Spring Framework :

- Architecture JEE
- Spring IOC
- Spring MVC
- Spring Integration : RMI, JaxWS, JaxRS, JMS, JMX, …
- Spring Security

Mohamed Youssfi

Laboratoire Signaux Systèmes Distribués et Intelligence Artificielle (SSDIA)

ENSET, Université Hassan II Casablanca, Maroc

Email : med@youssfi.net

Supports de cours : http://fr.slideshare.net/mohamedyoussfi9

Chaîne vidéo : http://youtube.com/mohamedYoussfi

Recherche : http://www.researchgate.net/profile/Youssfi_Mohamed/publications

Architecture J2EE

med@youssfi.net | ENSET Université Hassan II Mohammedia

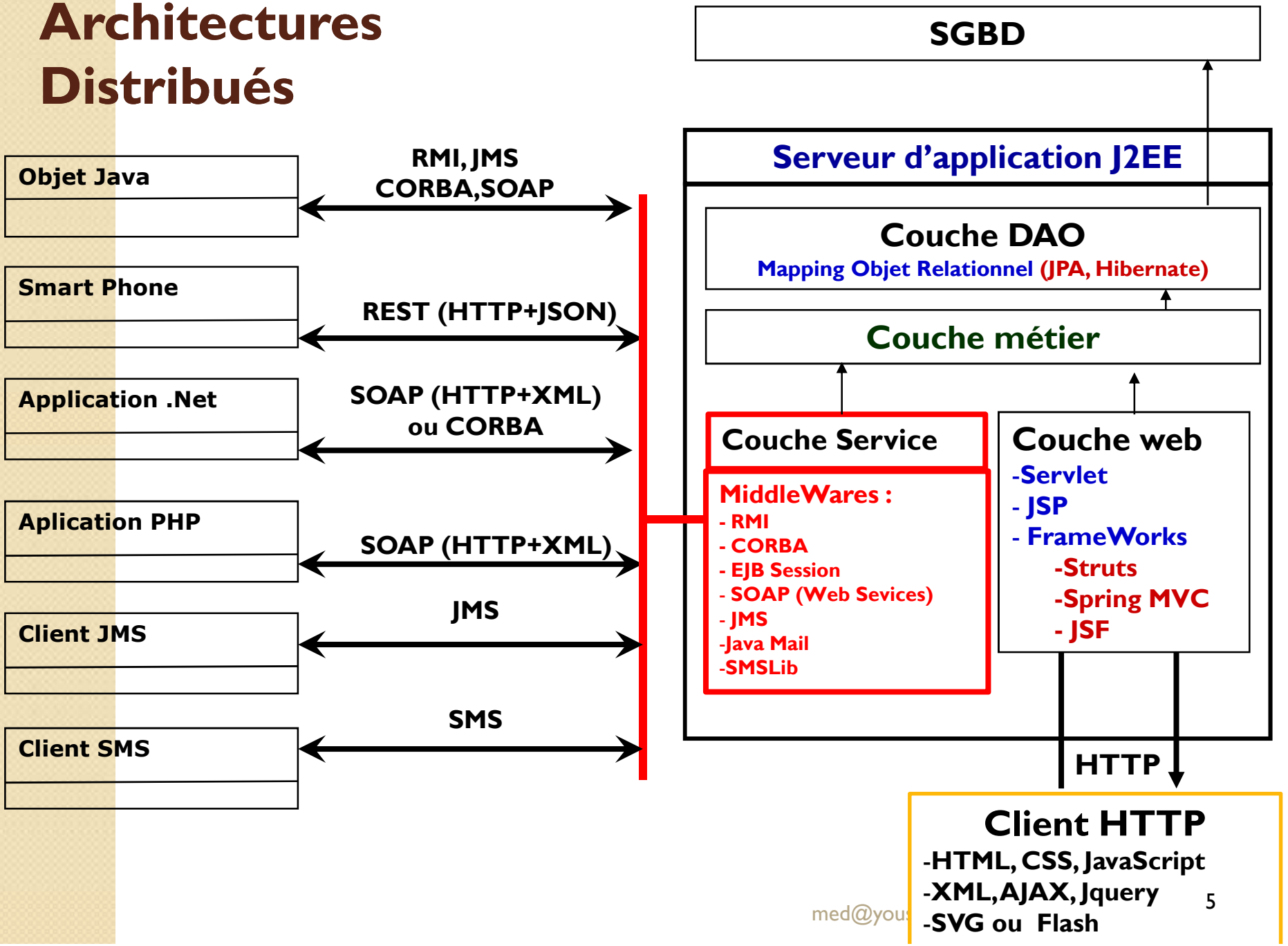
Exigences d'un projet informatique

- Exigences fonctionnelles:
  - Une application est créée pour répondre , tout d'abord, aux besoins fonctionnels des entreprises.
- Exigences Techniques :
  - Les performances:
    - Temps de réponse
    - Haute disponibilité et tolérance aux pannes
    - Eviter le problème de montée en charge
  - La maintenance:
    - Une application doit évoluer dans le temps.
    - Doit être fermée à la modification et ouverte à l'extension
  - Sécurité
  - Portabilité
  - Distribution
  - Capacité de communiquer avec d'autres applications distantes.
  - Capacité de fournir le service à différents type de clients (Desk TOP, Mobile, SMS, http…)
  - …..
  - Coût du logiciel

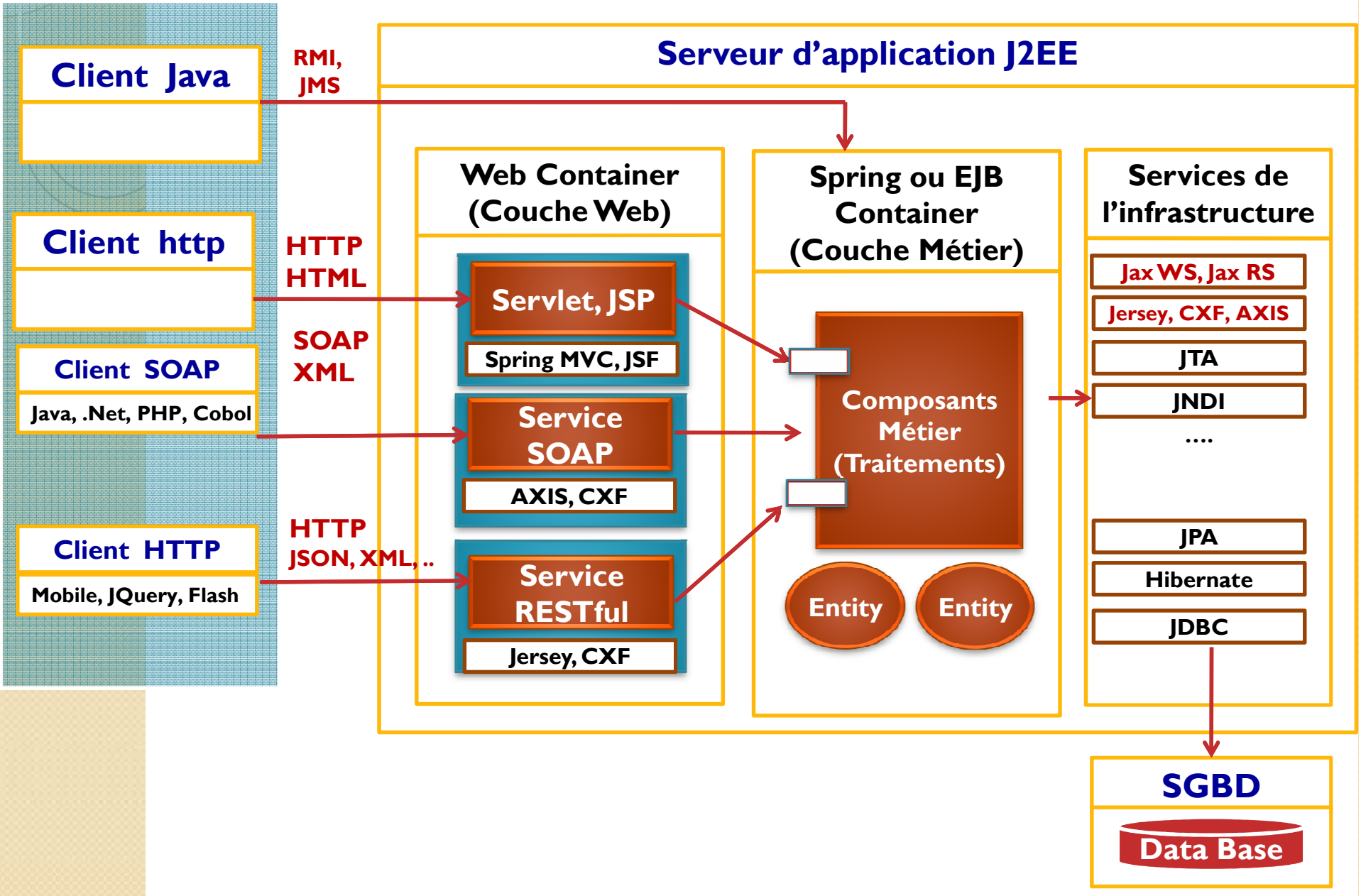
Constat

- Il est très difficile de développer un système logiciel qui respecte ces exigences sans utiliser l'expérience des autres :
  - Serveur d'application JEE:
    - JBOSS, Web Sphere,
    - GlassFish, Tomcat,
    - …
  - Framework pour l'Inversion de contrôle:
    - Spring (Conteneur léger)
    - EJB (Conteneur lourd)
  - Frameworks :
    - Mapping objet relationnel (ORM) : JPA, Hibernate, Toplink, …
    - Applications Web : Struts, JSF, SpringMVC
    - ….
  - Middlewares :
    - RMI, CORBA : Applications distribuées
    - JAXWS pour Web services SOAP
    - JAXRS pour les Web services RESTful
    - JMS : Communication asynchrone entre les application
    - …

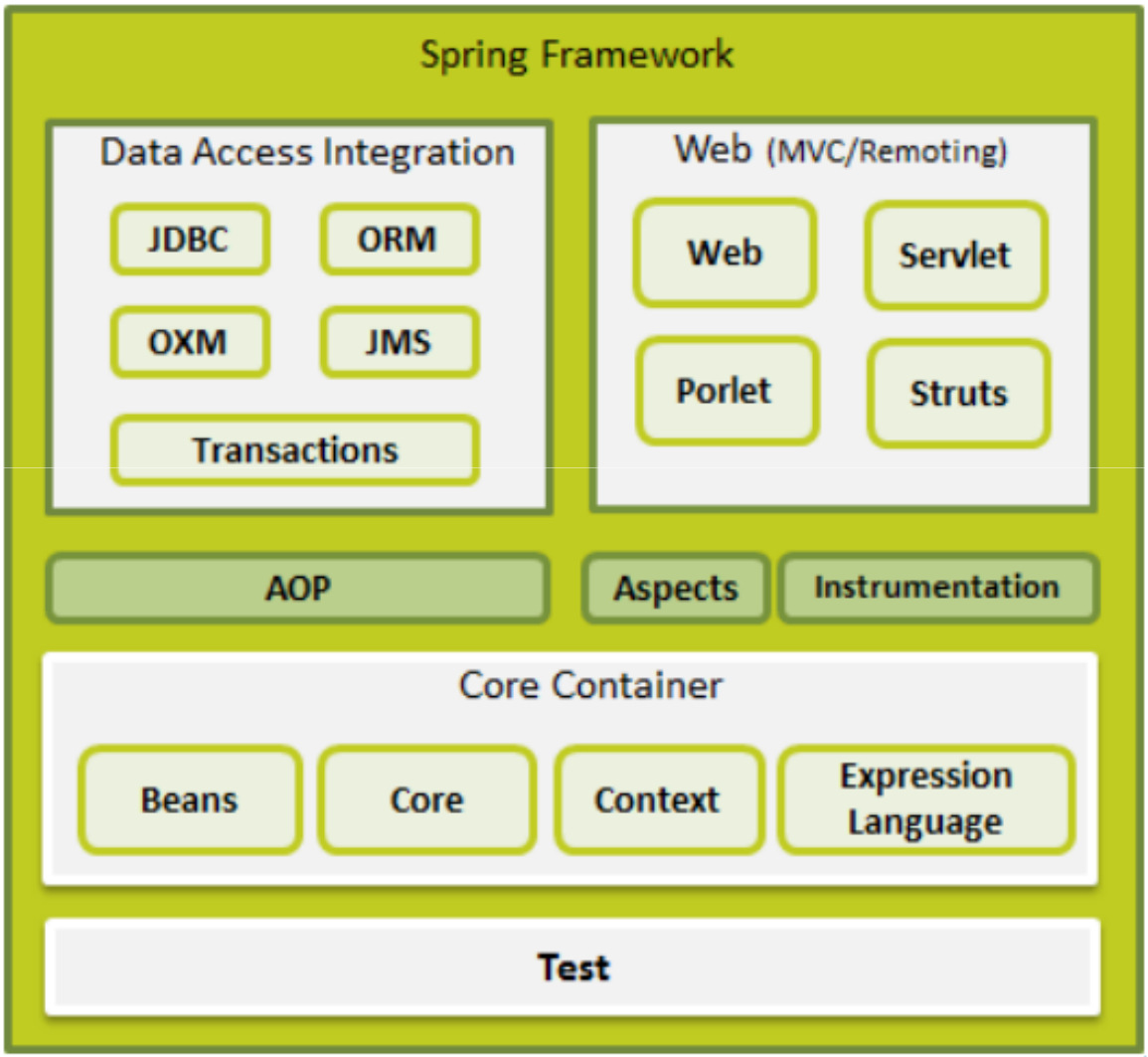
Architectures Distribués



Architecture J2EE



Spring Framework Architecture



Spring Framework Architecture

- Spring est modulaire , permettant de choisir les modules appropriés à votre application, sans être obligé d'autiliser le reste.
- Spring Framework fournit plus de 20 modules qui peuvent être utilisé dans els applications.

Core Container

- The Core Container consists of the Core, Beans, Context, and Expression Language modules whose detail is as follows:
  - The Core module provides the fundamental parts of the framework, including the IoC and Dependency Injection features.
  - The Bean module provides BeanFactory which is a sophisticated implementation of the factory pattern.
  - The Context module builds on the solid base provided by the Core and Beans modules and it is a medium to access any objects defined and configured. The ApplicationContext interface is the focal point of the Context module.
  - The Expression Language module provides a powerful expression language for querying and manipulating an object graph at runtime.

Data Access/Integration

- The Data Access/Integration layer consists of the JDBC, ORM, OXM, JMS and Transaction modules whose detail is as follows:
  - The JDBC module provides a JDBC-abstraction layer that removes the need to do tedious JDBC related coding.
  - The ORM module provides integration layers for popular object-relational mapping APIs, including JPA, JDO, Hibernate, and iBatis.
  - The OXM module provides an abstraction layer that supports Object/XML mapping implementations for JAXB, Castor, XMLBeans, JiBX and XStream.
  - The Java Messaging Service JMS module contains features for producing and consuming messages.
  - The Transaction module supports programmatic and declarative transaction management for classes that implement special interfaces and for all your POJOs.

Web

- The Web layer consists of the Web, Web-Servlet, Web-Struts, and Web-Portlet modules whose detail is as follows:
  - The Web module provides basic web-oriented integration features such as multipart file-upload functionality and the initialization of the IoC container using servlet listeners and a web-oriented application context.
  - The Web-Servlet module contains Spring's model-view-controller (MVC) implementation for web applications
  - The Web-Struts module contains the support classes for integrating a classic Struts web tier within a Spring application.
  - The Web-Portlet module provides the MVC implementation to be used in a portlet environment and mirrors the functionality of Web-Servlet module.

Autres

- There are few other important modules like AOP, Aspects, Instrumentation, Web and Test modules whose detail is as follows:
  - The AOP module provides aspect-oriented programming implementation allowing you to define method-interceptors and pointcuts to cleanly decouple code that implements functionality that should be separated.
  - The Aspects module provides integration with AspectJ which is again a powerful and mature aspect oriented programming (AOP) framework.
  - The Instrumentation module provides class instrumentation support and class loader implementations to be used in certain application servers.
  - The Test module supports the testing of Spring components with JUnit or TestNG frameworks.

Inversion de contrôle
ou
Injection de dépendances

med@youssfi.net

Rappels de quelque principes de conception

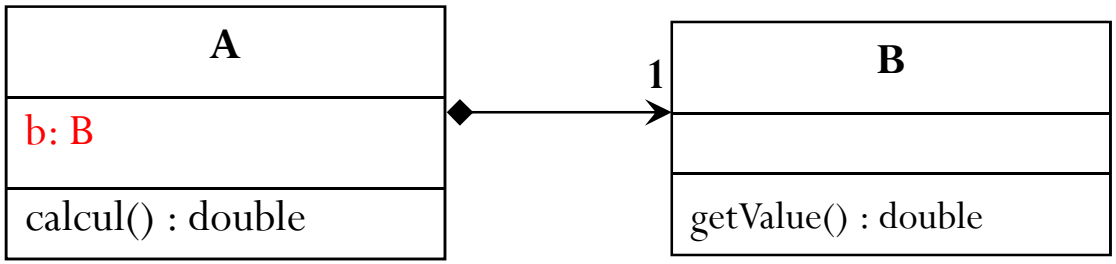
- Une application qui n'évolue pas meurt.
- Une application doit être fermée à la modification et ouverte à l'extension.
- Une application doit s'adapter aux changements
- Efforcez-vous à coupler faiblement vos classes.
- Programmer une interface et non une implémentation
- Etc..

Couplage Fort
et
Couplage faible

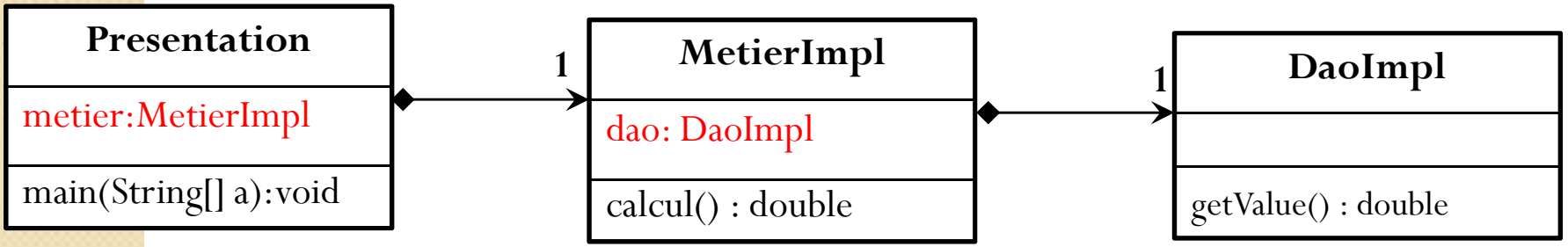
med@youssfi.net

Couplage fort

- Quand une classe A est lié à une classe B, on dit que la classe A est fortement couplée à la classe B.
- La classe A ne peut fonctionner qu'en présence de la classe B.
- Si une nouvelle version de la classe B (soit B2), est crée, on est obligé de modifier dans la classe A.
- Modifier une classe implique:
  - Il faut disposer du code source.
  - Il faut recompiler, déployer et distribuer la nouvelle application aux clients.
  - Ce qui engendre un cauchemar au niveau de la maintenance de l'application



Exemple de couplage fort




```
package metier;
import dao.DaoImpl;
public class MetierImpl {
  private DaoImpl dao;
 public MetierImpl() {
  dao=new DaoImpl();
 }
public double calcul(){
 double nb=dao.getValue();
 return 2*nb;
}
}
```

```
package dao;
public class DaoImpl {
 public double getValue(){
 return(5);
 }
}
```

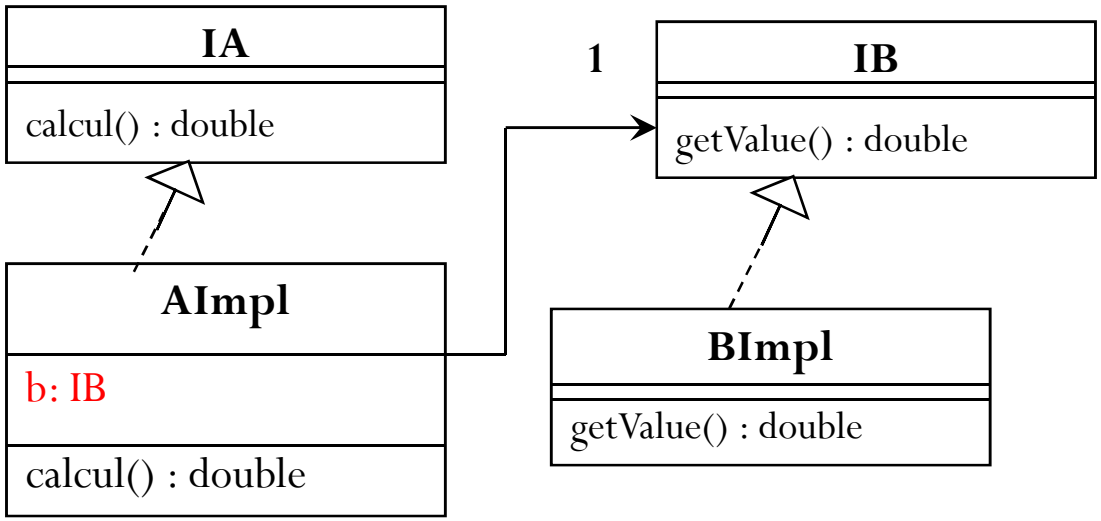
```
package pres;
import metier.MetierImpl;
public class Presentation {
private static MetierImpl metier;
public static void main(String[]
   args) {
metier=new MetierImpl();
System.out.println(metier.calcul());
}
}
```

Problèmes du couplage fort

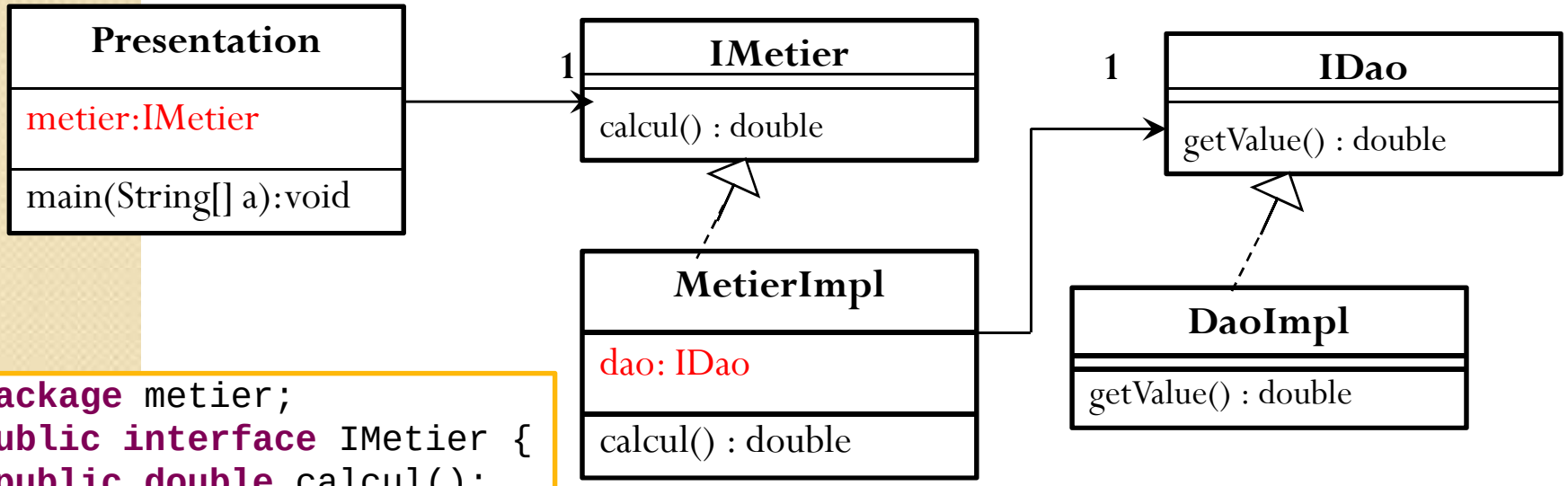
- Dans l'exemple précédent, les classes MetierImpl et DaoImpl sont liées par un couplage fort. De même pour les classe Presentation et MetierImpl
- Ce couplage fort n'a pas empêché de résoudre le problème au niveau fonctionnel.
- Mais cette conception nous ne a pas permis de créer une application fermée à la modification et ouverte à l'extension.
- En effet, la création d'une nouvelle version de la méthode getValue() de la classe DaoImpl, va nous obliger d'éditer le code source de l'application aussi bien au niveau de DaoImpl et aussi MetierImpl.
- De ce fait nous avons violé le principe « une application doit être fermée à la modification et ouverte à l'exetension»
- Nous allons voir que nous pourrons faire mieux en utilisant le couplage faible.

Couplage Faible.

