



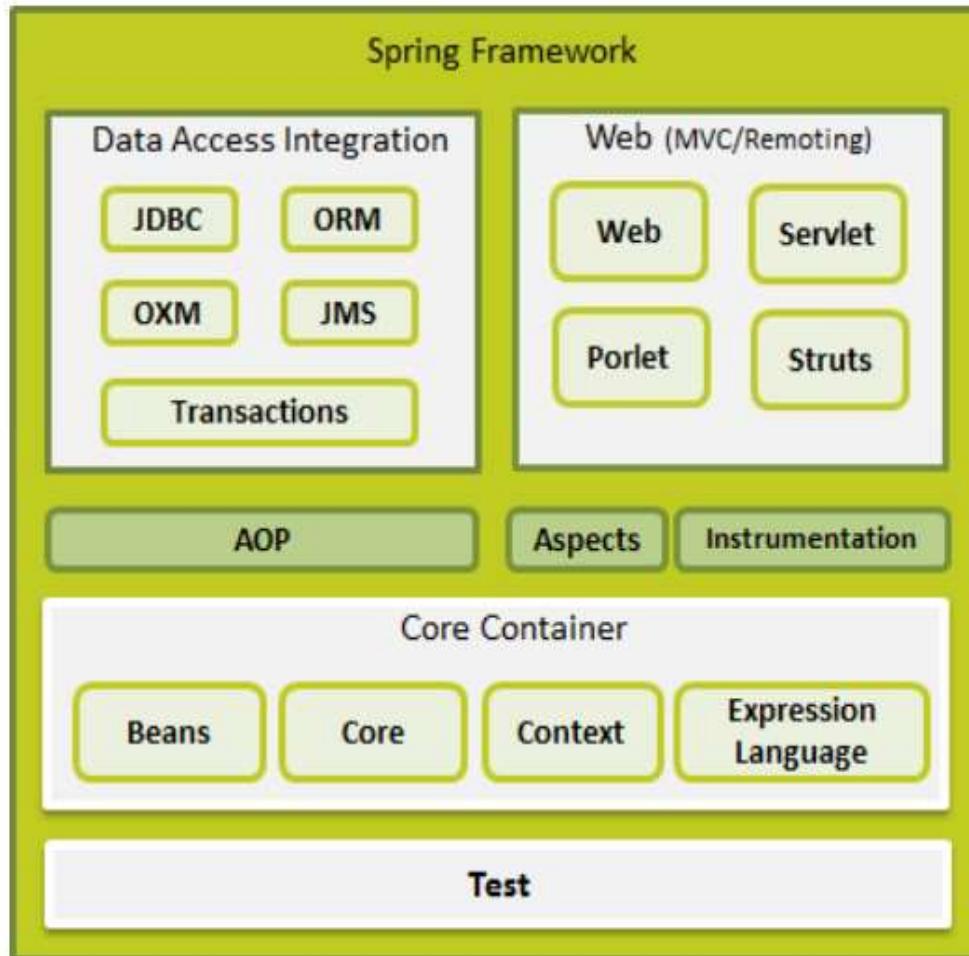
Spring

Java Application Framework

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.

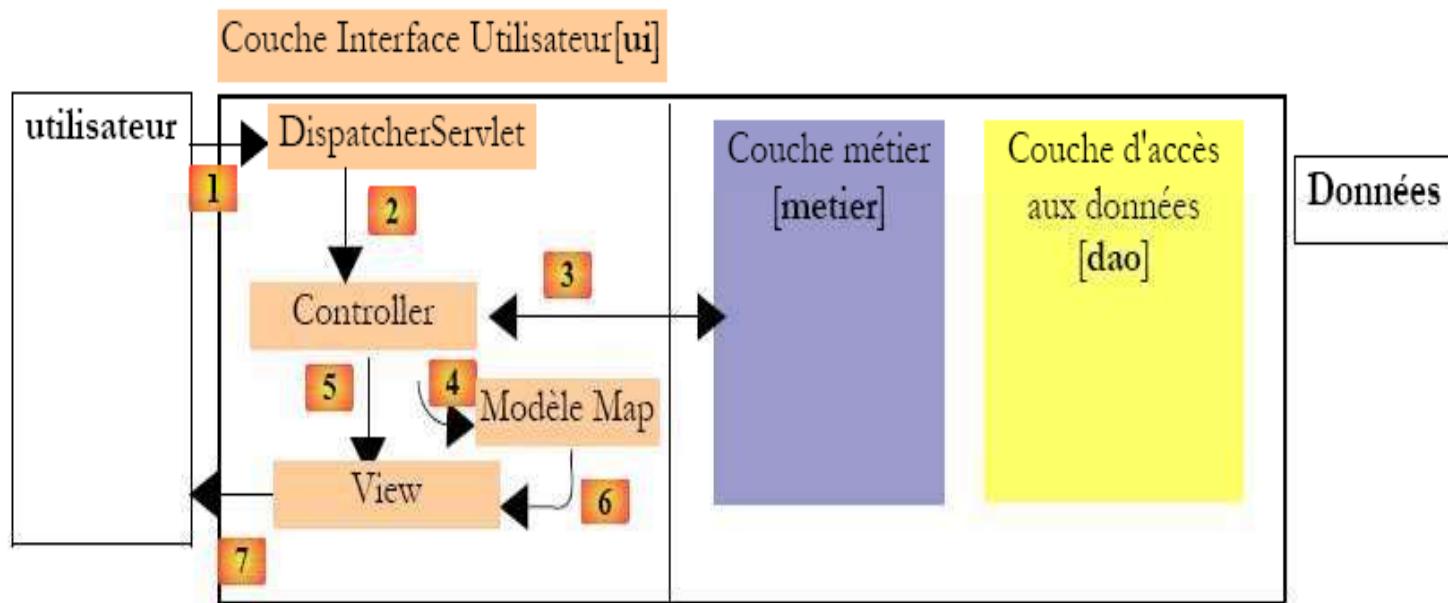
Spring Framework Architecture



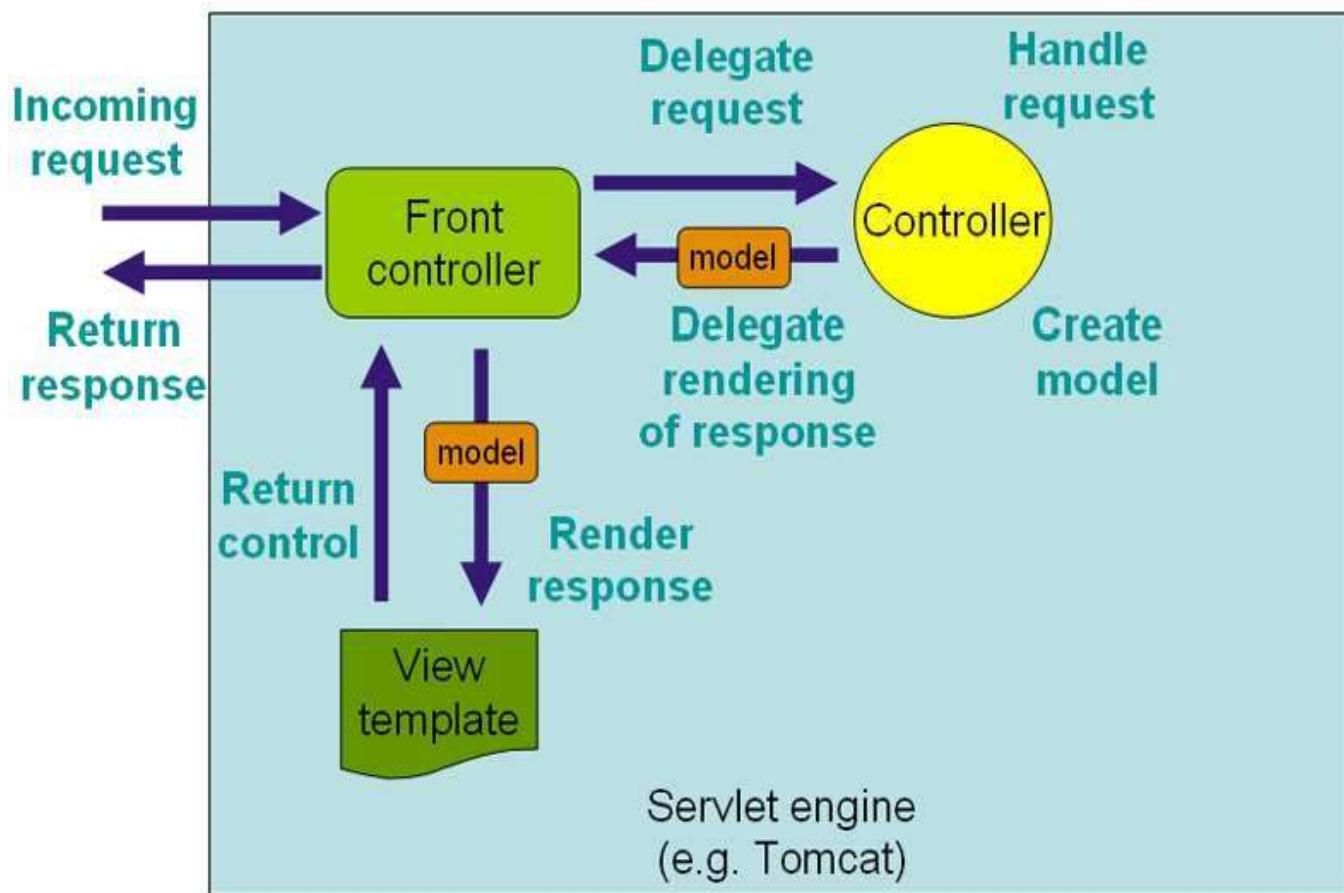
Spring MVC

Spring MVC

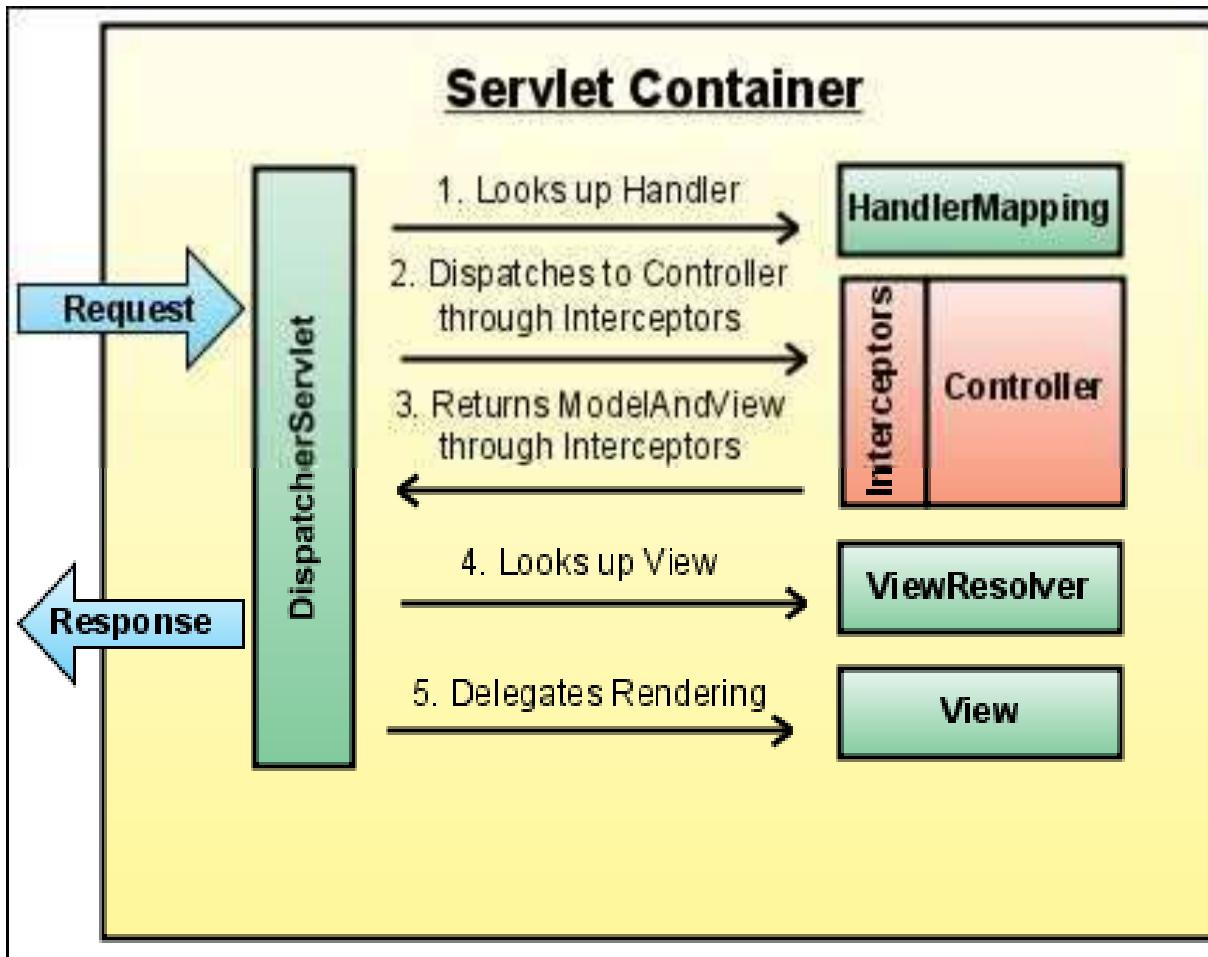
- Architecture de Spring MVC



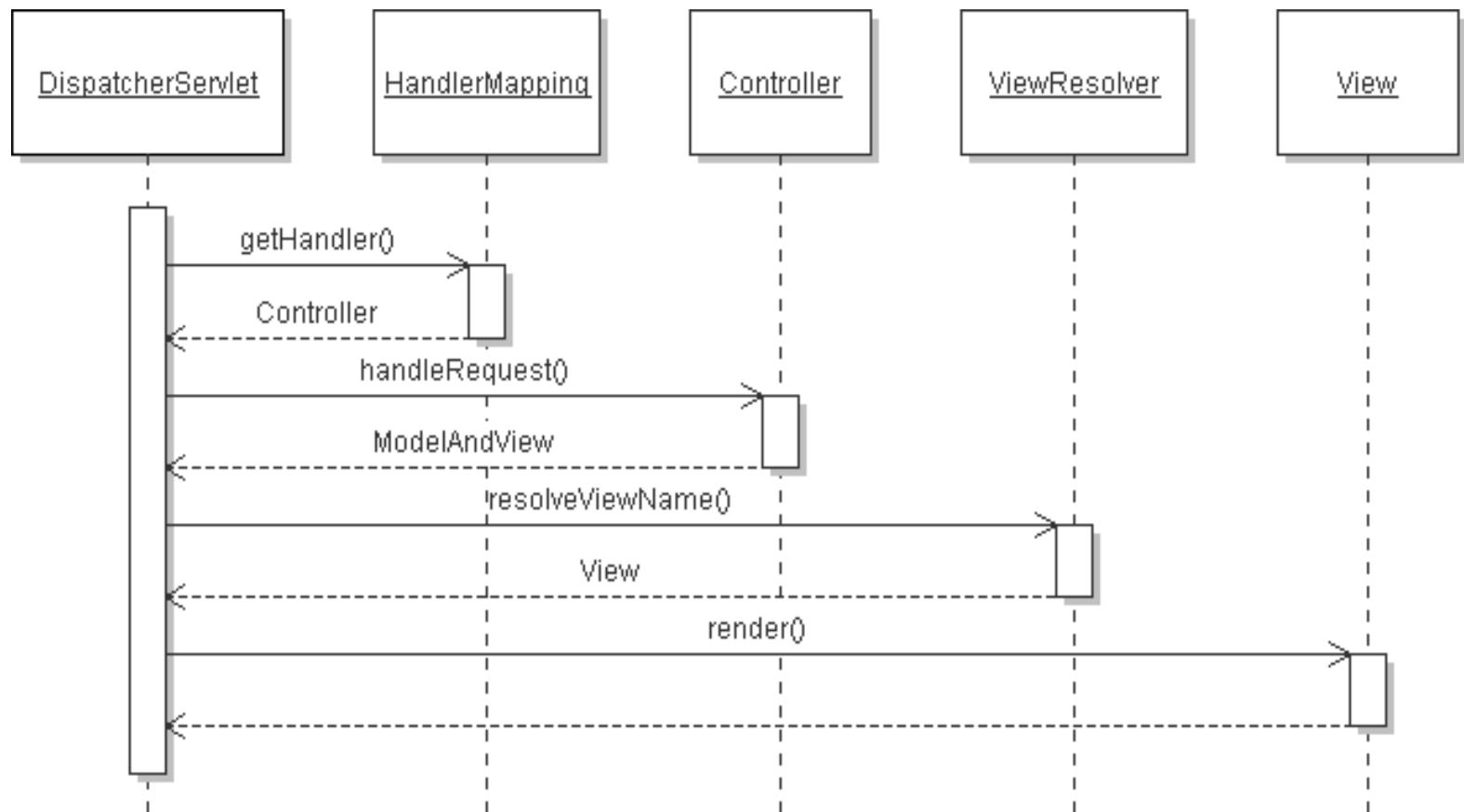
Spring MVC



Spring MVC Architecture

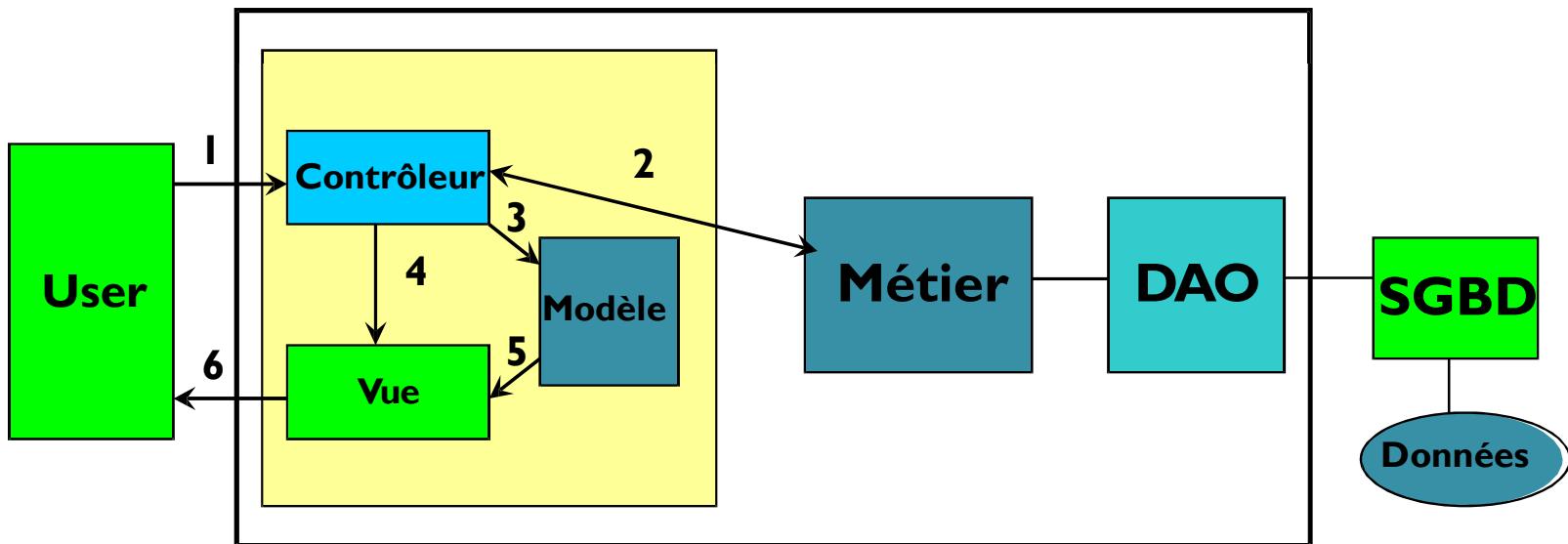


Spring MVC



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.

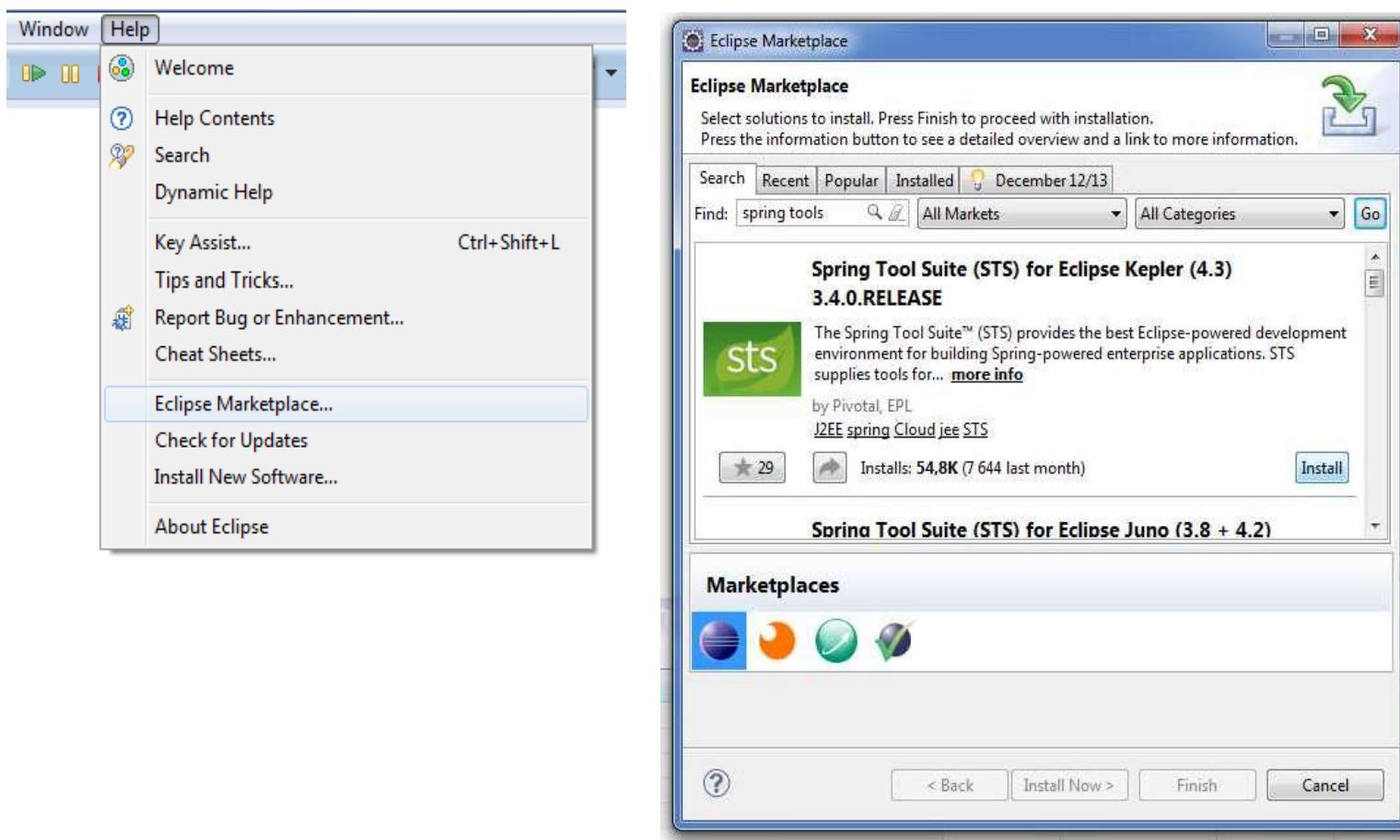
5- Le contrôleur **DispatcherServlet** demande à la vue choisie de s'afficher. Il s'agit d'une classe implémentant l'interface **org.springframework.web.servlet.View**

- Spring MVC propose différentes implémentations de cette interface pour générer des flux HTML, Excel, PDF,...

