:url>
        <a href="{url}">
          <spring:message code="suppression.supprimer.libelle" />
        </a>
      </td>
    </tr>
  </c:forEach>
</tbody>
</table>
</body>
</html>

```


Après déploiement, on obtient le résultat ci-dessous à l'adresse :
<http://localhost:8080/tutoriel-web-spring-M2i/afficherSuppressionListeCourses>.



IDOBJET	LIBELLE	QUANTITE	
1	Banane	3	Supprimer
2	Sucre blanc	75	Supprimer
3	Oeuf	1	Supprimer
4	Levure	1	Supprimer
5	Sel	1	Supprimer
6	Farine	150	Supprimer
7	Beurre	70	Supprimer
8	Chocolat	50	Supprimer

Voici le résultat après avoir cliqué sur le dernier lien de suppression.



IDOBJET	LIBELLE	QUANTITE	
1	Banane	3	Supprimer
2	Sucre blanc	75	Supprimer
3	Oeuf	1	Supprimer
4	Levure	1	Supprimer
5	Sel	1	Supprimer
6	Farine	150	Supprimer
7	Beurre	70	Supprimer

XII. Modification des données en base :

A. Objectifs :

Dans ce paragraphe, nous allons modifier des données dans la base de données. Pour cela, nous allons créer un formulaire avec une liste de valeurs qui seront mises à jour dans la base de données.

B. Gestion des « properties » :

Il faut ajouter de nouveaux libellés dans le fichier « messages_fr.properties ».

titre.modification.elementcourses=Modification d'élément de la liste de courses

Il faut créer un nouveau fichier d'internationalisation

« ValidationMessages_fr.properties » (également dans le dossier « src/main/resources »).

modification.course.quantite.notempty=La quantité est nécessaire.

modification.course.quantite.numerique=La quantité doit être numérique et positive.

C. Mise à jour des fichiers java suivants :

- Dans l'interface « IListeCoursesDAO » de la DAO, nous ajoutons la méthode de modification recevant une entité en paramètre.

Dans le fichier : IListeCoursesDAO.java

```
void modifierCourse(final Course pCourse);
```

- Tandis que dans l'interface « IServiceListeCourses » du service, nous ajoutons la méthode recevant en paramètres une liste d'entités.

Dans le fichier : IServiceListeCourses.java

```
void modifierCourses(final List<Course> pListeCourses);
```

L'implémentation de la méthode de la DAO « ListeCoursesDAO » supprime l'entité en base. Elle lève une exception uniquement si la requête modifie un nombre d'occurrences différent de 1 (ce qui aura pour conséquence de provoquer un rollback de transaction au niveau du service). Le libellé de l'exception contient le texte de la requête SQL.

Mise à jour du fichier : ListeCoursesDAO.java

```
public void modifierCourse(final Course pCourse) {
    final CriteriaBuilder ICriteriaBuilder =
        entityManager.getCriteriaBuilder();
    final CriteriaUpdate<Course> ICriteriaUpdate =
        ICriteriaBuilder.createCriteriaUpdate(Course.class);
    final Root<Course> IRoot = ICriteriaUpdate.from(Course.class);
    final Path<Course> IPath = IRoot.get("id");
    final Expression<Boolean> IExpression = ICriteriaBuilder.equal(IPath,
        pCourse.getId());
    ICriteriaUpdate.where(IExpression);
    ICriteriaUpdate.set("quantite", pCourse.getQuantite());
    final Query IQuery = entityManager.createQuery(ICriteriaUpdate);
    final int IRowCount = IQuery.executeUpdate();
    if (IRowCount != 1) {
        final org.hibernate.Query IHQuery =
            IQuery.unwrap(org.hibernate.Query.class);
        final String ISql = IHQuery.getQueryString();
        throw new RuntimeException("Nombre d'occurrences (" + IRowCount +
            ") modifiés différent de 1 pour " + ISql);
    }
}
```

L'implémentation de la méthode du service « ServiceListeCourses » parcourt les entités de la liste pour les passer en paramètre l'une après l'autre à la DAO.

Mise à jour du fichier : ServiceListeCourses.java

@Transactional