- Pour utiliser le couplage faible, nous devons utiliser les interfaces.
- Considérons une classe A qui implémente une interface IA, et une classe B qui implémente une interface IB.
- Si la classe A est liée à l'interface IB par une association, on dit que le classe A et la classe B sont liées par un couplage faible.
- Cela signifie que la classe B peut fonctionner avec n'importe quelle classe qui implémente l'interface IA.
- En effet la classe B ne connait que l'interface IA. De ce fait n'importe quelle classe implémentant cette interface peut être associée à la classe B, sans qu'il soit nécéssaire de modifier quoi que se soit dans la classe B.
- Avec le couplage faible, nous pourrons créer des application fermée à la modification et ouvertes à l'extension.



Exemple de coupage faible




```
package metier;
public interface IMetier {
 public double calcul();
}
```

```
package metier;
import dao.IDao;
public class MetierImpl
   implements IMetier {
 private IDao dao;
public double calcul() {
double nb=dao.getValue();
 return 2*nb;
}
// Getters et Setters
}
```

```
package dao;
public interface IDao {
public double getValue();
}
```

```
package dao;
public class DaoImpl implements IDao {
public double getValue() {
 return 5;
}
}
```

Injection des dépendances

- Injection par instanciation statique :

```
import metier.MetierImpl;
import dao.DaoImpl;
public class Presentation {
public static void main(String[] args) {
 DaoImpl dao=new DaoImpl();
 MetierImpl metier=new MetierImpl();
 metier.setDao(dao);
 System.out.println(metier.calcul());
}
}
```

Injection des dépendances

- Injection par instanciation dynamique par réflexion :
  - Fichier texte de configuration : config.txt

```
ext.DaoImp
metier.MetierImpl
```

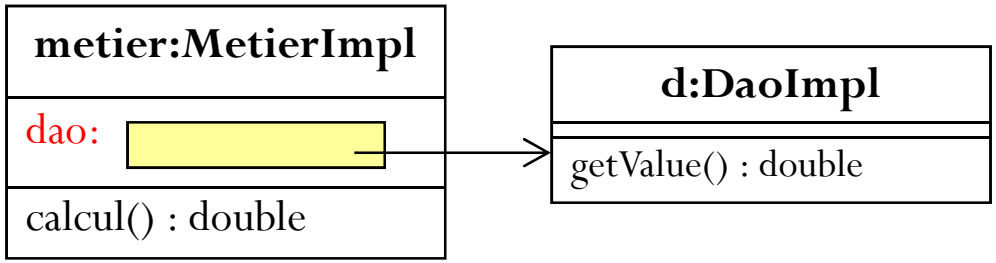
```
import java.io.*;import java.lang.reflect.*;
import java.util.Scanner; import metier.IMetier;
import dao.IDao;
public class Presentation {
public static void main(String[] args) {
try {
 Scanner scanner=new Scanner(new File("config.text"));
 String daoClassname=scanner.next();
 String metierClassName=scanner.next();
 Class cdao=Class.forName(daoClassname);
 IDao dao= (IDao) cdao.newInstance();
 Class cmetier=Class.forName(metierClassName);
 IMetier metier=(IMetier) cmetier.newInstance();
 Method meth=cmetier.getMethod("setDao",new Class[]{IDao.class});
 meth.invoke(metier, new Object[]{dao});
 System.out.println(metier.calcul());
} catch (Exception e) { e.printStackTrace(); }
}
}
```

Injection des dépendances avec Spring.

- L'injection des dépendance, ou l'inversion de contrôle est un concept qui intervient généralement au début de l'exécution de l'application.
- Spring IOC commence par lire un fichier XML qui déclare quelles sont différentes classes à instancier et d'assurer les dépendances entre les différentes instances.
- Quand on a besoin d'intégrer une nouvelle implémentation à une application, il suffirait de la déclarer dans le fichier xml de beans spring.

Injection des dépendances dans une application java standard

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE beans PUBLIC "-//SPRING//DTD BEAN 2.0//EN"
   "http://www.springframework.org/dtd/spring-beans-
   2.0.dtd" >
<beans>
  <bean id="d" class="dao.DaomImpl2"></bean>
  <bean  id="metier" class="metier.MetierImpl">
   <property name="dao" ref="d"></property>
  </bean>
</beans>
```



Injection des dépendances dans une application java standard

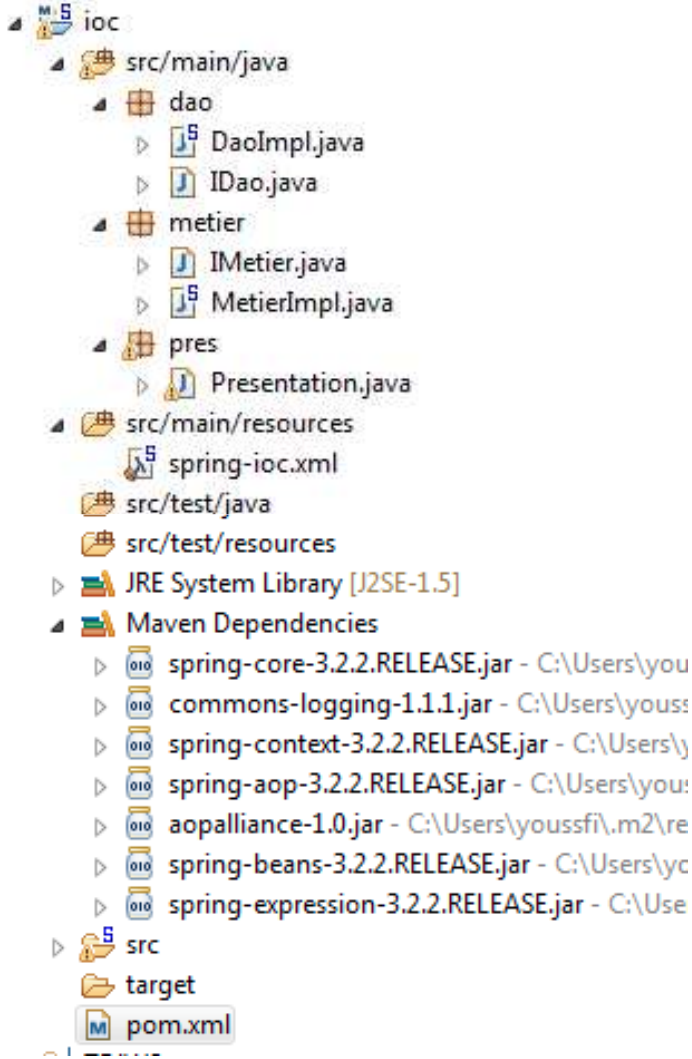
```
package pres;

import metier.IMetier;

import org.springframework.context.support.ClassPathXmlApplicationContext;

public class Presentation {
 public static void main(String[] args) {
        ClassPathXmlApplicationContext context=new
        ClassPathXmlApplicationContext(new String[]{"spring-ioc.xml"});
        IMetier metier=(IMetier) context.getBean("metier");
        System.out.println(metier.calcul());
 }
}
```

med@youssfi.net

Maven dependencies

```
<dependencies>
    <dependency>
      <groupId>org.springframework</groupId>
       <artifactId>spring-core</artifactId>
       <version>3.2.2.RELEASE</version>
      </dependency>
      <dependency>
        <groupId>org.springframework</groupId>
        <artifactId>spring-context</artifactId>
        <version>3.2.2.RELEASE</version>
     </dependency>
</dependencies>
```

Structure du projet



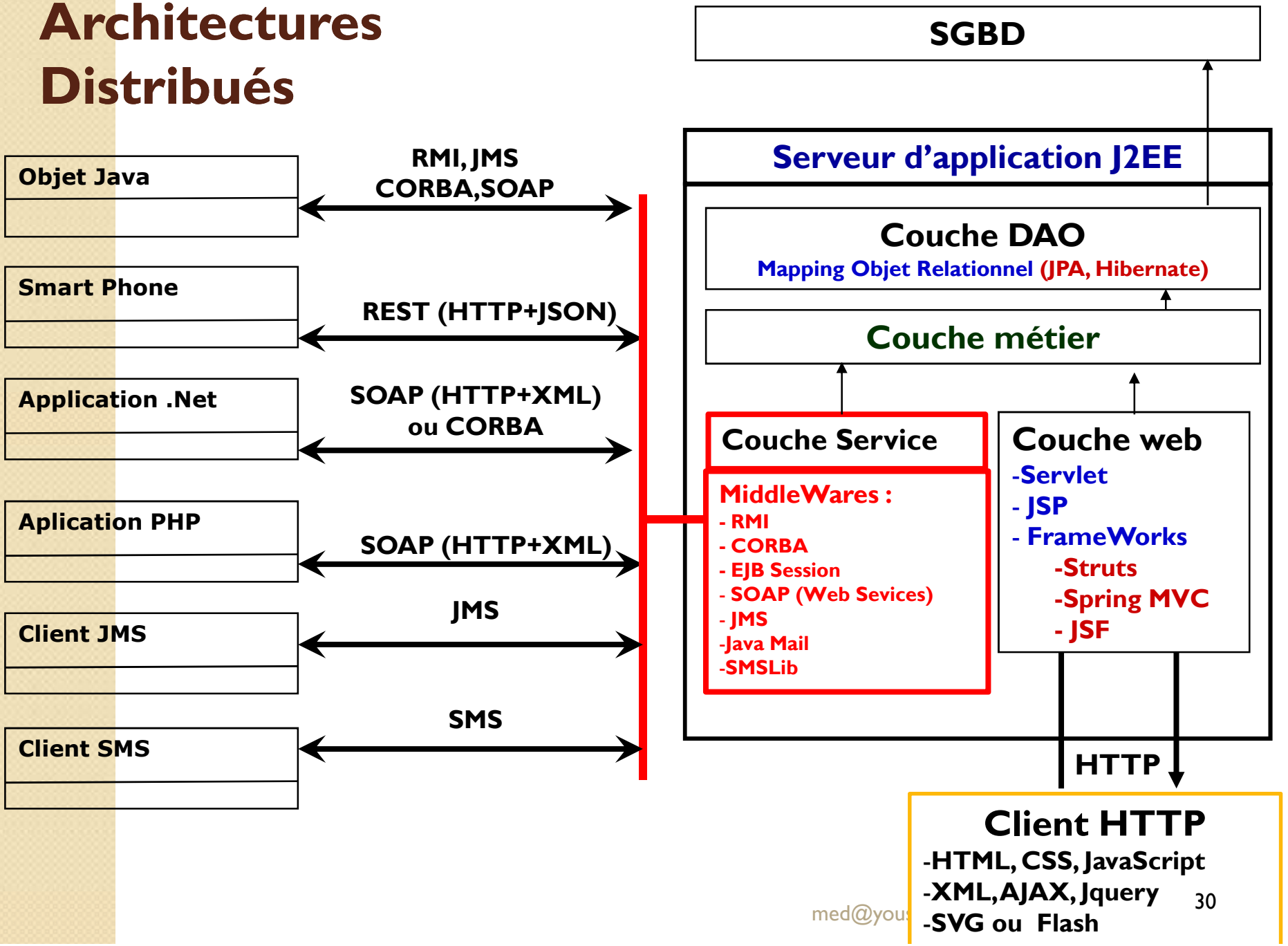
Injection des dépendances dans une application web

- Dans une application web, SpringIOC est appelé au démarrage du serveur en déclarant le listener ContextLoaderListener dans le fichier web.xml