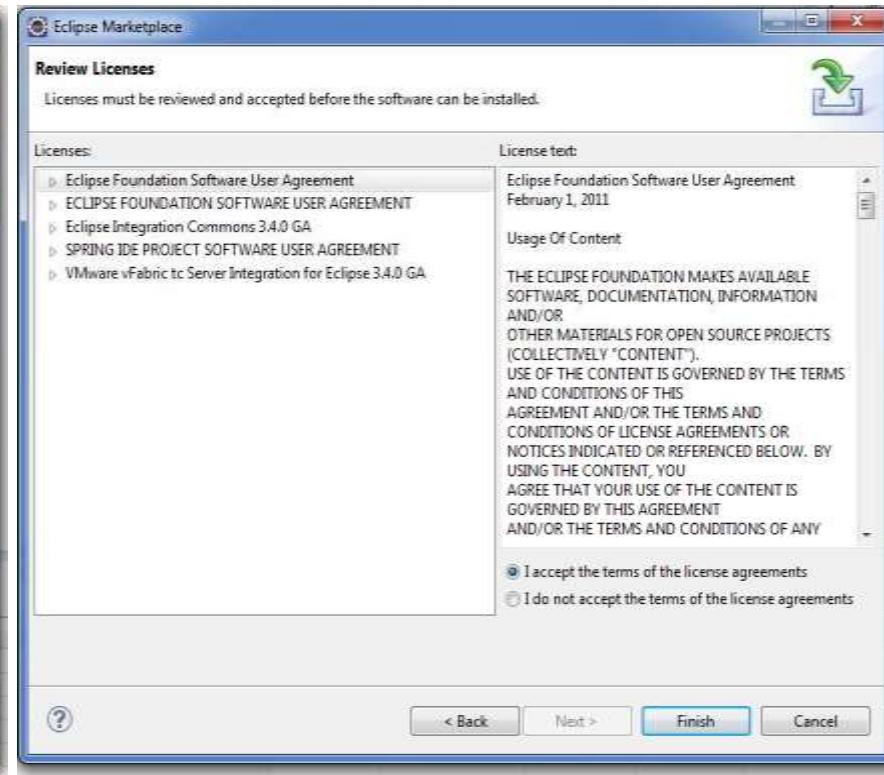
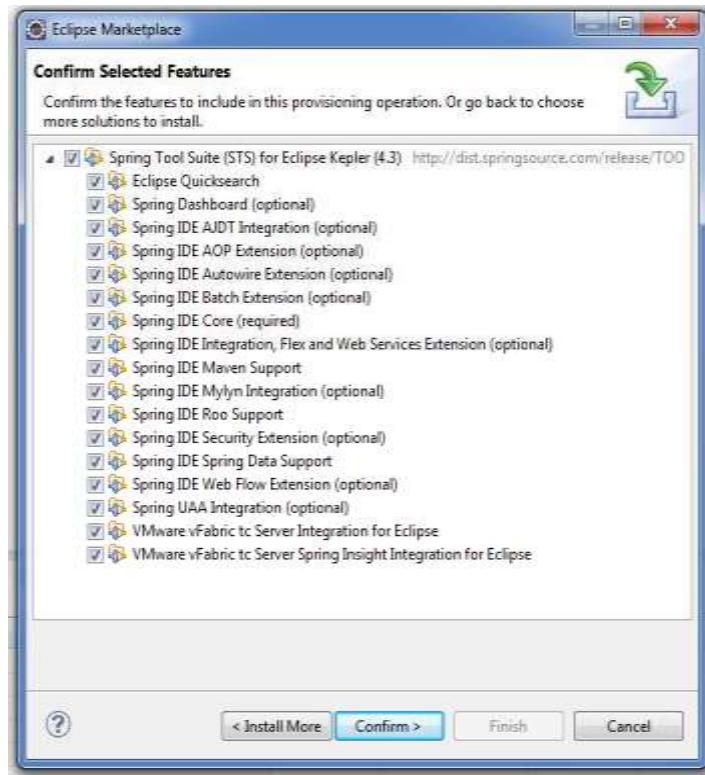
6. le générateur de vue **View** utilise le modèle **Map** préparé par le contrôleur **Controller** pour initialiser les parties dynamiques de la réponse qu'il doit envoyer au client.

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

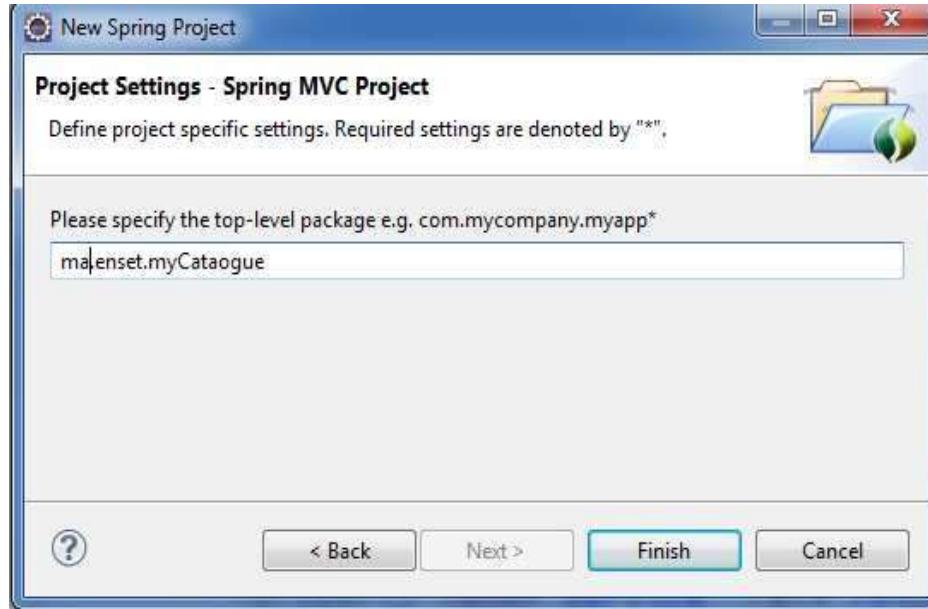
The screenshot shows two windows from the Eclipse IDE interface:

- Left Window: "Select a wizard"**
 - Header: "New" with a "Select a wizard" button.
 - Search bar: "Wizards:" containing the text "spri".
 - List: "Spring" category expanded, showing "Import Spring Getting Started Content", "Spring Bean Configuration File", "Spring Bean Definition", "Spring Project" (selected), "Spring Roo Project", "Spring Starter Project", and "Spring Web Flow Definition File".
 - Buttons at the bottom: "?", "< Back", "Next >" (highlighted in blue), "Finish", and "Cancel".
- Right Window: "New Spring Project"**
 - Header: "New Spring Project" with a "Spring Project" icon.
 - Text: "① Click 'Next' to load the template contents."
 - Form fields:
 - "Project name:" input field containing "CatalogueMVC".
 - "Use default location" checkbox checked.
 - "Location:" input field containing "C:\WS\SpringProjects\CatalogueMVC" with a "Browse..." button.
 - "Select Spring version:" dropdown set to "Default".
 - Templates:**
 - List: "Simple Spring Utility Project" and "Spring MVC Project" (selected).
 - Status: "requires downloading" and "Configure templates..." button.
 - Refresh button.
 - Description:**

A new Spring MVC web application development project
 - URL:** <http://dist.springsource.com/release/STS/help/org.springframework.templates.mvc-3.2.2.zip>
 - Working sets:**
 - Checkboxes: "Add project to working sets" and "Working sets:" dropdown with "Select..." button.
 - Buttons at the bottom: "?", "< Back", "Next >" (highlighted in blue), "Finish", and "Cancel".

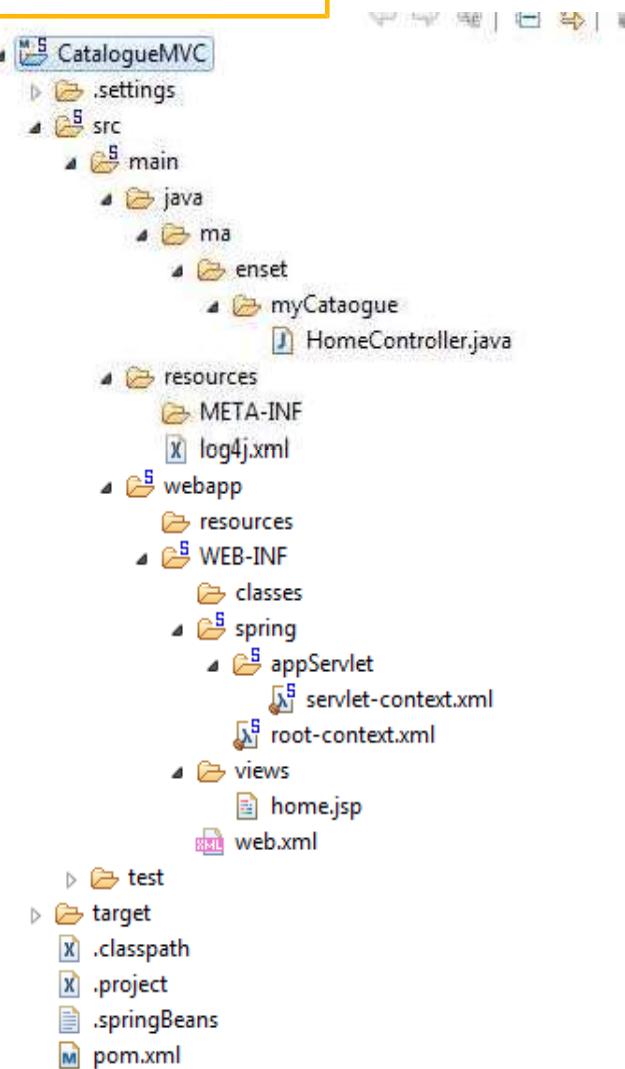
et

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</listene
      r- 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/
        schema/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 contrôleur 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/cont
    ext"
    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

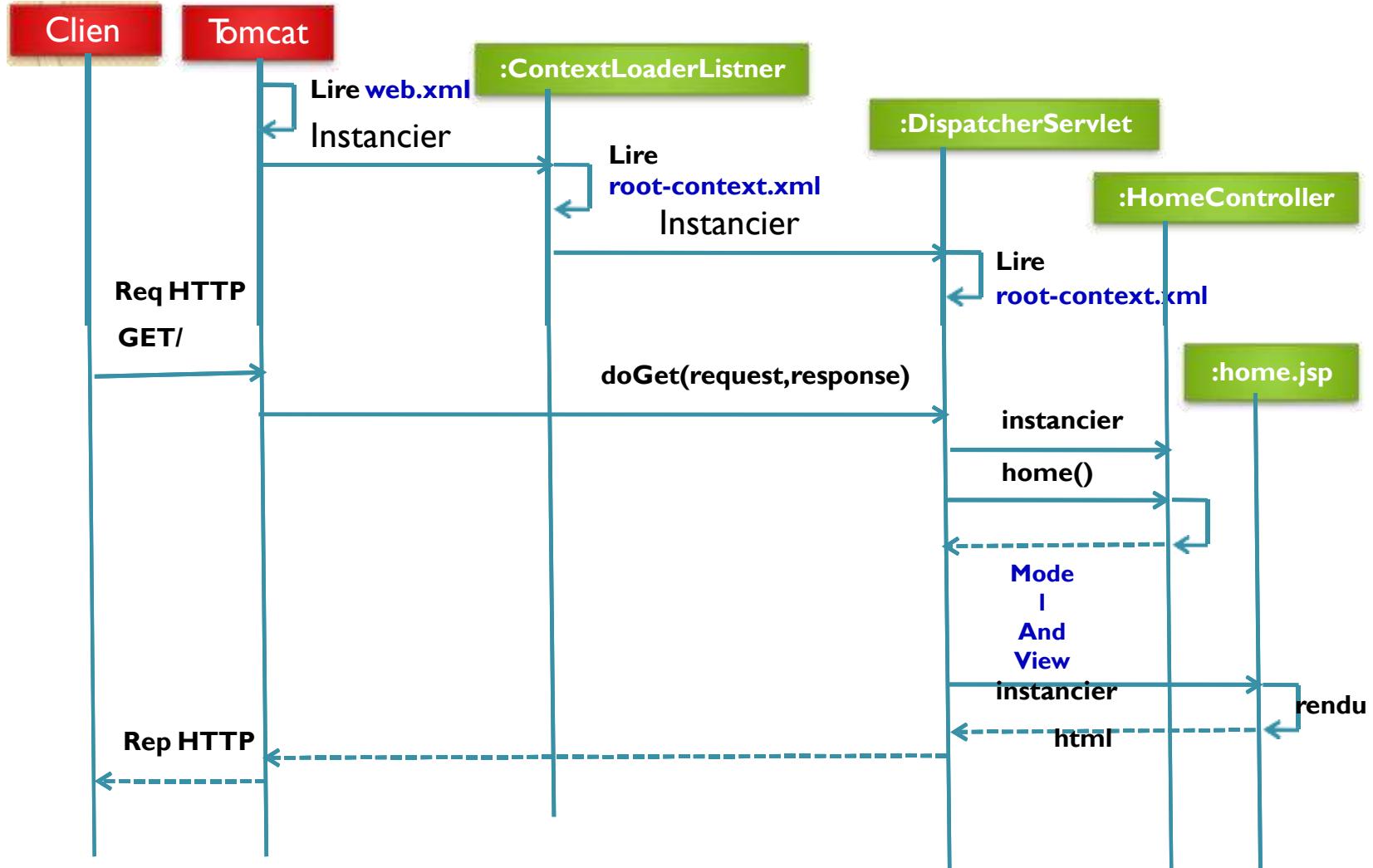
public class HomeController {
    private static final Logger Logger = LoggerFactory.getLogger(HomeController.class);
    /** Simply selects the home view to render by returning its
name. */ @RequestMapping(value = "/", method = RequestMethod.GET)
public String home(Locale locale, Model model) {
    Logger.info("Welcome home! The client Locale is {}.",
    DateFormat dateFormat =
    Locale); Date date = new Date();
    DateFormat.getDateTimeInstance(DateFormat.LONG, 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           </h1>
    <p> The time on the server is ${serverTime}.
    !
    </p>
  </body>
</html>
```



Functionnement



Application I

- 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 une implémentations de la couche DAO qui gère les produits qui sont stockés dans une liste de type HashMap.

Ecrans de l'application

Catalogue de produits

localhost:8080/sid/index

Référence:	<input type="text"/>				
Désignation:	<input type="text"/>				
Prix:	<input type="text" value="0.0"/>				
Quantité:	<input type="text" value="0"/>				
<input type="button" value="Save"/>					
REF	DESIGNATION	PRIX	QUANTITE		
AT980	Smart Phone GT	4500.0	8	Supprimer	Edit
HP675	Ordinateur HP	8000.0	5	Supprimer	Edit
aaaaaaaaaa	bbbbbbbbbbbb	900.0	67	Supprimer	Edit

Validation des données côté serveur

Catalogue de produits

localhost:8080/sid/saveProduit

Référence: ne peut pas être vide
la taille doit être entre 4 et 12

Désignation: ne peut pas être vide

Prix: doit être plus grand que 100

Quantité:

REF	DESIGNATION	PRIX	QUANTITE		
AT980	Smart Phone GT	4500.0	8	Supprimer	Edit
HP675	Ordinateur HP	8000.0	5	Supprimer	Edit
aaaaaaaaaa	bbbbbbbbbbbb	900.0	67	Supprimer	Edit

Saisie d'un produit

Catalogue de produits

localhost:8080/sid/saveProduit

Référence: AERT ne peut pas être vide
la taille doit être entre 4 et 12

Désignation: TEST ne peut pas être vide

Prix: 300 doit être plus grand que 100

Quantité: 23

Save

REF	DESIGNATION	PRIX	QUANTITE		
AT980	Smart Phone GT	4500.0	8	Supprimer	Edit
HP675	Ordinateur HP	8000.0	5	Supprimer	Edit
aaaaaaaaaa	bbbbbbbbbbbb	900.0	67	Supprimer	Edit

Après Ajout d'un produit

Catalogue de produits

localhost:8080/sid/saveProduit

Référence:	<input type="text"/>
Désignation:	<input type="text"/>
Prix:	0.0
Quantité:	0
<input type="button" value="Save"/>	

REF	DESIGNATION	PRIX	QUANTITE		
AT980	Smart Phone GT	4500.0	8	Supprimer	Edit
HP675	Ordinateur HP	8000.0	5	Supprimer	Edit
AERT	TEST	300.0	23	Supprimer	Edit
aaaaaaaa	bbbbbbbbbb	900.0	67	Supprimer	Edit

Edition d'un produit

Catalogue de produits

localhost:8080/sid/editProduit?ref=AERT

Référence:	AERT
Désignation:	TEST
Prix:	300.0
Quantité:	23
<input type="button" value="Save"/>	

REF	DESIGNATION	PRIX	QUANTITE		
AT980	Smart Phone GT	4500.0	8	Supprimer	Edit
HP675	Ordinateur HP	8000.0	5	Supprimer	Edit
AERT	TEST	300.0	23	Supprimer	Edit
aaaaaaaaaa	bbbbbbbbbbbbbb	900.0	67	Supprimer	Edit

Mise à jour d'un produit

Catalogue de produits

localhost:8080/sid/saveProduit

Référence:	<input type="text"/>
Désignation:	<input type="text"/>
Prix:	<input type="text" value="0.0"/>
Quantité:	<input type="text" value="0"/>
<input type="button" value="Save"/>	

REF	DESIGNATION	PRIX	QUANTITE		
AT980	Smart Phone GT	4500.0	8	Supprimer	Edit
HP675	Ordinateur HP	8000.0	5	Supprimer	Edit
AERT	TEST	2450.0	23	Supprimer	Edit
aaaaaaaaaa	bbbbbbbbbbbbb	900.0	67	Supprimer	Edit

Suppression d'un

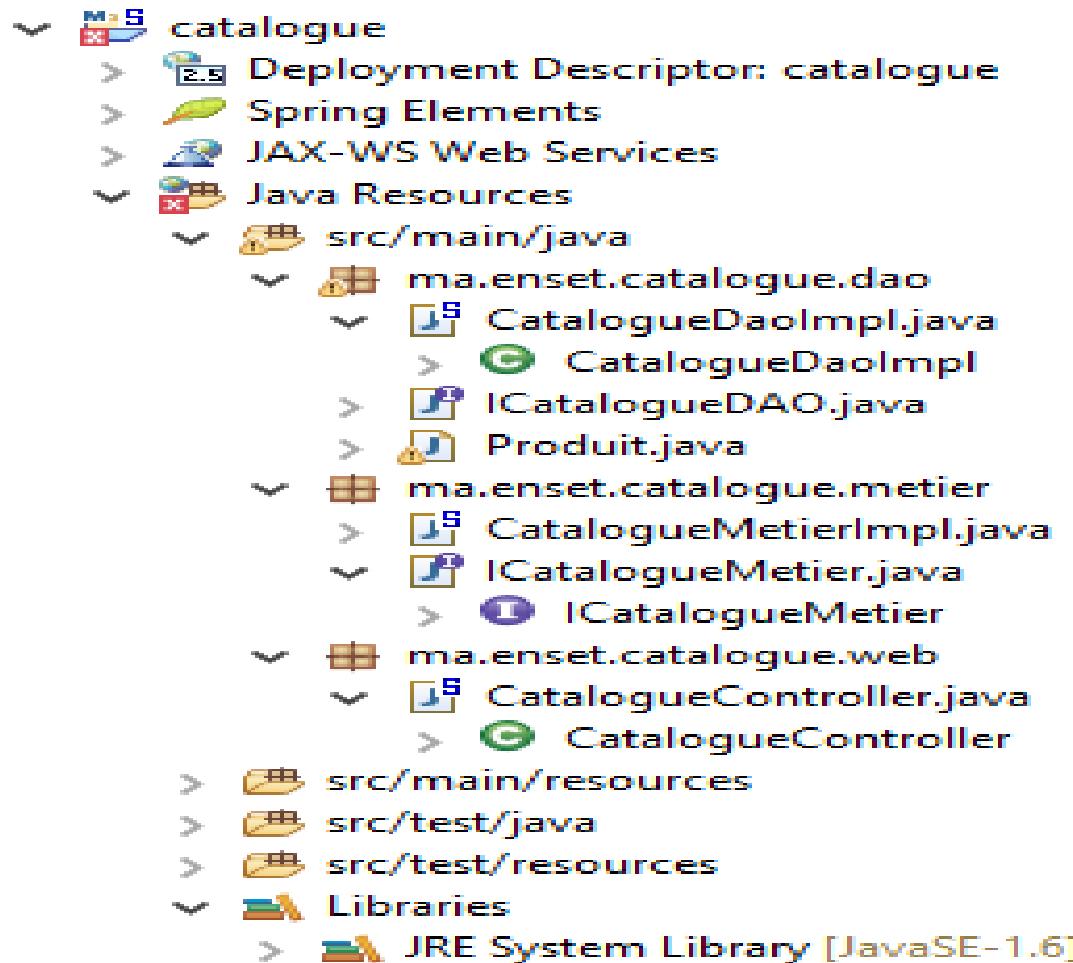
Catalogue de produits

localhost:8080/sid/deleteProduit?ref=aaaaaaaa

Référence:	<input type="text"/>
Désignation:	<input type="text"/>
Prix:	0.0
Quantité:	0
<input type="button" value="Save"/>	

REF	DESIGNATION	PRIX	QUANTITE		
AT980	Smart Phone GT	4500.0	8	Supprimer	Edit
HP675	Ordinateur HP	8000.0	5	Supprimer	Edit
AERT	TEST	2450.0	23	Supprimer	Edit

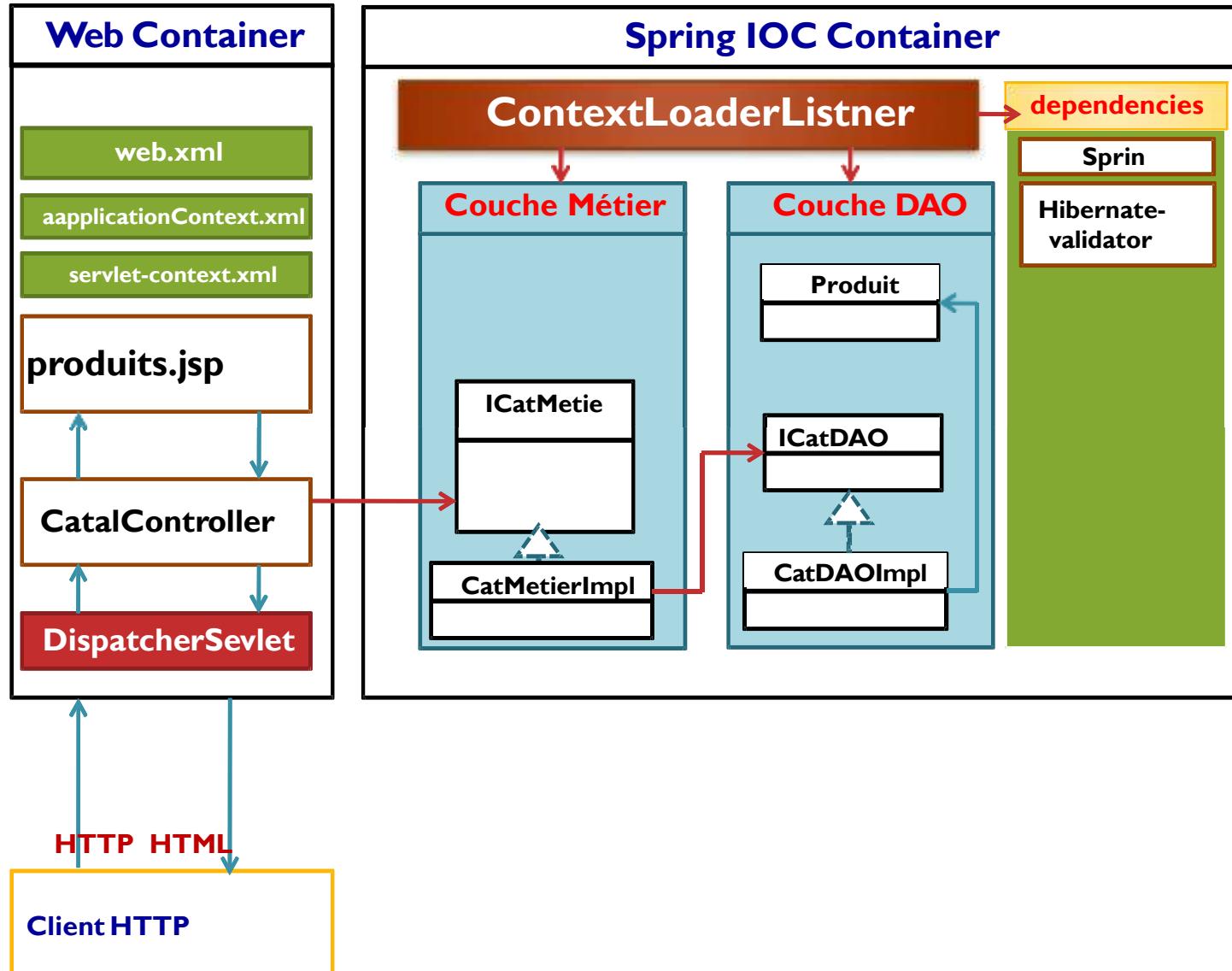
Structure du projet



Maven Dependencies

- ▶ Maven Dependencies
 - ▶ spring-context-3.1.1.RELEASE.jar - C:\Users\youssf\l.m2\repository\org\springframework\spring-context\3.1.1.RELEASE
 - ▶ spring-aop-3.1.1.RELEASE.jar - C:\Users\youssf\l.m2\repository\org\springframework\spring-aop\3.1.1.RELEASE
 - ▶ aopalliance-1.0.jar - C:\Users\youssf\l.m2\repository\org\springframework\spring-aop\3.1.1.RELEASE
 - ▶ spring-beans-3.1.1.RELEASE.jar - C:\Users\youssf\l.m2\repository\org\springframework\spring-beans\3.1.1.RELEASE
 - ▶ spring-core-3.1.1.RELEASE.jar - C:\Users\youssf\l.m2\repository\org\springframework\spring-core\3.1.1.RELEASE
 - ▶ spring-expression-3.1.1.RELEASE.jar - C:\Users\youssf\l.m2\repository\org\springframework\spring-expression\3.1.1.RELEASE
 - ▶ spring-asm-3.1.1.RELEASE.jar - C:\Users\youssf\l.m2\repository\org\springframework\spring-asm\3.1.1.RELEASE
 - ▶ spring-webmvc-3.1.1.RELEASE.jar - C:\Users\youssf\l.m2\repository\org\springframework\spring-webmvc\3.1.1.RELEASE
 - ▶ spring-context-support-3.1.1.RELEASE.jar - C:\Users\youssf\l.m2\repository\org\springframework\spring-context-support\3.1.1.RELEASE
 - ▶ spring-web-3.1.1.RELEASE.jar - C:\Users\youssf\l.m2\repository\org\springframework\spring-web\3.1.1.RELEASE
 - ▶ aspectjrt-1.6.10.jar - C:\Users\youssf\l.m2\repository\org\aspectj\aspectjrt\1.6.10
 - ▶ slf4j-api-1.6.6.jar - C:\Users\youssf\l.m2\repository\org\slf4j\slf4j-api\1.6.6
 - ▶ jcl-over-slf4j-1.6.6.jar - C:\Users\youssf\l.m2\repository\org\slf4j\jcl-over-slf4j\1.6.6
 - ▶ slf4j-log4j12-1.6.6.jar - C:\Users\youssf\l.m2\repository\org\slf4j\slf4j-log4j12\1.6.6
 - ▶ log4j-1.2.15.jar - C:\Users\youssf\l.m2\repository\log4j\log4j\1.2.15
 - ▶ javax.inject-1.jar - C:\Users\youssf\l.m2\repository\javax\inject\javax.inject\1
 - ▶ servlet-api-2.5.jar - C:\Users\youssf\l.m2\repository\javax\servlet\servlet-api\2.5
 - ▶ jsp-api-2.1.jar - C:\Users\youssf\l.m2\repository\javax\servlet\jsp\jsp-api\2.1
 - ▶ jstl-1.2.jar - C:\Users\youssf\l.m2\repository\javax\servlet\jstl\1.2
 - ▶ junit-4.7.jar - C:\Users\youssf\l.m2\repository\junit\junit\4.7
 - ▶ hibernate-validator-5.0.0.Final.jar - C:\Users\youssf\l.m2\repository\org\hibernate\hibernate-validator\5.0.0.Final
 - ▶ validation-api-1.1.0.Final.jar - C:\Users\youssf\l.m2\repository\javax\validation\validation-api\1.1.0.Final
 - ▶ jboss-logging-3.1.1.GA.jar - C:\Users\youssf\l.m2\repository\org\jboss\logging\jboss-logging\3.1.1.GA
 - ▶ classmate-0.8.0.jar - C:\Users\youssf\l.m2\repository\com\fasterxml\classmate\0.8.0
 - ▶ javax.el-2.2.4.jar - C:\Users\youssf\l.m2\repository\org\glassfish\web\javax.el\2.2.4
 - ▶ javax.el-api-2.2.4.jar - C:\Users\youssf\l.m2\repository\javax\el\javax.el-api\2.2.4

Architecture technique



Maven dependencies : Spring

pom.xml