```
public void modifierCourses(final List<Course> pListeCourses) {  
    for (final Course lCourse : pListeCourses) {  
        dao.modifierCourse(lCourse);  
    }  
}
```

D. Gestion de classe utilitaire :

Synoptique :

Le formulaire « ModificationForm » contient une liste qui comporte l'annotation « @Valid ». Cela provoquera la validation de chaque élément de la liste lorsque le formulaire sera validé en entrée de la méthode « modifier » du contrôleur.

Création du fichier : ModificationForm.java

```
package com.M2i.web.controller;
import java.util.List;
import javax.validation.Valid;
public class ModificationForm {
    @Valid
    private List<ModificationCourse> listeCourses;
    public void setListeCourses(final List<ModificationCourse>
pListeCourses) {
        listeCourses = pListeCourses;
    }
    public List<ModificationCourse> getListeCourses() {
        return listeCourses;
    }
}
```

Synoptique :

La classe « **ModificationCourse** » correspond à un élément de la liste contenue dans le formulaire « **ModificationForm** ». Nous pouvons remarquer que les annotations « **NotEmpty** » et « **Pattern** » comportent des valeurs « **message** » qui correspondent aux messages internationalisés contenus dans le fichier « **ValidationMessages_fr.properties** ».

Création du fichier : **ModificationCourse.java**

```
package com.M2i.web.controller;
import javax.validation.constraints.Pattern;
import org.hibernate.validator.constraints.NotEmpty;
public class ModificationCourse {
    private Integer id;
    private String libelle;
    @NotEmpty(message="{modification.course.quantite.notempty}")
    @Pattern(regex="\\d*",
    message="{modification.course.quantite.numerique}")
    private String quantite;
    public Integer getId() {
        return id;
    }
    public void setId(final Integer pId) {
        id = pId;
    }
    public String getLibelle() {
        return libelle;
    }
    public void setLibelle(final String pLibelle) {
        libelle = pLibelle;
    }
    public String getQuantite() {
        return quantite;
    }
    public void setQuantite(final String pQuantite) {
        quantite = pQuantite;
    }
}
```

E. Création du Controller de modification :

Synoptique :

Le contrôleur « **ModifierListeCoursesController** » comporte les méthodes « **afficher** » et « **modifier** ». La méthode « **afficher** » place la liste des courses dans le formulaire et la méthode « **modifier** » récupère la liste des courses du formulaire pour appeler la méthode de modification du service.

Création du fichier : **ModifierListeCoursesController.java**

```
package com.M2i.web.controller;
import java.util.LinkedList;
import java.util.List;
import javax.validation.Valid;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Controller;
import org.springframework.ui.ModelMap;
import org.springframework.validation.BindingResult;
import org.springframework.web.bind.annotation.ModelAttribute;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;
import com.M2i.web.bean.Course;
import com.M2i.web.services.IServiceListeCourses;
```

@Controller

```
public class ModifierListeCoursesController {  
    @Autowired  
    private IServiceListeCourses service;  
    @RequestMapping(value="/afficherModificationListeCourses", method  
    = RequestMethod.GET)  
    public String afficher(final ModelMap pModel) {  
        if (pModel.get("modification") == null) {  
            final List<Course> IListeCourses = service.rechercherCourses();  
            final ModificationForm IModificationForm = new ModificationForm();  
            final List<ModificationCourse> IListe = new  
            LinkedList<ModificationCourse>();  
            for (final Course ICourse : IListeCourses) {  
                final ModificationCourse IModificationCourse = new  
                ModificationCourse();  
                IModificationCourse.setId(ICourse.getId());  
                IModificationCourse.setLibelle(ICourse.getLibelle());  
                IModificationCourse.setQuantite(ICourse.getQuantite().toString());  
                IListe.add(IModificationCourse);  
            }  
            IModificationForm.setListeCourses(IListe);  
            pModel.addAttribute("modification", IModificationForm);  
        }  
        return "modification";  
    }
```