```
<context-param>
        <param-name>contextConfigLocation</param-name>
        <param-value>/WEB-INF/spring-beans.xml</param-value>
</context-param>
<listener>
        <listener-class>
org.springframework.web.context.ContextLoaderListener
        </listener-class>
</listener>
```

- Dans cette déclaration, CotextLoaderListener est appelé par Tomcat au moment du démarrage de l'application. Ce listener cherchera le fichier de beans spring « spring-beans.xml » stocké dans le dossier WEB-INF. ce qui permet de faire l'injection des dépendances entre MetierImpl et DaoImpl

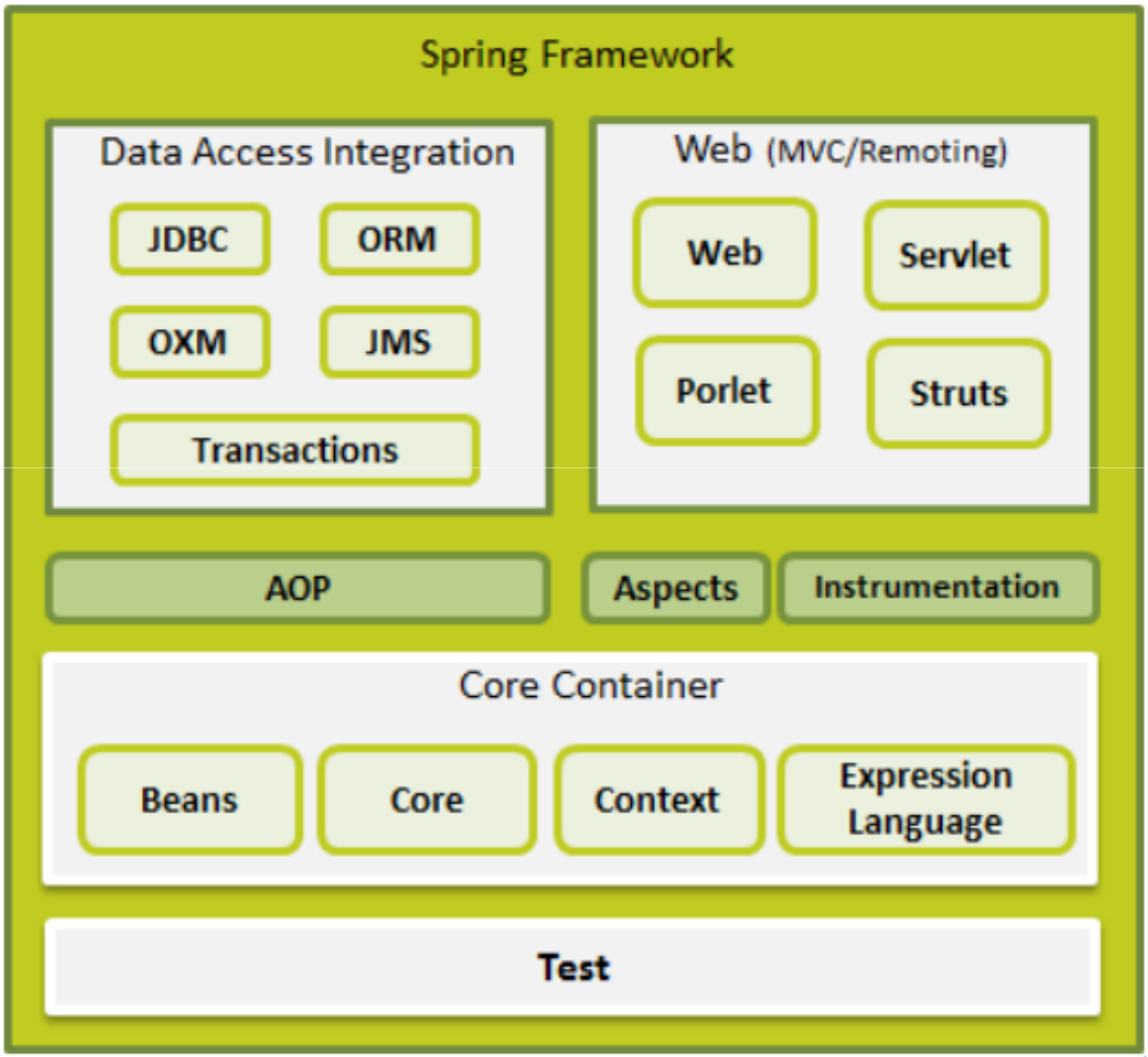
Architectures Distribués



Spring

- Spring est un framework open source JEE pour les applications n-tiers, dont il facilite le développement et les tests.
- Il est considéré comme un conteneur dit « léger », c'est-à-dire une infrastructure similaire à un serveur d'application J2EE.
- Il prend donc en charge la création d'objets et la mise en relation d'objets par l'intermédiaire d'un fichier de configuration qui décrit les objets à fabriquer et les relations de dépendances entre ces objets.
- Il permet de séparer le code métier du code technique
- Spring s'appuie principalement sur l'intégration de trois concepts clés :
  - l'inversion de contrôle ou injection de dépendance (IoC).
  - la programmation orientée aspect (AOP).
  - une couche d'abstraction.
- Ce framework, grâce à sa couche d'abstraction, ne concurrence pas d'autres frameworks dans une couche spécifique d'un modèle architectural MVC mais s'avère un framework multi-couches pouvant s'insérer au niveau de toutes les couches.

Spring Framework Architecture

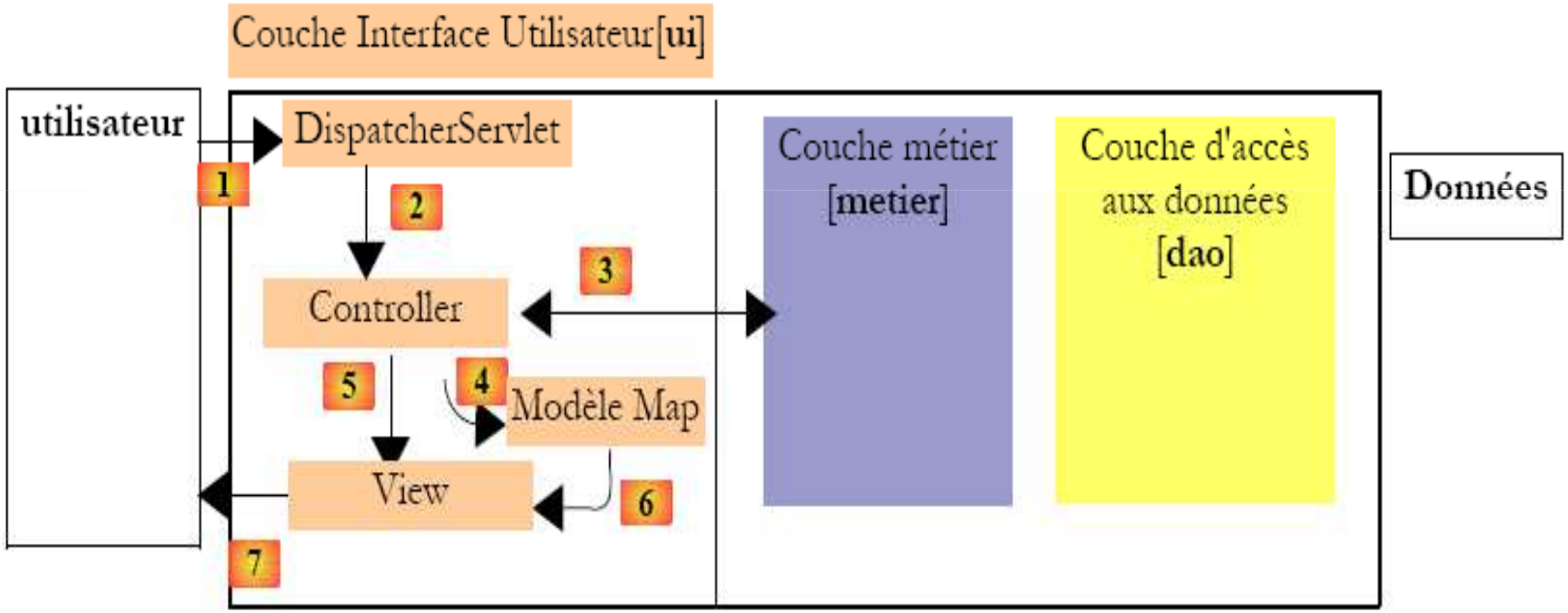


Spring MVC

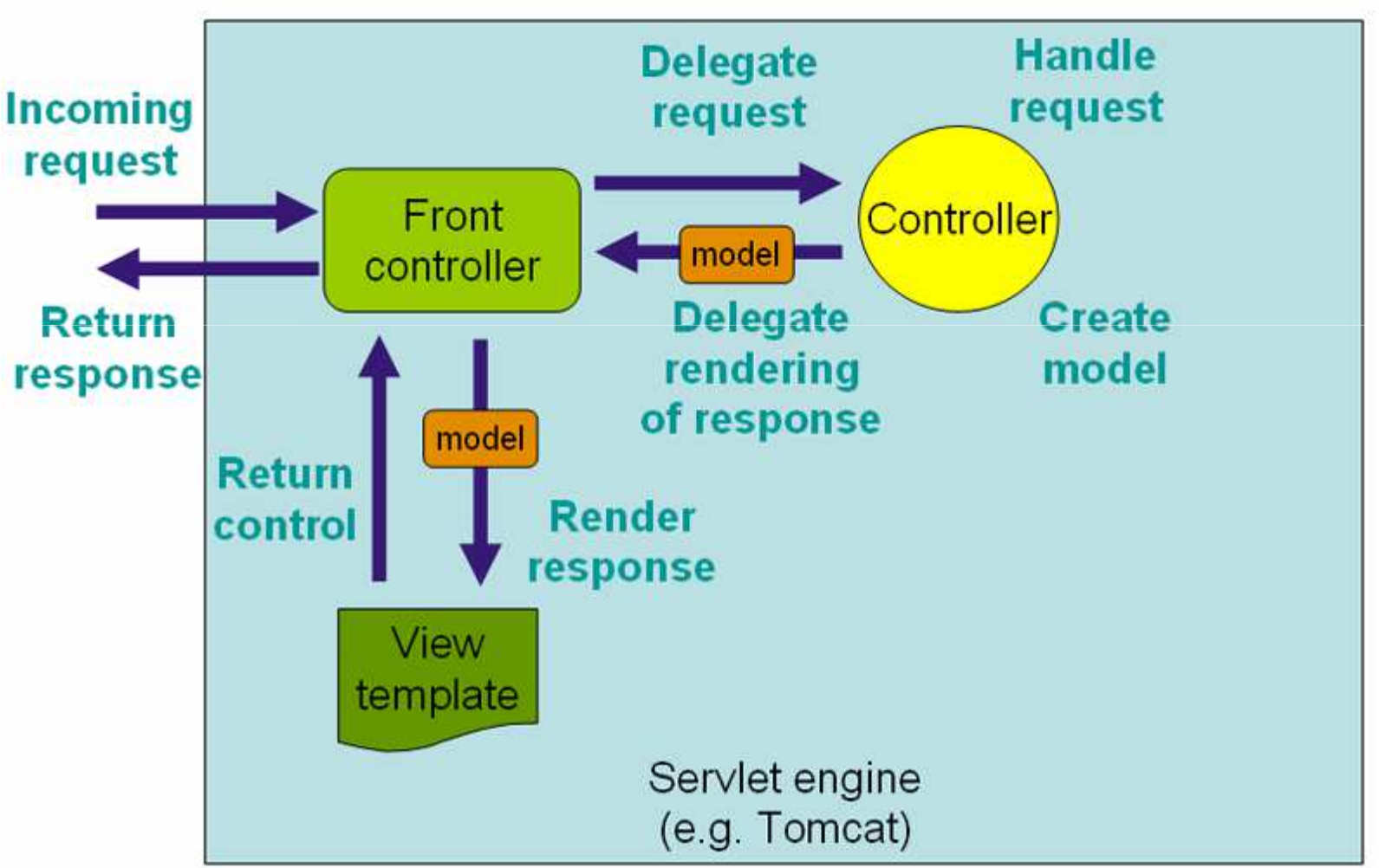
med@youssfi.net

Spring MVC

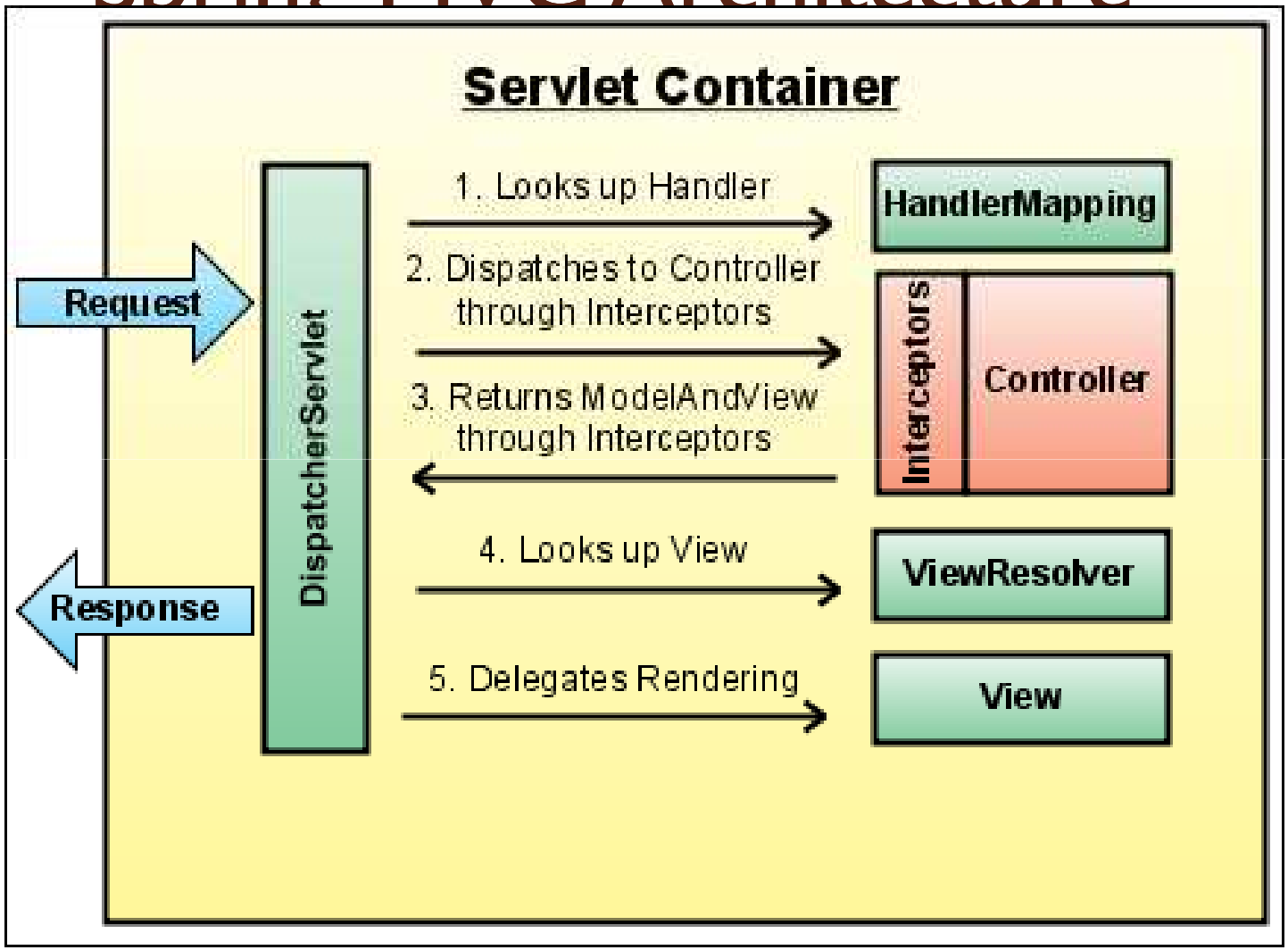
- Architecture de Spring MVC



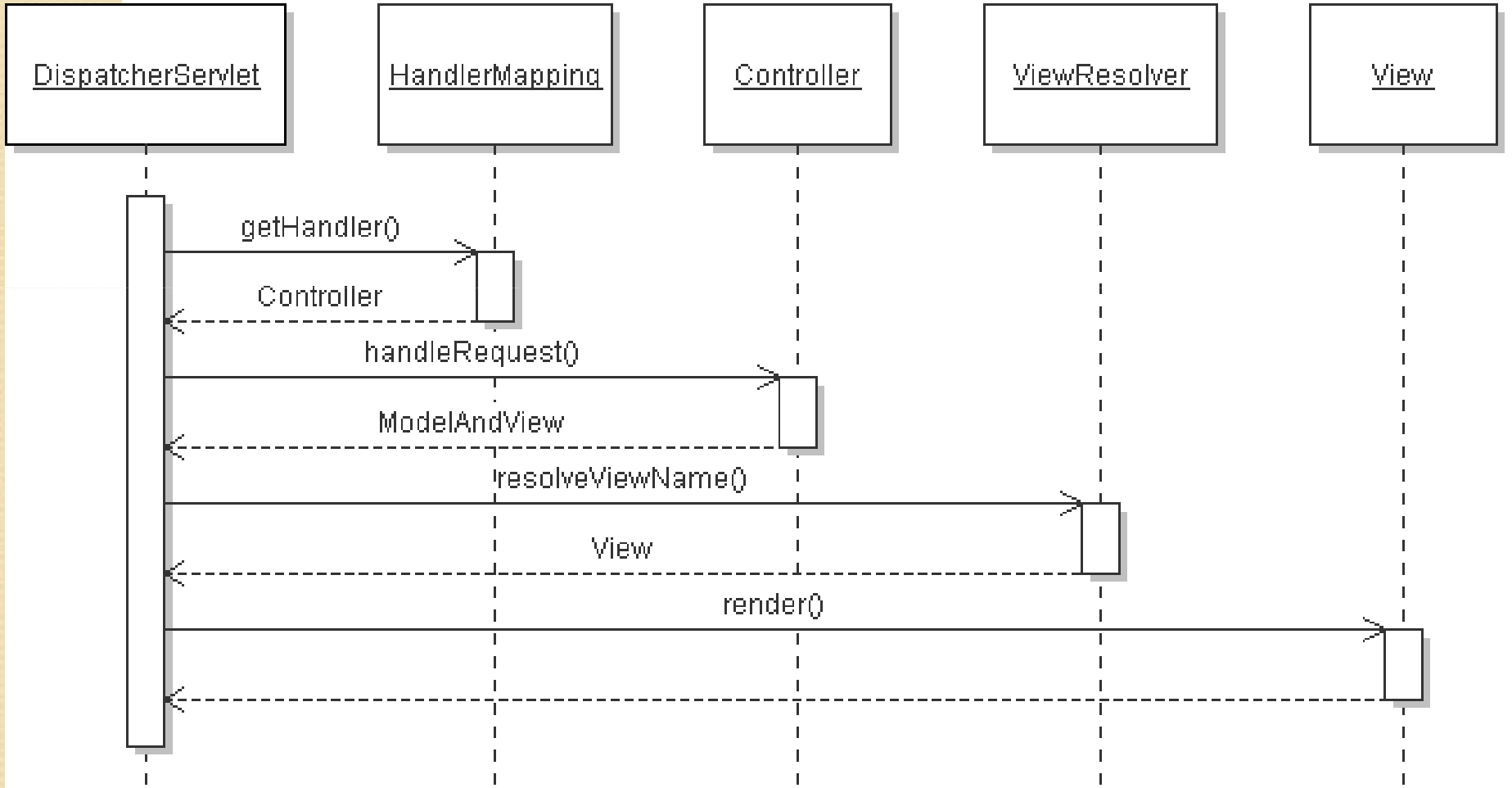
Spring MVC


Incoming request
Delegate request
Handle request
Front controller
Controller
model
Return response
Delegate rendering of response
Create model
model
Return control
Render response
View template
Servlet engine (e.g. Tomcat)

Spring MVC Architecture

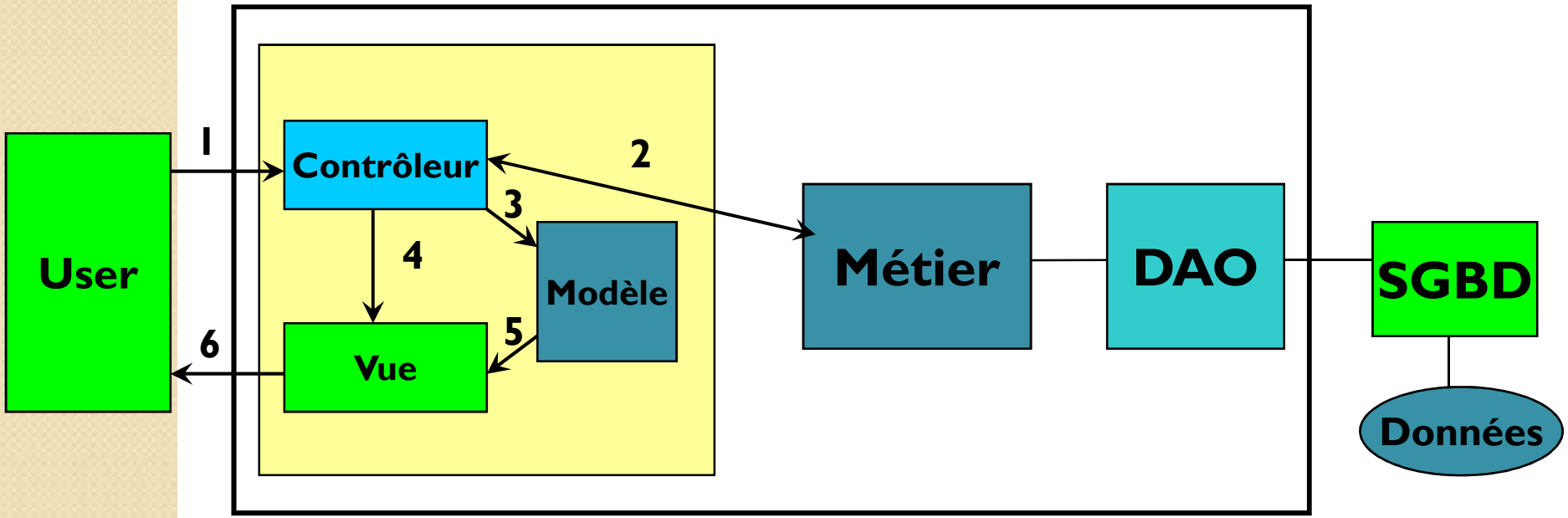


Spring MVC


DispatcherServlet
HandlerMapping
Controller
ViewResolver
View
getHandler()
Controller
handleRequest()
ModelAndView
resolveViewName()
View
render()

Architecture d'une application respectant le modèle MVC

- Au sein de l'architecture 3tier, l'architecture MVC peut être représentée comme suit :



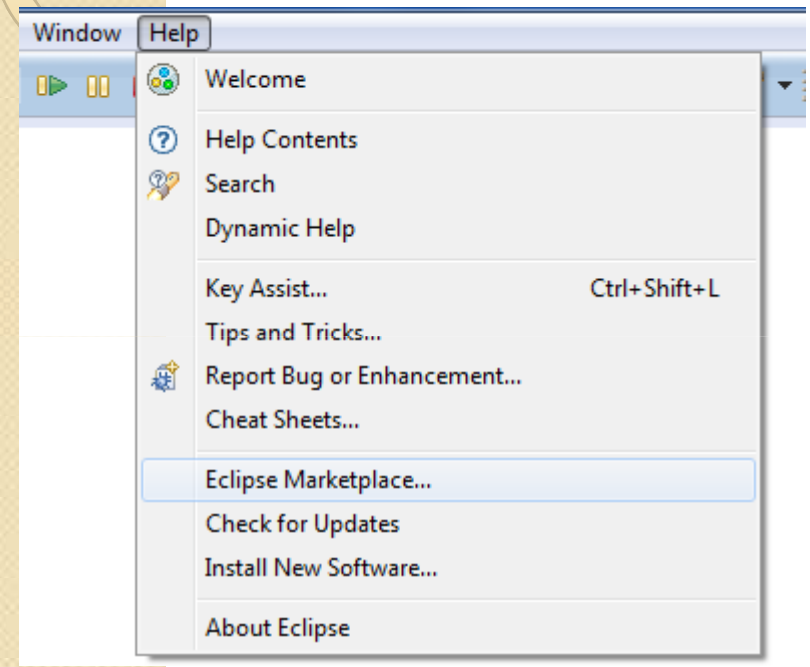
Spring MVC

- 1- le client fait une demande au contrôleur. Celui-ci voit passer toutes les demandes des clients. C'est la porte d'entrée de l'application. C'est le **C** de MVC. Ici le contrôleur est assuré par une servlet générique : **org.springframework.web.servlet.DispatcherServlet**
- 2- le contrôleur principal [DispatcherServlet] fait exécuter l'action demandée par l'utilisateur par une classe implémentant l'interface : **org.springframework.web.servlet.mvc.Controller**
  - A cause du nom de l'interface, nous appellerons une telle classe un contrôleur secondaire pour le distinguer du contrôleur principal [**DispatcherServlet**] ou simplement contrôleur lorsqu'il n'y a pas d'ambiguïté.
- 3- le contrôleur **[Controller]** traite une demande particulière de l'utilisateur. Pour ce faire, il peut avoir besoin de l'aide de la couche métier. Une fois la demande du client traitée, celle-ci peut appeler diverses réponses. Un exemple classique est :
  - une page d'erreurs si la demande n'a pu être traitée correctement
  - une page de confirmation sinon

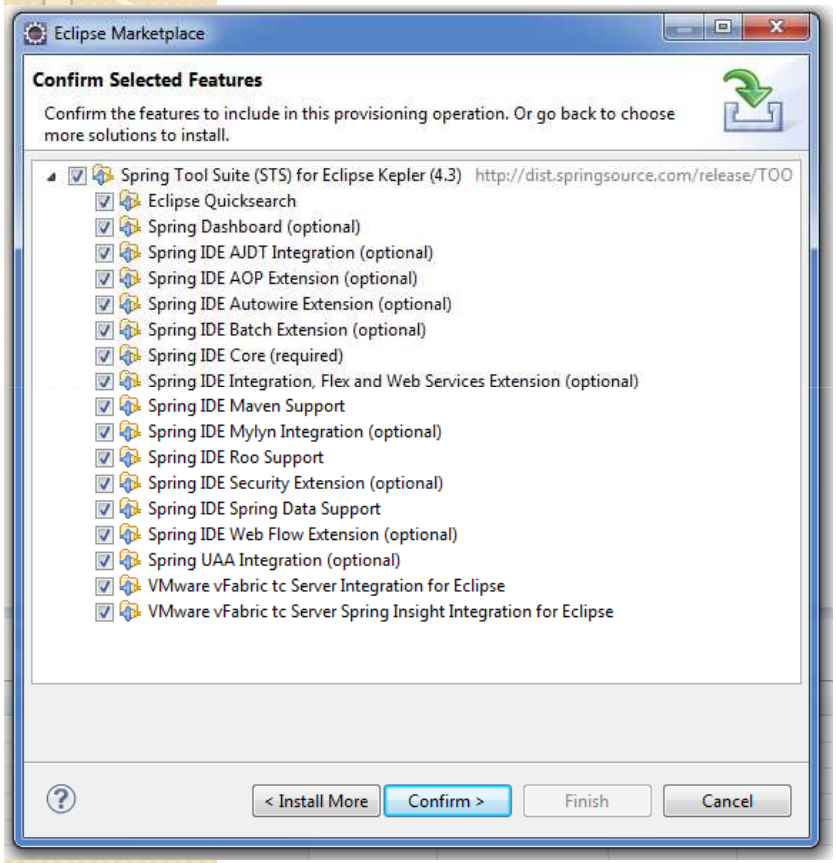
Spring MVC

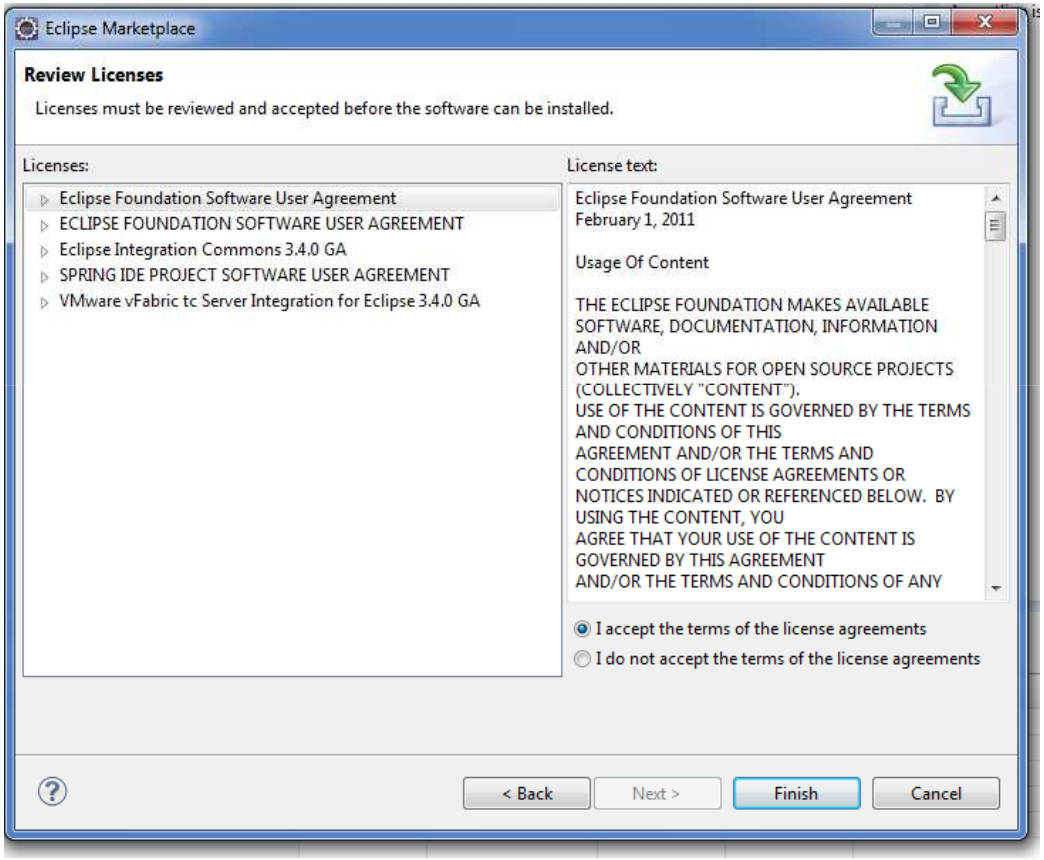
- 4- Le contrôleur choisit la réponse (= vue) à envoyer au client. Choisir la réponse à envoyer au client nécessite plusieurs étapes :
  - choisir l'objet qui va générer la réponse. C'est ce qu'on appelle la vue **V**, le **V** de MVC. Ce choix dépend en général du résultat de l'exécution de l'action demandée par l'utilisateur.
  - lui fournir les données dont il a besoin pour générer cette réponse. En effet, celle-ci contient le plus souvent des informations calculées par la couche métier ou le contrôleur lui-même. Ces informations forment ce qu'on appelle le modèle **M** de la vue, le **M** de MVC. Spring MVC fournit ce modèle sous la forme d'un dictionnaire de type **java.util.Map**.
  - Cette étape consiste donc en le choix d'une vue **V** et la construction du modèle **M** nécessaire à celle-ci.
- 5- Le contrôleur **DispatcherServlet** demande à la vue choisie de s'afficher. Il s'agit d'une classe implémentant l'interface **org.springframework.web.servlet.View**
  - Spring MVC propose différentes implémentations de cette interface pour générer des flux HTML, Excel, PDF, ...
- 6. le générateur de vue **View** utilise le modèle **Map** préparé par le contrôleur **Controller** pour initialiser les parties dynamiques de la réponse qu'il doit envoyer au client.
- 7. la réponse est envoyée au client. La forme exacte de celle-ci dépend du générateur de vue. Ce peut être un flux HTML, XML, PDF, Excel, ...

Installation du plugin : spring tools pour eclipse

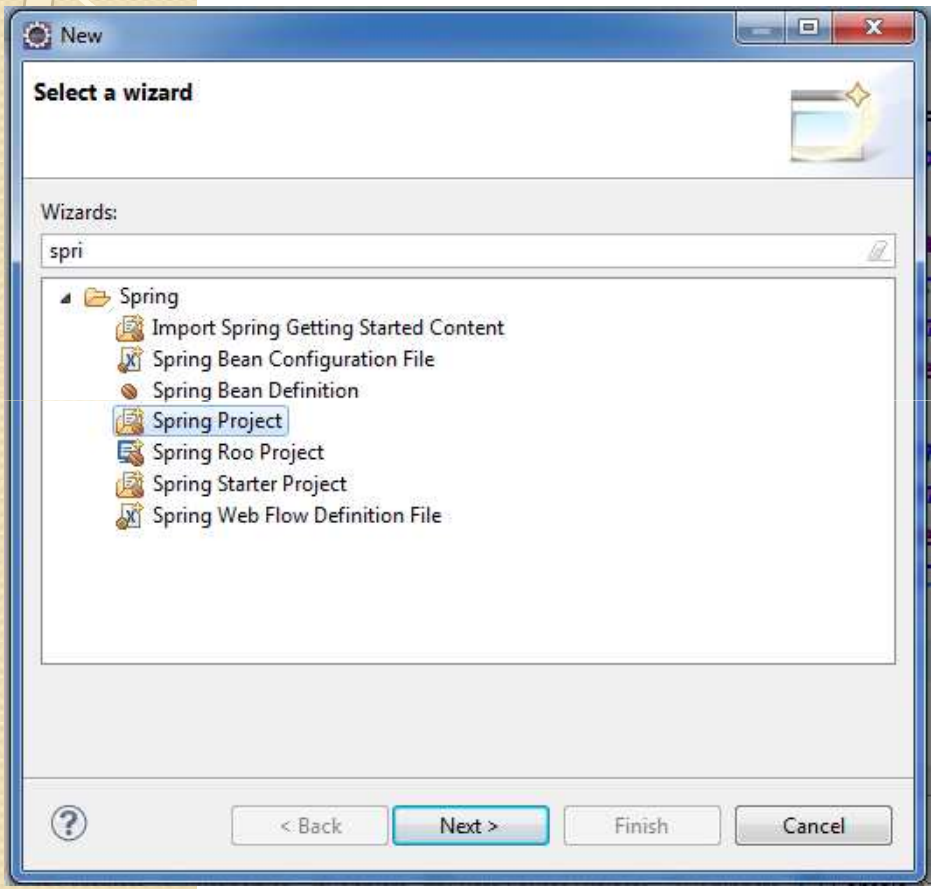


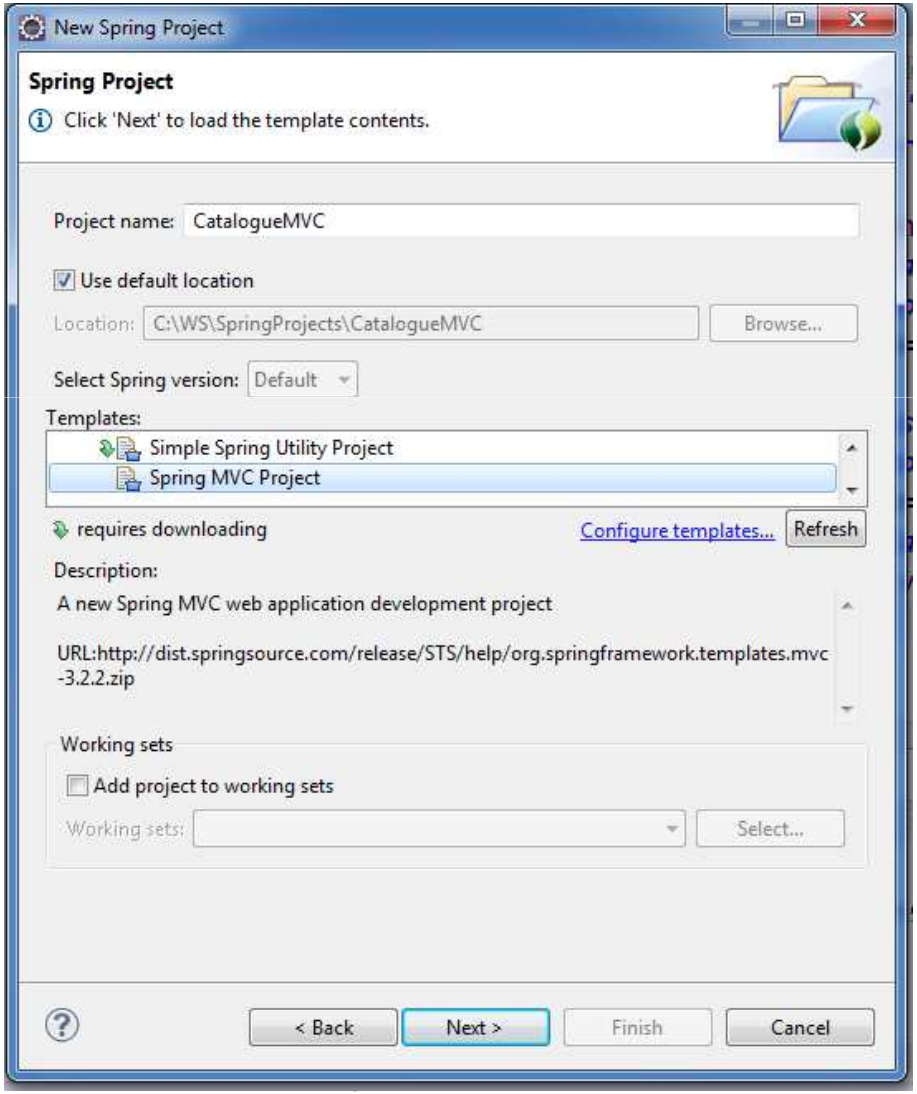
Installation du plugin : spring tools pour eclipse



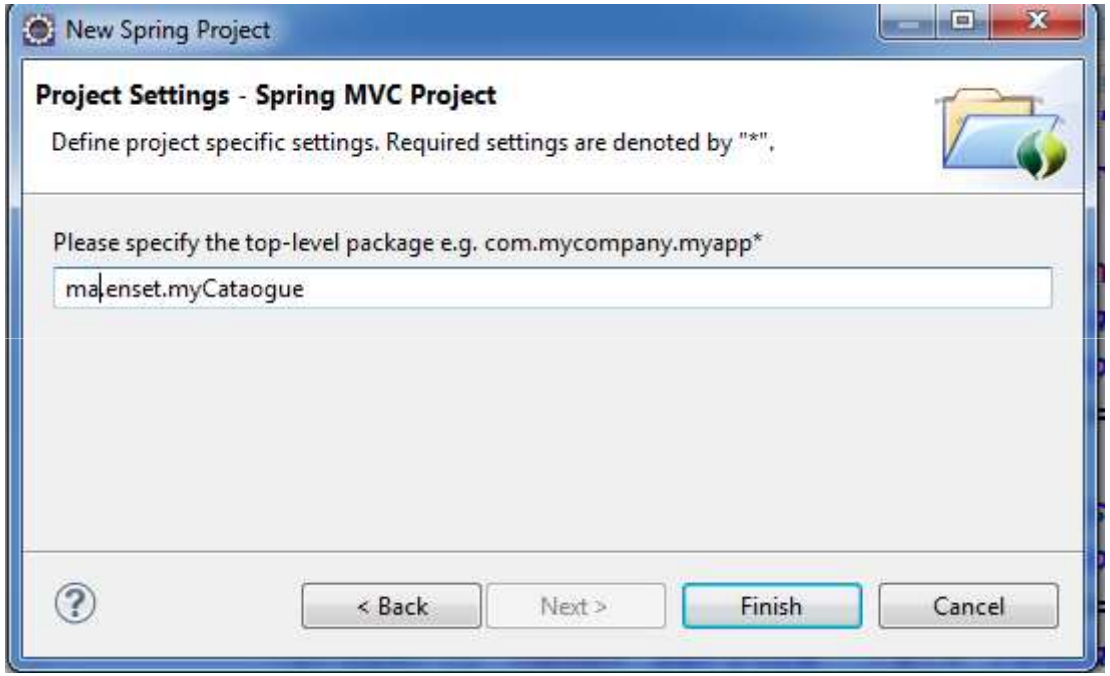


Création d'un projet Spring



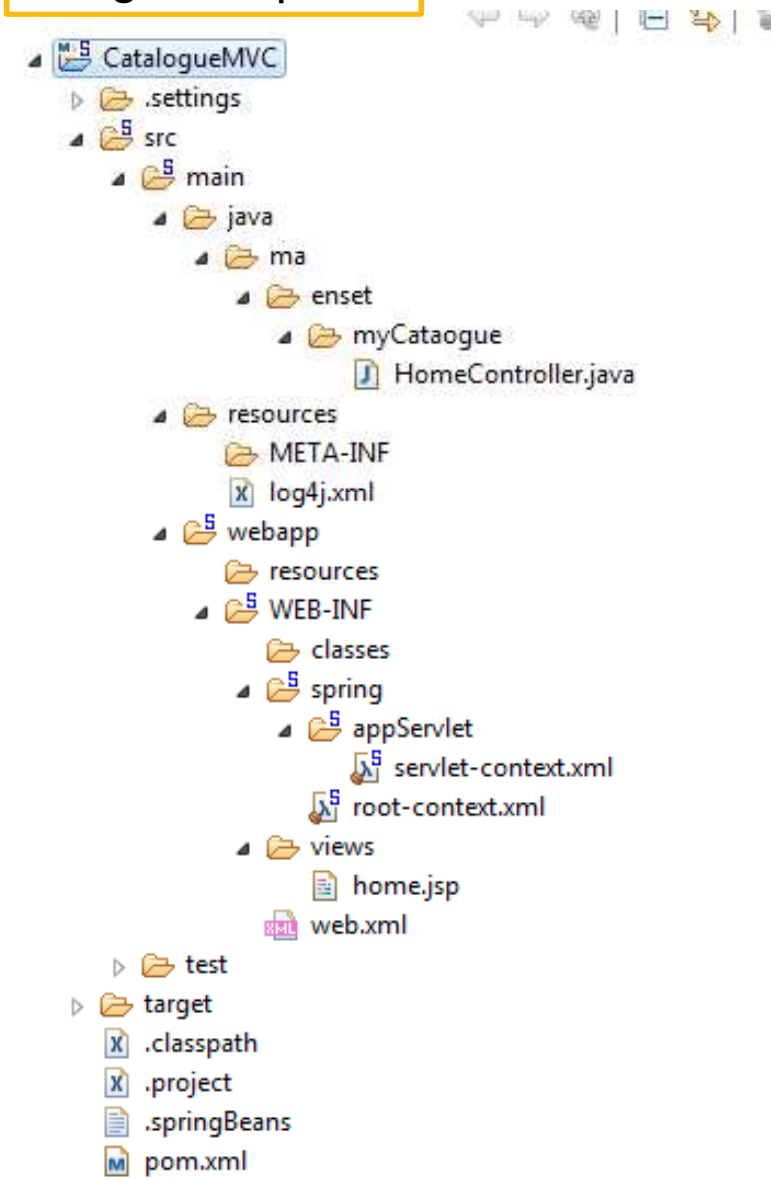


Création d'un projet Spring



Structure du projet

Navigator Explorer



web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app version="2.5" xmlns="http://java.sun.com/xml/ns/javaee"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
   http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd">
<!-- The definition of the Root Spring Container shared by all Servlets and
   Filters -->
<context-param>
   <param-name>contextConfigLocation</param-name>
   <param-value>/WEB-INF/spring/root-context.xml</param-value>
</context-param>
<!-- Creates the Spring Container shared by all Servlets and Filters -->
<listener>
<listener-
   class>org.springframework.web.context.ContextLoaderListener</listener-
   class>
</listener>
```