```
<properties>
    <java-version>1.8</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

Entité Produit

```
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

```
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

```
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

Interface IcatalogueMetier

```
import java.util.List;
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

```
import java.util.List;
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);
}

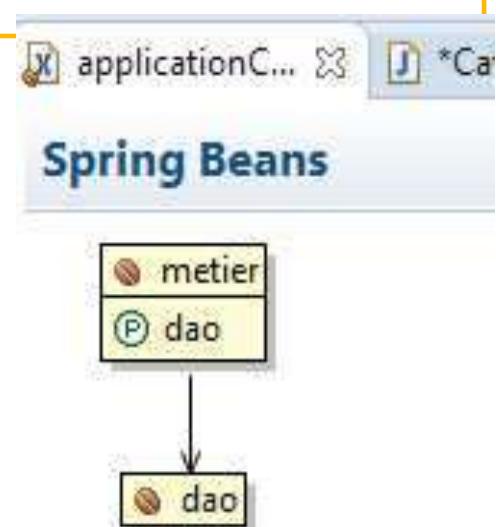
@Override
public void deleteProduit(String refernce) {
    dao.deleteProduit(refernce);
}

@Override
public void updateProduit(Produit p) {
    dao.updateProduit(p);
}
```

INJECTION DES DEPENDANCES

Root-context.xml

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

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/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>
```

Catalogue Controller

```
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";
}
```

Catalogue Controller

```
@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>
            </table>
        </f:form>
    </div>
</body>
```

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>
```

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 II

- On souhaite 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
 - Dans cette deuxième implémentation, nous allons supposer que les produits sont stockés dans une base de données de type MySQL.

Maven dependencies : Spring

pom.xml

```
<properties>
    <java-version>1.8</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

```
<!-- Hibernate 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>
```

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 Security

```
<!-- 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>
```

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

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);  
}  
}
```

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

/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://localhost: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"/>
</beans>
```

Couche Web :Contrôleur

```
import java.util.List;
import javax.validation.Valid;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.validation.BindingResult;
import org.springframework.web.bind.annotation.*;

@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

http://localhost:8080/catalogue/index

ID Produit:

Nom Produit:

Prix: 0.0

Save

ID	Nom	Prix		
12	aaaaaa	0.0	Supprimer	Edit
13	aaaaaa	0.0	Supprimer	Edit
20	pppppp	0.0	Supprimer	Edit
25	AZM	0.0	Supprimer	Edit

http://localhost:8080/catalogue/saveProduit

ID Produit:

Nom Produit: ne peut pas être vide
la taille doit être entre 3 et 60

Prix: 0.0

Save

ID	Nom	Prix
----	-----	------

http://localhost:8080/catalogue/saveProduit

ID Produit:

Nom Produit:

Prix: 0.0

Save

ID	Nom	Prix		
12	aaaaaa	0.0	Supprimer	Edit
13	aaaaaa	0.0	Supprimer	Edit
20	pppppp	0.0	Supprimer	Edit
25	AZM	0.0	Supprimer	Edit
26	HP980	9800.0	Supprimer	Edit

http://localhost:8080/catalogue/editProduit?id=26

ID Produit: 26

Nom Produit: HP980

Prix: 9800.0

Save

ID	Nom	Prix		
12	aaaaaa	0.0	Supprimer	Edit
13	aaaaaa	0.0	Supprimer	Edit
20	pppppp	0.0	Supprimer	Edit
25	AZM	0.0	Supprimer	Edit
26	HP980	9800.0	Supprimer	Edit

Vue :produits.jsp

http://localhost:8080/catalogue/saveProduit

ID Produit:

Nom Produit:

Prix:

Save

ID	Nom	Prix		
12	aaaaaa	0.0	Supprimer	Edit
13	aaaaaa	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:

Save

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

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>
        </table>
    </f:form>
</body>
</html>
```

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

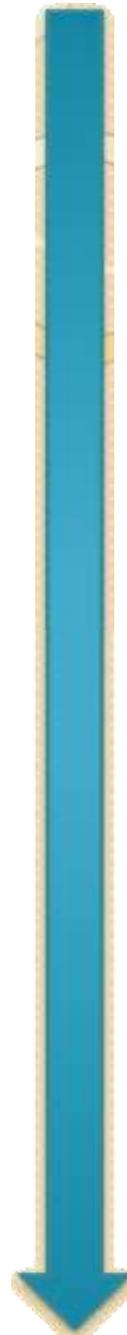
```
<table border="1" width="80%>
  <tr>
    <th>ID</th><th>Nom</th><th>PrixD</th>
  </tr>
  <c:forEach items="${produits}" var="p">
    <tr>
      <td>${p.idProduit }</td>
      <td>${p.nomProduit }</td>
      <td>${p.prix }</td>
      <td><a href="deleteProduit?id=${p.idProduit }">Supprimer</a></td>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

Architectures Logicielles Distribuées basées sur les Micro-Services

Plan

- Evolution des approches de programmation
- Industrialisation du génie logiciel
- Exigences des systèmes logiciels
- Principe de l'inversion de contrôle
- Exemples d'architectures logicielles modernes
- Problème des applications monolithiques
- Architectures logicielles basées sur les Micro-services
- Mise en œuvre des architectures basées sur les Micro-services : Spring Boot

Evolution des techniques de programmation



Langage Machine

Assembleur

Chaque famille de CPU possède son porophore jeu d'instructions

Langage bas niveau

Programmation Procédurale

Sous programmes: Procédures, fonctions
(Basic, Pascal, C, Fortran,...)

Programmation Orientée
Objet

Objet = Etat + Comportement + Identité
Concepts fondamentaux : Objet, classe, Héritage,
polymorphisme, encapsulation (C++, JAVA, C#, ...)

Programmation Orientée
Objet Distribués

Objets distribués sur plusieurs machines
Middlewares : (RMI, CORBA, JMS, ...)

Programmation Orientée
composants

Objets distribués, réutilisables, configurables,
Interchangeables, évolutifs, mobiles, surveillable à
chaud : Conteneur (EJB, Spring) : AOP

Programmation Orientée
Services

Composant disponibles à d'autres applications
distantes hétérogènes via des protocoles (http)
transportant des données: XML, JSON => SOAP
et REST

Programmation Orientée
Agents

? Service + Intelligence + Apprentissage+ ...

Industrialisation du génie logiciel

- Le processus du développement logiciel est, aujourd'hui complètement industrialisé.
- Un logiciel est construit à base de composants
 - Réutilisables
 - Interchangeable
 - Évolutifs
 - Reconfigurables
 - Mobiles
 - Surveillables à chaud
 - ...

Avènement des technologies Open Source

- En dix ans le développement logiciel a évolué en grande partie grâce aux technologies Open Sources.
- Celles ci permettent aux développeurs
 - de ne pas réinventer la roue
 - et de se concentrer plus sur les aspects métiers des applications
 - et utiliser les Framework pour résoudre les aspects techniques.
- Exemples :
 - Spring, Struts, Hibernate (Coté Serveur)
 - AngularJS, BootStrap (Coté Client)
 - RMI, CORBA, JMS, JAXWS, JAXRS (Systèmes Distribués)

Avènement des méthodologies Agile

- Les méthodes Agile de gestion projet et de développement comme Scrum ou l'eXtreme Programming (XP) partent du constat que le cycle de développement en cascade est un échec.
- Développement en cascade:
 - spécifier pendant X mois,
 - puis ensuite coder Y mois,
 - tester Z mois,
 - pour livrer au client un projet qui ne correspond plus tout à fait à ses attentes.

Avènement des méthodologies Agile

- Les méthodes Agile prônent
 - Les tests avant le développement (Test Driven Development : TDD)
 - Des cycles de développement itératifs,
 - L'implication du client final tout au long du projet,
 - Des spécifications réduites en début de projet,
 - etc.
- Les méthodologies Agile sont pragmatiques,
- On fait leurs preuves et sont prisées par de grands industriels de logiciels.

Industrialiser le cycle de vie : améliorations, maintenance, corrections

- Pour industrialiser les composants logiciels, il est nécessaire d'utiliser des outils qui permettent d'automatiser le processus de fabrication des logiciels
 - **Frameworks de tests** : JUnit
 - Automatiser les Tests Unitaires
 - **Outils d'intégration continue** : Maven
 - Ils jouent le rôle de chef d'orchestre pour piloter et automatiser le processus de développement logiciel :
 - Gérer les dépendances
 - Lancer la compilation des sources
 - Lancer les Test Unitaires
 - Générer les packages (jar, war, ear)
 - Installer les packages dans le repository
 - Déployer l'application dans le serveur
 - Générer la documentation du projet
 - ...

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:
 - Montée en charge
 - Haute disponibilité et tolérance aux pannes
 - La maintenance:
 - 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...)
 -
- **Exigences financières :Coût du logiciel**

Comment faire surmonter toutes contraintes

- Il est très difficile de développer un système logiciel qui respecte ces exigences sans **utiliser l'expérience des autres** :
 - Bâtir l'application sur une architecture d'entreprise: (JEE,. Net)
 - **Framework pour l'Inversion de contrôle:**
 - Permettre au développeur de se concentrer sur le code métier (**Exigences fonctionnelles**)
 - Le Framework s'occupe du code Technique (**Exigence Technique**)
 - Grâce à la **Programmation Orientée Aspect (AOP)**
 - Exemple :Spring,EJB pour l'architecture JEE
 - Utiliser des modèles de conceptions confirmés :Design Patterns
 - **Frameworks :**
 - Mapping objet relationnel (ORM) :JPA, Hibernate,Toplink,...
 - Applications Web coté serveur :Struts,JSF,SpringMVC
 - Applications web coté client :AngularJS, BootStrap
 - ...
 - **Middlewares pour les applications Distribuées :**
 - RMI,CORBA :Applications distribuées
 - JAX WS pour Web services basés SOAP (HTTP+XML)
 - JAXRS pour les Web services RESTful (HTTP+(JSON/XML,...))
 - JMS :Communication asynchrone entre les application
 - ...

Exemple : sans utiliser d'inversion de contrôle

```
public void virement(int c1, int c2, double mt) {  
    /* Création d'une transaction */  
    EntityTransaction transaction=entityManager.getTransaction();  
    /* Démarrer la transaction */  
    transaction.begin();  
    try {
```

Code Technique

```
    /* Code métier */  
    retirer(c1,mt);  
    verser(c2,mt);
```

Code Métier

```
    /* valider la transaction */  
    transaction.commit();  
} catch (Exception e) {  
    /* Annuler la transaction en cas d'exception */  
    transaction.rollback();  
    e.printStackTrace();  
}
```

Code Technique

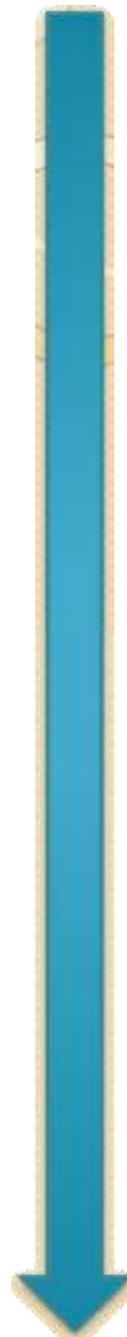
Exemple : en utilisant l'inversion de contrôle

```
@Transactional  
public void virement(int c1, int c2, double mt) {  
    retirer(c1,mt);  
    verser(c2,mt);  
}
```

Code Métier

Ici, avec l'annotation **@Transactional**, nous avons délégué la gestion des transaction au conteneur Spring IOC

Evolution des techniques de programmation



Langage Machine

Assembleur

Chaque famille de CPU possède son porophore jeu d'instructions

Langage bas niveau

Programmation Procédurale

Sous programmes: Procédures, fonctions
(Basic, Pascal, C, Fortran,..)

Programmation Orientée
Objet

Objet = Etat + Comportement + Identité
Concepts fondamentaux : Objet, classe, Héritage,
polymorphisme, encapsulation (C++, JAVA, C#, ..)

Programmation Orientée
Objet Distribués

Objets distribués sur plusieurs machines
Middlewares : (RMI, CORBA, JMS, ...)