```

    @RequestMapping(value="/modifierModificationListeCourses",
    method = RequestMethod.POST)
    public String modifier(@Valid @ModelAttribute(value="modification")
    final ModificationForm
    pModification,
    final BindingResult pBindingResult, final ModelMap pModel) {
    if (!pBindingResult.hasErrors()) {
    final List<Course> lListeCourses = new LinkedList<Course>();
    for (final ModificationCourse lModificationCourse :
    pModification.getListeCourses()) {
    final Course lCourse = new Course();
    lCourse.setId(lModificationCourse.getId());
    final Integer lQuantite =
    Integer.valueOf(lModificationCourse.getQuantite());
    lCourse.setQuantite(lQuantite);
    lListeCourses.add(lCourse);
    }
    service.modifierCourses(lListeCourses);
    }
    return afficher(pModel);
    }
}

```

F. Gestion de la page JSP pour la modification :

Synoptique :

La JSP affiche un tableau des différents éléments de la liste de courses. La colonne des quantités comporte des champs de saisie permettant de modifier les différentes quantités. Nous remarquons l'utilisation de la variable « status » qui permet de nommer les champs du formulaire et de filtrer les messages d'erreur selon l'occurrence de la liste des courses.

Création la page JSP : /vues/modification.jsp

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
isELIgnored="false" 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"%>

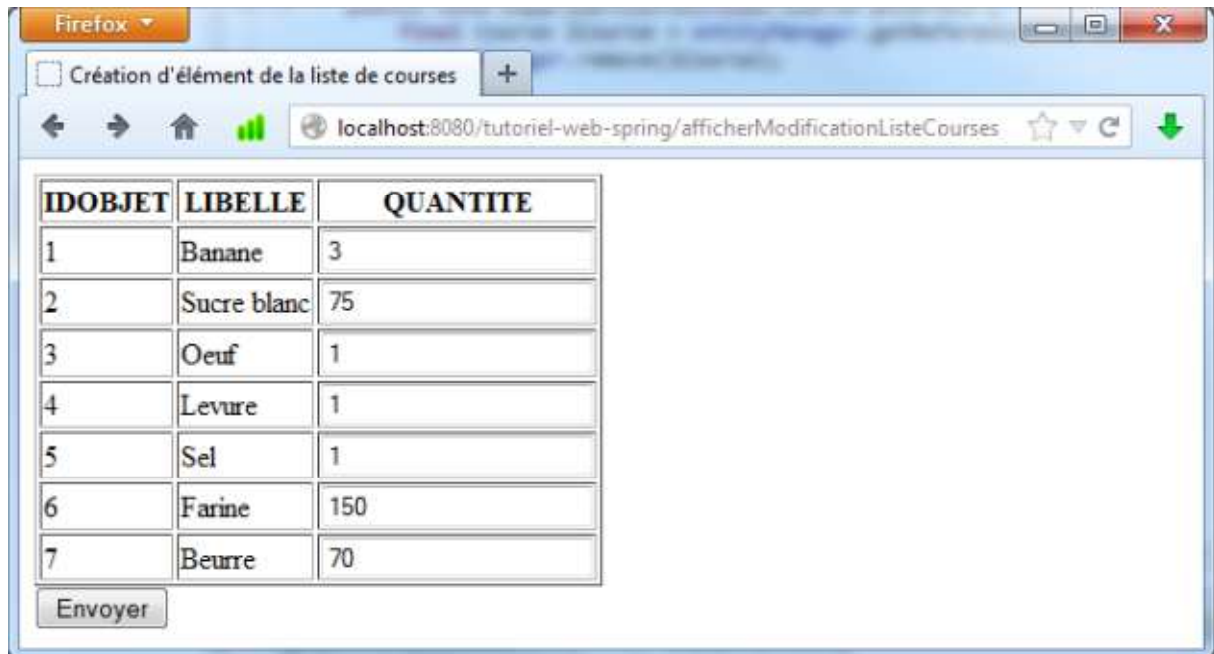
<%@taglib uri="http://www.springframework.org/tags" prefix="spring"%>
<%@taglib uri="http://www.springframework.org/tags" prefix="spring"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
    <head>
        <title><spring:message code="titre.creation.elementcourses"/></title>
    </head>
```

```

<body>
  <form:form method="post" modelAttribute="modification"
  action="modifierModificationListeCourses">
    <table border="1">
      <thead>
        <tr>
          <th><spring:message code="colonne.identifiant"/></th>
          <th><spring:message code="colonne.libelle"/></th>
          <th><spring:message code="colonne.quantite"/></th>
        </tr>
      </thead>
      <tbody>
        <c:forEach items="${modification.listeCourses}" var="course"
        varStatus="status">
          <tr>
            <td><c:out value="${course.id}"/>
            <input type="hidden" name="listeCourses[${status.index}].id"
            value="${course.id}"/></td>
            <td><c:out value="${course.libelle}"/>
            <input type="hidden"
            name="listeCourses[${status.index}].libelle"
            value="${course.libelle}"/></td>
            <td><input type="text"
            name="listeCourses[${status.index}].quantite"
            value="${course.quantite}"/><br/>
            <b><i><form:errors path="listeCourses[${status.index}].quantite"
            /></i></b></td>
          </tr>
        </c:forEach>
      </tbody>
    </table>
    <input type="submit"/>
  </form:form>
</body>
</html>