web.xml

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

>/WEB-INF/spring/root-context.xml

- Ce fichier est lu par ContextLoaderListener, au démarrage du serveur .
- C'est un fichier dans lequel contexte de l'application sera construit
- ContextLoaderListener représente Spring IOC
- c'est donc un fichier pour l'injection des dépendances
- Pour le moment, il est vide

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

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

</beans>
```

>/WEB-INF/spring/appServlet/servlet-context.xml

- Ce fichier est lu par DispatcherServlet qui représente le controleur web de l'application

```
<?xml version="1.0" encoding="UTF-8"?>
<beans:beans xmlns="http://www.springframework.org/schema/mvc"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:beans="http://www.springframework.org/schema/beans"
xmlns:context="http://www.springframework.org/schema/context"
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/context http://www.springframework.org/schema/context/spring-context.xsd">
<!-- Enables the Spring MVC @Controller programming model -->
<annotation-driven />
<!-- Handles HTTP GET requests for /resources/** by efficiently serving up static
   resources in the ${webappRoot}/resources directory -->
<resources mapping="/resources/**" location="/resources/" />
<!-- Resolves views selected for rendering by @Controllers to .jsp resources in the
   /WEB-INF/views directory -->
<beans:bean class="org.springframework.web.servlet.view.InternalResourceViewResolver">
   <beans:property name="prefix" value="/WEB-INF/views/" />
   <beans:property name="suffix" value=".jsp" />
</beans:bean>
<context:component-scan base-package="ma.enset.myCataogue" />
</beans:beans>
```

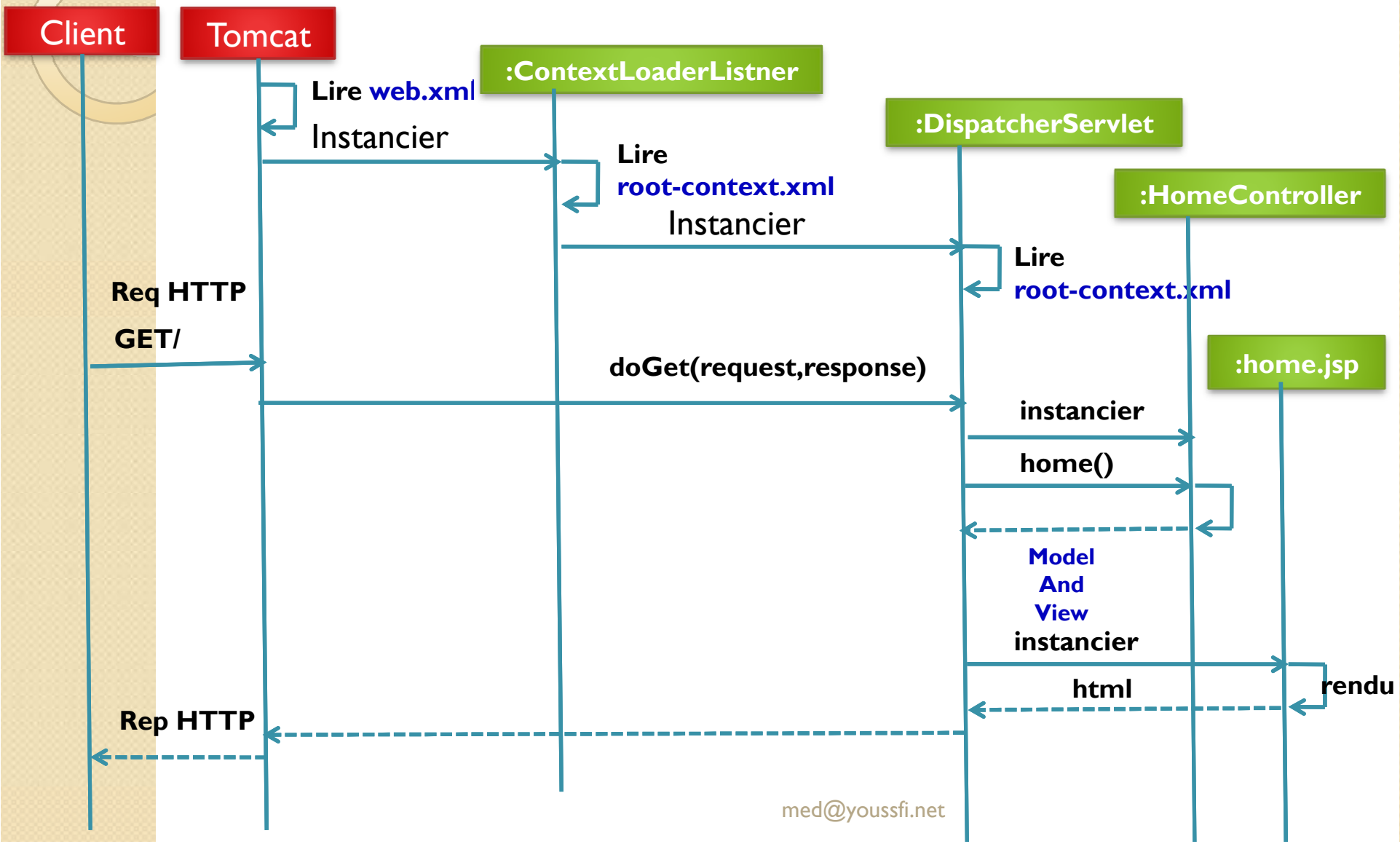
Un exemple de contrôleur Spring MVC

```
package ma.enset.myCataogue;
import java.text.*;import java.util.*;import org.slf4j.*;import
   org.springframework.stereotype.Controller;
import org.springframework.ui.Model; import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;
/** Handles requests for the application home page. */
@Controller
public class HomeController {
   private static final Logger logger = LoggerFactory.getLogger(HomeController.class);
   /** Simply selects the home view to render by returning its name. */
@RequestMapping(value = "/", method = RequestMethod.GET)
public String home(Locale locale, Model model) {
   logger.info("Welcome home! The client locale is {}.", locale);
   Date date = new Date();
   DateFormat dateFormat = DateFormat.getDateTimeInstance(DateFormat.LONG, DateFormat.LONG, locale);
   String formattedDate = dateFormat.format(date);
   model.addAttribute("serverTime", formattedDate );
   return "home";
}
}
```

Un exemple de vue JSP