Programmation Orientée
composants

Objets distribués, réutilisables, configurables,
Interchangeables, évolutifs, mobiles, surveillable à
chaud : Conteneur (EJB, Spring) : AOP

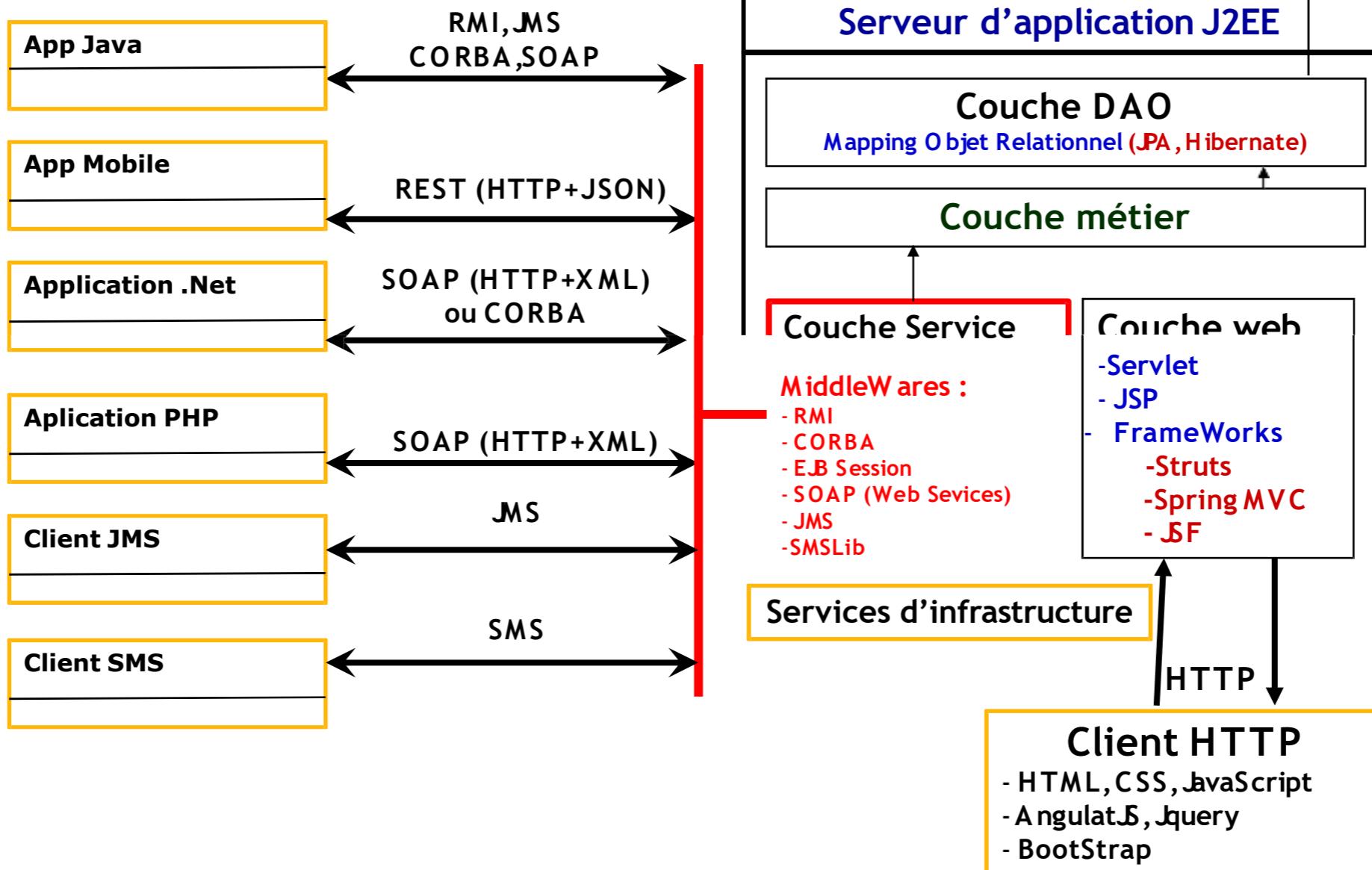
Programmation Orientée
Services

Composant disponibles à d'autres applications
distantes hétérogènes via des protocoles (http)
transportant des données: XML, JSON => SOAP
et REST

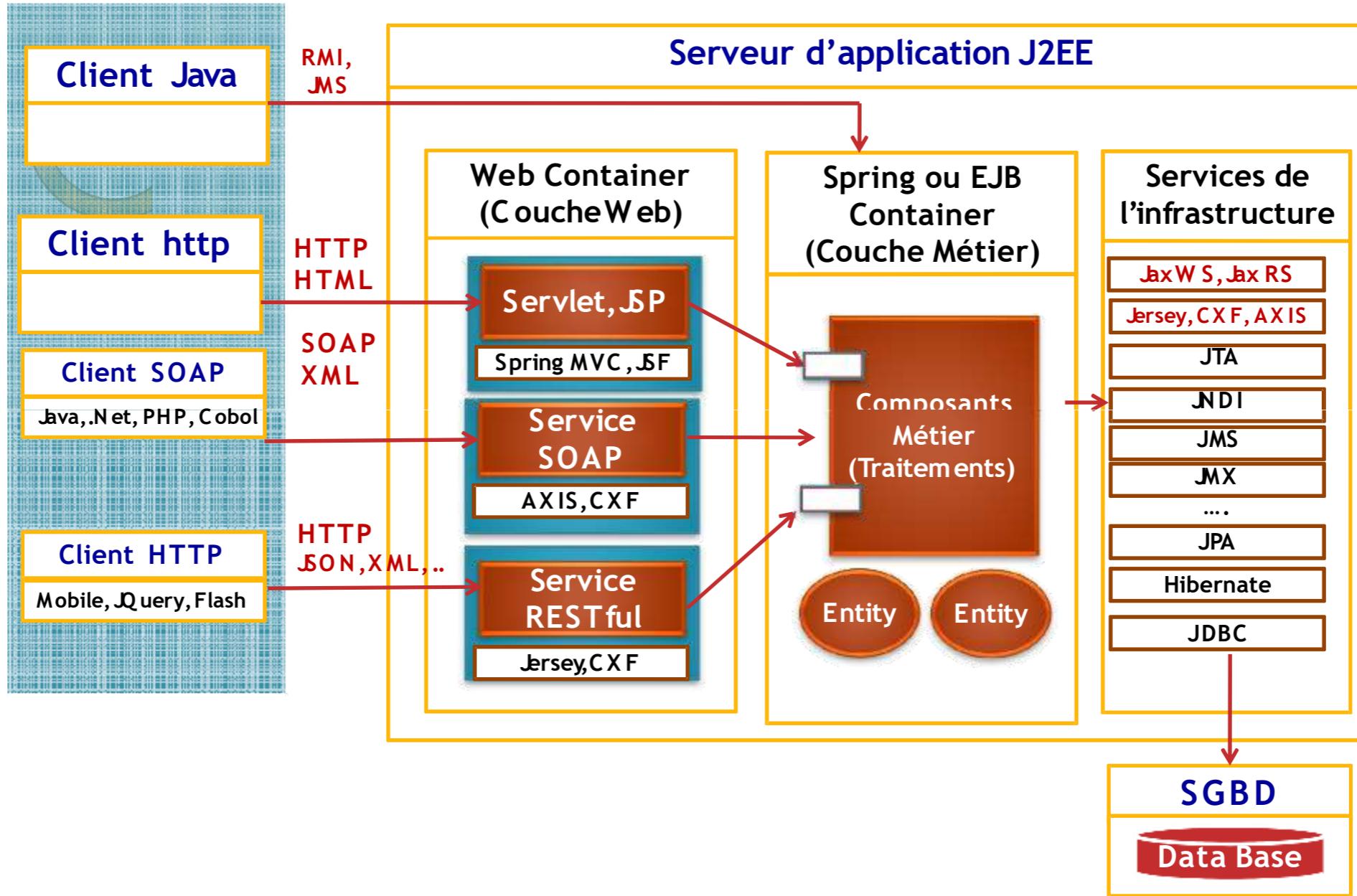
Programmation Orientée
Agents

? Service + Intelligence + Apprentissage+ ...

Architectures Distribués

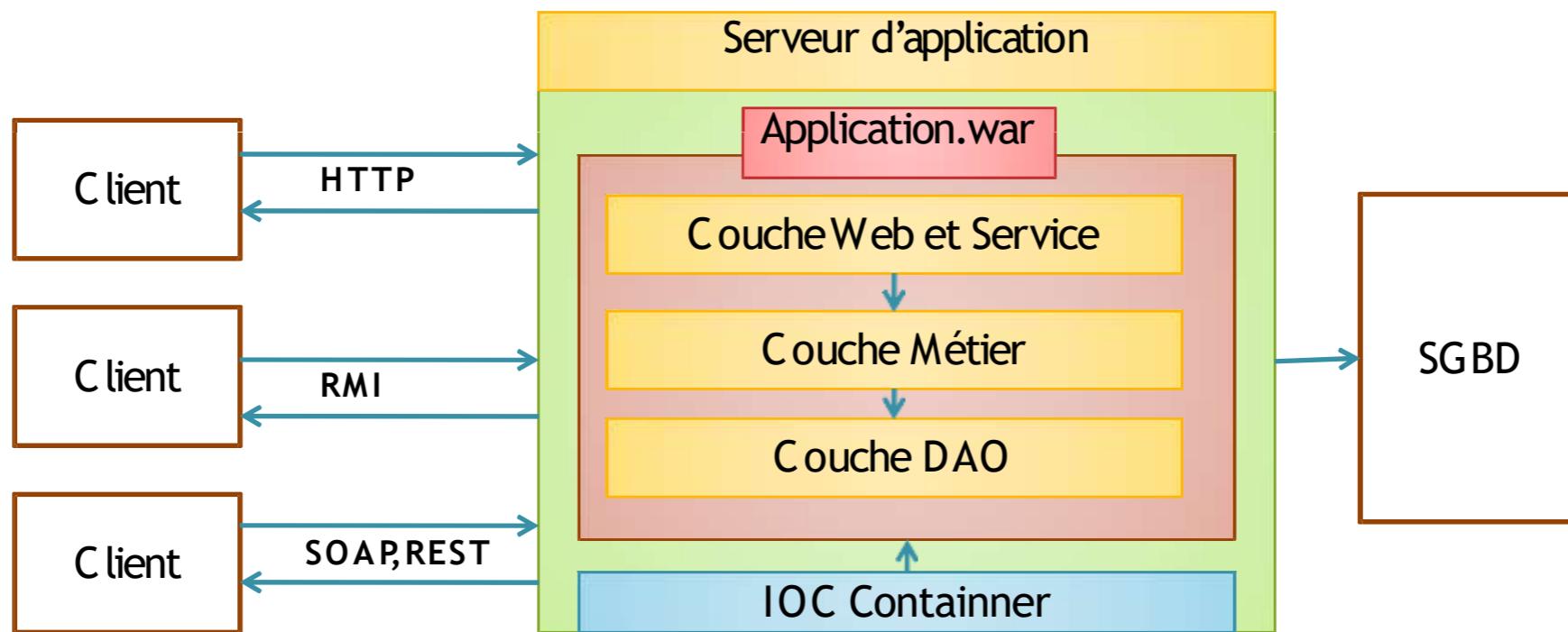


Architecture J2EE



De l'application monolithique aux architectures micro services

- Une application monolithique est une application qui est développée en un seul bloc (war, jar, Ear) et déployée d'une manière unitaire dans un serveur d'application



Problèmes des applications monolithiques

- Les principaux problème des applications monolithiques sont :
 - Elles centralisent tous les besoins fonctionnels
 - Elles sont réalisées dans une seule technologie.
 - Chaque modification nécessite de :
 - Tester les régressions
 - Redéployer toute l'application
 - Difficile à faire évoluer au niveau fonctionnel
 - Livraison en bloc (Le client attend beaucoup de temps pour commencer à voir les premières versions)

Les Micro services

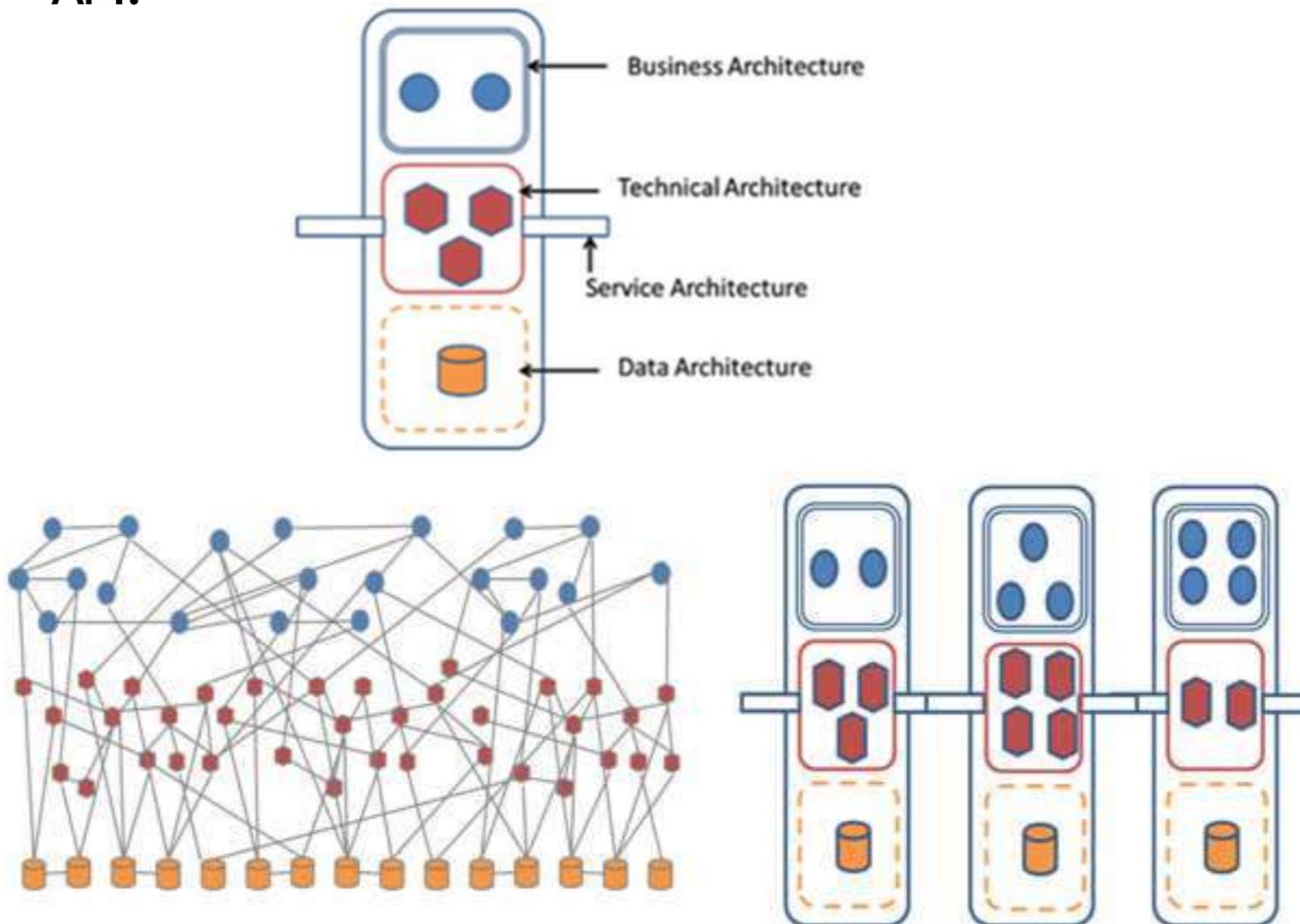
- Les micro services sont une approche d'architecture et de développement d'une **application composées de petits services.**
- L'idée étant de découper un grand problème en petites unités implémentée sous forme de micro-services
- Chaque service est **responsable d'une fonctionnalité,**
- Chaque micro-service est **développé, testé et déployé séparément** des autres.
- Chaque service **tourne dans un processus séparé.**
- La seule relation entre les différents micro services est l'échange de données effectué à travers les différentes APIs qu'ils exposent. (**SOAP, REST, RMI, CORBA, JMS,...**)
- Lorsqu'on **les combinent**, ces micro services peuvent **réaliser des opérations très complexes.**

Aspects clef des micro services

- Chaque micro service peut être conçu à l'aide de n'importe quel **outil** et développé avec n'importe quel **langage et technologie**.
- Ils sont **faiblement couplés** puisque chaque micro service est physiquement séparé des autres,
- **Indépendance relative entre les différentes équipes** qui développent les différents micro services (en partant du principe que les APIs qu'ils exposent sont définis à l'avance).
- **Facilité des tests** et du **déploiement** ou de la livraison continue.

Micro service

- Un micro service combine les trois couches « **métier**, **technique** et **données** » en une seule, accessible via des API.





- MISE EN ŒUVRE DES ARCHITECTURES BASÉES SUR LES MICRO-SERVICES

Spring Boot

- Spring Boot est un Framework qui permet de créer des applications basées sur des micro services.
- Atouts de Spring Boot :
 - Faciliter le développement d'applications complexes.
 - Faciliter à l'extrême l'injection des dépendances
 - Réduire à l'extrême les fichier de configurations
 - Faciliter la gestion des dépendances Maven.
 - Auto Configuration : la plupart des beans sont créés si le ou les jar(s) adéquats sont dans le classpath.
 - Fournir un conteneur de servlet embarqué (Tomcat, Jetty)
 - Créer une application autonome (jar ou war)

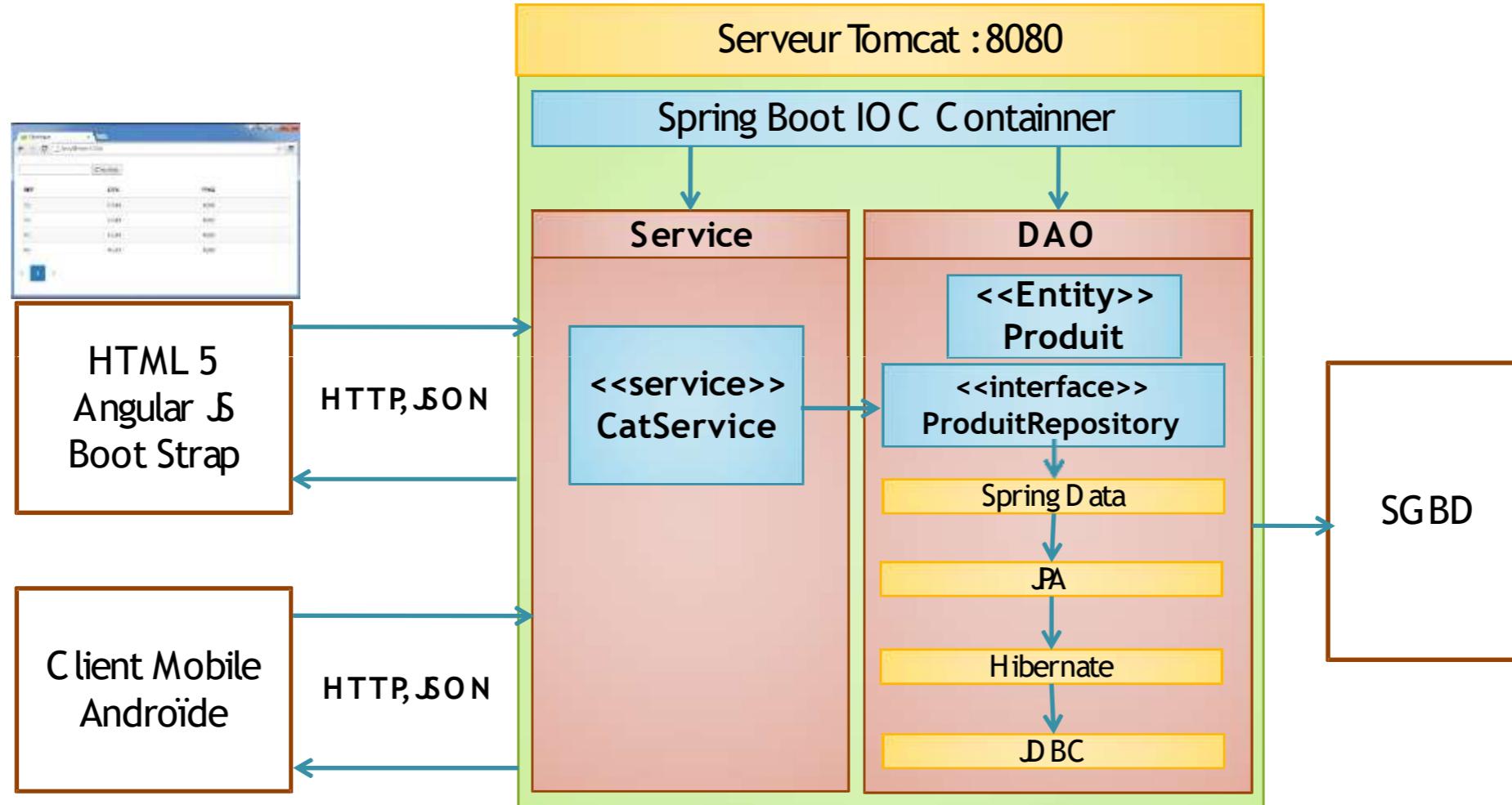


- EXEMPLE
D'APPLICATION
SPRING BOOT

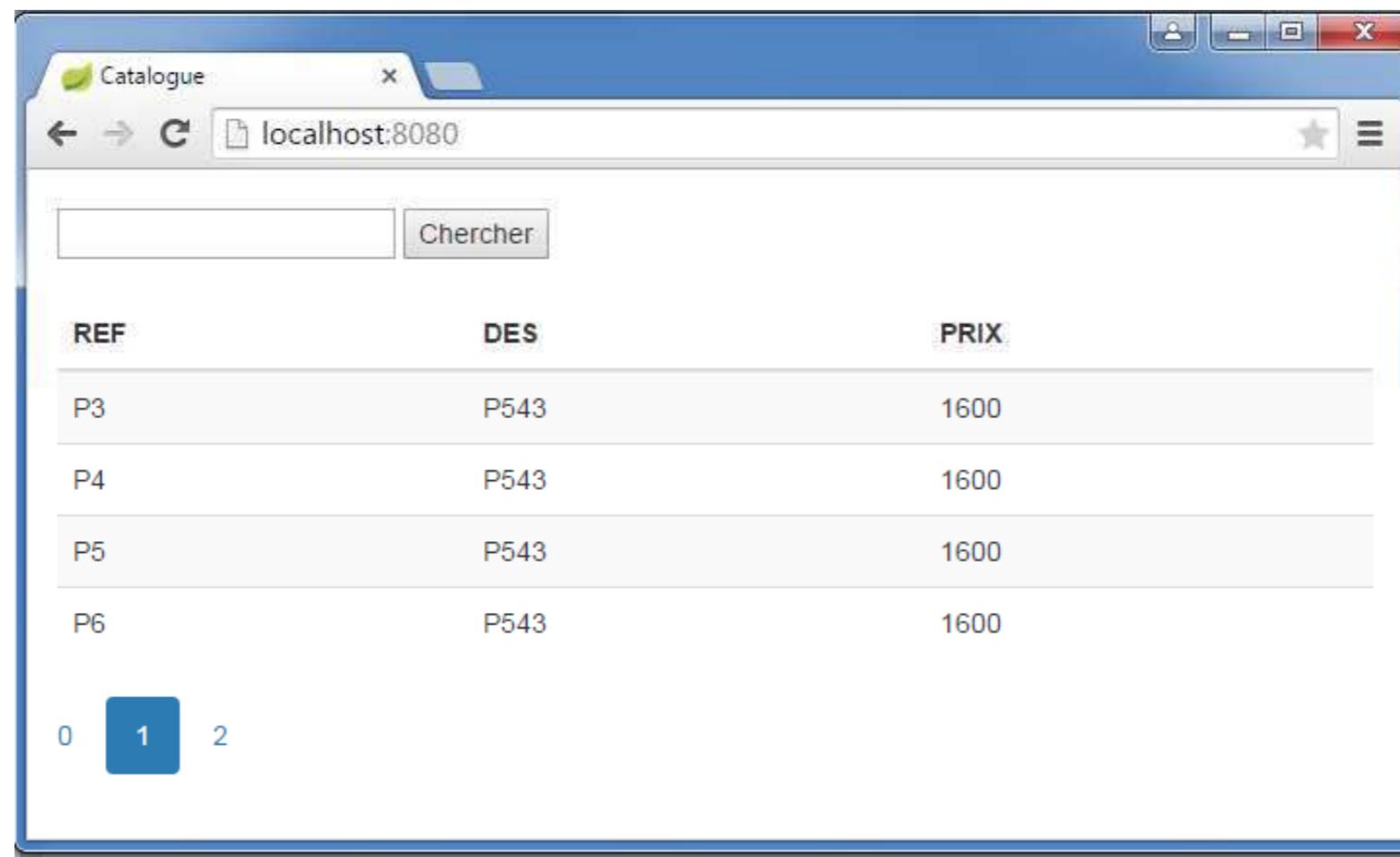
Premier Exemple d'application

- On souhaite créer une application qui permet de gérer des produits.
- Chaque produit est défini par :
 - Sa référence de type String
 - Sa désignation de type String
 - Son prix
- L'application permet de :
 - Ajouter de nouveaux produits
 - Consulter les produits
 - Chercher les produits par mot clé
 - Consulter un produit
 - Mettre à jour un produit
 - Supprimer un produit
- Les données sont stockées dans une base de données MySQL
- L'application est un service Restful basée sur Spring Boot
- La couche web respecte MVC côté Client et basée sur :
 - HTML5
 - CSS, Bootstrap
 - Angular JS