```

Après déploiement, on obtient le résultat ci-dessous à l'adresse :
<http://localhost:8080/tutoriel-web-spring-M2i/afficherModificationListeCourses>.



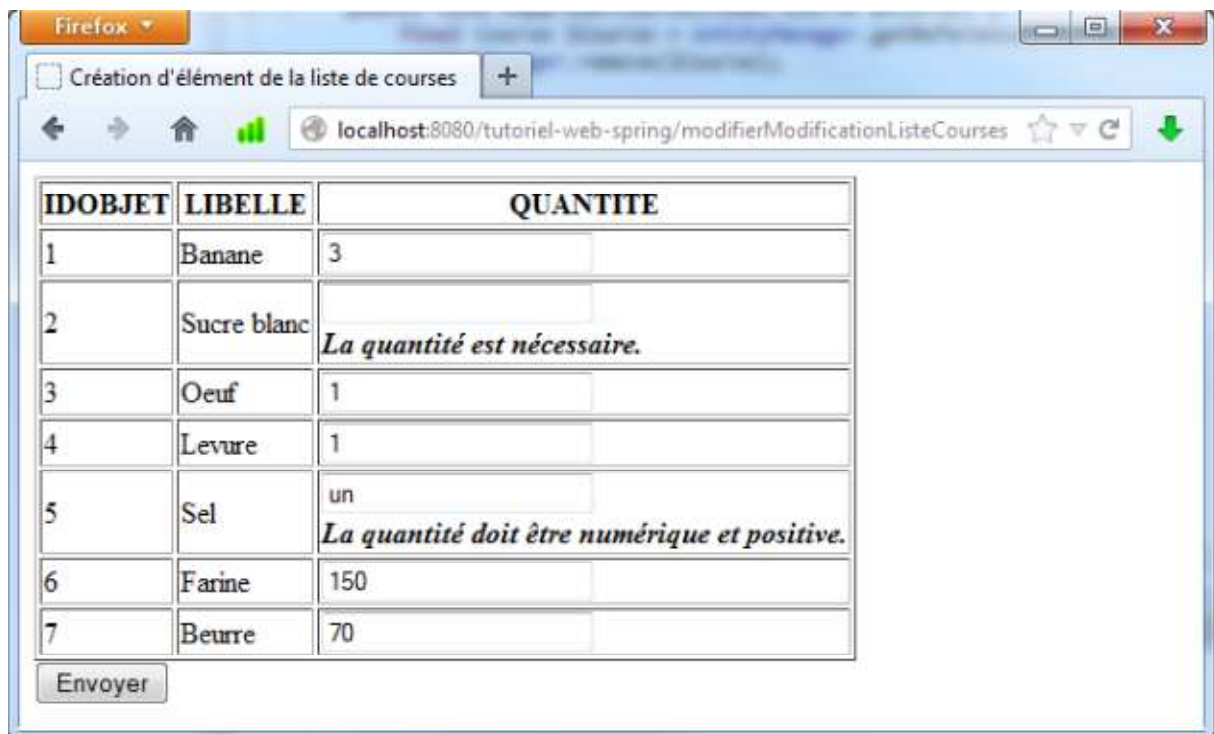
Création d'élément de la liste de courses +