```
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
<%@ page session="false" %>
<html>
  <head>
      <title>Home</title>
  </head>
<body>
  <h1> Hello world!  </h1>
  <P>  The time on the server is ${serverTime}. </P>
</body>
</html>
```

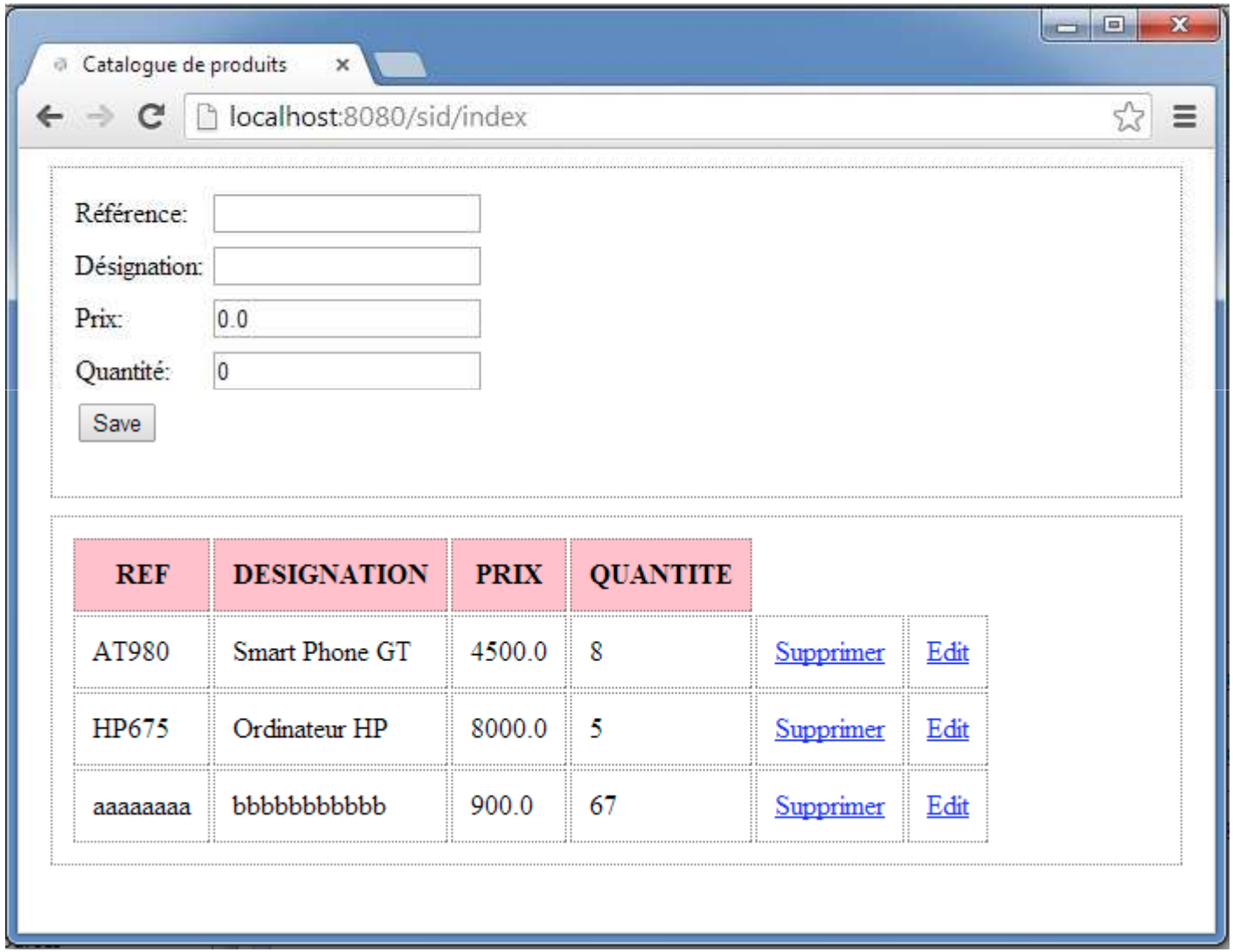
Functionnement



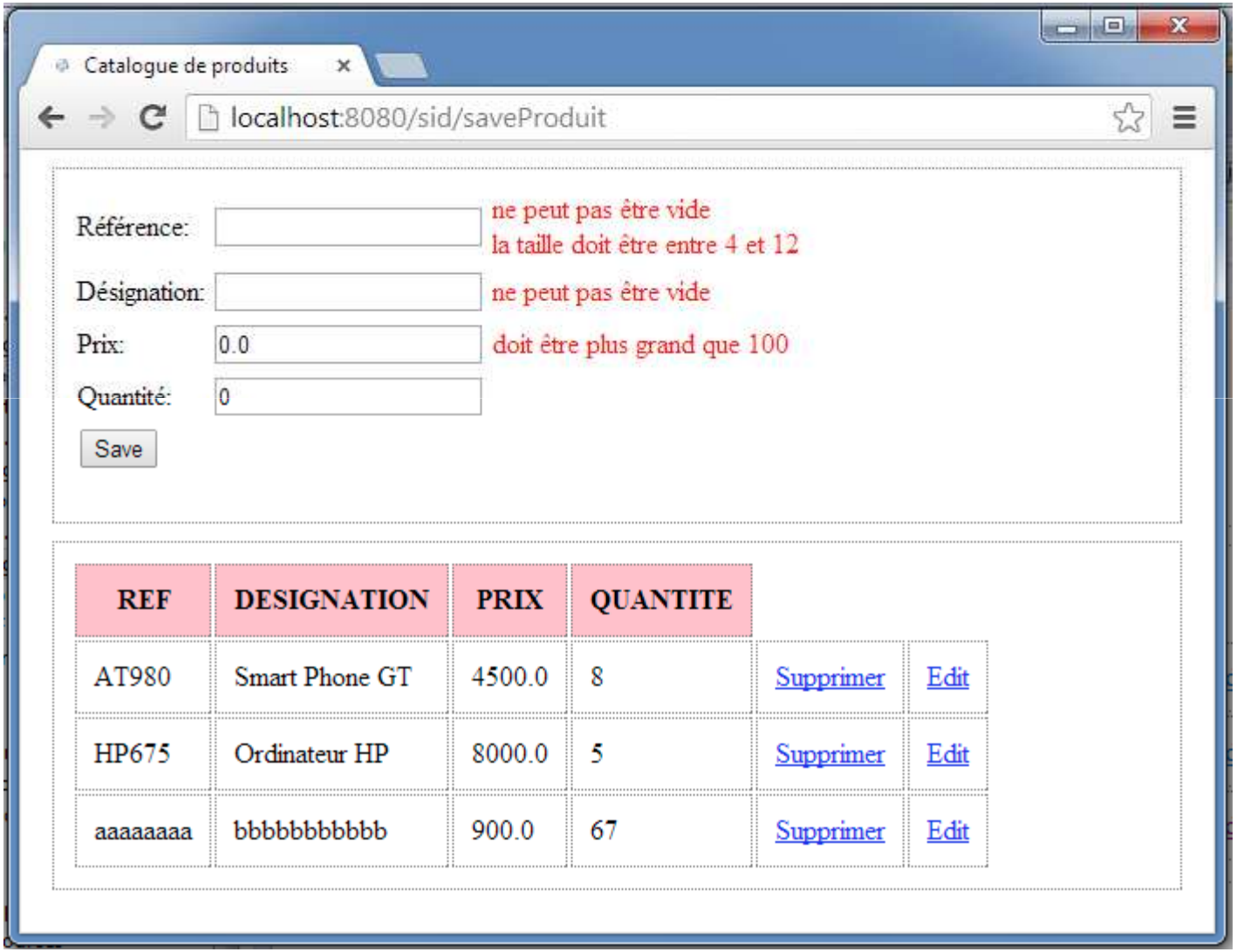
Application 1

- On souhaite créer une application qui permet de gérer des produits. Chaque produit est défini par sa référence (de type String), sa désignation, son prix et sa quantité. L'application doit permettre les opérations suivantes :
  - Ajouter un nouveau produit
  - Consulter tous les produits
  - Consulter les produits dont le nom contient un mot clé.
  - Consulter un produit
  - Supprimer un produit
  - Mettre à jour un produit.
- Cette application se compose de trois couches DAO, Métier et Présentation.
- Elle doit être fermée à la modification et ouverte à l'extension.
- L'injection des dépendances sera effectuée en utilisant Spring IOC.
- Nous allons définir deux implémentations de la couche DAO.
  - Une implémentation qui gère les produits qui sont stockés dans une liste de type HashMap.
  - Dans la deuxième implémentation, nous allons supposer que les produits sont stockés dans une base de données de type MySQL.

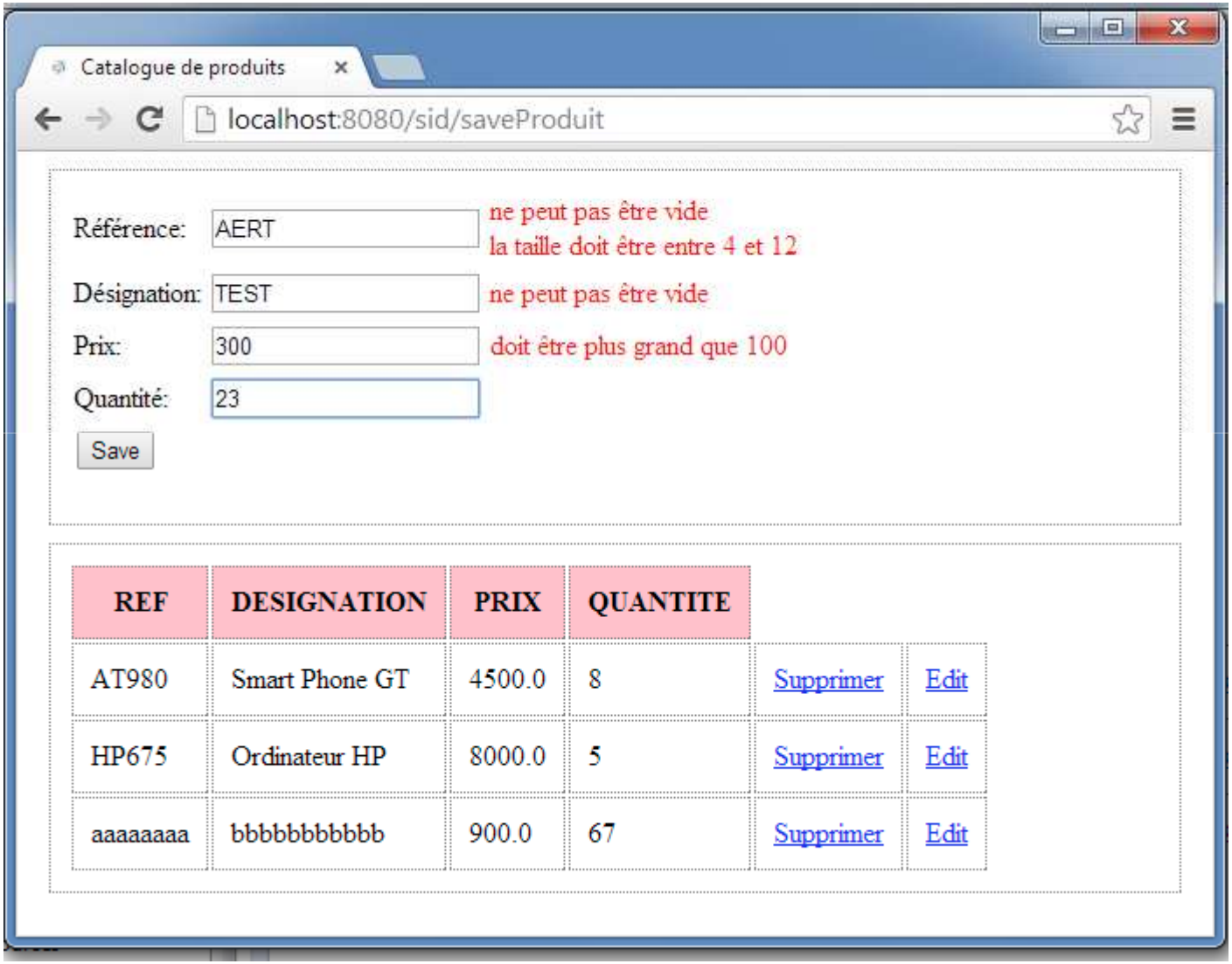
Ecrans de l'application



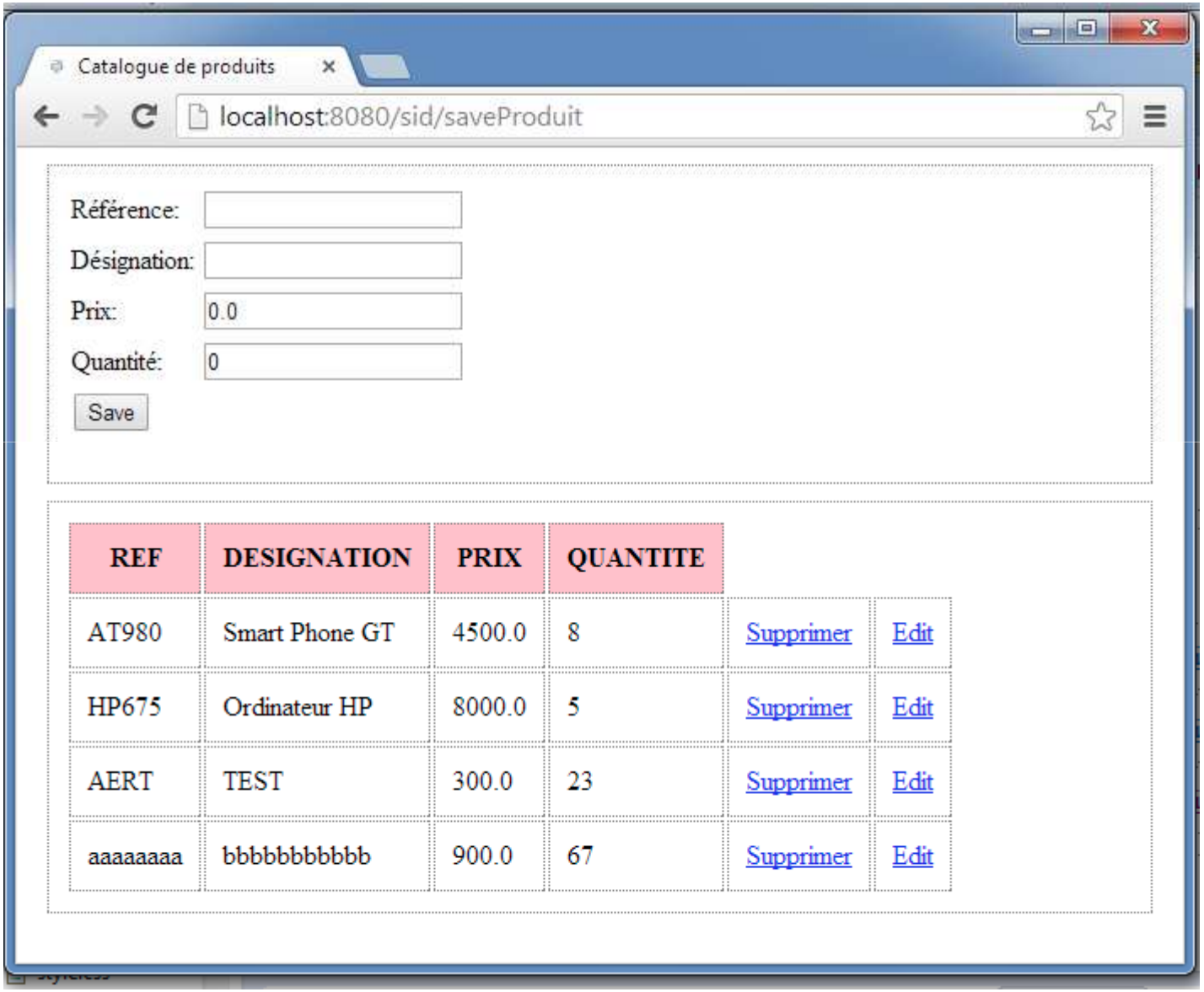
Validation des données coté serveur



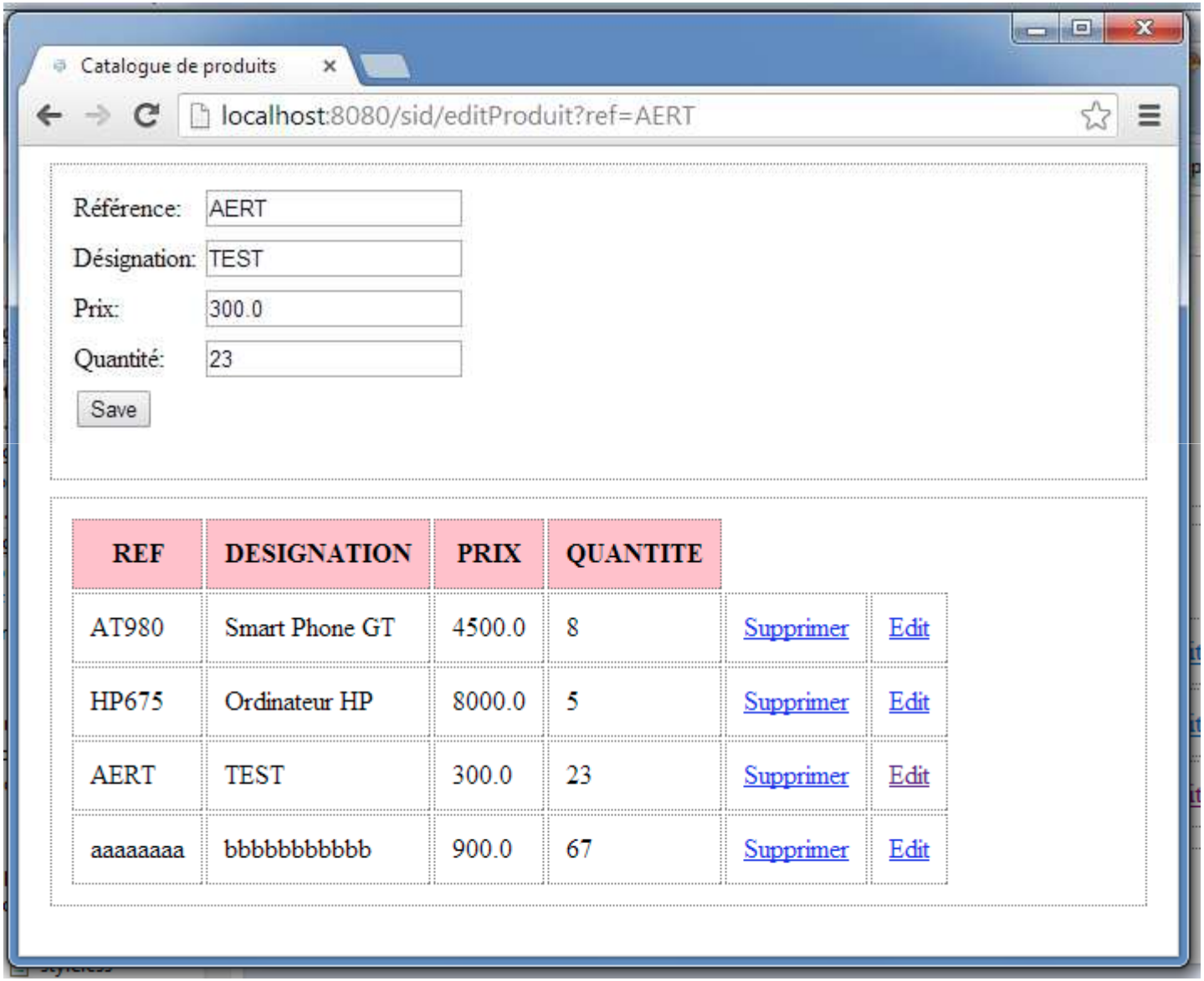
Saisie d'un produit



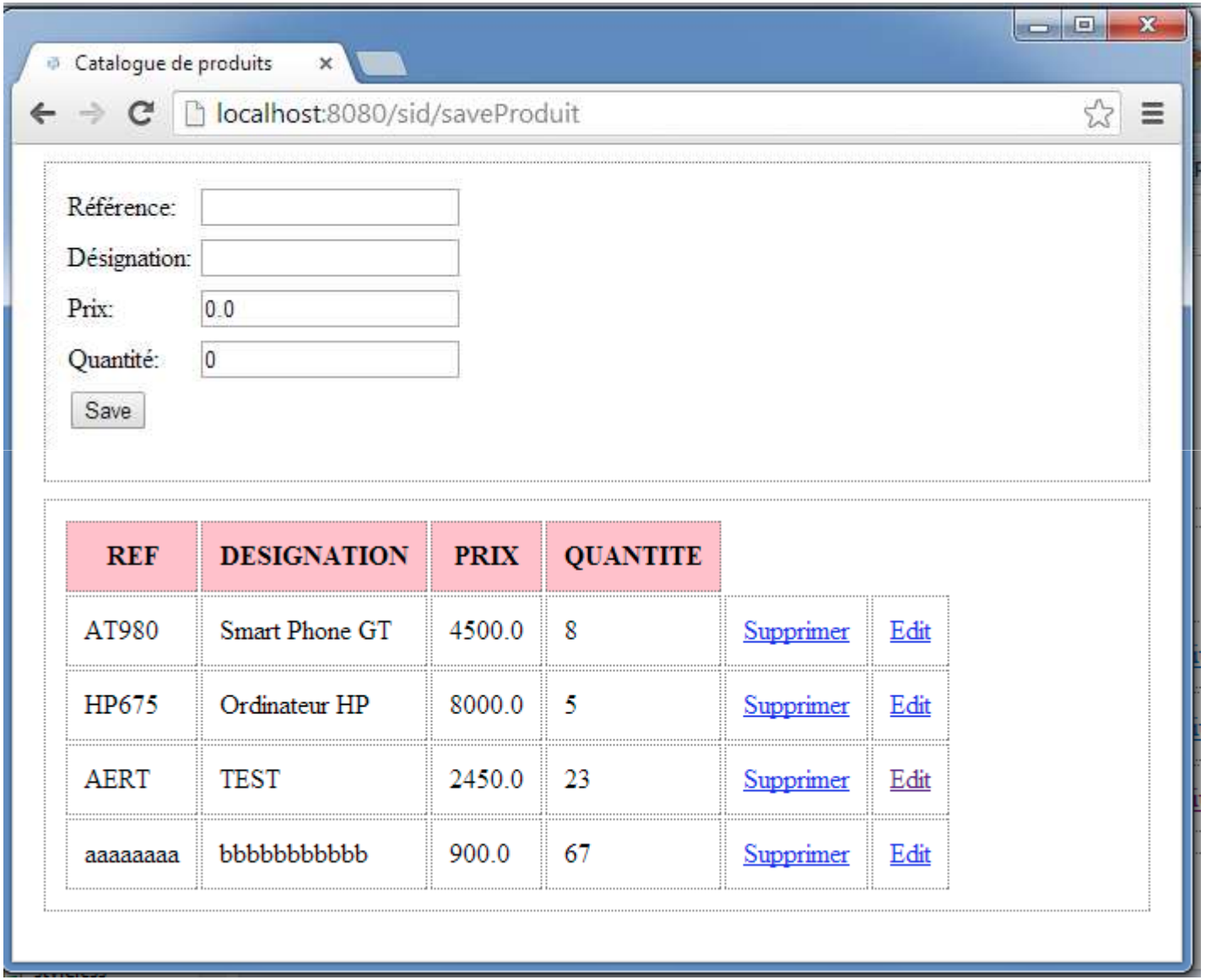
Après Ajout d'un produit



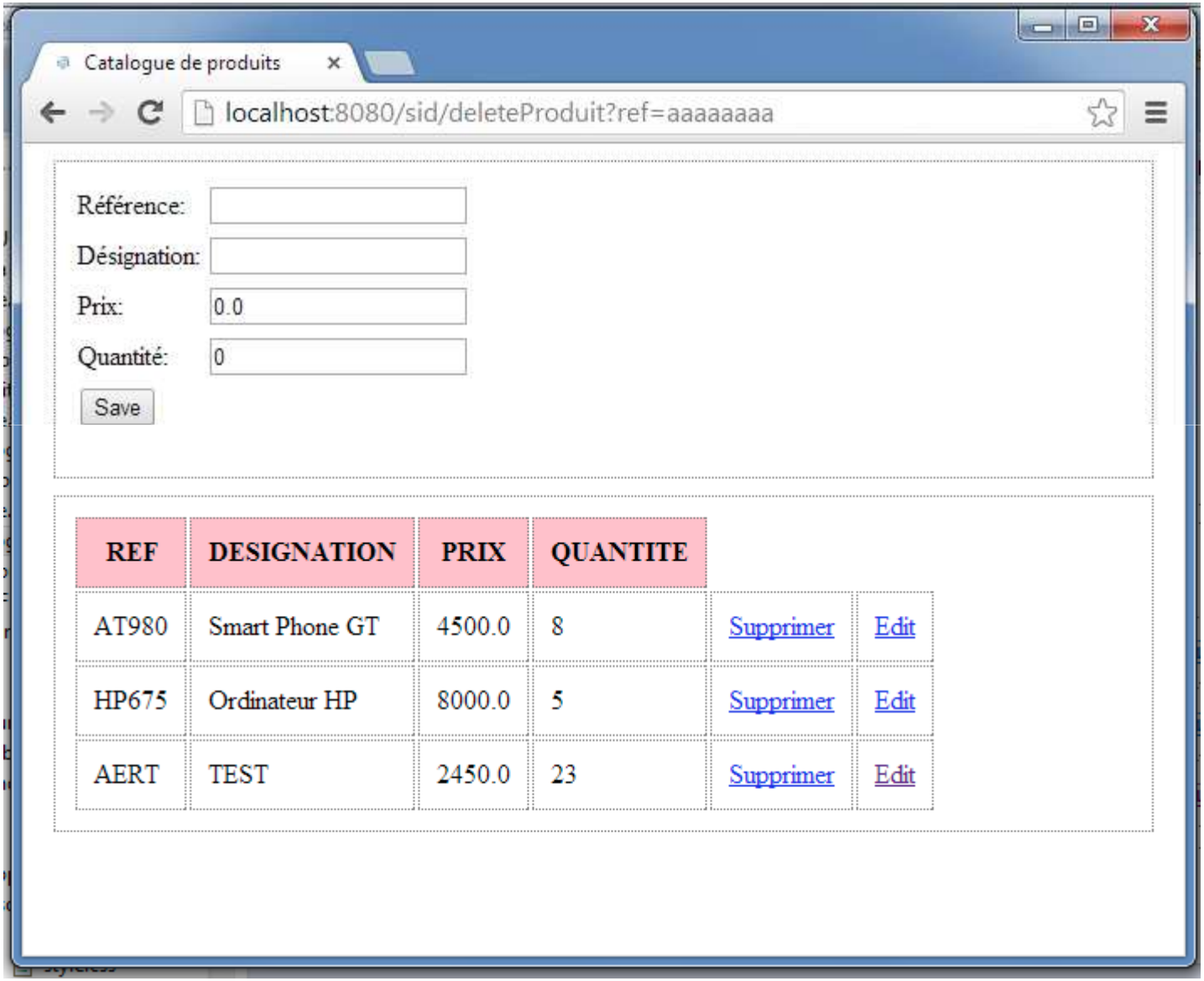
Edition d'un produit



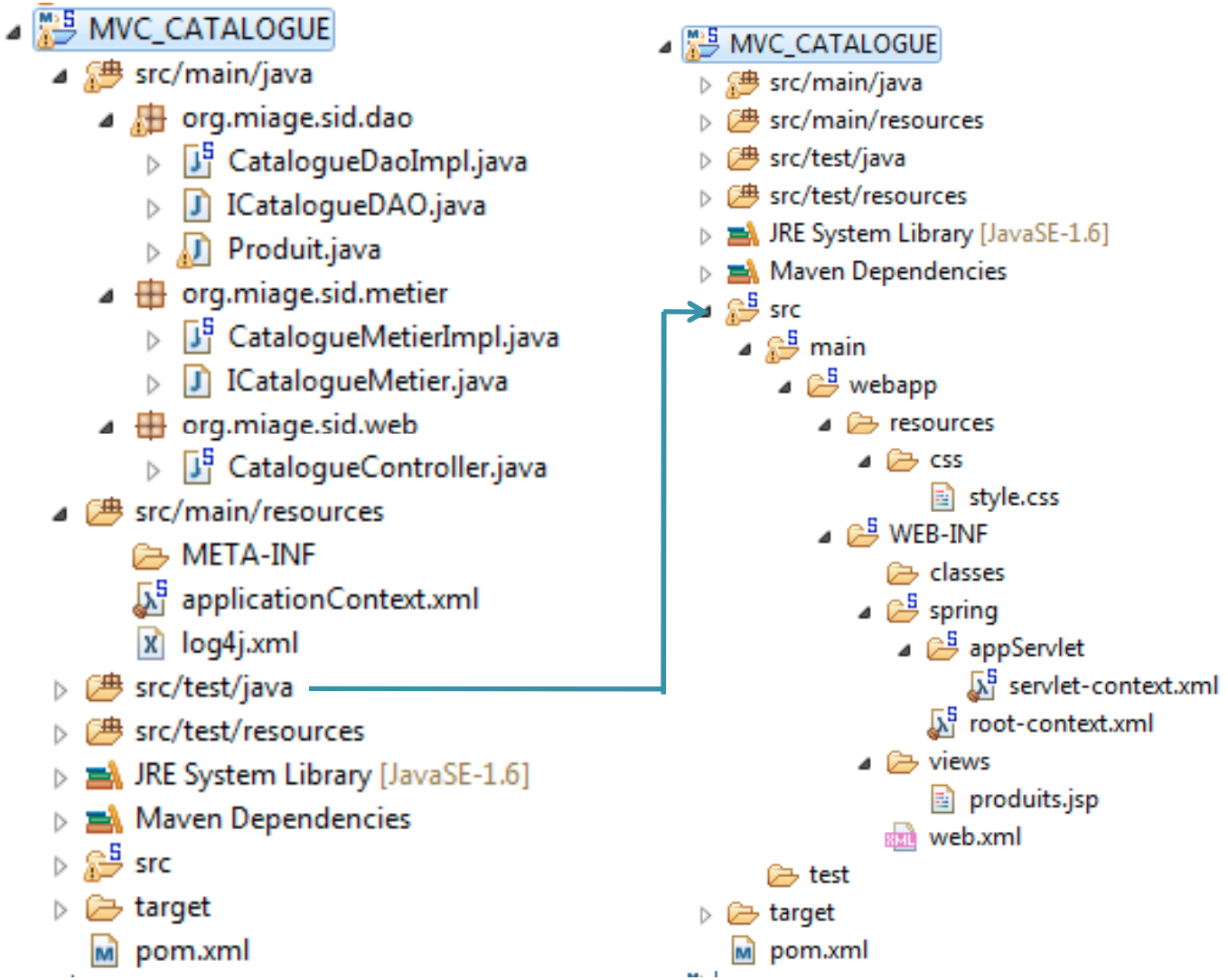
Mise à jour d'un produit



Suppression d'un produit

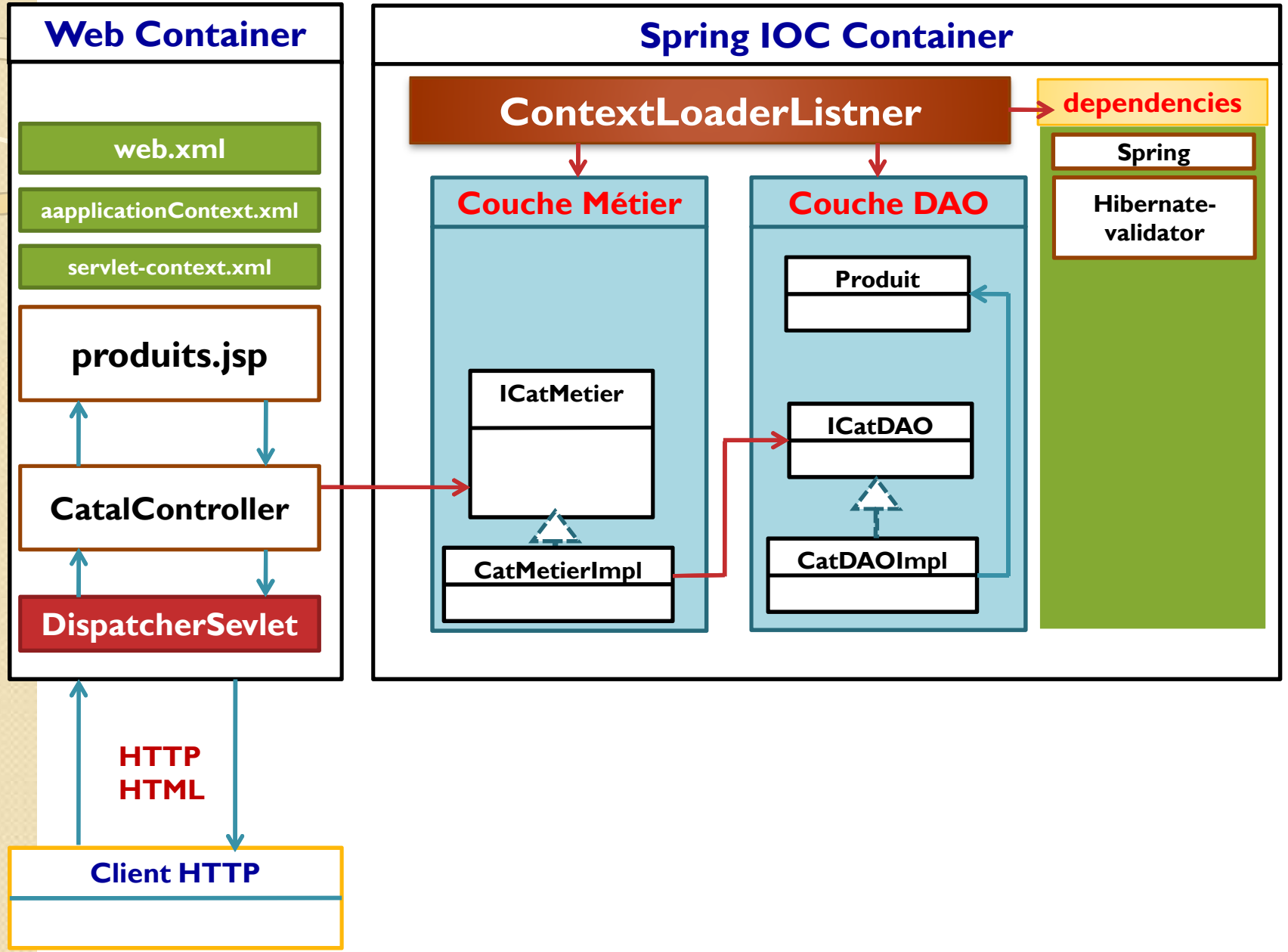


Structure du projet



Maven Dependencies

Architecture technique



Maven dependencies : Spring

pom.xml

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

Maven Dependencies

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

Maven Dependencies

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

Maven Dependencies

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

Maven Dependencies

```
<!-- @Inject -->
<dependency>
    <groupId>javax.inject</groupId>
    <artifactId>javax.inject</artifactId>
    <version>1</version>
</dependency>

<!-- Servlet -->
<dependency>
    <groupId>javax.servlet</groupId>
    <artifactId>servlet-api</artifactId>
    <version>2.5</version>
    <scope>provided</scope>
</dependency>
<dependency>
    <groupId>javax.servlet.jsp</groupId>
    <artifactId>jsp-api</artifactId>
    <version>2.1</version>
    <scope>provided</scope>
</dependency>
<dependency>
    <groupId>javax.servlet</groupId>
    <artifactId>jstl</artifactId>
    <version>1.2</version>
</dependency>
```

Maven Dependencies

```
<!-- Test -->
<dependency>
  <groupId>junit</groupId>
  <artifactId>junit</artifactId>
  <version>4.7</version>
  <scope>test</scope>
</dependency>
<!-- Hibernate Validator -->
<dependency>
  <groupId>org.hibernate</groupId>
  <artifactId>hibernate-validator</artifactId>
  <version>5.0.0.Final</version>
</dependency>
```

COUCHE DAO

med@youssfi.net

Entité Produit

```
package org.miage.sid.dao;
import java.io.Serializable;
import javax.validation.constraints.DecimalMin;
import javax.validation.constraints.Size;
import org.hibernate.validator.constraints.NotEmpty;
public class Produit implements Serializable {
@NotEmpty
@Size(min=4,max=12)
  private String reference;
@NotEmpty
  private String designation;
@DecimalMin(value="100")
  private double prix;
  private int quantite;
 // Constructeurs
 // Getters et Setters
}
```

Interface ICatalogueDAO

```
package org.miage.sid.dao;
import java.util.List;
public interface ICatalogueDAO {
  public void addProduit(Produit p);
  public List<Produit> getAllProduits();
  public List<Produit> getProduits(String mc);
  public Produit getProduit(String reference);
  public void deleteProduit(String refernce);
  public void updateProduit(Produit p);
}
```

Implémentation CatalogueDAOImpl

```
package org.miage.sid.dao;
import java.util.*;
import org.apache.log4j.Logger;
public class CatalogueDaoImpl implements ICatalogueDAO {
private Map<String, Produit> produits=new HashMap<String, Produit>();
   Logger logger=Logger.getLogger(CatalogueDaoImpl.class);
   @Override
   public void addProduit(Produit p) {
        produits.put(p.getReference(), p);
   }
   @Override
   public List<Produit> getAllProduits() {
        Collection<Produit> prods=produits.values();
        return new ArrayList<Produit>(prods);
   }
```

Implémentation CatalogueDAOImpl

```
@Override
public List<Produit> getProduits(String mc) {
   List<Produit> prods=new ArrayList<Produit>();
   for(Produit p:produits.values())
        if(p.getDesignation().indexOf(mc)>=0)
           prods.add(p);
   return prods;
}
@Override
public Produit getProduit(String reference) {
   return produits.get(reference);
}
@Override
public void deleteProduit(String refernce) {
   produits.remove(refernce);
}
```

Implémentation CatalogueDAOImpl

```
@Override
public void updateProduit(Produit p) {
   produits.put(p.getReference(),p);
}
public void init(){
   logger.info("Initialisation du catalogue");
   this.addProduit(new Produit("HP675","Ordinateur HP", 8000, 5));
   this.addProduit(new Produit("AEP65","Impriomante AE",760, 80));
   this.addProduit(new Produit("AT980","Smart Phone GT", 4500, 8));
 }
}
```

COUCHE METIER

med@youssfi.net

Interface IcatalogueMetier

```
package org.miage.sid.metier;
import java.util.List;
import org.miage.sid.dao.Produit;
public interface ICatalogueMetier {
  public void addProduit(Produit p);
  public List<Produit> getAllProduits();
  public List<Produit> getProduits(String mc);
  public Produit getProduit(String reference);
  public void deleteProduit(String refernce);
  public void updateProduit(Produit p);
}
```

Implémentation CatalogueMetierImpl