Architecture



Exemple de Vue à implémenter

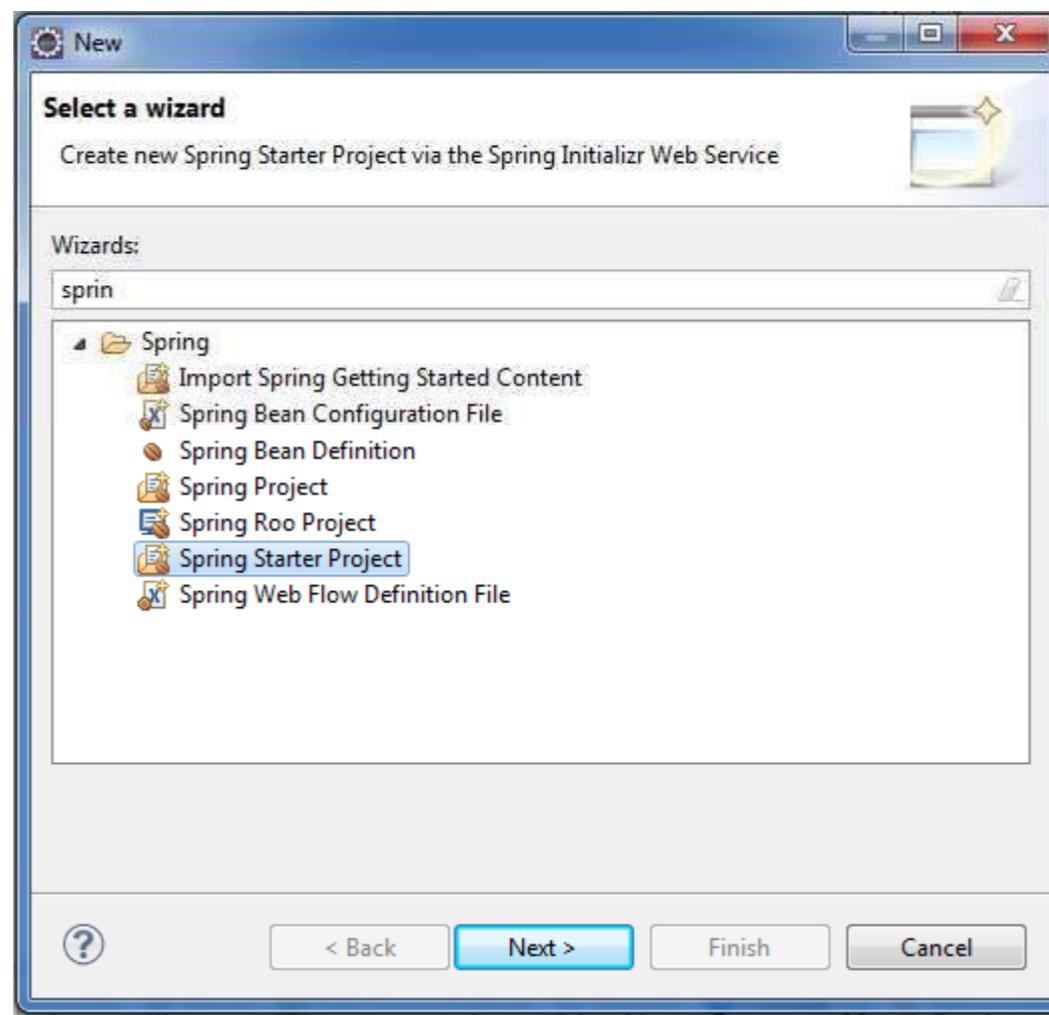


The screenshot shows a web browser window titled "Catalogue" with the URL "localhost:8080". The page displays a table of products with columns: REF, DES, and PRIX. There are four rows of data, each containing the same values: P3, P543, 1600. Below the table is a navigation bar with buttons labeled 0, 1, and 2, where button 1 is highlighted.

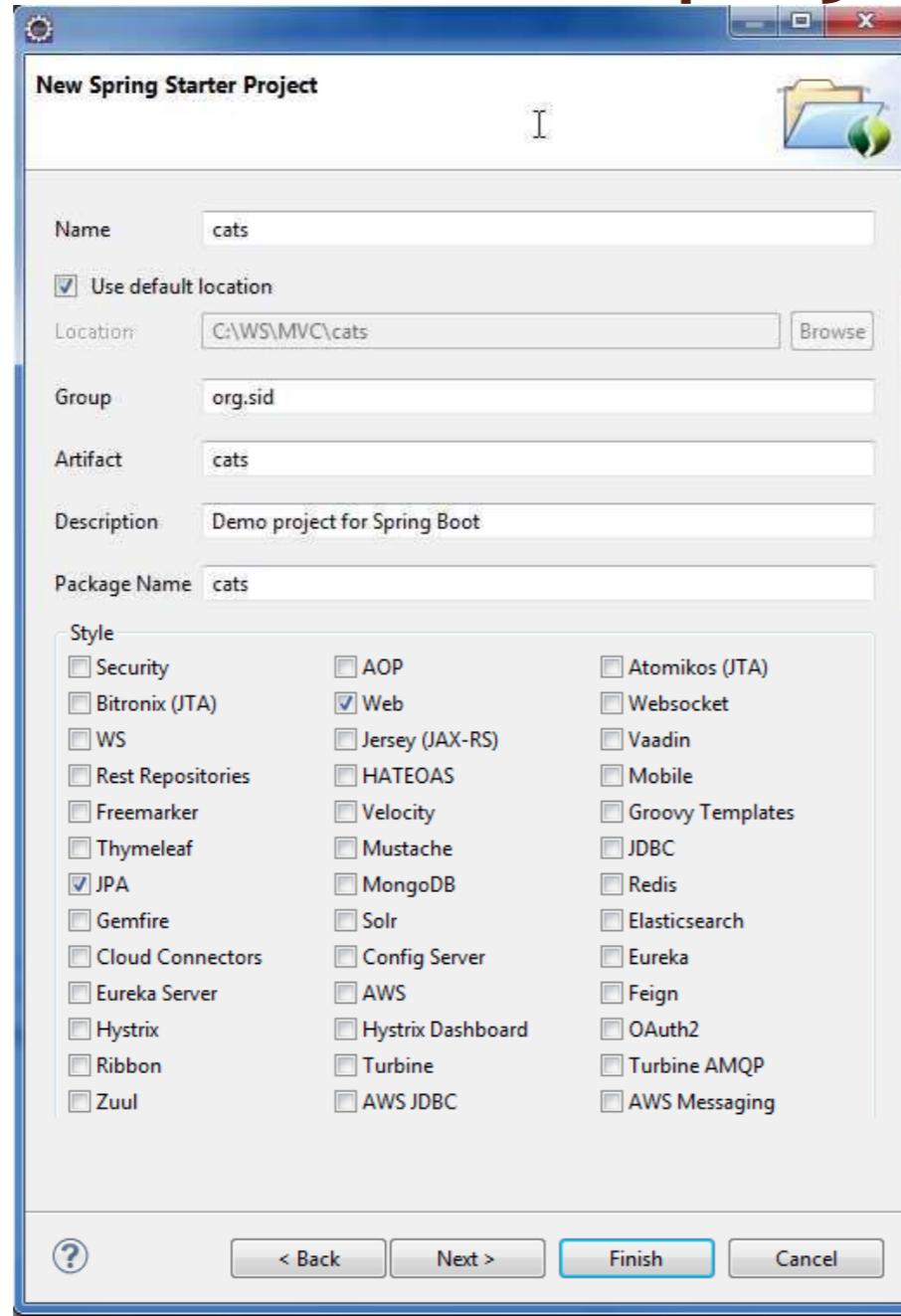
REF	DES	PRIX
P3	P543	1600
P4	P543	1600
P5	P543	1600
P6	P543	1600

0 1 2

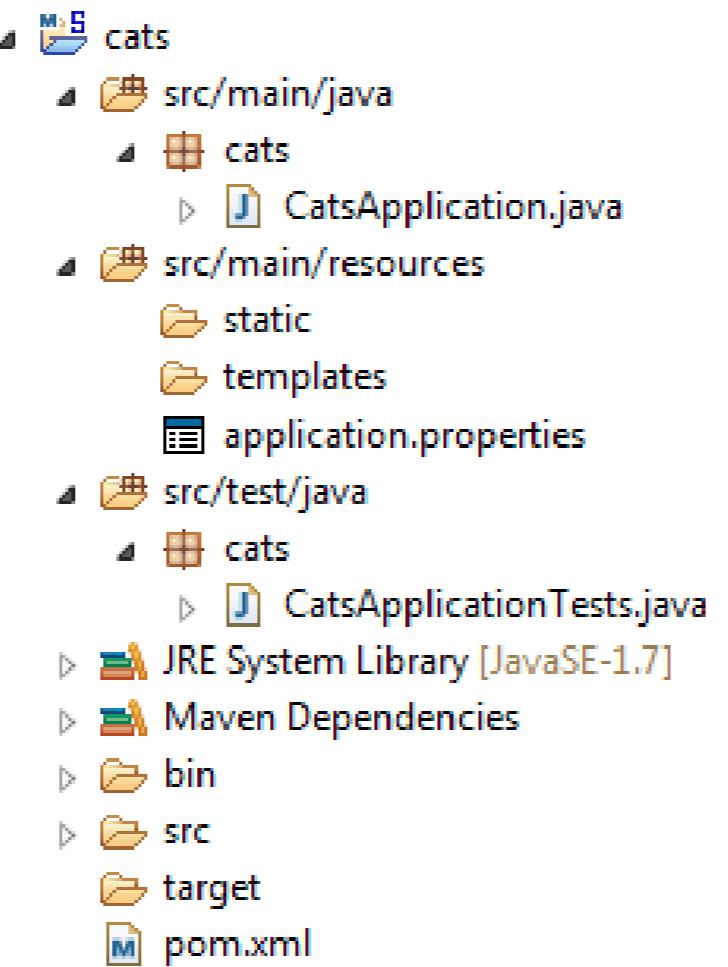
Création d'un projet Spring Boot



Création d'un projet Spring Boot



Structure du projet

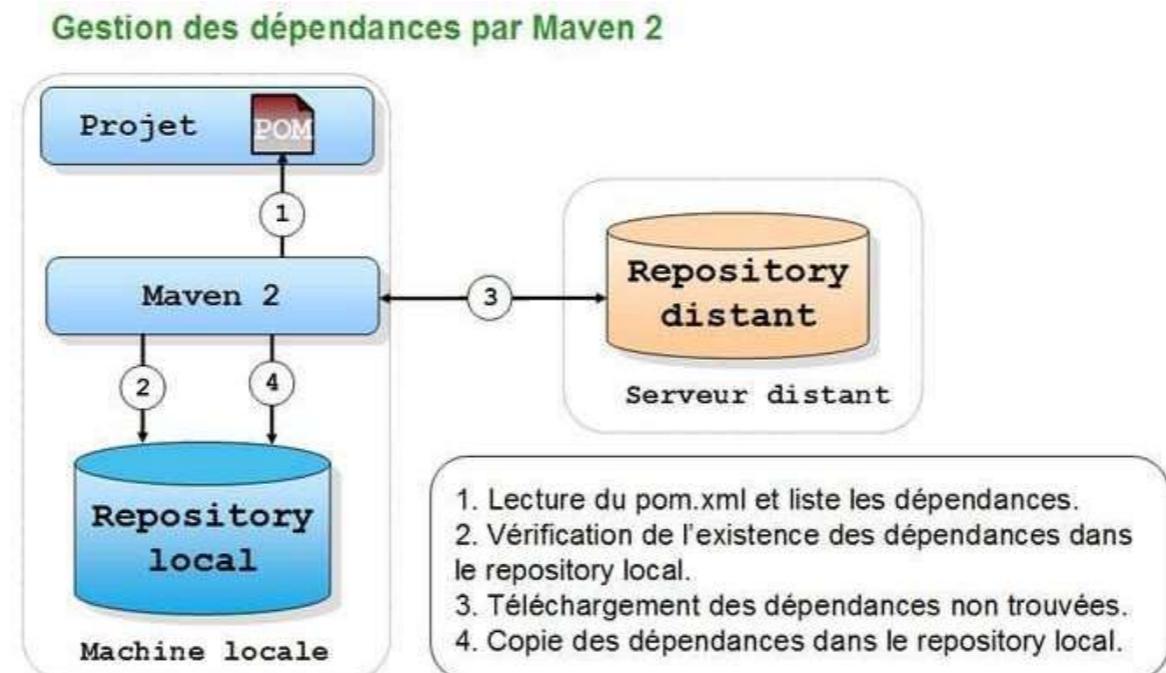


Maven

- Maven, géré par l'organisation *Apache Software Foundation.* (*Jakarta Project*), est un **outil pour la gestion et l'automatisation de production des projets logiciels Java en général et Java EE en particulier.**
- L'objectif recherché est de
 - produire un logiciel à partir de ses sources,
 - en optimisant les tâches réalisées à cette fin
 - et en garantissant le bon ordre de fabrication.
 - Compiler, Tester, Contrôler, produire les packages livrables
 - Publier la documentation et les rapports sur la qualité
- Apports :
 - Simplification du processus de construction d'une application
 - Fournit les bonnes pratiques de développement
 - Tend à uniformiser le processus de construction logiciel
 - Vérifier la qualité du code
 - Faciliter la maintenance d'un projet

Maven : POM

- Maven utilise un paradigme connu sous le nom de **Project Object Model (POM)** afin de :
 - Décrire un projet logiciel,
 - Ses dépendances avec des modules externes
 - et l'ordre à suivre pour sa production.
- Il est livré avec un grand nombre de tâches (**GOLS**) prédéfinies, comme la compilation du code Java ou encore sa modularisation.



pom.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
<modelVersion>4.0.0</modelVersion>

<groupId>org.sid</groupId>
<artifactId>cats</artifactId>
<version>0.0.1-SNAPSHOT</version>
<packaging>jar</packaging>
<name>cats</name>
<description>Demo project for Spring Boot</description>
<parent>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-parent</artifactId>
    <version>1.2.3.RELEASE</version>
    <relativePath/> <!-- lookup parent from repository -->
</parent>
<properties>
    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
    <start-class>cats.CatsApplication</start-class>
    <java.version>1.7</java.version>
</properties>
```

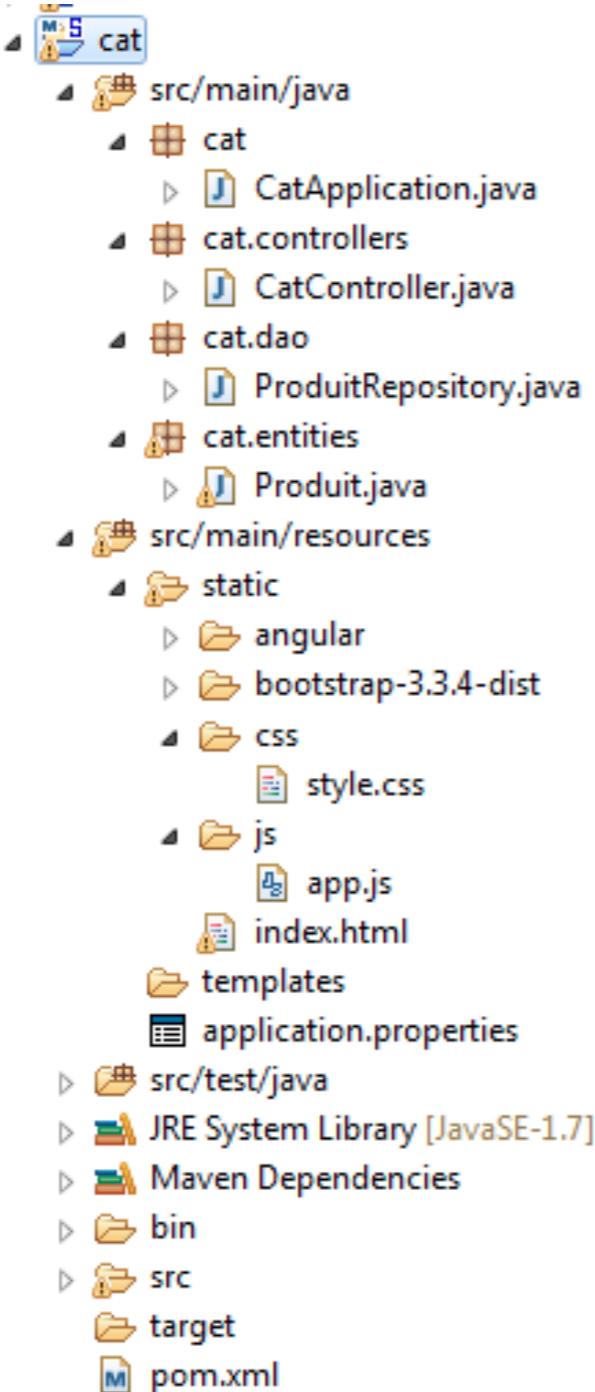
pom.xml

```
<dependencies>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-data-jpa</artifactId>
    </dependency>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-web</artifactId>
    </dependency>
    <dependency>
        <groupId>mysql</groupId>
        <artifactId>mysql-connector-java</artifactId>
    </dependency>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-test</artifactId>
        <scope>test</scope>
    </dependency>
</dependencies>
```

pom.xml

```
<build>
  <plugins>
    <plugin>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-maven-plugin</artifactId>
    </plugin>
  </plugins>
</build>

</project>
```



Structure du projet

application.properties

```
# DataSource settings:  
spring.datasource.url = jdbc:mysql://localhost:3306/db_boot_cat5  
spring.datasource.username = root  
spring.datasource.password =  
spring.datasource.driverClassName = com.mysql.jdbc.Driver  
  
# Specify the DBMS  
spring.jpa.database = MYSQL  
# Show or not log for each sql query  
spring.jpa.show-sql = true  
# Hibernate ddl auto (create, create-drop, update)  
spring.jpa.hibernate.ddl-auto = update  
# Naming strategy  
spring.jpa.hibernate.naming-strategy = org.hibernate.cfg.ImprovedNamingStrategy  
  
spring.jpa.properties.hibernate.dialect = org.hibernate.dialect.MySQL5Dialect
```

Spring Boot Application

```
package cat;
import org.springframework.boot.SpringApplication;
import
    org.springframework.boot.autoconfigure.SpringBootApplication
@SpringBootApplication
public class CatApplication {
    public static void main(String[] args) {
        SpringApplication.run(CatApplication.class, args);
    }
}
```

ENTITÉS

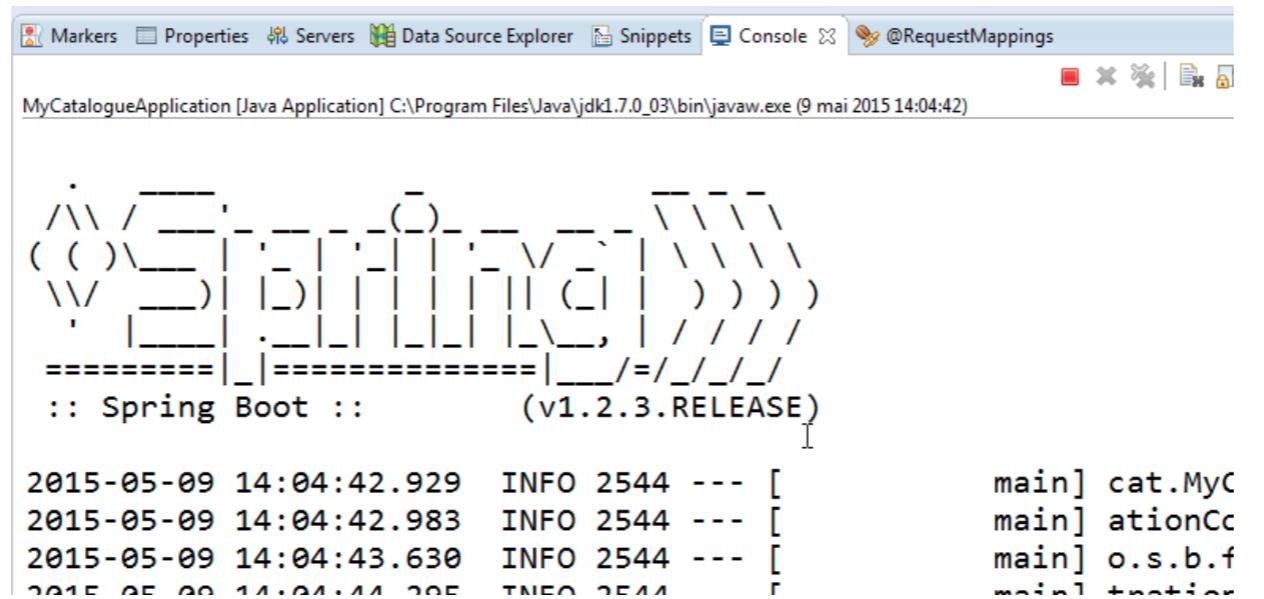
Entité Produit

```
package cat.entities;  
@Entity  
public class Produit implements Serializable {  
    @Id  
    @GeneratedValue(strategy=GenerationType.IDENTITY)  
    private Long reference;  
    private String designation;  
    private double prix;  
    // getters et Setters  
    // Constructeurs  
}
```

COUCHE DAO

Tester les entités

- Démarrer l'application et vérifier si la table produits a été bien créée.



The screenshot shows the Eclipse IDE interface with the 'Console' tab selected. The title bar indicates the project is 'MyCatalogueApplication [Java Application]' and the path is 'C:\Program Files\Java\jdk1.7.0_03\bin\javaw.exe'. The log output is as follows:

```
.
.
.
=====
:: Spring Boot ::      (v1.2.3.RELEASE)

2015-05-09 14:04:42.929  INFO 2544 --- [           main] cat.MyC
2015-05-09 14:04:42.983  INFO 2544 --- [           main] ationCc
2015-05-09 14:04:43.630  INFO 2544 --- [           main] o.s.b.f
2015-05-09 14:04:44.205  INFO 2544 --- [           main] tationCc
```

Nom	Type	Interclassement	Attributs	Null	Défaut	Extra
<u>reference</u>	bigint(20)			Non	Aucune	AUTO_INCREMENT
<u>designation</u>	varchar(255) latin1_swedish_ci			Oui	NULL	
<u>prix</u>	double			Non	Aucune	

Couche DAO

```
package cat.dao;

public interface IProduitRepository extends JpaRepository<Produit, Long> {
    @Query("select p from Produit p where p.designation like :x")
    public Page<Produit> produitParMC(@Param("x")String mc,Pageable p);
    public List<Produit> findByDesignation(String des);
    public Page<Produit> findByDesignation(String des,Pageable p);
}
```

COUCHEWEB

Le contrôleur Rest

```
package cat.controllers;

@RestController
public class CatalogueController {

    @Autowired
    private IProduitRepository produitRepository;

    @RequestMapping("/save")
    public Produit saveProduit(Produit p){
        produitRepository.save(p);
        return p;
    }

    @RequestMapping("/all")
    public List<Produit> getProduits(){
        return produitRepository.findAll();
    }
}
```

Le contrôleur Rest

```
@RequestMapping("/produits")
public Page<Produit> getProduits(int page){
    return produitRepository.findAll(new PageRequest(page, 5));
}

@RequestMapping("/produitsParMC")
public Page<Produit> getProduits(String mc,int page){
    return produitRepository.produitParMC("%"+mc+"%", new
PageRequest(page, 5));
}

@RequestMapping("/get")
public Produit getProduit(Long ref){
    return produitRepository.findOne(ref);
}
```

Le contrôleur Rest

```
@RequestMapping("/delete")
public boolean delete(Long ref){
    produitRepository.delete(ref);
    return true;
}

@RequestMapping("/update")
public Produit update(Produit p){
    produitRepository.saveAndFlush(p);
    return p;
}
```

Ajouter un produit

Coté Client

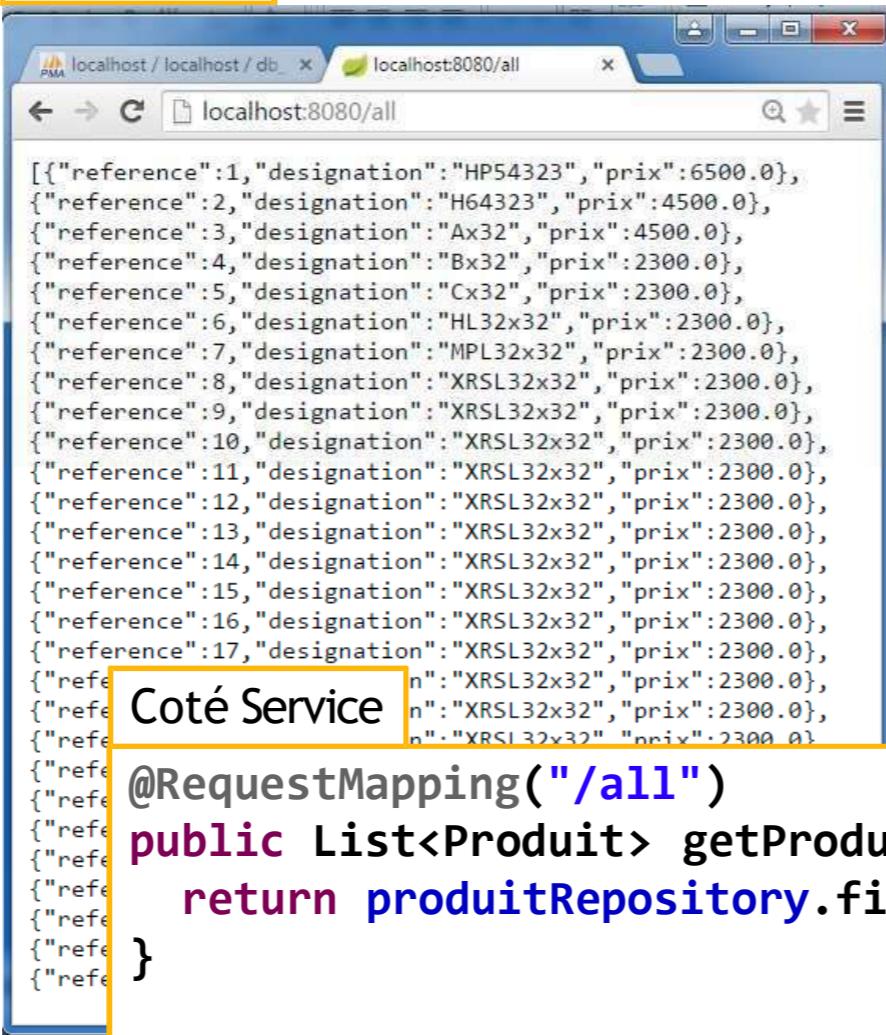


Coté Service

```
@RequestMapping("/save")
public Produit saveProduit(Produit p){
    produitRepository.save(p);
    return p;
}
```

Consulter tous les produits

Coté Client



Consulter une page de produits

Coté Client

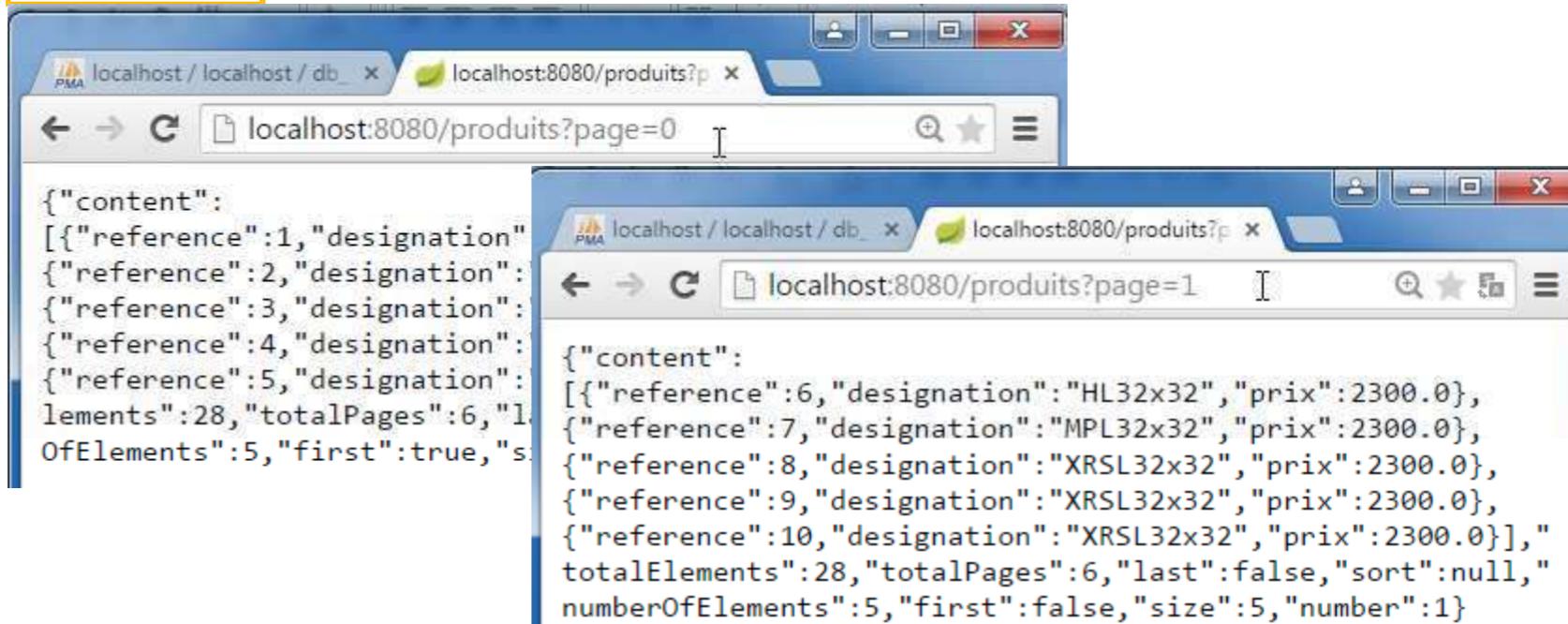


Coté Service

```
@RequestMapping("/produits")
public Page<Produit> getProduits(int page){
    return produitRepository.findAll(new PageRequest(page, 5));
}
```

Consulter une page de produits

Coté Client



Coté Service

```
@RequestMapping("/produits")
public Page<Produit> getProduits(int page){
    return produitRepository.findAll(new PageRequest(page, 5));
}
```

Chercher les produits par mot clé

Coté Client

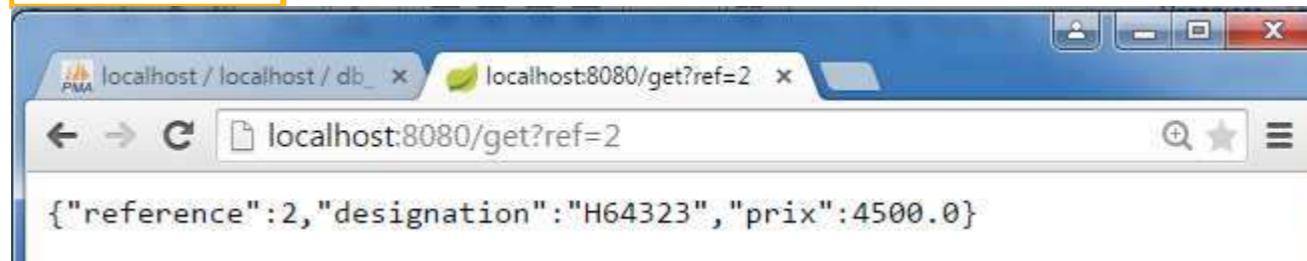


Coté Service

```
@RequestMapping("/produitsParMC")
public Page<Produit> getProduits(String mc, int page){
    return produitRepository.produitParMC("%" + mc + "%", new PageRequest(page, 5));
}
```

Consulter un produit

Coté Client

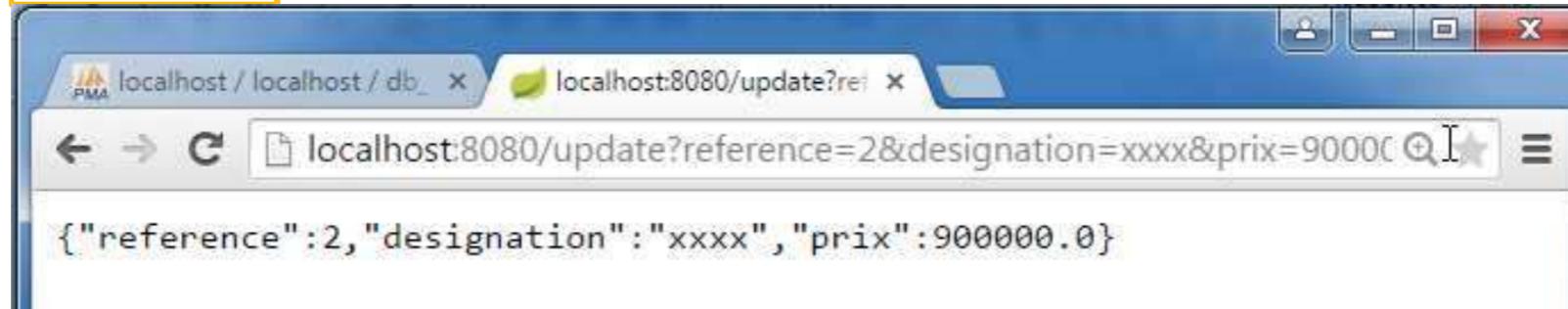


Coté Service

```
@RequestMapping("/get")
public Produit getProduit(Long ref){
    return produitRepository.findOne(ref);
}
```

Mettre à jour à produit

Coté Client

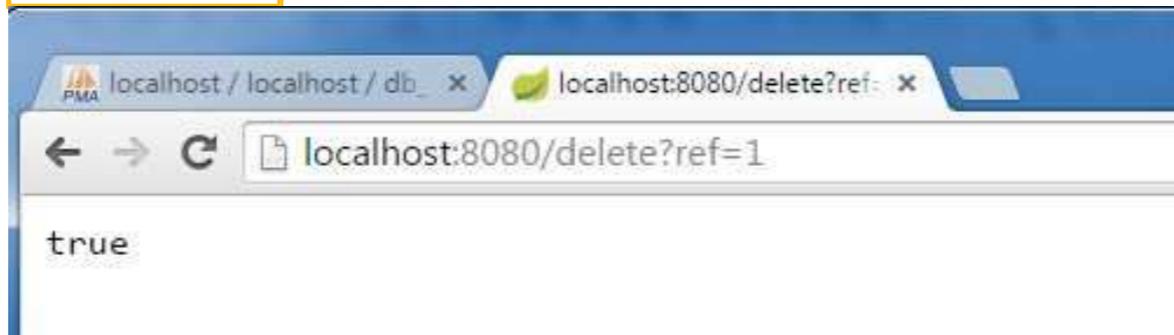


Coté Service

```
@RequestMapping("/update")
public Produit update(Produit p){
    produitRepository.saveAndFlush(p);
    return p;
}
```

Supprimer un produit

Coté Client

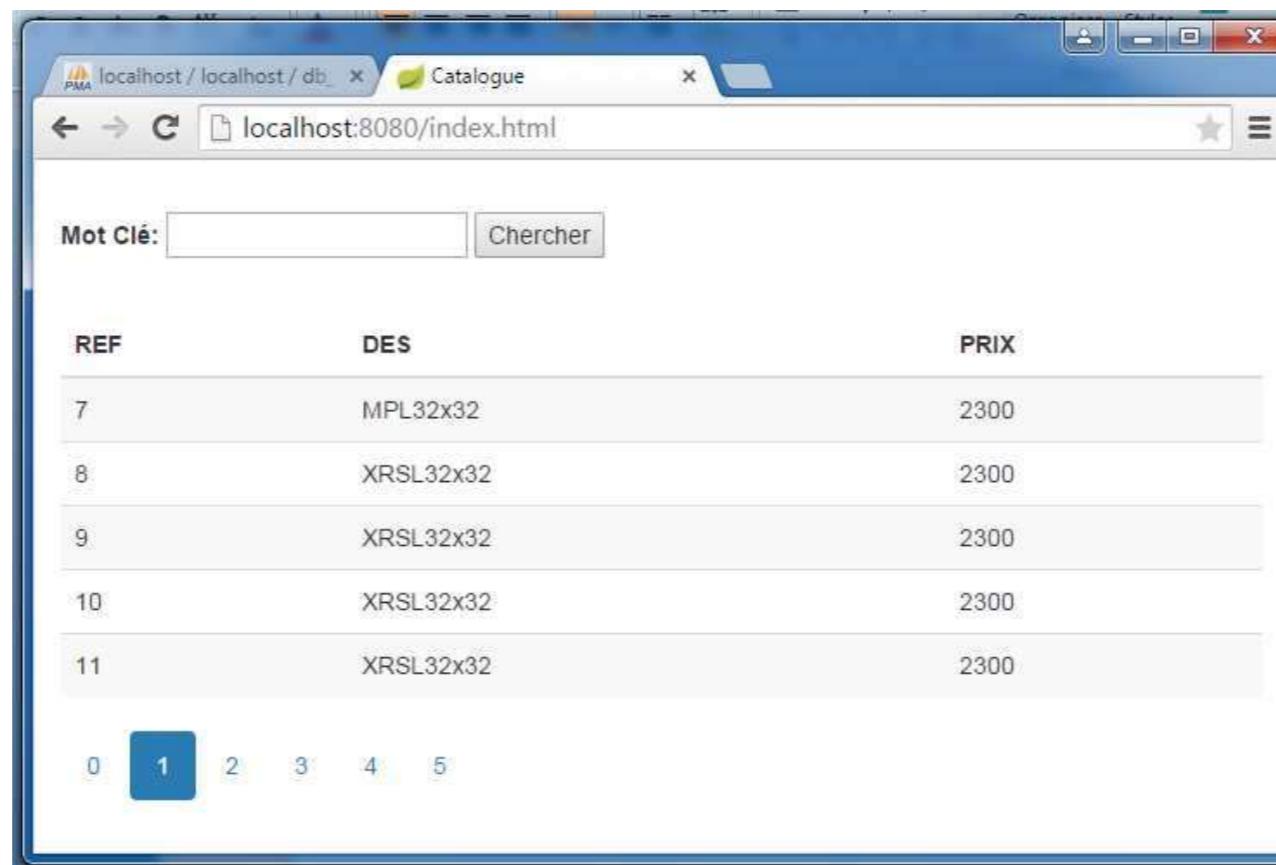


Coté Service

```
@RequestMapping("/delete")
public boolean delete(Long ref){
    produitRepository.delete(ref);
    return true;
}
```

Couche Présentation Web Coté Client

- HTML5
- Java Script avec le framework Angular JS
- CSS avec le framework BootStrap



A screenshot of a web browser window titled "Catalogue". The address bar shows "localhost:8080/index.html". The page displays a search form with a "Mot Clé:" input field and a "Chercher" button. Below the form is a table with columns "REF", "DES", and "PRIX". The table contains five rows of data:

REF	DES	PRIX
7	MPL32x32	2300
8	XRSL32x32	2300
9	XRSL32x32	2300
10	XRSL32x32	2300
11	XRSL32x32	2300

At the bottom, there is a navigation bar with buttons labeled 0, 1, 2, 3, 4, 5, where the number 1 is highlighted.

index.html

```
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>Catalogue</title>
    <link rel="stylesheet" type="text/css" href="bootstrap-3.3.4-
dist/css/bootstrap.min.css"/>
    <link rel="stylesheet" type="text/css" href="css/style.css"/>
</head>
<body ng-app="MyCat" ng-controller="CatController" >
    <div class="container spacer">
        <form>
            <label>Mot Clé:</label>
            <input type="text" ng-model="motCle">
            <button ng-click="charger()">Chercher</button>
        </form>
    </div>
```

index.html

```
<div class="container spacer">
  <table class="table table-striped">
    <thead>
      <tr>
        <th>REF</th><th>DES</th><th>PRIX</th>
      </tr>
    </thead>
    <tbody>
      <tr ng-repeat="p in produits.content">
        <td>{{p.reference}}</td>
        <td>{{p.designation}}</td>
        <td>{{p.prix}}</td>
      </tr>
    </tbody>
  </table>
</div>
```

index.html

```
<div class="container">
  <ul class="nav nav-pills">
    <li ng-class="{active:$index==pageCourante}"
        class="clickable" ng-repeat="p in pages track by $index">
      <a ng-click="gotoPage($index)">{{$index}}</a>
    </li>
  </ul>
</div>
<script type="text/javascript"
src="angular/angular.min.js"></script>
<script type="text/javascript" src="js/app.js"></script>
</body>
</html>
```

app.js