localhost:8080/tutoriel-web-spring/afficherModificationListeCourses

IDOBJET	LIBELLE	QUANTITE
1	Banane	3
2	Sucre blanc	75
3	Oeuf	1
4	Levure	1
5	Sel	1
6	Farine	150
7	Beurre	70

Envoyer

Voici le résultat d'une validation en erreur.



Création d'élément de la liste de courses +

localhost:8080/tutoriel-web-spring/modifierModificationListeCourses

IDOBJET	LIBELLE	QUANTITE
1	Banane	3
2	Sucre blanc	<i>La quantité est nécessaire.</i>
3	Oeuf	un
4	Levure	1
5	Sel	<i>La quantité doit être numérique et positive.</i>
6	Farine	150
7	Beurre	70

Envoyer

Et celui d'une modification réussie.

Firefox

Création d'élément de la liste de courses

localhost:8080/tutoriel-web-spring/modifierModificationListeCourses

IDOBJET	LIBELLE	QUANTITE
1	Banane	30
2	Sucre blanc	75
3	Oeuf	1
4	Levure	1
5	Sel	1
6	Farine	150
7	Beurre	70

Envoyer

XIII. Unification de l'application par un menu

Dans ce paragraphe, nous allons ajouter aux pages Web précédemment créées un menu qui permettra de naviguer dans l'application. Pour cela, nous allons utiliser les Tiles. Il faut donc rajouter les dépendances Maven suivantes. Les dépendances « tiles-jsp » et « slf4j-log4j12 » permettent l'utilisation de Tiles.

Ce fichier du Framework Spring MVC : c'est d'avoir des templates JSP qui seront inclus dans un template principal qui sera un menu.

Fichier POM.XML à vérifier :

```
<dependency>
  <groupId>org.apache.tiles</groupId>
  <artifactId>tiles-jsp</artifactId>
  <version>3.0.8</version>
</dependency>
```

```
<dependency>
  <groupId>org.slf4j</groupId>
  <artifactId>slf4j-log4j12</artifactId>
  <version>1.5.11</version>
</dependency>
```

(Si absent ou version différente mettre à jour.)

A. Gestion des fichiers « properties » :

Il faut ajouter un nouveau libellé dans le fichier « messages_fr.properties ».
titre.application=Application de liste de courses

B. Mise à jour des fichiers XML :

Il faut modifier le fichier « dispatcher-servlet.xml » comme ci-dessous. Le bean « `InternalResourceViewResolver` » (déjà présent dans le fichier) doit se trouver après les nouvelles déclarations (simplement parce que Spring teste les « `ViewResolver` » dans l'ordre de déclaration, ce qui permet de résoudre une ressource avec « `UrlBasedViewResolver` » puis d'essayer avec « `InternalResourceViewResolver` » en deuxième).

Le bean « `UrlBasedViewResolver` » (depuis Spring 1.0) utilise « `TilesView` » pour traiter les vues. Le bean « `TilesConfigurer` » charge la configuration dans le fichier « `/WEB-INF/tiles.xml` ».