```
package org.miage.sid.metier;
import java.util.List; import org.miage.sid.dao.ICatalogueDAO;
import org.miage.sid.dao.Produit;
public class CatalogueMetierImpl implements ICatalogueMetier {
   private ICatalogueDAO dao;
   /*Setter setDao pour l'injection*/
   public void setDao(ICatalogueDAO dao) {
        this.dao = dao;
   }
   @Override
   public void addProduit(Produit p) {
        dao.addProduit(p);
   }
@Override
public List<Produit> getAllProduits() {
   return dao.getAllProduits();
}
```

Implémentation CatalogueMetierImpl

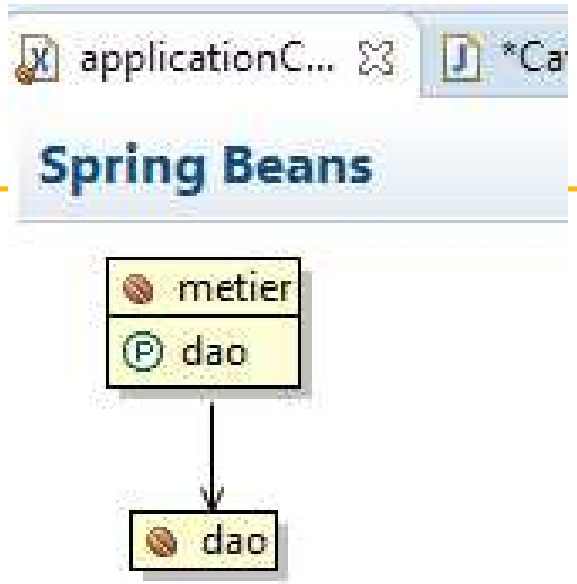
```
@Override
public List<Produit> getProduits(String mc) {
   return dao.getProduits(mc);
}
@Override
public Produit getProduit(String reference) {
   return dao.getProduit(reference);
}
@Override
public void deleteProduit(String refernce) {
   dao.deleteProduit(refernce);
}
@Override
public void updateProduit(Produit p) {
   dao.updateProduit(p);
}
}
```

INJECTION DES DEPENDANCES

med@youssfi.net

applicationContext.xml

```xml
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.springframework.org/schema/beans
   http://www.springframework.org/schema/beans/spring-beans.xsd">
  <bean id="dao" class="org.miage.sid.dao.CatalogueDaoImpl" init-method="init"></bean>
  <bean id="metier" class="org.miage.sid.metier.CatalogueMetierImpl">
    <property name="dao" ref="dao"></property>
  </bean>
</beans>
```



COUCHE WEB

med@youssfi.net

web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app version="2.5" xmlns="http://java.sun.com/xml/ns/javaee"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
   http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd">
<!-- The definition of the Root Spring Container shared by all Servlets and Filters -->
<context-param>
   <param-name>contextConfigLocation</param-name>
   <param-value>classpath*:applicationContext.xml</param-value>
</context-param>
<!-- Creates the Spring Container shared by all Servlets and Filters -->
<listener>
<listener-class>org.springframework.web.context.ContextLoaderListener</listener-class>
</listener>
```

web.xml

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

/WEB-INF/spring/appServlet/servlet-context.xml

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

/WEB-INF/spring/appServlet/servlet-context.xml

```
<!-- DispatcherServlet Context: defines this servlet's request-processing
infrastructure -->

<!-- Enables the Spring MVC @Controller programming model -->

<annotation-driven />

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

<resources mapping="/resources/**" location="/resources/" />

<!-- Resolves views selected for rendering by @Controllers to .jsp
resources in the /WEB-INF/views directory -->

<beans:bean
class="org.springframework.web.servlet.view.InternalResourceViewResolver">

     <beans:property name="prefix" value="/WEB-INF/views/" />

     <beans:property name="suffix" value=".jsp" />

</beans:bean>

<context:component-scan base-package="org.miage.sid" />

</beans:beans>
```

CatalogueController

```
package org.miage.sid.web;
import javax.validation.Valid;
import org.miage.sid.dao.Produit;
import org.miage.sid.metier.ICatalogueMetier;
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.RequestMapping;
import org.springframework.web.bind.annotation.RequestParam;
@Controller
public class CatalogueController {
  @Autowired
  private ICatalogueMetier metier;
```

CatalogueController

```
@RequestMapping("/index")
public String index(Model model){
   model.addAttribute("produit",new Produit());
   model.addAttribute("produits", metier.getAllProduits());
   return "produits";
}
@RequestMapping("/saveProduit")
public String save(@Valid Produit p,BindingResult bindingResult,Model
   model){
   if(bindingResult.hasErrors()){
        model.addAttribute("produits", metier.getAllProduits());
        return "produits";
   }
   metier.addProduit(p);
   model.addAttribute("produit",new Produit());
   model.addAttribute("produits", metier.getAllProduits());
   return "produits";
}
```

CatalogueController

```
@RequestMapping("/deleteProduit")
public String delete(@RequestParam  String ref,Model model){
   metier.deleteProduit(ref);
   model.addAttribute("produit",new Produit());
   model.addAttribute("produits", metier.getAllProduits());
   return "produits";
}
@RequestMapping("/editProduit")
public String edit(@RequestParam  String ref,Model model){
   model.addAttribute("produit",metier.getProduit(ref));
   model.addAttribute("produits", metier.getAllProduits());
   return "produits";
}
}
```

La vue produits.jsp

```
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
<%@ taglib uri="http://www.springframework.org/tags/form" prefix="f" %>
<!DOCTYPE html">
<html>
<head>
 <meta charset="UTF-8">
<title>Catalogue de produits</title>
 <link rel="stylesheet" type="text/css" href="<%=request.getContextPath()
  %>/resources/css/style.css">
</head>
<body>
 <div id="formProduits">
   <f:form modelAttribute="produit" method="post" action="saveProduit">
     <table>
       <tr>
         <td>Référence:</td>
         <td><f:input path="reference"/></td>
         <td><f:errors path="reference" cssClass="error"/></td>
       </tr>
```

La vue produits.jsp

```
        <tr>
          <td>Désignation:</td>
          <td><f:input path="designation"/></td>
          <td><f:errors path="designation" cssClass="error"/></td>
        </tr>
        <tr>
          <td>Prix:</td> <td><f:input path="prix"/></td>
          <td><f:errors path="prix" cssClass="error"/></td>
        </tr>
        <tr>
          <td>Quantité:</td><td><f:input path="quantite"/></td>
          <td><f:errors path="quantite" cssClass="error"/></td>
        </tr>
        <tr>
         <td><input type="submit" value="Save"></td>
        </tr>
      </table>
    </f:form>
  </div>
```

La vue produits.jsp

```
<div id="listProduits">
   <table class="table1">
     <tr>
       <th>REF</th><th>DESIGNATION</th>
       <th>PRIX</th><th>QUANTITE</th>
     </tr>
     <c:forEach items="${produits}" var="p">
       <tr>
         <td>${p.reference}</td>
         <td>${p.designation}</td>
         <td>${p.prix}</td>
         <td>${p.quantite}</td>
         <td><a href="deleteProduit?ref=${p.reference}">Supprimer</a></td>
         <td><a href="editProduit?ref=${p.reference}">Edit</a></td>
       </tr>
     </c:forEach>
   </table>
 </div>
</body>
</html>
```

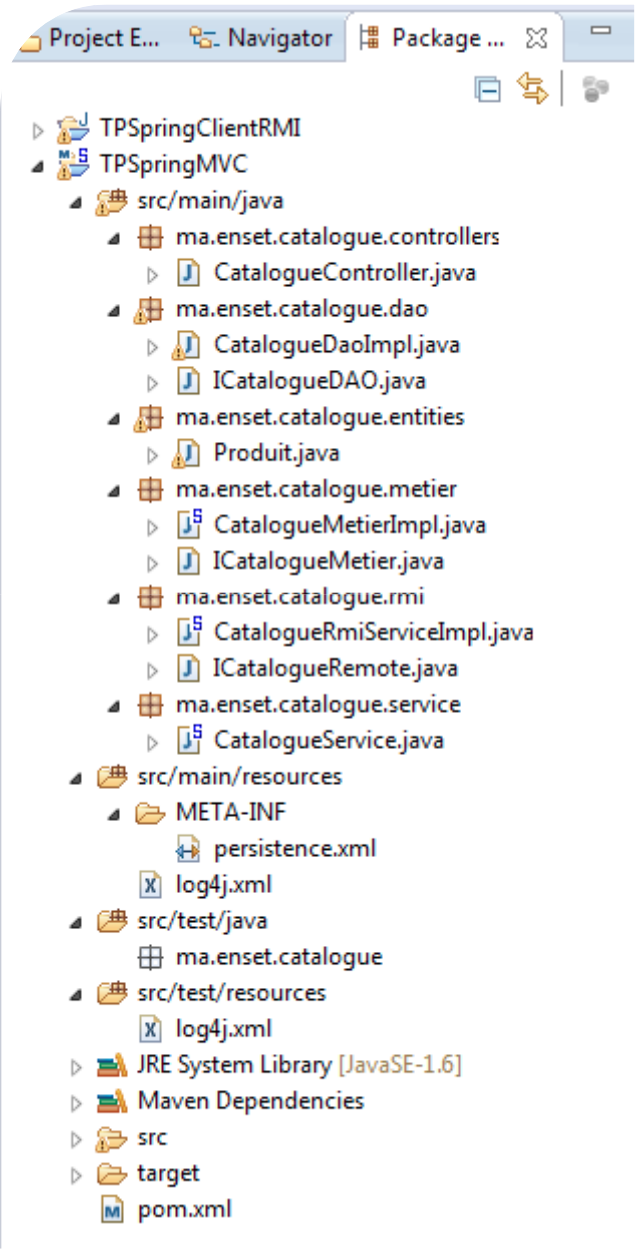
La feuille de style css : style.css

```
div{
   padding: 10px;
   margin: 10px;
   border: 1px dotted gray;
}
.table1 th{
   border: 1px dotted gray;
   padding: 10px;
   margin: 10px;
   background: pink;
}
.table1 td{
   border: 1px dotted gray;
   padding: 10px;
   margin: 10px;
   background: white;
}
.error{
   color: red;
}
```

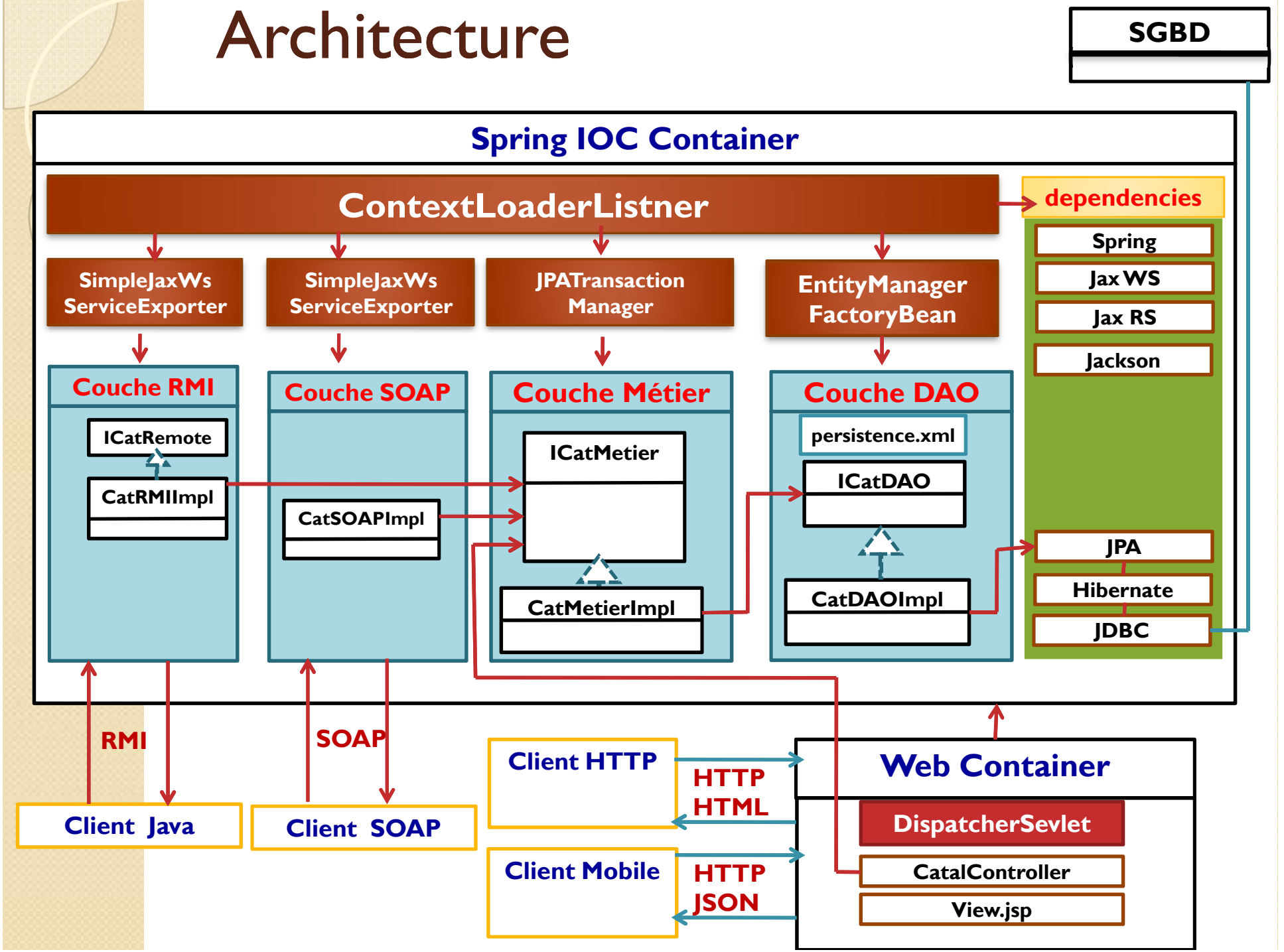
Application

- On souhait créer une application qui permet de gérer des produits.
- Chaque produit est défini par son code, son nom et son prix.
- L'application doit permettre de:
  - Saisir et ajouter un nouveau produit
  - Afficher tous les produits
  - Supprimer un produit
  - Ajouter une liste de produits envoyés par un client HTTP dans un fichier Excel
- Les fonctionnalités de cette application doivent être accessibles via :
  - Un client HTTP (Navigateur Web)
  - Un client SOAP (Application .Net)
  - Un client RMI (Client Lourd Swing Java)
  - Un client Androïde (HTTP, JSON)

Structure du projet



Architecture



Maven dependencies : Spring

pom.xml

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

Maven Dependencies

```
<!-- @Inject -->
<dependency>
   <groupId>javax.inject</groupId>
   <artifactId>javax.inject</artifactId>
   <version>1</version>
</dependency>
<!-- AspectJ -->
<dependency>
   <groupId>org.aspectj</groupId>
   <artifactId>aspectjrt</artifactId>
   <version>${org.aspectj-version}</version>
</dependency>
<!-- Spring -->
<dependency>
   <groupId>org.springframework</groupId>
   <artifactId>spring-context</artifactId>
   <version>${org.springframework-version}</version>
   <exclusions>
   <!-- Exclude Commons Logging in favor of SLF4j -->
         <exclusion>
           <groupId>commons-logging</groupId>
           <artifactId>commons-logging</artifactId>
         </exclusion>
   </exclusions>
</dependency>
```

Maven Dependencies : Spring

```
<!-- Spring -->
<dependency>
   <groupId>org.springframework</groupId>
   <artifactId>spring-webmvc</artifactId>
   <version>${org.springframework-version}</version>
</dependency>
<dependency>
   <groupId>org.springframework</groupId>
   <artifactId>spring-orm</artifactId>
   <version>${org.springframework-version}</version>
</dependency>
<dependency>
   <groupId>org.springframework</groupId>
   <artifactId>spring-beans</artifactId>
   <version>${org.springframework-version}</version>
</dependency>
<dependency>
   <groupId>org.springframework</groupId>
   <artifactId>spring-tx</artifactId>
   <version>3.2.2.RELEASE</version>
</dependency>
```

Maven Dependencies

```
<!-- Hibernte  JPA -->
<dependency>
   <groupId>org.hibernate</groupId>
   <artifactId>hibernate-commons-annotations</artifactId>
   <version>3.2.0.Final</version>
</dependency>
<dependency>
   <groupId>org.hibernate</groupId>
   <artifactId>hibernate-entitymanager</artifactId>
   <version>3.6.0.Final</version>
</dependency>
<dependency>
   <groupId>org.hibernate</groupId>
   <artifactId>hibernate-validator</artifactId>
   <version>4.1.0.Final</version>
</dependency>
<dependency>
   <groupId>org.hibernate.javax.persistence</groupId>
   <artifactId>hibernate-jpa-2.0-api</artifactId>
   <version>1.0.0.Final</version>
</dependency>
```

Maven Dependencies : Logging

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

Maven Dependencies : Lo4j

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

med@youssfi.net

Maven Dependencies :Web

```
<!-- Servlet -->
<dependency>
   <groupId>javax.servlet</groupId>
   <artifactId>servlet-api</artifactId>
   <version>2.5</version>
   <scope>provided</scope>
</dependency>
<dependency>
   <groupId>javax.servlet.jsp</groupId>
   <artifactId>jsp-api</artifactId>
   <version>2.1</version>
   <scope>provided</scope>
</dependency>
<dependency>
   <groupId>javax.servlet</groupId>
   <artifactId>jstl</artifactId>
   <version>1.2</version>
</dependency>
```

Maven Dependencies

```
<!-- Test -->
<dependency>
    <groupId>junit</groupId>
    <artifactId>junit</artifactId>
    <version>4.7</version>
    <scope>test</scope>
</dependency>
<!- MySQL JDBC Driver -->
<dependency>
    <groupId>mysql</groupId>
    <artifactId>mysql-connector-java</artifactId>
    <version>5.1.6</version>
</dependency>
<!- To generate JSON with jackson -->
<dependency>
    <groupId>org.codehaus.jackson</groupId>
    <artifactId>jackson-core-asl</artifactId>
    <version>1.9.10</version>
</dependency>
<dependency>
    <groupId>org.codehaus.jackson</groupId>
    <artifactId>jackson-mapper-asl</artifactId>
    <version>1.9.10</version>
</dependency>
```

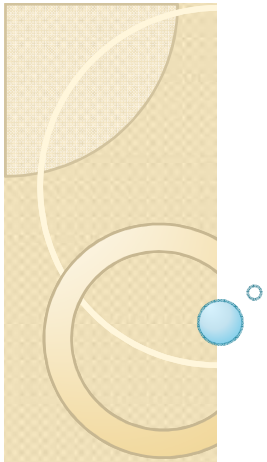
Maven Dependencies : Spring Securuty

```
<!-- Spring Security -->
<dependency>
   <groupId>org.springframework.security</groupId>
   <artifactId>spring-security-core</artifactId>
   <version>3.2.0.RELEASE</version>
</dependency>
<dependency>
   <groupId>org.springframework.security</groupId>
   <artifactId>spring-security-config</artifactId>
   <version>3.2.0.RELEASE</version>
</dependency>
<dependency>
   <groupId>org.springframework.security</groupId>
   <artifactId>spring-security-web</artifactId>
   <version>3.2.0.RELEASE</version>
</dependency>
</dependencies>
```

Maven Dependencies : File Upload et Excel

```
<!-- Apache Commons Upload -->
<dependency>
   <groupId>commons-fileupload</groupId>
   <artifactId>commons-fileupload</artifactId>
   <version>1.2.2</version>
</dependency>
<dependency>
   <groupId>commons-io</groupId>
   <artifactId>commons-io</artifactId>
   <version>1.3.2</version>
</dependency>
<!-- Apache Excel -->
<dependency>
    <groupId>org.apache.poi</groupId>
    <artifactId>poi</artifactId>
    <version>3.7</version>
 </dependency>
```

med@youssfi.net



Entities

Entité : Produit

```
package ma.enset.catalogue.entities;
import java.io.Serializable;import javax.persistence.*;
import javax.validation.constraints.Size;
import org.hibernate.validator.constraints.NotEmpty;
@Entity
public class Produit implements Serializable {
   @Id
   @GeneratedValue(strategy=GenerationType.IDENTITY)
  private Long idProduit;
   @NotEmpty
   @Size(max=60,min=3)
   private String nomProduit;
   private double prix;
  // Constructeur sans paramètre
  // Constructeur avec paramètres (nom, prix)
  // Getters et Setters
}
```

Couche DAO

med@youssfi.net

Interface DAO

```
package ma.enset.catalogue.dao;

import java.util.List;
import ma.enset.catalogue.entities.Produit;
public interface ICatalogueDAO {
    public void addProduit(Produit p);
    public List<Produit> listProduits();
    public void deleteProduit(Long idP);
    public Produit editProduit(Long idP);
    public void updateProduit(Produit p);
public List<Produit> chercherProduits(String motCle);
}
```

Implémentation DAO

```
package ma.enset.catalogue.dao;
import java.util.List;import javax.persistence.EntityManager;
import javax.persistence.PersistenceContext;
import javax.persistence.Query;import ma.enset.catalogue.entities.Produit;
public class CatalogueDaoImpl implements ICatalogueDAO {
   @PersistenceContext
   private EntityManager entityManager;

   @Override
   public void addProduit(Produit p) {
     entityManager.persist(p);
   }
   @Override
   public List<Produit> listProduits() {
     Query req=entityManager.createQuery("select p from Produit p");
     return req.getResultList();
   }
```

Implémentation DAO

```
    @Override
    public void deleteProduit(Long idP) {
       Produit p=entityManager.find(Produit.class,idP);
       entityManager.remove(p);
    }
    @Override
    public Produit editProduit(Long idP) {
       Produit p=entityManager.find(Produit.class,idP);
       return p;
    }
    @Override
    public void updateProduit(Produit p) {
       entityManager.merge(p);
    }
@Override
public List<Produit> chercherProduits(String motCle) {
   Query req=entityManager.createQuery("select p from Produit p where p.nomProduit like :x");
   req.setParameter("x", "%"+motCle+"%");
return req.getResultList();
}
}
```

Couche Métier

Interface Métier

```
package ma.enset.catalogue.metier;
import java.util.List;
import ma.enset.catalogue.entities.Produit;
public interface ICatalogueMetier {
public void addProduit(Produit p);
    public List<Produit> listProduits();
    public void deleteProduit(Long idP);
    public Produit editProduit(Long idP);
    public void updateProduit(Produit p);
public List<Produit> chercherProduits(String
  motCle);
}
```

Implémentation métier

```
package ma.enset.catalogue.metier;
import java.util.List;
import org.springframework.transaction.annotation.Transactional;
import ma.enset.catalogue.dao.ICatalogueDAO;
import ma.enset.catalogue.entities.Produit;
public class CatalogueMetierImpl implements ICatalogueMetier {
private ICatalogueDAO dao;
public void setDao(ICatalogueDAO dao) {
  this.dao = dao;
}
@Override
@Transactional
public void addProduit(Produit p) {
  dao.addProduit(p);
}
```

Implémentation métier

```
@Override
public List<Produit> listProduits() { return dao.listProduits(); }
@Override
@Transactional
public void deleteProduit(Long idP) { dao.deleteProduit(idP); }
@Override
public Produit editProduit(Long idP) { return dao.editProduit(idP); }
@Override
@Transactional
public void updateProduit(Produit p) {
  dao.updateProduit(p);
}
@Override
@Transactional
public List<Produit> chercherProduits(String motCle) {
   return dao.chercherProduits(motCle);
}
}
```

Web Service SOAP

```
package ma.enset.catalogue.service;
import java.util.List;
import javax.jws.WebParam;
import javax.jws.WebService;
import org.springframework.beans.factory.annotation.Autowired;
import ma.enset.catalogue.entities.Produit;
import ma.enset.catalogue.metier.ICatalogueMetier;
@WebService
public class CatalogueService {
    private ICatalogueMetier metier;
     public void setMetier(ICatalogueMetier metier) {
         this.metier = metier;
     }
    public List<Produit> listProduits(){
         return metier.listProduits();
    }
    public Produit getProduit(@WebParam(name="idProduit")Long idP){
         return metier.editProduit(idP);
    }}
```

Service RMI : Interface Remote

```
package ma.enset.catalogue.rmi;
import java.rmi.Remote;
import java.rmi.RemoteException;
import java.util.List;
import ma.enset.catalogue.entities.Produit;
public interface ICatalogueRemote extends Remote {
  public void addProduit(Produit p)throws RemoteException;;
  public List<Produit> listProduits()throws RemoteException;;
  public void deleteProduit(Long idP)throws RemoteException;;
  public Produit editProduit(Long idP)throws RemoteException;;
  public void updateProduit(Produit p)throws RemoteException;;
}
```

Service RMI : Implémentation du service RMI

```
package ma.enset.catalogue.rmi;
import java.rmi.RemoteException;
import java.util.List;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Component;
import ma.enset.catalogue.entities.Produit;
import ma.enset.catalogue.metier.ICatalogueMetier;
public class CatalogueRmiServiceImpl implements ICatalogueRemote {
   private ICatalogueMetier metier;
   public void setMetier(ICatalogueMetier metier) {
        this.metier = metier;
   }
@Override
public void addProduit(Produit p) throws RemoteException {
   metier.addProduit(p);
}
```