```
var app=angular.module("MyCat",[]);
app.controller("CatController",function($scope,$http){
    $scope.produits=[];
    $scope.motCle=null;
    $scope.pageCourante=0;
    $scope.charger=function(){
        $http.get("/produitsParMC?mc="+$scope.motCle+"&page="+$scope.pageCourante)
            .success(function(data){
                $scope.produits=data;
                $scope.pages=new Array(data.totalPages)
            });
    };
    $scope.gotoPage=function(p){
        $scope.pageCourante=p;
        $scope.charger();
    };
});
```

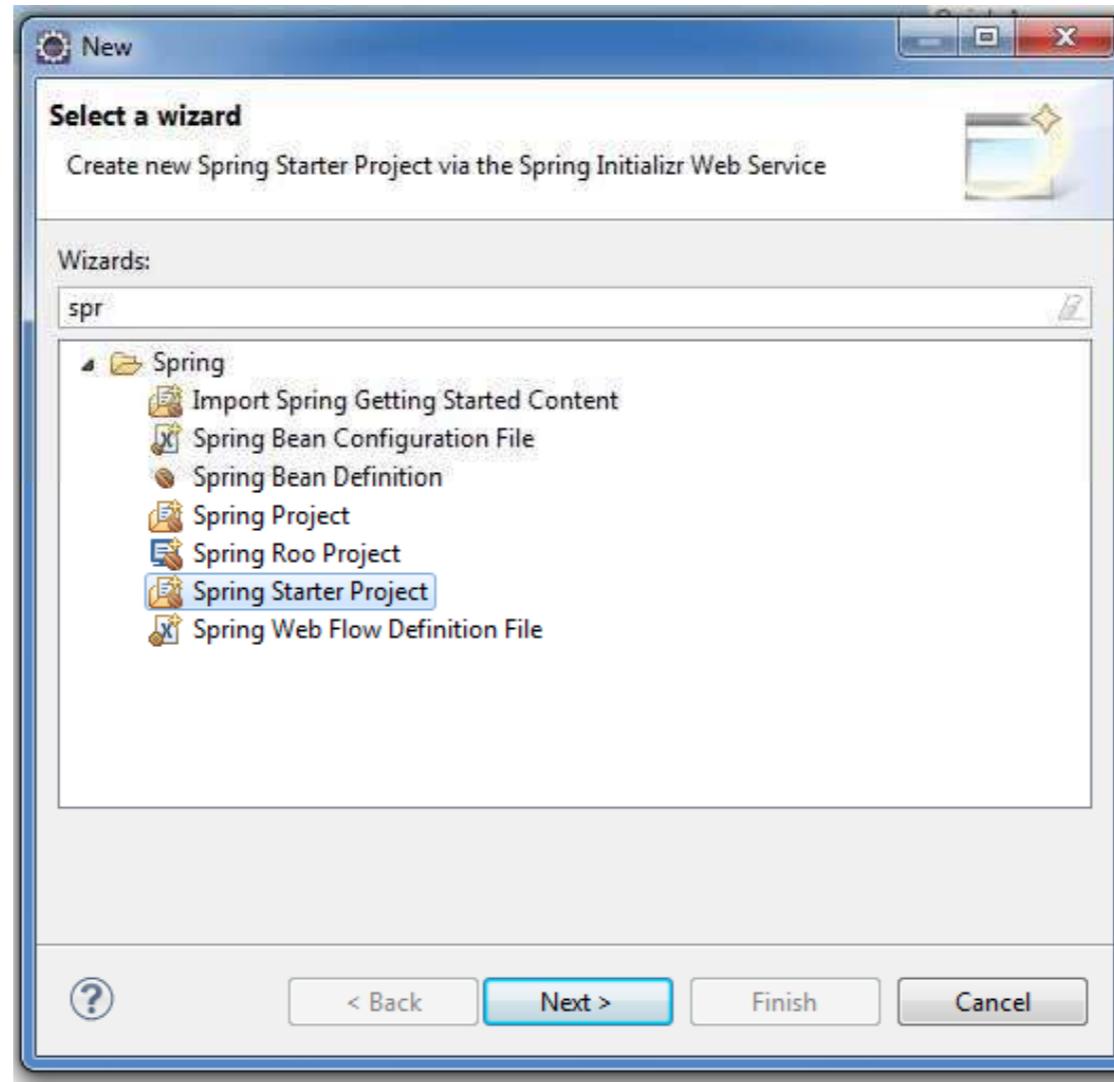
style.css

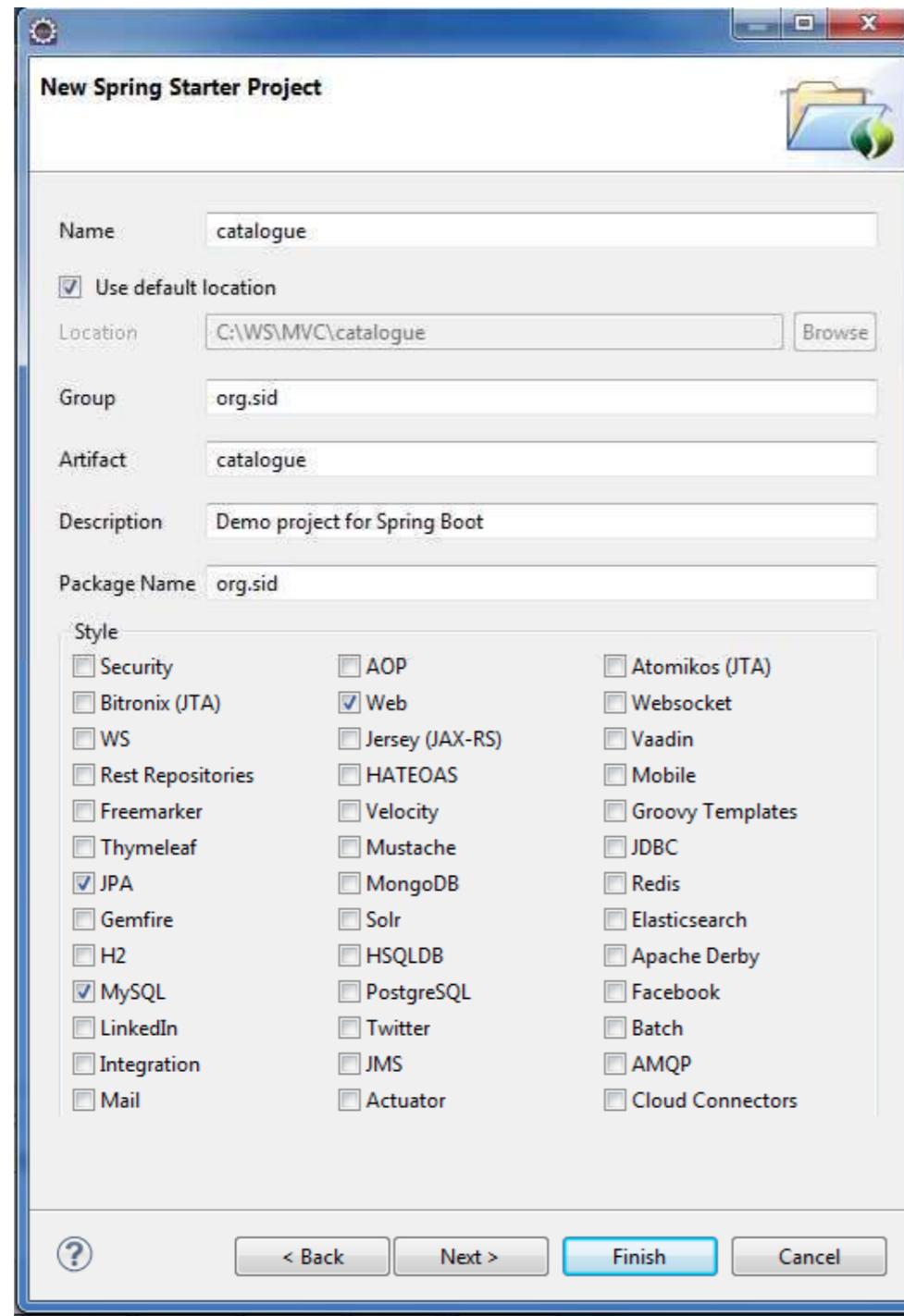
```
.spacer{  
    margin-top: 30px;  
}  
.clickable{  
    cursor: pointer;  
}
```

Deuxième Application

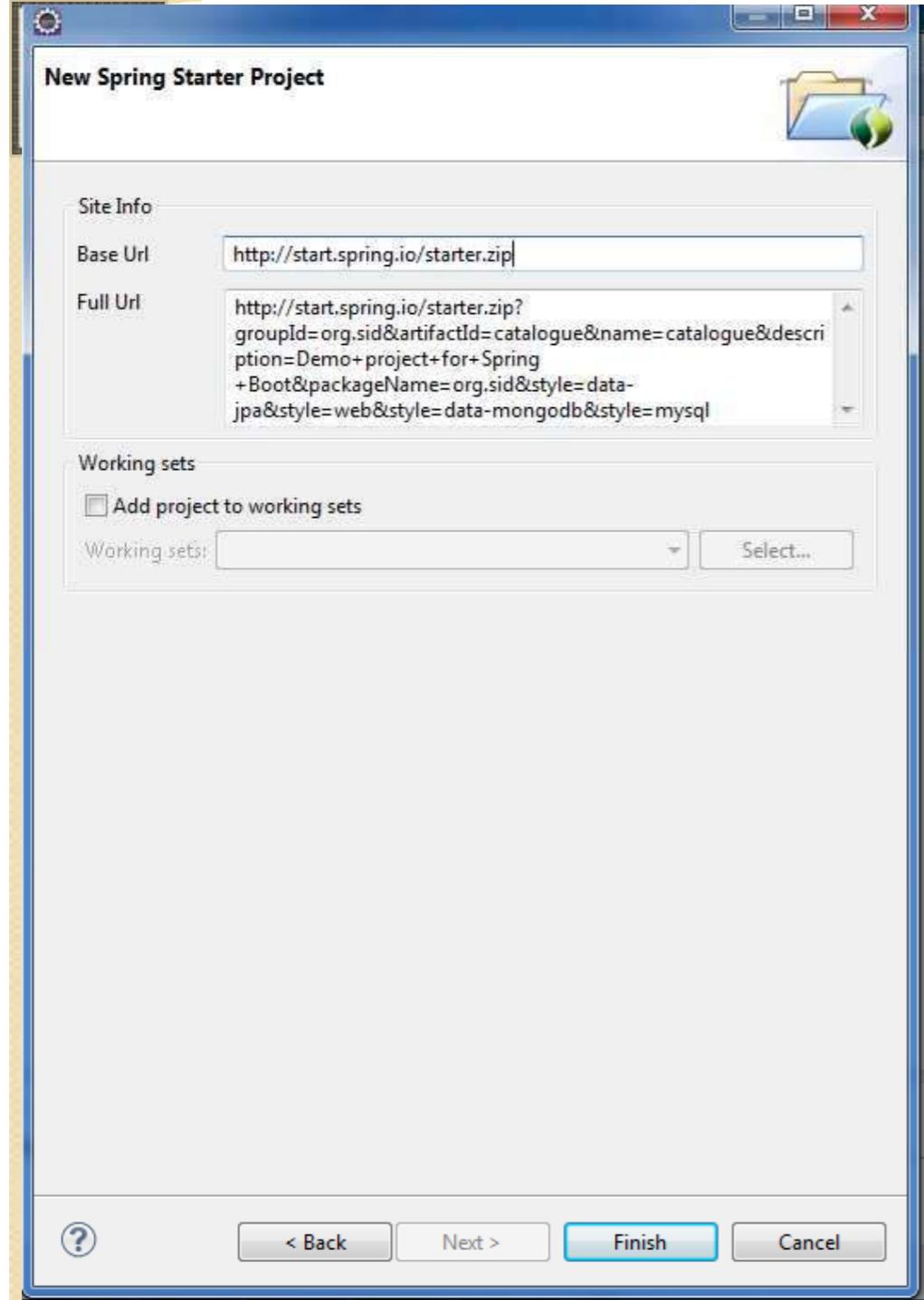
- Supposant que l'on souhaite créer une application qui permet de gérer le catalogue des produits appartenant à des catégories.
- Chaque produit est défini par :
 - Sa référence de type String
 - Sa désignation de type String
 - Son prix de type double
 - Sa quantité de type int
 - Sa disponibilité de type boolean
 - Sa photo
- Une catégorie est définie par :
 - Son code de type Long (Auto Increment)
 - Son nom de type String
- L'application doit permettre
 - D'ajouter une nouvelle catégorie
 - Ajouter un produit appartenant à une catégorie
 - Consulter toutes les catégories
 - Consulter les produits dont le nom contient un mot clé
 - Consulter les produits d'une catégorie
 - Consulter un produit
 - Mettre à jour un produit
 - Supprimer une catégorie
- L'injection des dépendances sera assurée par Spring IOC

Création d'un projet Spring Starter



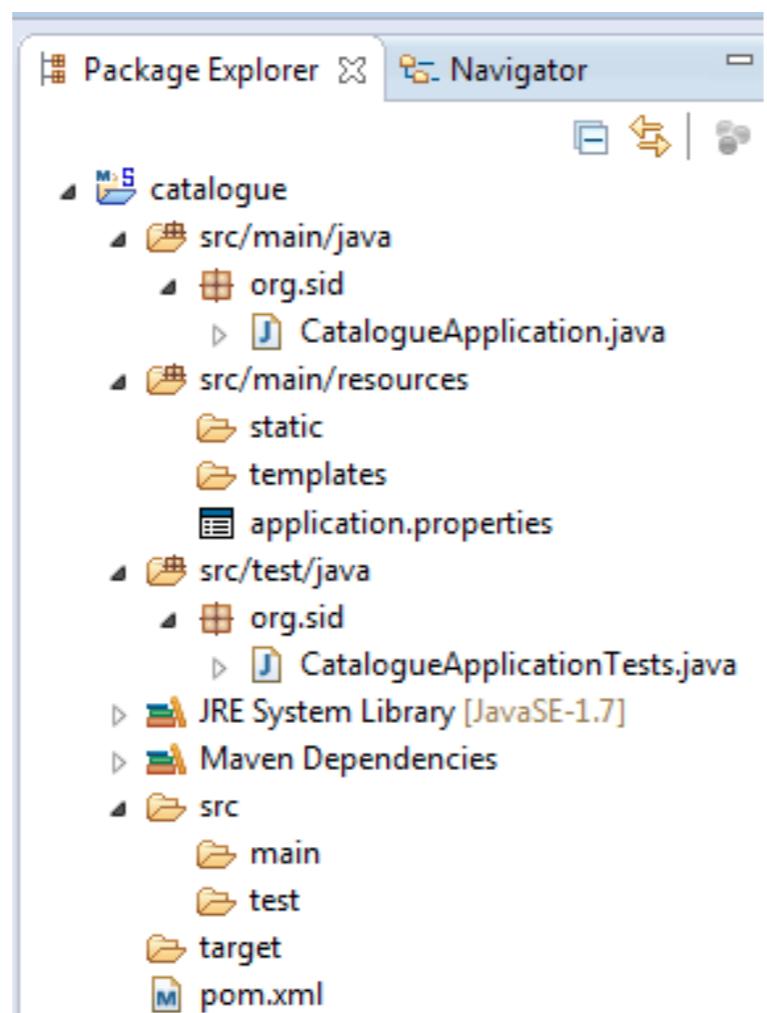


Création d'un projet Spring Starter



Création d'un projet Spring Starter

Structure du projet



Pom.xml

```
<modelVersion>4.0.0</modelVersion>
<groupId>org.sid</groupId>
<artifactId>catalogue</artifactId>
<version>0.0.1-SNAPSHOT</version>
<packaging>jar</packaging>

<name>catalogue</name>
<description>Demo project for Spring Boot</description>

<parent>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-parent</artifactId>
  <version>1.2.3.RELEASE</version>
  <relativePath/> <!-- lookup parent from repository -->
</parent>
```

Pom.xml

```
<properties>
    <project.build.sourceEncoding>UTF-
    8</project.build.sourceEncoding>
    <start-class>org.sid.CatalogueApplication</start-class>
    <java.version>1.7</java.version>
</properties>
<dependencies>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-data-jpa</artifactId>
    </dependency>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-web</artifactId>
    </dependency>
```

Pom.xml

```
<dependency>
    <groupId>mysql</groupId>
    <artifactId>mysql-connector-java</artifactId>
    <scope>runtime</scope>
</dependency>
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-test</artifactId>
    <scope>test</scope>
</dependency>
</dependencies>
<build>
    <plugins>
        <plugin>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-maven-plugin</artifactId>
        </plugin>
    </plugins>
</build>
```

application.properties

```
# DataSource settings:  
spring.datasource.url =  
    jdbc:mysql://localhost:3306/db_boot  
spring.datasource.username = root  
spring.datasource.password =  
spring.datasource.driverClassName = com.mysql.jdbc.Driver  
  
# Specify the DBMS  
spring.jpa.database = MYSQL  
# Show or not log for each sql query  
spring.jpa.show-sql = true  
# Hibernate ddl auto (create, create-drop,  
update) spring.jpa.hibernate.ddl-auto = update  
# Naming strategy  
spring.jpa.hibernate.naming-strategy =  
org.hibernate.cfg.ImprovedNamingStrategy  
  
spring.jpa.properties.hibernate.dialect =  
org.hibernate.dialect.MySQL5Dialect # View Resolver  
spring.view.prefix: /WEB-INF/views/  
spring.view.suffix: .jsp
```

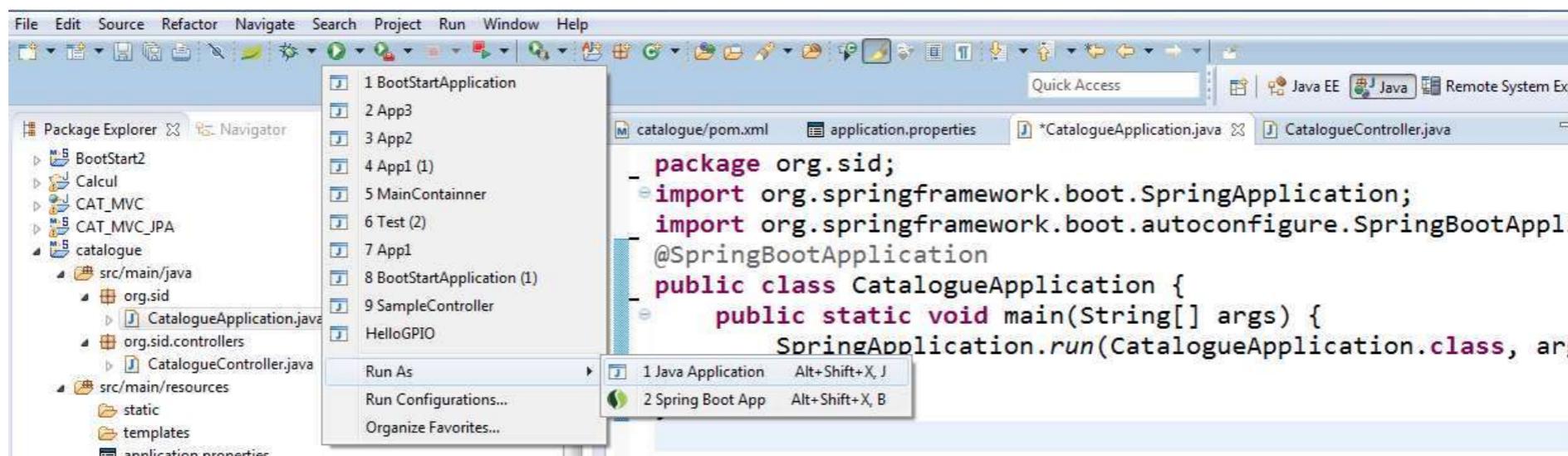
SpringBootApplication

```
package org.sid;  
import org.springframework.boot.SpringApplication;  
import org.springframework.boot.autoconfigure.SpringBootApplication;  
@SpringBootApplication  
public class CatalogueApplication {  
    public static void main(String[] args) {  
        SpringApplication.run(CatalogueApplication.class, args);  
    }  
}
```

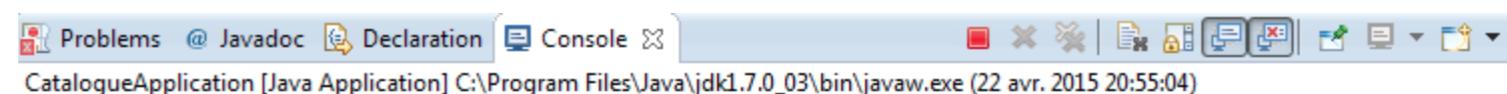
Premier Controleur Spring

```
package org.sid.controllers;
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.ResponseBody;
@Controller
public class CatalogueController {
    @RequestMapping(value="/index")
    @ResponseBody
    public String index(){
        return "Test";
    }
    @RequestMapping(value="/test")
    public String test(){
        return "Test";
    }
}
```

Exécuter l'application SpringBoot

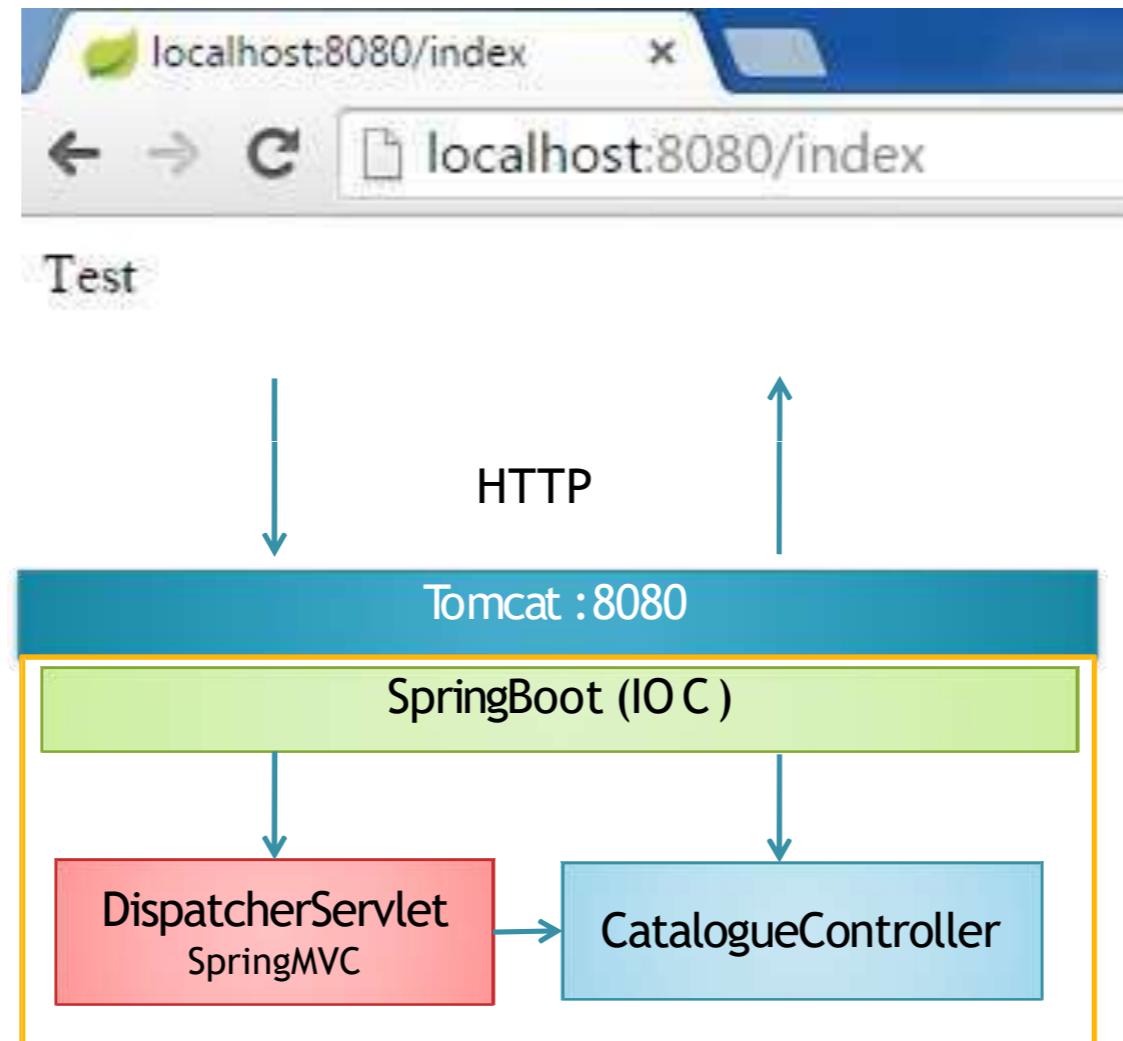


● Run As > Java Application



```
.....(ASCII art representing a stylized tree or logo)
:: Spring Boot ::      (v1.2.3.RELEASE)
[2015-04-22 20:55:05.420] INFO 7904 --- [main] org.sid.CatalogueApplication : Starting CatalogueApplication on med@youssfi.net [ENSEEIHT Université Hassan II de Casablanca]
[2015-04-22 20:55:05.477] INFO 7904 --- [main] o.s.b.SpringApplication : Application 'CatalogueApplication' running in "prod" profile on port 8080
[2015-04-22 20:55:06.167] INFO 7904 --- [main] o.s.b.SpringApplication : Started application in 0.738 seconds (JVM: 0.738s / Tomcat: 0.000s)
```

Tester l'application



Travailler Coté serveur avec JSP et JSTL