Fichier : `/WEB-INF/dispatcher-servlet.xml` (à mettre à jour)

```
<bean
class="org.springframework.web.servlet.view.UrlBasedViewResolver">
<property name="viewClass"
value="org.springframework.web.servlet.view.tiles3.TilesView" />
</bean>
```

```
<bean id="tilesConfigurer"
class="org.springframework.web.servlet.view.tiles3.TilesConfigurer">
<property name="definitions">
<list>
<value>/WEB-INF/tiles.xml</value>
</list>
</property>
</bean>
```

Le fichier « `tiles.xml` » contient la définition des différents tiles. Nous reprenons les noms des ressources utilisées dans les contrôleurs, ce qui nous évite de modifier les contrôleurs.

Fichier : /WEB-INF/tiles.xml (à créer)

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<!DOCTYPE tiles-definitions PUBLIC
"-//Apache Software Foundation//DTD Tiles Configuration 3.0//EN"
"http://tiles.apache.org/dtds/tiles-config_3_0.dtd">
<tiles-definitions>
<definition name="listeCourses" template="/vues/page.jsp">
<put-attribute name="principal" value="/vues/listeCourses.jsp" />
</definition>
<definition name="creation" template="/vues/page.jsp">
<put-attribute name="principal" value="/vues/creation.jsp" />
</definition>
<definition name="suppression" template="/vues/page.jsp">
<put-attribute name="principal" value="/vues/suppression.jsp" />
</definition>
<definition name="modification" template="/vues/page.jsp">
<put-attribute name="principal" value="/vues/modification.jsp" />
</definition>
</tiles-definitions>
```

C. Gestion des pages JSP :

La JSP « page.jsp » contient le squelette des pages (le « template » dans le fichier « tiles.xml »). Le tag « tiles:insertAttribute » permet d'inclure les JSP contenant le corps de la page et définis dans le fichier « tiles.xml ».

Fichier : /vues/page.jsp (à créer)

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
isELIgnored="false"
pageEncoding="ISO-8859-1"%>
<%@taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c"%>
<%@taglib uri="http://www.springframework.org/tags" prefix="spring"%>
<%@taglib uri="http://tiles.apache.org/tags-tiles" prefix="tiles"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/
loose.dtd">
```



```

<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<title><spring:message code="titre.application"/></title>
</head>
<body>
<table>
<tbody>
<tr>
<td valign="top">
<table>
<tbody>
<tr><td>
<c:url value="/afficherListeCourses" var="url" />
<a href="{url}">
<spring:message code="titre.listecourses"/>
</a>
</td></tr>
<tr><td>
<c:url value="/afficherCreationListeCourses" var="url" />
<a href="{url}">
<spring:message code="titre.creation.elementcourses"/>
</a>
</td></tr>
<tr><td>
<c:url value="/afficherSuppressionListeCourses" var="url" />
<a href="{url}">
<spring:message code="titre.suppression.elementcourses"/>
</a>
</td></tr>
<tr><td>
<c:url value="/afficherModificationListeCourses" var="url" />
<a href="{url}">
<spring:message code="titre.modification.elementcourses"/>
</a>
</td></tr>
</tbody>
</table>

```

```

</td>
<td valign="top">
<tiles:insertAttribute name="principal" />
</td>
</tr>
</tbody>
</table>
</body>
</html>

```

Modification des fichiers JSP existants :

Il faut modifier les JSP « listeCourses.jsp », « creation.jsp », « suppression.jsp » et « modification.jsp » pour ne garder que les parties dans la balise « body ».