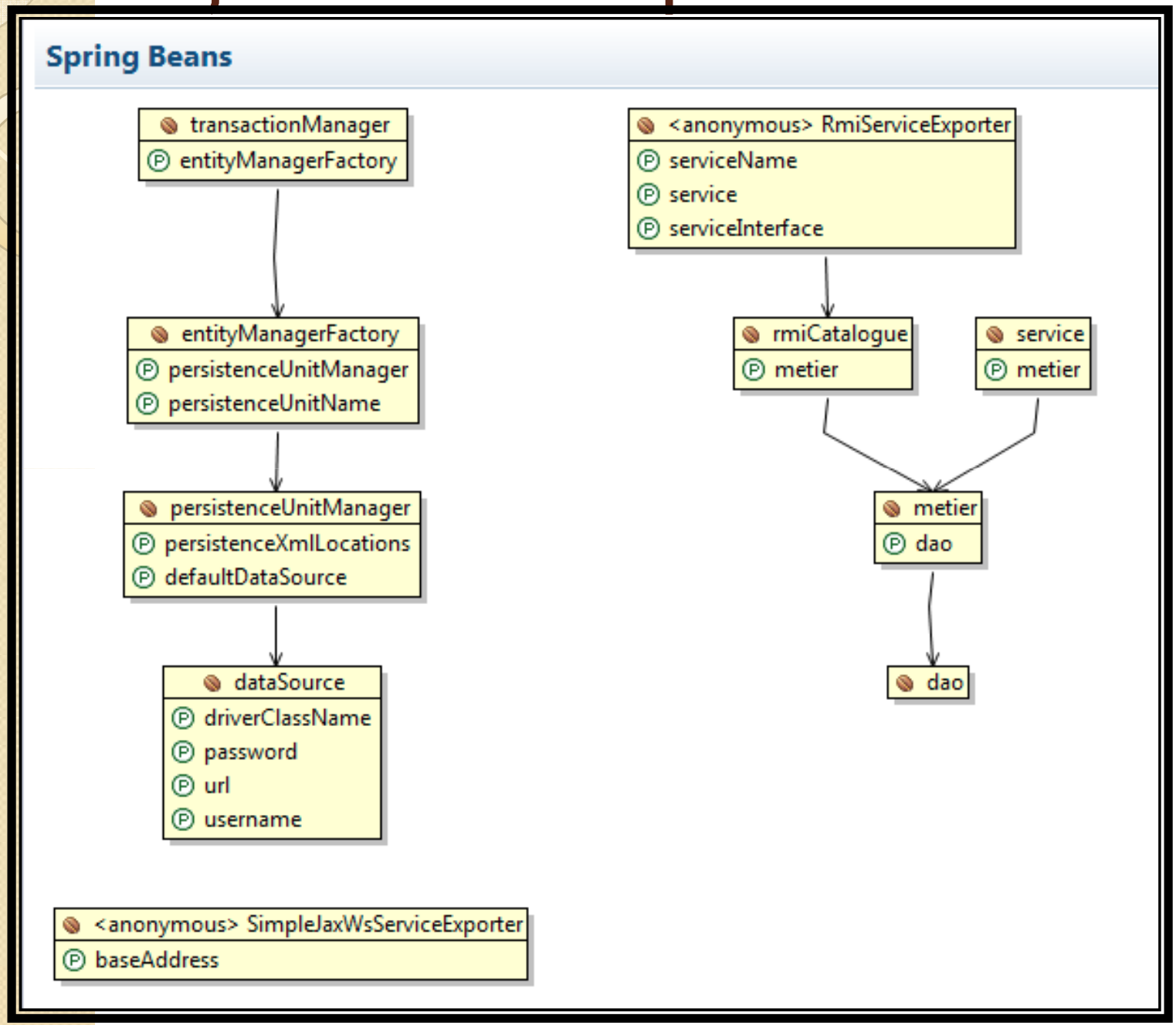
Service RMI : Implémentation du service RMI

```
@Override
public List<Produit> listProduits() throws RemoteException {
   return metier.listProduits();
}
@Override
public void deleteProduit(Long idP) throws RemoteException {
   metier.deleteProduit(idP);
}
@Override
public Produit editProduit(Long idP) throws RemoteException {
   return metier.editProduit(idP);
}
@Override
public void updateProduit(Produit p) throws RemoteException {
   metier.updateProduit(p);
}
}
```

main/java/resources/META-INF/persistence.xml

```xml
<?xml version="1.0" encoding="UTF-8"?>
<persistence xmlns="http://java.sun.com/xml/ns/persistence"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation="http://java.sun.com/xml/ns/persistence
   http://java.sun.com/xml/ns/persistence/persistence_2_0.xsd"
   version="2.0">
    <persistence-unit name="MY_P_U" transaction-type="RESOURCE_LOCAL">
        <provider>org.hibernate.ejb.HibernatePersistence</provider>
       <properties>
            <property name="hibernate.show_sql" value="true"/>
            <property name="hibernate.hbm2ddl.auto" value="update"/>
        </properties>
    </persistence-unit>
</persistence>
```

Injection des dépendances



Injection des dépendances

/WEB-INF/spring/root-context.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:context="http://www.springframework.org/schema/context"
xmlns:p="http://www.springframework.org/schema/p"
xmlns:tx="http://www.springframework.org/schema/tx"
xsi:schemaLocation="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-3.2.xsd
http://www.springframework.org/schema/context http://www.springframework.org/schema/context/spring-context-3.1.xsd">
<!-- Root Context: defines shared resources visible to all other web components -->
<context:annotation-config></context:annotation-config>
<bean id="dao" class="ma.enset.catalogue.dao.CatalogueDaoImpl"></bean>
<bean id="metier" class="ma.enset.catalogue.metier.CatalogueMetierImpl">
  <property name="dao" ref="dao"></property>
</bean>
```

Injection des dépendances

/WEB-INF/spring/root-context.xml

```
<bean id="datasource"
   class="org.springframework.jdbc.datasource.DriverManagerDataSource">
     <property name="driverClassName" value="com.mysql.jdbc.Driver"></property>
     <property name="url" value="jdbc:mysql://locahost:3306/DB_SID_CAT"></property>
     <property name="username" value="root"></property>
     <property name="password" value=""></property>
   </bean>
<bean id="persistenceUnitManager"
   class="org.springframework.orm.jpa.persistenceunit.DefaultPersistenceUnitManager">
  <property name="persistenceXmlLocations">
    <list>
      <value>classpath*:META-INF/persistence.xml</value>
    </list>
  </property>
  <property name="defaultDataSource" ref="dataSource"></property>
</bean>
```

Injection des dépendances

/WEB-INF/spring/root-context.xml

```
<bean id="entityManagerFactory"
   class="org.springframework.orm.jpa.LocalContainerEntityManagerFactoryBean">
  <property name="persistenceUnitManager"
   ref="persistenceUnitManager"></property>
  <property name="persistenceUnitName" value="MY_P_U"></property>
</bean>
<bean id="transactionManager"
   class="org.springframework.orm.jpa.JpaTransactionManager">
  <property name="entityManagerFactory" ref="entityManagerFactory"></property>
</bean>
<tx:annotation-driven transaction-manager="transactionManager"/>
```

Injection des dépendances

/WEB-INF/spring/root-context.xml

```
<bean id="service" class="ma.enset.catalogue.service.CatalogueService">
 <property name="metier" ref="metier"></property>
</bean>
<bean class="org.springframework.remoting.jaxws.SimpleJaxWsServiceExporter">
  <property name="baseAddress" value="http://localhost:8080/services/"></property>
</bean>
<bean id="rmiCatalogue" class="ma.enset.catalogue.rmi.CatalogueRmiServiceImpl">
 <property name="metier" ref="metier"></property>
</bean>
<bean class="org.springframework.remoting.rmi.RmiServiceExporter">
  <property name="serviceName" value="catService"></property>
  <property name="service" ref="rmiCatalogue"></property>
  <property name="serviceInterface"
   value="ma.enset.catalogue.rmi.ICatalogueRemote"></property>
</bean>
</beans>
```

Couche Web : Contrôleur

```
package ma.enset.catalogue.controllers;
import java.util.List; import javax.validation.Valid;
import ma.enset.catalogue.entities.Produit; import ma.enset.catalogue.metier.ICatalogueMetier;
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.*;
@Controller
public class CatalogueController {
@Autowired
private ICatalogueMetier metier;
@RequestMapping(value="/index")
public String index(Model model){
   model.addAttribute("produit", new Produit());
   model.addAttribute("produits", metier.listProduits());
   return "produits";
}
```

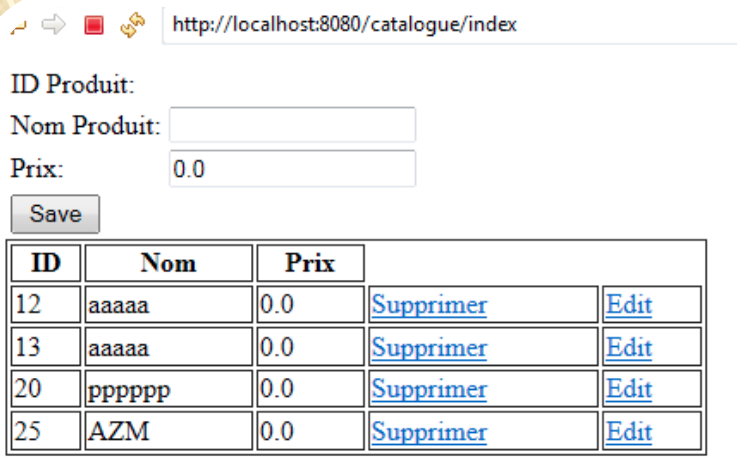
Couche Web : Contrôleur

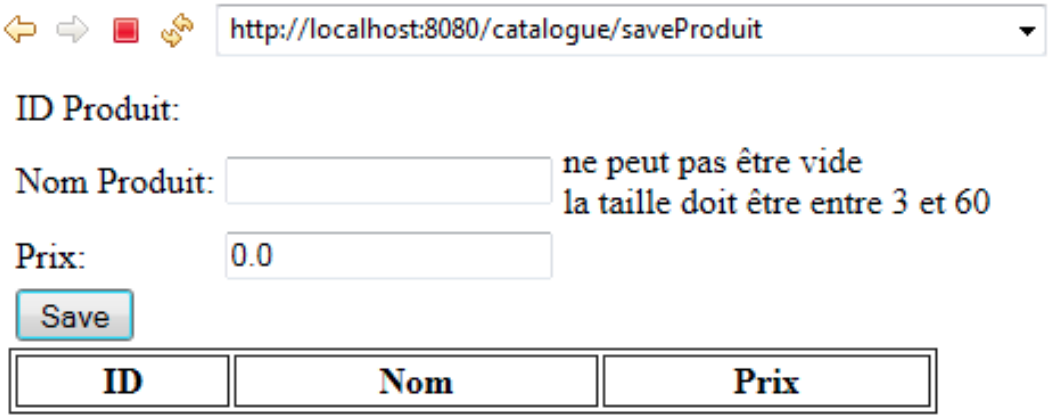
```
@RequestMapping(value="/saveProduit")
public String saveProduit(@Valid Produit p,BindingResult
   bindingResult,Model model){
if(bindingResult.hasErrors()) return "produits";
if(p.getIdProduit()==null)
 metier.addProduit(p);
else metier.updateProduit(p);
model.addAttribute("produit", new Produit());
model.addAttribute("produits", metier.listProduits());
return "produits";
}
@RequestMapping(value="/deleteProduit")
public String delete(@RequestParam(value="id") Long id, Model model){
   metier.deleteProduit(id);
   model.addAttribute("produit", new Produit());
   model.addAttribute("produits", metier.listProduits());
   return "produits";
}
```

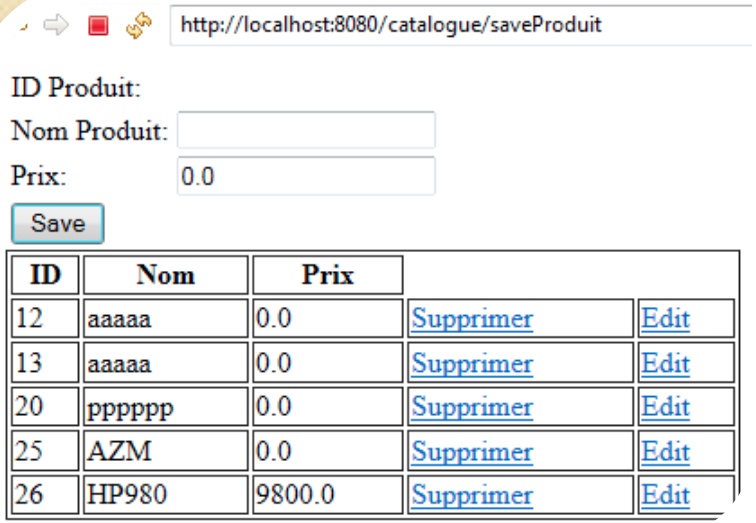
Couche Web : Contrôleur

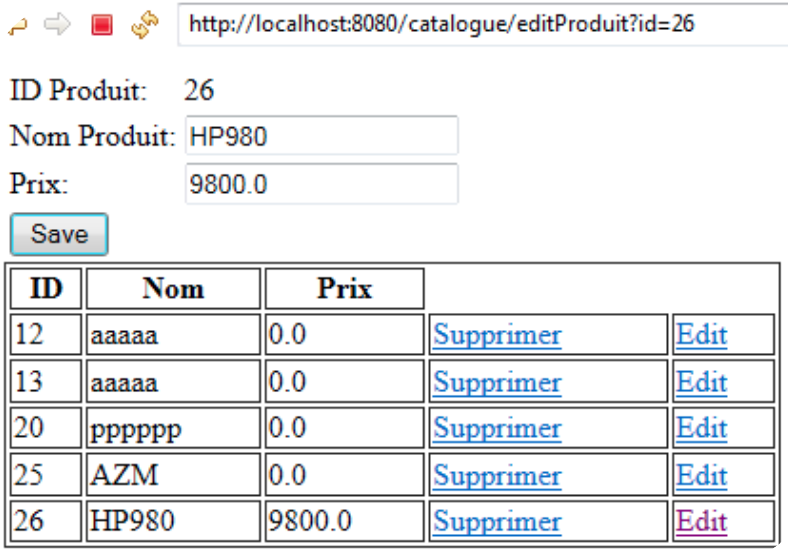
```
@RequestMapping(value="/editProduit/{id}")
public String edit(@PathVariable(value="id") Long id, Model model){
Produit p=metier.editProduit(id);
model.addAttribute("produit", p);
model.addAttribute("produits", metier.listProduits());
return "produits";
}
@RequestMapping(value="/listProduits/{mc}")
@ResponseBody
public List<Produit> listProduits(@PathVariable(value="mc")String mc){
   return metier.chercherProduits(mc);
}
}
```

Vue : produits.jsp









Vue : produits.jsp

http://localhost:8080/catalogue/saveProduit

ID Produit:
Nom Produit:
Prix: 0.0
Save

ID	Nom	Prix		
12	aaaaa	0.0	Supprimer	Edit
13	aaaaa	0.0	Supprimer	Edit
20	pppppp	0.0	Supprimer	Edit
25	AZM	0.0	Supprimer	Edit
26	HP980900	9800.0	Supprimer	Edit

http://localhost:8080/catalogue/deleteProduit?id=25

ID Produit:
Nom Produit:
Prix: 0.0
Save

ID	Nom	Prix		
12	aaaaa	0.0	Supprimer	Edit
13	aaaaa	0.0	Supprimer	Edit
20	pppppp	0.0	Supprimer	Edit
26	HP980900	9800.0	Supprimer	Edit

localhost:8080/catalogue/ × | Struts - med@youssfi.net × | localhost:808/services/?ws × | Insert title here

localhost:8080/catalogue/listProduits/a


```
[{"idProduit":12,"nomProduit":"aaaaa","prix":0.0},
{"idProduit":13,"nomProduit":"aaaaa","prix":0.0}]
```

Vue : produits.jsp

```
<%@ 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="f" %>
<!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>Insert title here</title>
</head>
<body>
  <f:form action="saveProduit" modelAttribute="produit">
     <table>
      <tr>
        <td>ID Produit:<f:hidden path="idProduit"/></td>
        <td>${produit.idProduit}</td>
      </tr>
```

Vue : produits.jsp

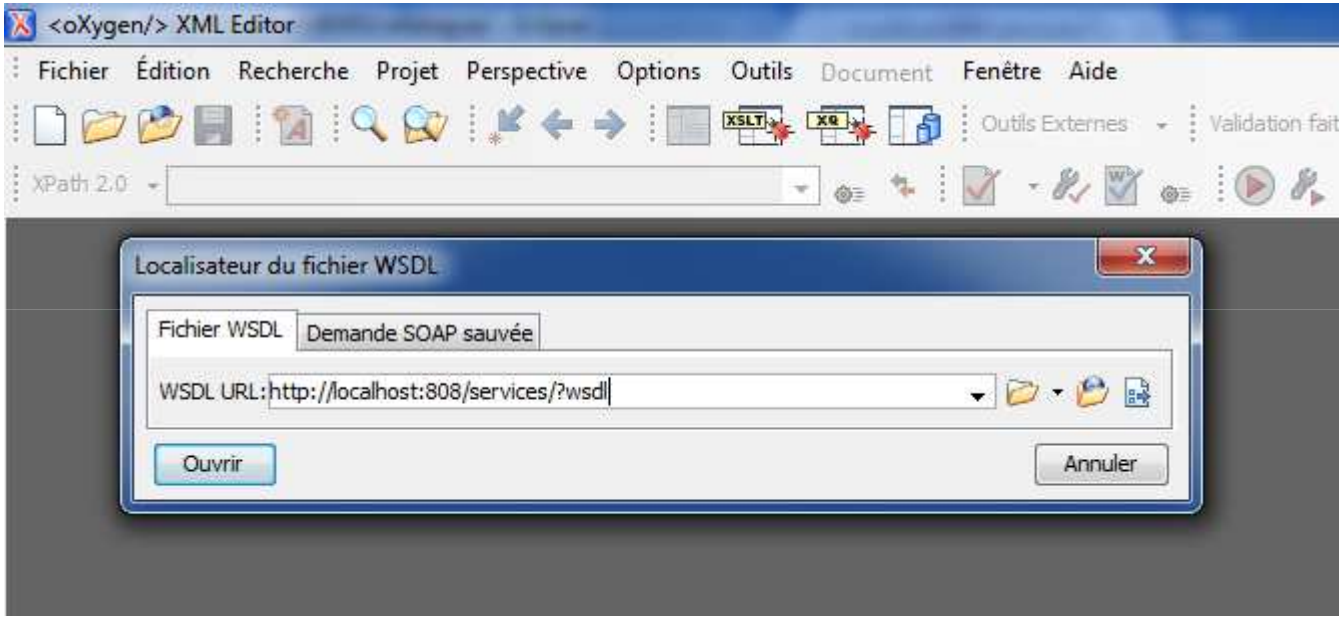
```
      <tr>
        <td>Nom Produit:</td>
        <td><f:input path="nomProduit"/></td>
        <td><f:errors path="nomProduit" cssClass="error"/></td>
      </tr>
      <tr>
        <td>Prix:</td>
        <td><f:input path="prix"/></td>
        <td><f:errors path="prix" cssClass="error"/></td>
      </tr>
      <tr>
        <td><input type="submit" value="Save"></td>
      </tr>
     </table>
  </f:form>
</body>
```

Vue : produits.jsp

```
<table border="1" width="80%">
  <tr>
   <th>ID</th><th>Nom</th><th>Prix</th>
  </tr>
  <c:forEach items="${produits}" var="p">
     <tr>
       <td>${p.idProduit }</td>
       <td>${p.nomProduit }</td>
       <td>${p.prix }</td>
       <td><a href="deleteProduit?id=${p.idProduit }">Supprimer</a></td>
       <td><a href="editProduit/${p.idProduit }">Edit</a></td>
     </tr>
  </c:forEach>
</table>
</html>
```

Tester le web service

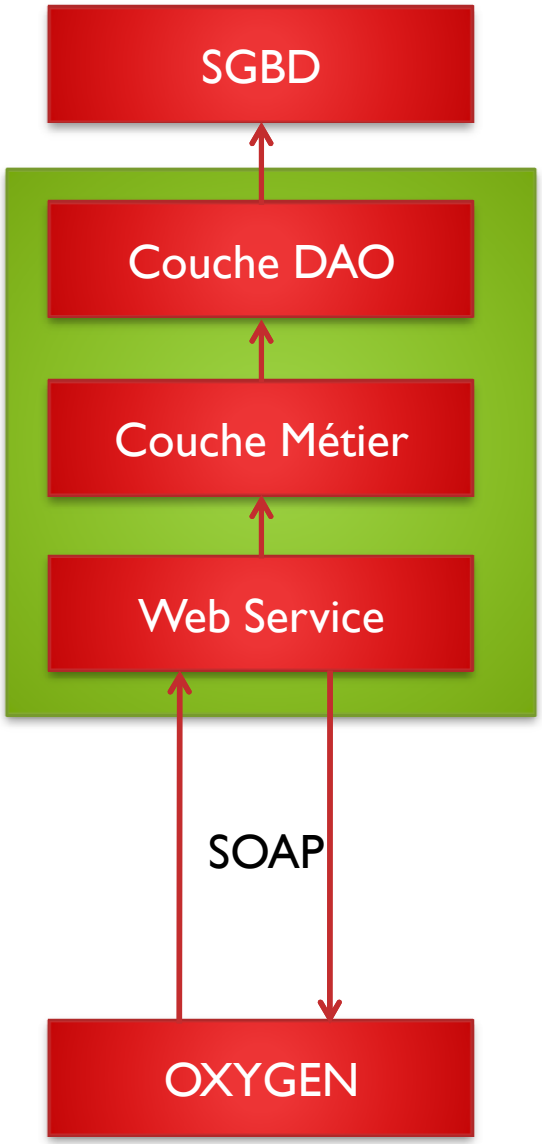
Tester les méthodes avec oxygen



Requête et réponse SOAP

```
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
    <SOAP-ENV:Header/>
    <SOAP-ENV:Body>
        <listProduits xmlns="http://service.catalogue.enset.ma/"/>
    </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

```
<?xml version="1.0" ?>
<S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
   <S:Body>
      <ns2:listProduitsResponse xmlns:ns2="http://service.catalogue.enset.ma/">
         <return>
            <idProduit>12</idProduit>
            <nomProduit>aaaaa</nomProduit>
            <prix>0.0</prix>
         </return>
         <return>
            <idProduit>13</idProduit>
            <nomProduit>aaaaa</nomProduit>
            <prix>0.0</prix>
         </return>
         <return>
            <idProduit>20</idProduit>
            <nomProduit>pppppp</nomProduit>
            <prix>0.0</prix>
         </return>
         <return>
            <idProduit>26</idProduit>
            <nomProduit>HP980900</nomProduit>
            <prix>9800.0</prix>
         </return>
      </ns2:listProduitsResponse>
   </S:Body>
</S:Envelope>
```