Fichier : /vues/listeCourses.jsp (à modifier)

```

<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
isELIgnored="false"
pageEncoding="ISO-8859-1"%>
<%@taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c"%>
<%@taglib uri="http://www.springframework.org/tags" prefix="spring"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/
loose.dtd">
<table border="1">
<thead>
<tr>
<th><spring:message code="colonne.identifiant"/></th>
<th><spring:message code="colonne.libelle"/></th>
<th><spring:message code="colonne.quantite"/></th>
</tr>
</thead>

```

```

<tbody>
<c:forEach items="${listeCourses}" var="course">
<tr>
<td><c:out value="${course.id}"/></td>
<td><c:out value="${course.libelle}"/></td>
<td><c:out value="${course.quantite}"/></td>
</tr>
</c:forEach>
</tbody>
</table>

```

Fichier : /vues/creation.jsp (à modifier)

```

<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
isELIgnored="false"
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"%>
<%@taglib uri="http://www.springframework.org/tags" prefix="spring"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/
loose.dtd">
<form:form method="post" modelAttribute="creation"
action="creerCreationListeCourses">
<spring:message code="creation.elementcourses.libelle.libelle" />
<form:input path="libelle"/>
<b><i><form:errors path="libelle" cssclass="error"/></i></b><br>
<spring:message code="creation.elementcourses.libelle.quantite"/>
<form:input path="quantite"/>
<b><i><form:errors path="quantite" cssclass="error"/></i></b><br>
<input type="submit"/>
</form:form>
<table border="1">

```

```

<thead>
<tr>
<th><spring:message code="colonne.identifiant"/></th>
<th><spring:message code="colonne.libelle"/></th>
<th><spring:message code="colonne.quantite"/></th>
</tr>
</thead>
<tbody>
<c:forEach items="{listeCourses}" var="course">
<tr>
<td><c:out value="{course.id}"/></td>
<td><c:out value="{course.libelle}"/></td>
<td><c:out value="{course.quantite}"/></td>
</tr>
</c:forEach>
</tbody>
</table>

```

Fichier : /vues/suppression.jsp (à modifier)

```

<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
isElIgnored="false"
pageEncoding="ISO-8859-1"%>
<%@taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c"%>
<%@taglib uri="http://www.springframework.org/tags" prefix="spring"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/
loose.dtd">
<table border="1">
<thead>
<tr>
<th><spring:message code="colonne.identifiant"/></th>
<th><spring:message code="colonne.libelle"/></th>
<th><spring:message code="colonne.quantite"/></th>
<th>&nbsp;</th>
</tr>
</thead>

```

```

<tbody>
<c:forEach items="{listCourses}" var="course">
<tr>
<td><c:out value="{course.id}"/></td>
<td><c:out value="{course.libelle}"/></td>
<td><c:out value="{course.quantite}"/></td>
<td>
<c:url value="/supprimerSuppressionListeCourses" var="url">
<c:param name="idCourse" value="{course.id}"/>
</c:url>
<a href="{url}">
<spring:message code="suppression.supprimer.libelle" />
</a>
</td>
</tr>
</c:forEach>
</tbody>
</table>

```

Fichier : /vues/modification.jsp (à modifier)