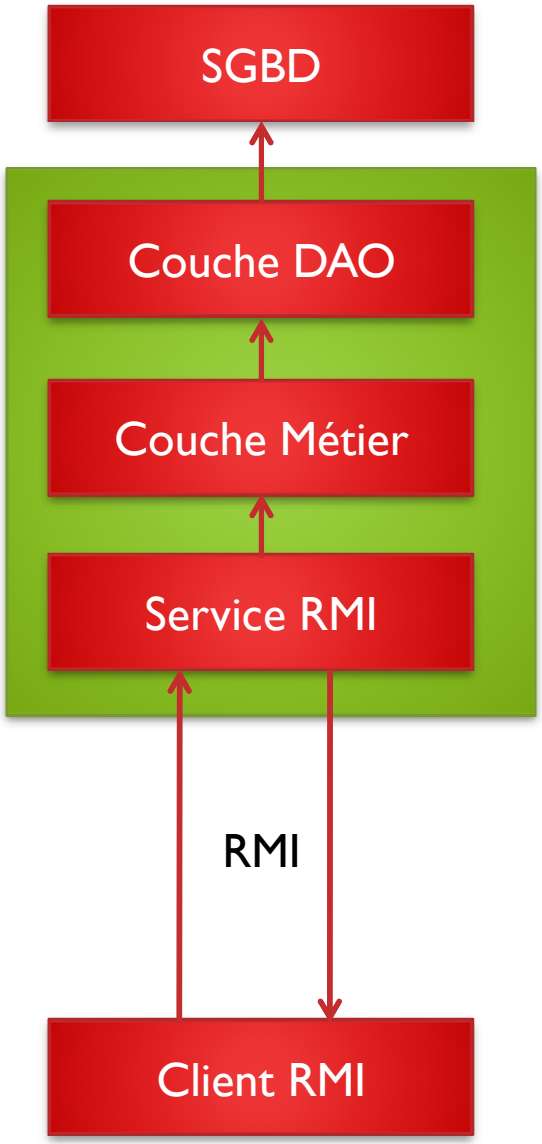
Requête et réponse SOAP

```
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
   <SOAP-ENV:Header/>
   <SOAP-ENV:Body>
      <getProduit xmlns="http://service.catalogue.enset.ma/">
         <idProduit xmlns="">26</idProduit>
      </getProduit>
   </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

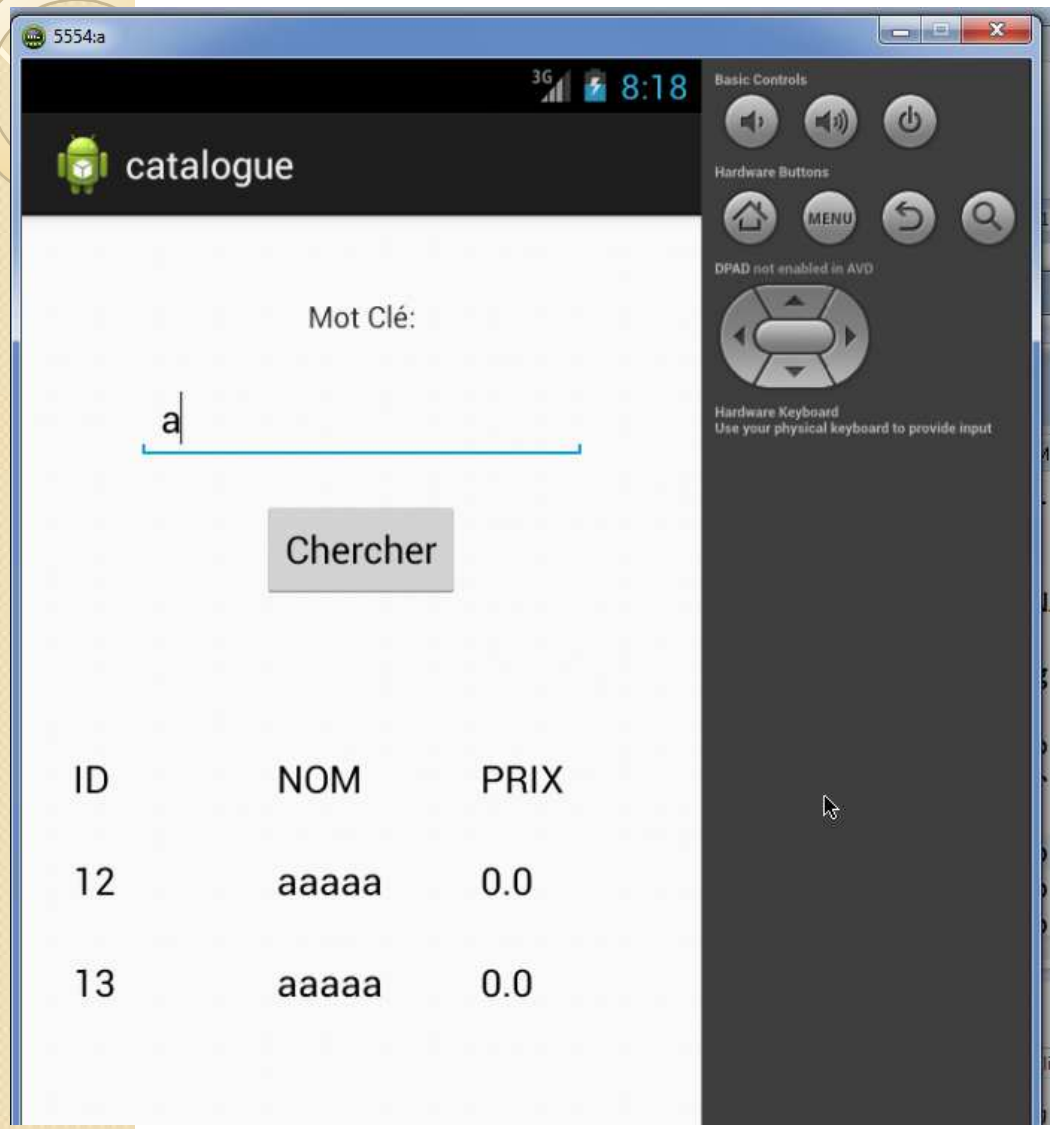
```
<?xml version="1.0" ?>
<S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
   <S:Body>
      <ns2:getProduitResponse xmlns:ns2="http://service.catalogue.enset.ma/">
         <return>
            <idProduit>26</idProduit>
            <nomProduit>HP980900</nomProduit>
            <prix>9800.0</prix>
         </return>
      </ns2:getProduitResponse>
   </S:Body>
</S:Envelope>
```

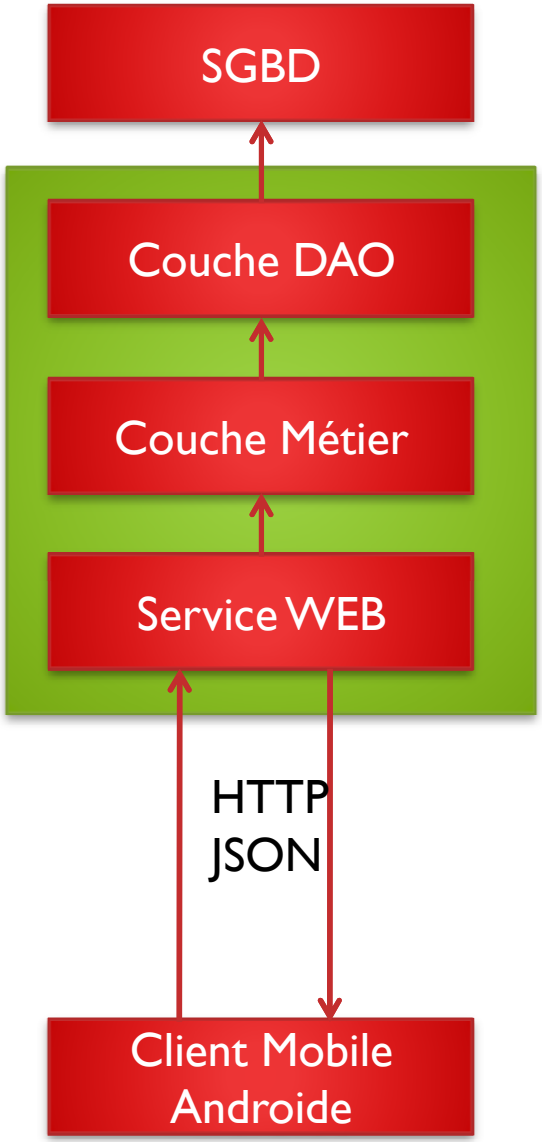
Client RMI

```
import java.rmi.Naming;
import java.util.List;
import ma.enset.catalogue.entities.Produit;
import ma.enset.catalogue.rmi.ICatalogueRemote;
public class ClientRMICatalogue {

public static void main(String[] args) {
try {
   ICatalogueRemote stub=(ICatalogueRemote)
   Naming.lookup("rmi://localhost:1099/catService");
   List<Produit> prods=stub.listProduits();
   for(Produit p:prods){
        System.out.println(p.getNomProduit());
        System.out.println(p.getPrix());
   }
   } catch (Exception e) {
        e.printStackTrace();
   }
}
}
```

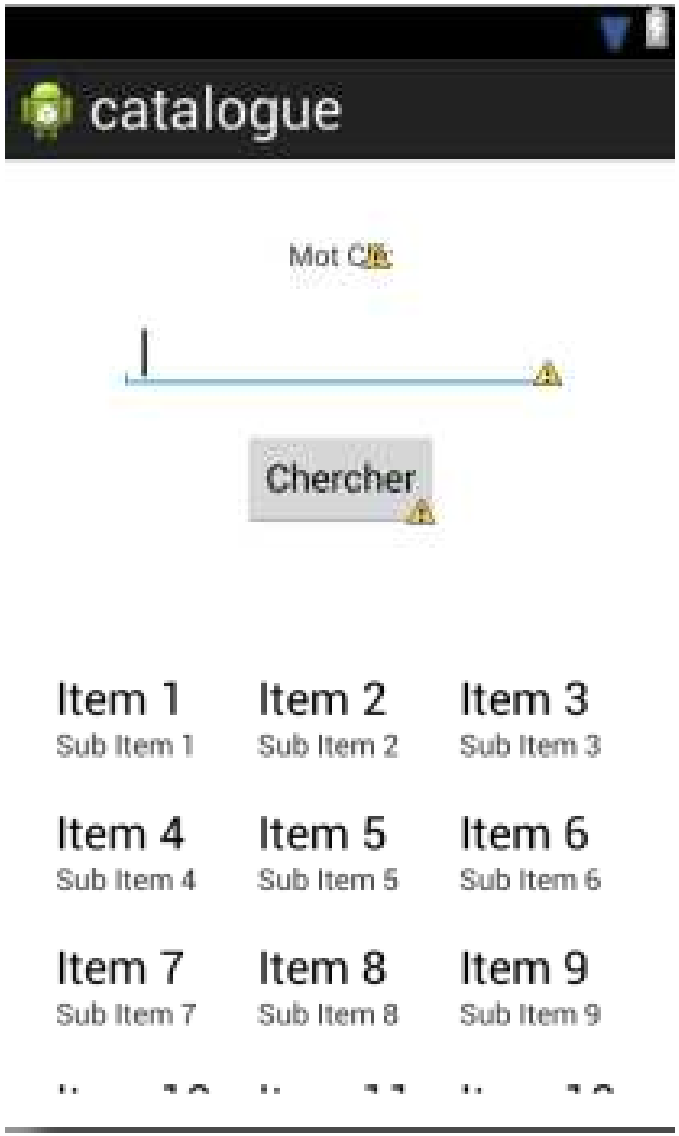


Client Androïde





Vue


catalogue
Mot C
Chercher
Item 1
Sub Item 1
Item 2
Sub Item 2
Item 3
Sub Item 3
Item 4
Sub Item 4
Item 5
Sub Item 5
Item 6
Sub Item 6
Item 7
Sub Item 7
Item 8
Sub Item 8
Item 9
Sub Item 9

L'activité

```
package com.example.catalogue;
import java.io.*;import org.apache.http.*;import org.apache.http.client.*;
import org.apache.http.client.methods.HttpGet;
import org.apache.http.impl.client.DefaultHttpClient;
import org.json.*;import android.os.*;import android.app.Activity;
import android.view.*;import android.view.View.OnClickListener;
import android.widget.*;
public class MainActivity extends Activity implements OnClickListener {
   private EditText editTextMC;      private Button buttonOK;
   private GridView gridViewProduits;
@Override
protected void onCreate(Bundle savedInstanceState) {
   super.onCreate(savedInstanceState);
   setContentView(R.layout.activity_main);
   buttonOK=(Button) findViewById(R.id.button1);
   editTextMC=(EditText) findViewById(R.id.editTextMC);
   gridViewProduits=(GridView) findViewById(R.id.gridView1);
   buttonOK.setOnClickListener(this);
StrictMode.ThreadPolicy threadPolicy=new
StrictMode.ThreadPolicy.Builder().permitAll().build();
StrictMode.setThreadPolicy(threadPolicy);
}
```

L'activité

```
@Override
public void onClick(View arg0) {
   Toast.makeText(getApplicationContext(), "Chargement",Toast.LENGTH_LONG).show();
   StringBuilder reponseHTTP = new StringBuilder();
    HttpClient client = new DefaultHttpClient();
    String mc=editTextMC.getText().toString();
    HttpGet httpGet = new HttpGet
    ("http://192.168.1.79:8080/catalogue/listProduits/"+mc);
    try {
      HttpResponse response = client.execute(httpGet);
      StatusLine statusLine = response.getStatusLine();
      int statusCode = statusLine.getStatusCode();
      if (statusCode == 200) {
        HttpEntity entity = response.getEntity();
        InputStream content = entity.getContent();
        BufferedReader reader = new BufferedReader(new InputStreamReader(content));
        String line;
        while ((line = reader.readLine()) != null) {
          reponseHTTP.append(line);
        }
```

L'activité

```
        JSONArray jsonArray=new JSONArray(reponseHTTP.toString());
        String[] produits=new String[(jsonArray.length()+1)*3];
    int index=-1;
      produits[++index]="ID";produits[++index]="NOM";produits[++index]="PRIX";
      for (int i = 0; i < jsonArray.length(); i++) {
        JSONObject jsonObject = jsonArray.getJSONObject(i);
        produits[++index]=jsonObject.getString("idProduit");
        produits[++index]=jsonObject.getString("nomProduit");
        produits[++index]=jsonObject.getString("prix");
      }
      ArrayAdapter<String> adapter = new ArrayAdapter<String>
        (this, android.R.layout.simple_list_item_1, produits);
        gridViewProduits.setAdapter(adapter);
      } else {
        Toast.makeText(getApplicationContext(), "Code ="+statusCode,
Toast.LENGTH_LONG).show();
      }    } catch (Exception e) {
    Toast.makeText(getApplicationContext(), e.getMessage(), Toast.LENGTH_LONG).show();
      e.printStackTrace();
    }}}
```

Upload d'un fichier Format Excel qui contient les produits

http://localhost:8080/catalogue/saveUploadedFile

Fichier Excel (Format.xls): esktop\ex\Classeur1.xls Parcourir...

Upload

Content

Nom	Prix
Aaaaa	600.0
Bbbbb	600.0
Ccccc	12.0
Ddddd	78.0
Eeeee	12.0

Classeur.x1.xls

C4 Nom

	A	B	C	D	E	F
1						
2						
3						
4			Nom	Prix		
5			Aaaaa	600		
6			Bbbbb	600		
7			Ccccc	12		
8			Ddddd	78		
9			Eeeee	12		
10						
11						
12						

Dépendances

```
<!-- Apache Commons Upload -->
<dependency>
   <groupId>commons-fileupload</groupId>
   <artifactId>commons-fileupload</artifactId>
   <version>1.2.2</version>
</dependency>

<!-- Apache Commons Upload -->
<dependency>
   <groupId>commons-io</groupId>
   <artifactId>commons-io</artifactId>
   <version>1.3.2</version>
</dependency>
<!-- Apache Excel -->
  <dependency>
        <groupId>org.apache.poi</groupId>
        <artifactId>poi</artifactId>
        <version>3.7</version>
   </dependency>
```

Classe UploadedFile

```
package ma.enset.catalogue.controllers;

import org.springframework.web.multipart.commons.CommonsMultipartFile;

public class UploadedFile {
  private CommonsMultipartFile file;

public CommonsMultipartFile getFile() {
return file;
}

public void setFile(CommonsMultipartFile file) {
this.file = file;
}

}
```

Controleur

```
package ma.enset.catalogue.controllers;
import java.io.File;import java.io.FileOutputStream;
import java.util.ArrayList;import java.util.HashMap;
import java.util.List;import java.util.Map;
import javax.servlet.http.HttpServletRequest;import javax.servlet.http.HttpServletResponse;
import org.apache.poi.hssf.usermodel.HSSFRow;import org.apache.poi.hssf.usermodel.HSSFSheet;
import org.apache.poi.hssf.usermodel.HSSFWorkbook;
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.ModelAttribute;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.servlet.HandlerExceptionResolver;
import org.springframework.web.servlet.ModelAndView;
import ma.enset.catalogue.entities.Produit;
import ma.enset.catalogue.metier.ICatalogueMetier;

@Controller
public class UploadProduitsController implements HandlerExceptionResolver {
    @Autowired
    private ICatalogueMetier metier;
   @RequestMapping(value="/formUpload")
public String formUpload(@ModelAttribute(value="form") UploadedFile form){
   return "formUpload";
}
```

Contrôleur : Sauvegarde du fichier

```
@RequestMapping(value="/saveUploadedFile")
public String saveUploadedFile(
  @ModelAttribute(value="form") UploadedFile form,
  BindingResult bindingResult,Model model){
  if(!bindingResult.hasErrors()){
  try {
    // Enregistrement du fichier
String filePath=
  System.getProperty("java.io.tmpdir")+"/"+form.getFile().getOriginalFi
  lename();
FileOutputStream outputStream=new FileOutputStream(new File(filePath));
outputStream.write(form.getFile().getFileItem().get());
outputStream.close();
System.out.println(form.getFile().getSize());
```

Contrôleur : Traitemet du fichier Excel

```
// Interprétation du fichier Excel
        HSSFWorkbook workbook=new HSSFWorkbook(form.getFile().getInputStream());
                HSSFSheet f1=workbook.getSheet("Feuil1");
                int l1=f1.getFirstRowNum();
                int l2=f1.getLastRowNum();
                List<Produit> produits=new ArrayList<Produit>();
                for(int i=l1+1;i<=l2;i++){
                Produit p=new Produit();
                HSSFRow r1=f1.getRow(i);
                int n1=r1.getFirstCellNum();
                p.setNomProduit(r1.getCell(n1).getStringCellValue());
                p.setPrix(r1.getCell(n1+1).getNumericCellValue());
                metier.addProduit(p);
                produits.add(p);
                }
                model.addAttribute("produits", produits);
} catch (Exception e) {
throw new RuntimeException(e);
}}
return "formUpload";
}
```

Contrôleur : Gestion des Exceptions

```
@Override
public ModelAndView resolveException(HttpServletRequest request,
HttpServletResponse response, Object o, Exception e) {
   Map<Object, Object> model=new HashMap<Object, Object>();
   model.put("errors", e.getMessage());
   model.put("form", new UploadedFile());
   return new ModelAndView("formUpload",(Map)model);
 }
}
```

Vue : formUpload.jsp

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
    pageEncoding="ISO-8859-1"%>
<%@taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<%@taglib prefix="f" uri="http://www.springframework.org/tags/form" %>
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<title>Insert title here</title>
</head>
<body>
   <f:form commandName="form" action="saveUploadedFile" enctype="multipart/form-data"
   method="post">
     <table>
       <tr>
         <td>Fichier Excel (Froamt XLS):</td>
         <td><f:input type="file" path="file"/>
         <td><f:errors path="*"></f:errors>${errors}</td>
       </tr>
       <tr>  <td><input type="submit" value="Upload"/></td>    </tr>
     </table>
   </f:form>
```

Vue : formUpload.jsp

```
<h3>Content</h3>
        <table border="1">
          <tr>
            <th>Nom</th><th>Prix</th>
          </tr>
          <c:forEach  items="${produits}" var="p">
            <tr>
              <td>${p.nomProduit}</td>
              <td>${p.prix}</td>
            </tr>
          </c:forEach>
        </table>
</body>
</html>
```

Limitation de la taille des fichiers uploadé

Fichier : **/WEB-INF/spring/appServlet/servlet-context.xml**

```
<beans:bean
   class="org.springframework.web.multipart.commons.CommonsMultipartResolver"
   id="multipartResolver">
  <beans:property name="maxUploadSize" value="100000"></beans:property>
</beans:bean>
```

SPRING SECURITY

med@youssfi.net

web.xml

- Déclarer le filtre DelegatingFilterProxy dans le fichier web.xml
- Toutes les requêtes HTTP passent par ce filtre.
- Le nom du filtre est : **springSecurityFilterChain**
- Ce nom devrait correspondre au nom d'un bean spring qui sera déployé par ContextLoaderListener et qui contient les règles de sécurité à exécuter.

```
 <!-- Spring Security -->
<filter>
    <filter-name>springSecurityFilterChain</filter-name>
    <filter-
  class>org.springframework.web.filter.DelegatingFilterProxy</filter-class>
</filter>
<filter-mapping>
    <filter-name>springSecurityFilterChain</filter-name>
    <url-pattern>/*</url-pattern>
</filter-mapping>
```

Base de données

- Créer deux tables :
  - Users : qui contient les utilisateurs autorisés à accéder à l'application
  - Roles : qui contient les rôles de chaque utilisateur

#	Nom	Type	Interclassement	Attributs	Null	Défaut	Extra
1	ID_USER	int(11)			Non	*Aucune*	AUTO_INCREMENT
2	username	varchar(15)	latin1_swedish_ci		Non	*Aucune*	
3	PASSWORD	varchar(100)	latin1_swedish_ci		Non	*Aucune*	
4	ACTIVED	tinyint(1)			Non	*Aucune*	

Nom	Type	Interclassement	Attributs	Null	Défaut	Extra
ID_ROLE	int(11)			Non	*Aucune*	AUTO_INCREMENT
ID_USER	int(11)			Non	*Aucune*	
ROLE_NAME	varchar(20)	latin1_swedish_ci		Non	*Aucune*	

ID_USER	username	PASSWORD	ACTIVED
1	admin1	e00cf25ad42683b3df678c61f42c6bda	1
2	admin2	c84258e9c39059a89ab77d846ddab909	1
3	user	ee11cbb19052e40b07aac0ca060c23ee	1

ID_ROLE	ID_USER	ROLE_NAME
1	1	ROLE_ADMIN_CAT
2	1	ROLE_ADMIN_PROD
3	2	ROLE_ADMIN_PROD
4	3	ROLE_USER

Base de données : Table Users

```
--
-- Structure de la table `users`
--
CREATE TABLE IF NOT EXISTS `users` (
  `ID_USER` int(11) NOT NULL AUTO_INCREMENT,
  `username` varchar(15) NOT NULL,
  `PASSWORD` varchar(100) NOT NULL,
  `ACTIVED` tinyint(1) NOT NULL,
  PRIMARY KEY (`ID_USER`),
  UNIQUE KEY `LOGIN` (`username`)
) ENGINE=InnoDB  DEFAULT CHARSET=latin1 AUTO_INCREMENT=4 ;
--
-- Contenu de la table `users`
--
INSERT INTO `users` (`ID_USER`, `username`, `PASSWORD`, `ACTIVED`) VALUES
(1, 'admin1', 'e00cf25ad42683b3df678c61f42c6bda', 1),
(2, 'admin2', 'c84258e9c39059a89ab77d846ddab909', 1),
(3, 'user', 'ee11cbb19052e40b07aac0ca060c23ee', 1);
```

Base de données : Table roles

```
--
-- Structure de la table `roles`
--
CREATE TABLE IF NOT EXISTS `roles` (
  `ID_ROLE` int(11) NOT NULL AUTO_INCREMENT,
  `ID_USER` int(11) NOT NULL,
  `ROLE_NAME` varchar(20) NOT NULL,
  PRIMARY KEY (`ID_ROLE`),
  KEY `ID_USER` (`ID_USER`)
) ENGINE=InnoDB  DEFAULT CHARSET=latin1 AUTO_INCREMENT=5 ;
--
-- Contenu de la table `roles`
--
INSERT INTO `roles` (`ID_ROLE`, `ID_USER`, `ROLE_NAME`) VALUES
(1, 1, 'ROLE_ADMIN_CAT'),
(2, 1, 'ROLE_ADMIN_PROD'),
(3, 2, 'ROLE_ADMIN_PROD'),
(4, 3, 'ROLE_USER');
--
-- Contraintes pour la table `roles`
--
ALTER TABLE `roles`
  ADD CONSTRAINT `roles_ibfk_1` FOREIGN KEY (`ID_USER`) REFERENCES `users` (`ID_USER`);
```

root-context.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:tx="http://www.springframework.org/schema/tx"
xmlns:context="http://www.springframework.org/schema/context"
xmlns:s="http://www.springframework.org/schema/security"
xsi:schemaLocation="http://www.springframework.org/schema/security
    http://www.springframework.org/schema/security/spring-security-3.2.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-3.2.xsd
http://www.springframework.org/schema/context http://www.springframework.org/schema/context/spring-context-3.2.xsd">

   <bean id="datasource" class="org.springframework.jdbc.datasource.DriverManagerDataSource">
     <property name="driverClassName" value="com.mysql.jdbc.Driver"></property>
     <property name="url" value="jdbc:mysql://localhost:3306/DB_SID_CAT"></property>
     <property name="username" value="root"></property>
     <property name="password" value=""></property>
   </bean>
```

root-context.xml

```
<s:http>
     <s:intercept-url pattern="/produits/**" access="ROLE_ADMIN_PROD"/>
     <s:intercept-url pattern="/categories/**" access="ROLE_ADMIN_CAT"/>
     <s:form-login login-page="/login" default-target-url="/produits/index"
      authentication-failure-url="/login" />
     <s:logout logout-success-url="/logout" />
   </s:http>
   <s:authentication-manager>
    <s:authentication-provider>
      <s:password-encoder hash="md5"></s:password-encoder>
      <s:jdbc-user-service data-source-ref="datasource"
     users-by-username-query="select username,password, actived
      from users where username=?"
      authorities-by-username-query="select u.username, r.role_name from users u, roles r
      where u.id_user = r.id_user and u.username =? " />
 <!--
      <s:user-service>
        <s:user name="admin1" password="admin1" authorities="ROLE_ADMIN_PROD"/>
        <s:user name="admin2" authorities="ROLE_ADMIN_CAT,ROLE_ADMIN_PROD" password="admin2" />
      </s:user-service>
         -->
    </s:authentication-provider>
   </s:authentication-manager>
</beans>
```

Contrôleur

```
package org.enset.sid.controllers;

import org.springframework.stereotype.Controller;
import
   org.springframework.web.bind.annotation.RequestMa
   pping;

@Controller
public class LoginController {
@RequestMapping("/login")
   public String login(){
   return "login";
   }
}
```

login.jsp

```
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<title>Insert title here</title>
</head>
<body>
 <form action="j_spring_security_check" method="post">
   <table>
     <tr>
       <td>Login</td>
       <td><input type="text" name="j_username"></td>
     </tr>
     <tr>
      <td>Pass word</td>
      <td><input type="password" name="j_password"></td>
     </tr>
     <tr>
      <td><input type="submit" value="Login"></td>
     </tr>
   </table>
 </form>
</body>
</html>
```