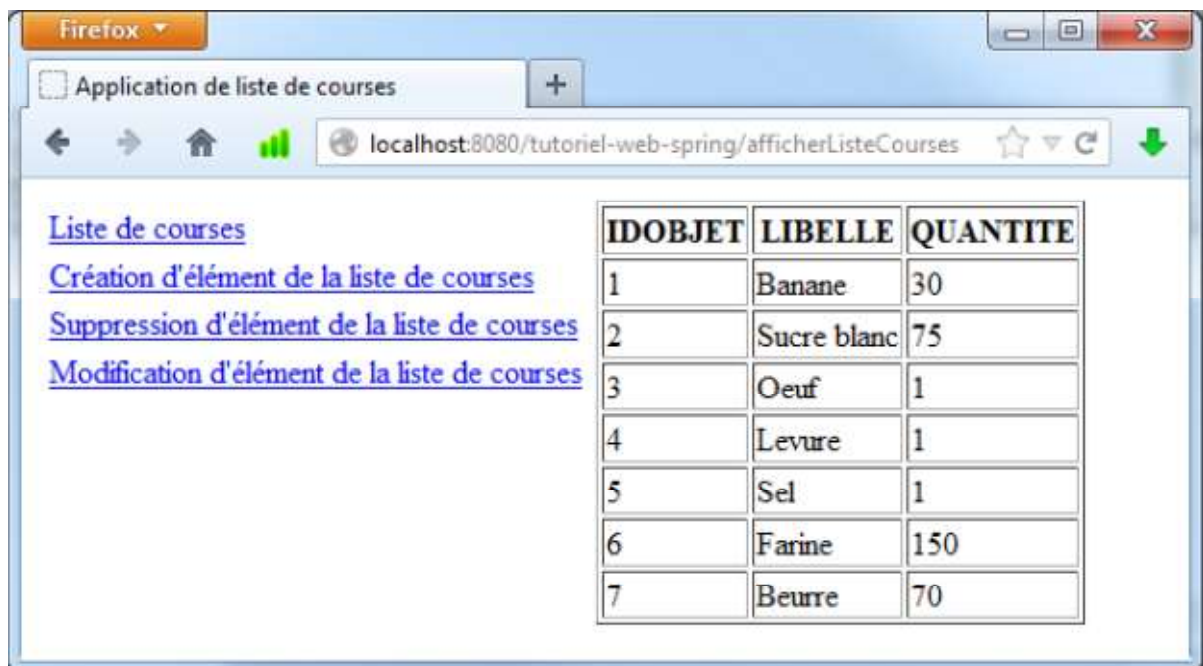
```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
isElIgnored="false"
pageEncoding="ISO-8859-1"%>
<%@taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c"%>
<%@taglib uri="http://www.springframework.org/tags/form"
prefix="form"%>
<%@taglib uri="http://www.springframework.org/tags" prefix="spring"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/
loose.dtd">
<form:form method="post" modelAttribute="modification"
action="modifierModificationListeCourses">
<table border="1">
<thead>
<tr>
<th><spring:message code="colonne.identifiant"/></th>
<th><spring:message code="colonne.libelle"/></th>
<th><spring:message code="colonne.quantite"/></th>
</tr>
</thead>
<tbody>
<c:forEach items="${modification.listeCourses}" var="course"
varStatus="status">
<tr>
<td>
<c:out value="${course.id}"/>
<input type="hidden" name="listeCourses[${status.index}].id"
value="${course.id}"/>
</td>
<td>
<c:out value="${course.libelle}"/>
<input type="hidden" name="listeCourses[${status.index}].libelle"
value="${course.libelle}"/>
</td>
```

```

<td>
<input type="text" name="listeCourses[${status.index}].quantite"
value="${course.quantite}"/><br/>
<b><i><form:errors path="listeCourses[${status.index}].quantite" /></i></b>
</td>
</tr>
</c:forEach>
</tbody>
</table>
<input type="submit"/>
</form:form>

```

Après déploiement, on obtient le résultat ci-dessous à l'adresse :
<http://localhost:8080/tutoriel-web-spring-M2i/afficherListeCourses>.



Les liens dans le menu à gauche permettent de naviguer entre les pages.

The screenshot shows a web browser window with the title "Application de liste de courses". The address bar displays "localhost:8080/tutorial-web-spring/afficherCreationListeCourses". On the left side, there is a menu with four links: "Liste de courses", "Création d'élément de la liste de courses", "Suppression d'élément de la liste de courses", and "Modification d'élément de la liste de courses". The main content area contains a form with two input fields labeled "Libellé" and "Quantité", followed by an "Envoyer" button. Below the form is a table with three columns: "IDOBJET", "LIBELLE", and "QUANTITE". The table contains seven rows of data representing grocery items.

IDOBJET	LIBELLE	QUANTITE
1	Banane	30
2	Sucre blanc	75
3	Oeuf	1
4	Levure	1
5	Sel	1
6	Farine	150
7	Beurre	70