```
<dependency>
    <groupId>org.apache.tomcat.embed</groupId>
    <artifactId>tomcat-embed-jasper</artifactId>
    <scope>provided</scope>
</dependency>

<dependency>
    <groupId>javax.servlet</groupId>
    <artifactId>jstl</artifactId>
</dependency>
```



Exemple d'application

- Supposant que l'on souhaite créer une application qui permet de gérer le catalogue des produits appartenant à des catégories.
- Chaque produit est défini par :
 - Sa référence de type String
 - Sa désignation de type String
 - Son prix de type double
 - Sa quantité de type int
 - Sa disponibilité de type boolean
 - Sa photo
- Une catégorie est définie par :
 - Son code de type Long (Auto Increment)
 - Son nom de type String
- L'application doit permettre
 - D'ajouter une nouvelle catégorie
 - Ajouter un produit appartenant à une catégorie
 - Consulter toutes les catégories
 - Consulter les produits dont le nom contient un mot clé
 - Consulter les produits d'une catégorie
 - Consulter un produit
 - Mettre à jour un produit
 - Supprimer une catégorie
- L'injection des dépendances sera assurée par Spring IOC

ENTITÉS

Entités : Produit et Categorie

Diagramme de classes : Modèle Objet



MLDR : Modèle Relationnel

- CATEGORIES (ID_CAT, NOM_CAT)
- PRODUITS (REFDES, PRIX, QTE, DISPO, PHOTO, #ID_CAT)

Entité : Produit

```
package dao;
import java.io.Serializable; import javax.persistence.*;
@Entity
public class Produit implements Serializable {
    @Id
    private String reference;
    private String designation; private double prix;    private int quantite;
    private boolean disponible;  private String photo;
    @ManyToOne
    private Categorie categorie;
    public Produit() { }
    public Produit(String reference, String designation, double prix, int
quantite, boolean disponible, String photo) {
        this.reference = reference; this.designation = designation;
        this.prix = prix; this.quantite = quantite; this.disponible = disponible;
        this.photo = photo;
    }
    // Getters et Setters
}
```

Entité : Categorie

```
package dao;
import java.io.Serializable; import
java.util.Collection; import javax.persistence.*;
@Entity
public class Categorie implements Serializable {
    @Id
    @GeneratedValue(strategy=GenerationType.IDENTITY)
    private Long idCategorie;
    private String nomCategorie;
    @OneToMany(mappedBy="categorie",fetch=FetchType.LAZY)
    private Collection<Produit> produits;
    public Categorie() { }
    public Categorie(String nomCategorie) {
        this.nomCategorie = nomCategorie;
    }
    // Getters et Setters
    @JsonIgnore
    @XmlTransient
    public Collection<Produit> getProduits() {
        return produits;
    }
}
```

Redémarrer l'application Spring Boot

- Les tables devraient être générées

Table Categories :

Nom	Type	Interclassement	Attributs	Null	Défaut	Extra
<u>id_categorie</u>	bigint(20)			Non	Aucune	AUTO_INCREMENT
<u>nom_categorie</u>	varchar(255)	latin1_swedish_ci		Oui	NULL	

Table Produits :

#	Nom	Type	Interclassement	Attributs	Null	Défaut	Extra
1	<u>reference</u>	varchar(255)	latin1_swedish_ci		Non	Aucune	
2	<u>designation</u>	varchar(255)	latin1_swedish_ci		Oui	NULL	
3	<u>disponible</u>	bit(1)			Non	Aucune	
4	<u>photo</u>	varchar(255)	latin1_swedish_ci		Oui	NULL	
5	<u>prix</u>	double			Non	Aucune	
6	<u>quantite</u>	int(11)			Non	Aucune	
7	<u>id_cat</u>	bigint(20)			Oui	NULL	

Couche DAO

```
package org.sid.dao;  
import org.sid.entities.Categorie;  
import org.springframework.data.jpa.repository.JpaRepository;  
  
public interface CategorieRepository extends JpaRepository<Categorie, Long> {  
}
```

```
package org.sid.dao;  
import java.util.List;  
import org.sid.entities.*;  
import org.springframework.data.domain.Page;  
import org.springframework.data.domain.Pageable;  
import org.springframework.data.jpa.repository.JpaRepository;  
import org.springframework.data.repository.query.Param;  
  
public interface ProduitRepository extends JpaRepository<Produit, String> {  
    public List<Produit> findByDesignation(@Param("mc")String mc);  
    public Page<Produit> findByCategorie(Categorie categorie,Pageable pageable);  
}
```



MVC CLIENT SIDE

- HTML, CSS, JAVA SCRIPT
- ANDROÏDE CLIENT
- IOS CLIENT

Contrôleur :Gestion des produits et des catégories

```
package org.sid.controllers;
import java.util.List;
import org.sid.dao.CategorieRepository; import
org.sid.dao.ProduitRepository; import org.sid.entities.Categorie; import
org.sid.entities.Produit;
import
org.springframework.beans.factory.annotation.Autowired;
import org.springframework.data.domain.Page;
import org.springframework.data.domain.PageRequest;
import org.springframework.stereotype.Controller;
import
org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.ResponseBody;

@Controller
public class MyCatalogueController {
    @Autowired
    private CategorieRepository categorieRepository;
    @Autowired
    private ProduitRepository produitRepository;

    @RequestMapping(value="/saveCat")
    @ResponseBody
    public Categorie saveCategorie(Categorie c){
```

Contrôleur :Gestion des produits et des catégories

```
@RequestMapping(value="/allCat")
@ResponseBody
public List<Categorie> allCategories(){
    return categorieRepository.findAll();
}

@RequestMapping(value="/saveProduit")
@ResponseBody
public Produit saveProduit(Produit p){
    return
        produitRepository.save(p);
}

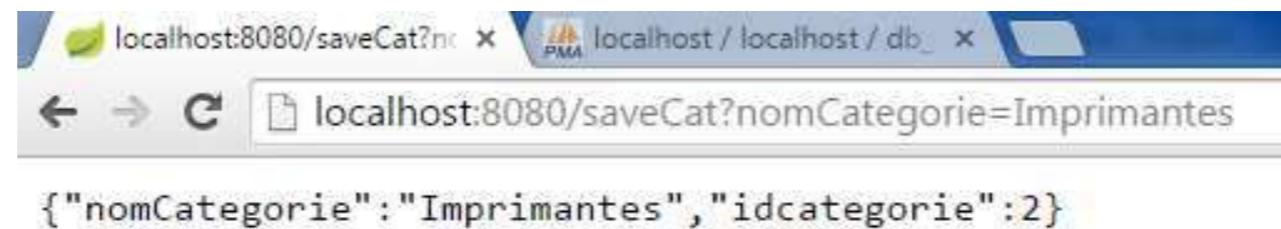
@RequestMapping(value="/allProduits")
@ResponseBody
public List<Produit> allProduits(){
    return produitRepository.findAll();
}
```

Contrôleur :Gestion des produits et des catégories

```
@RequestMapping(value="/pageProduits")
@ResponseBody
public Page<Produit> pageProduits(int page){
    return produitRepository.findAll(new PageRequest(page, 3));
}

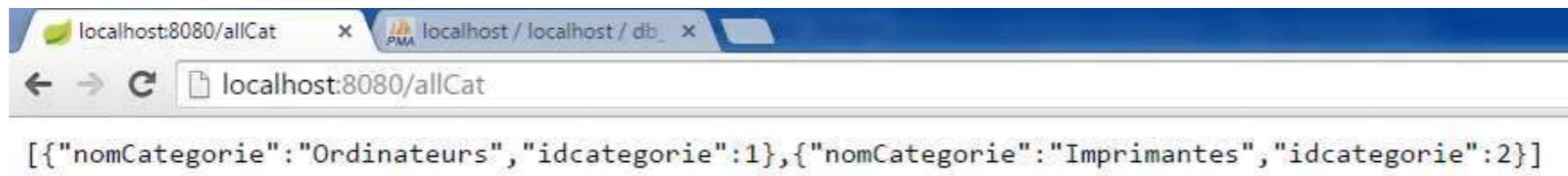
@RequestMapping(value="/produitsParCat")
@ResponseBody
public Page<Produit> produitsParCat(Categorie c,int page){
    return produitRepository.findByCategorie(c, new
    PageRequest(page, 3));
}
@RequestMapping(value="/produitsParMC")
@ResponseBody
public List<Produit> produitsParCat(String mc){
return produitRepository.findByDesignation(mc);
}
}
```

Ajouter une catégorie



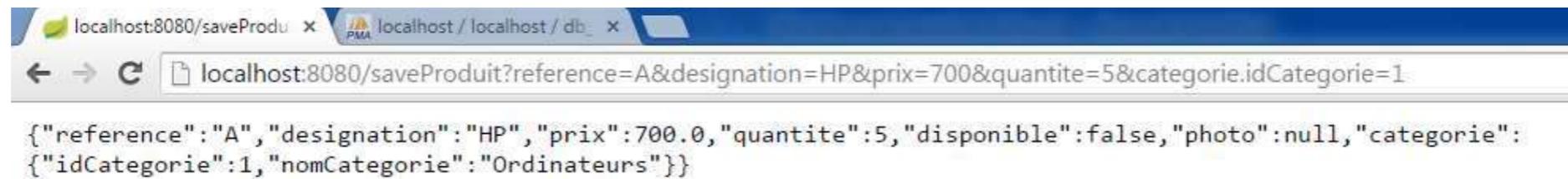
```
@RequestMapping(value="/saveCat")
@ResponseBody
public Categorie saveCategorie(Categorie c){
    return categorieRepository.save(c);
}
```

Consulter toutes les catégories



```
@RequestMapping(value="/allCat")
@ResponseBody
public List<Categorie> allCategories(){
    return categorieRepository.findAll();
}
```

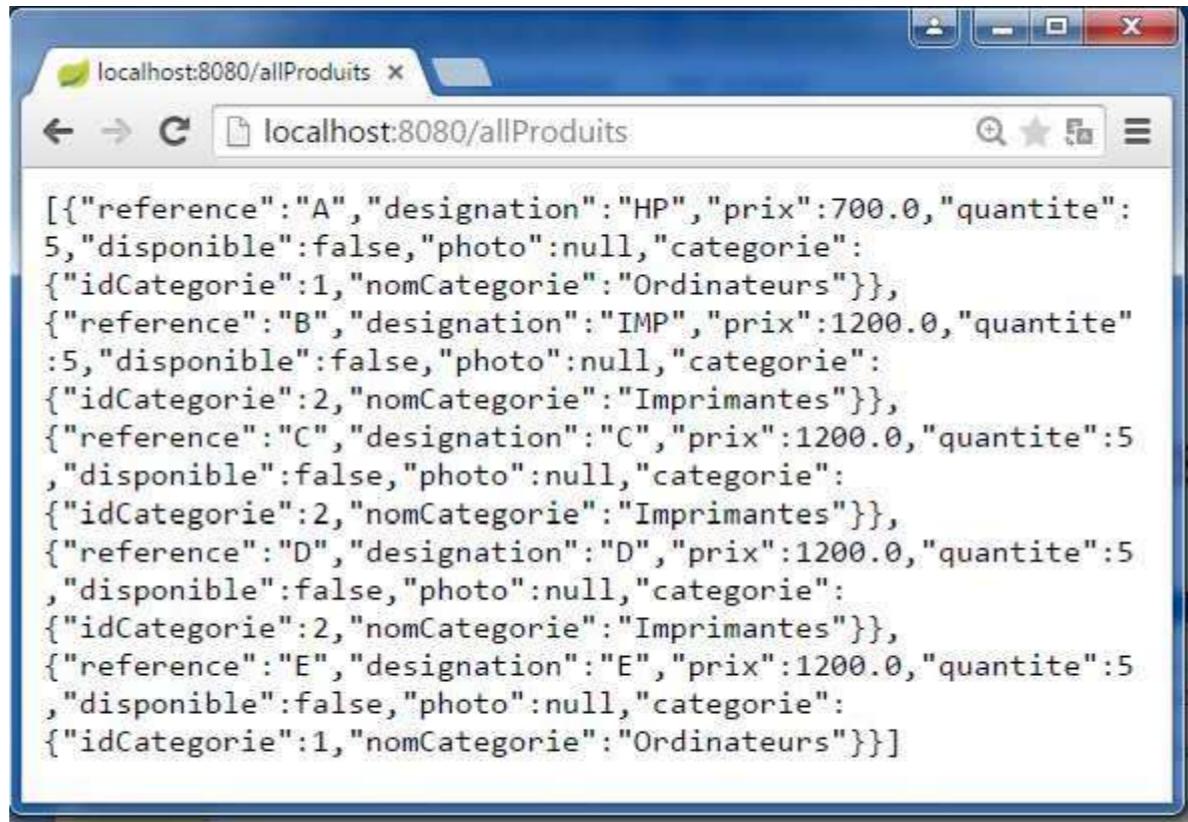
Ajouter un produit



```
{"reference": "A", "designation": "HP", "prix": 700.0, "quantite": 5, "disponible": false, "photo": null, "categorie": {"idCategorie": 1, "nomCategorie": "Ordinateurs"}}
```

```
@RequestMapping(value="/saveProduit")
@ResponseBody
public Produit saveProduit(Produit p){
    return produitRepository.save(p);
}
```

Consulter tous les produits

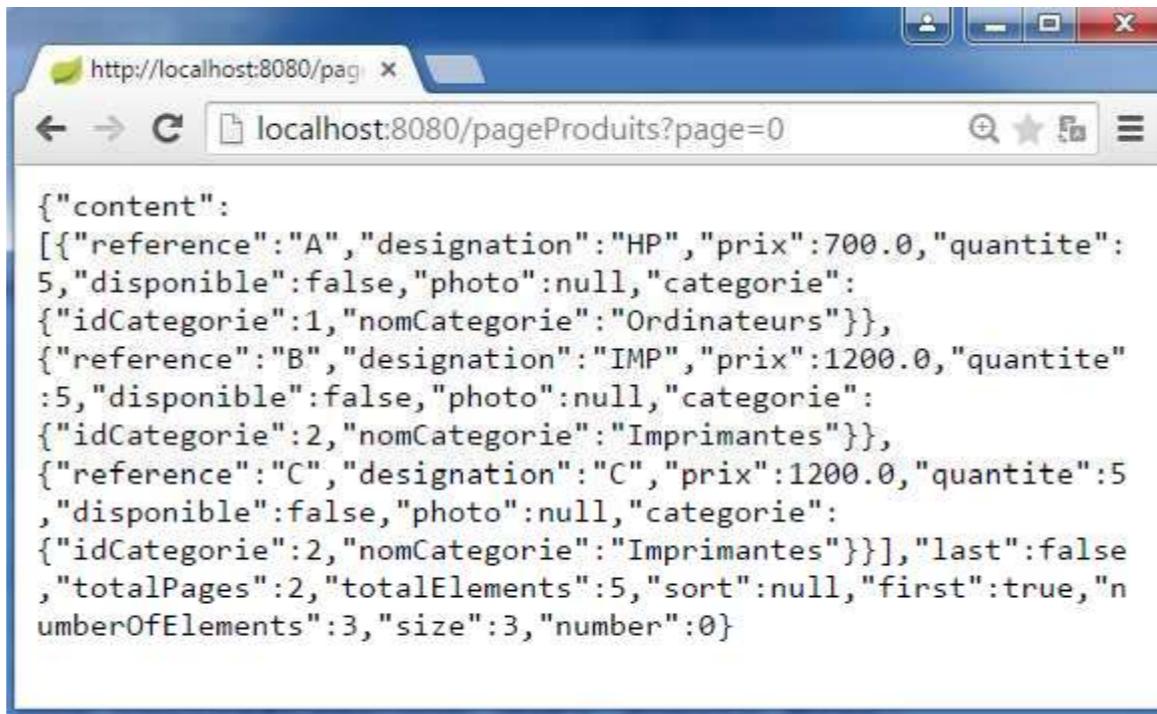


A screenshot of a web browser window titled "localhost:8080/allProduits". The address bar also shows "localhost:8080/allProduits". The content area displays the following JSON array:

```
[{"reference": "A", "designation": "HP", "prix": 700.0, "quantite": 5, "disponible": false, "photo": null, "categorie": {"idCategorie": 1, "nomCategorie": "Ordinateurs"}}, {"reference": "B", "designation": "IMP", "prix": 1200.0, "quantite": 5, "disponible": false, "photo": null, "categorie": {"idCategorie": 2, "nomCategorie": "Imprimantes"}}, {"reference": "C", "designation": "C", "prix": 1200.0, "quantite": 5, "disponible": false, "photo": null, "categorie": {"idCategorie": 2, "nomCategorie": "Imprimantes"}}, {"reference": "D", "designation": "D", "prix": 1200.0, "quantite": 5, "disponible": false, "photo": null, "categorie": {"idCategorie": 2, "nomCategorie": "Imprimantes"}}, {"reference": "E", "designation": "E", "prix": 1200.0, "quantite": 5, "disponible": false, "photo": null, "categorie": {"idCategorie": 1, "nomCategorie": "Ordinateurs"}}]
```

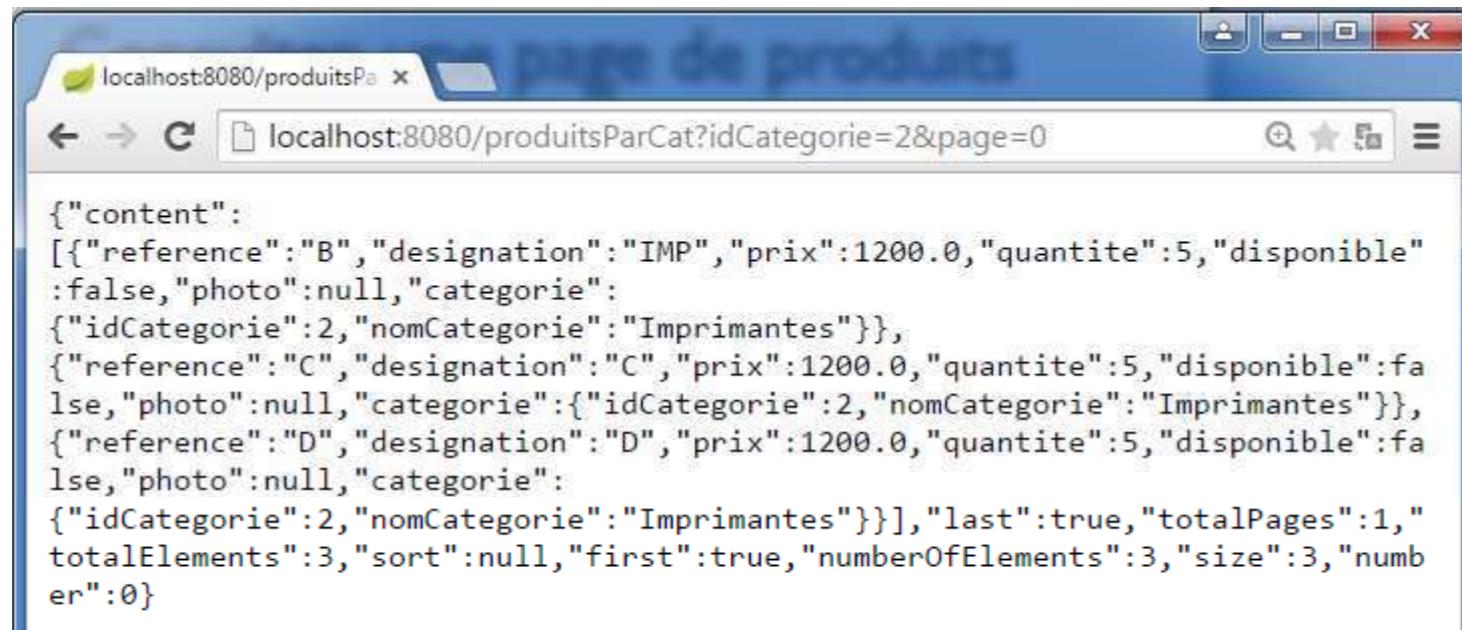
```
@RequestMapping(value="/allProduits")
@ResponseBody
public List<Produit> allProduits(){
    return produitRepository.findAll();
}
```

Consulter une page de produits



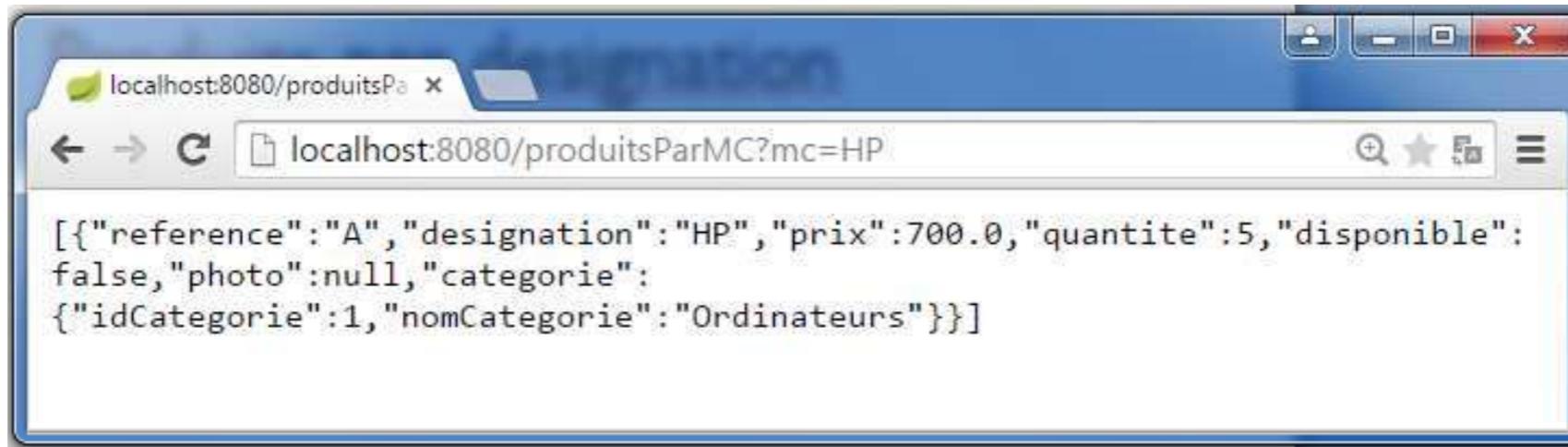
```
@RequestMapping(value="/pageProduits")
@ResponseBody
public Page<Produit> pageProduits(int page){
    return produitRepository.findAll(new PageRequest(page, 3));
}
```

Consulter une page de produits



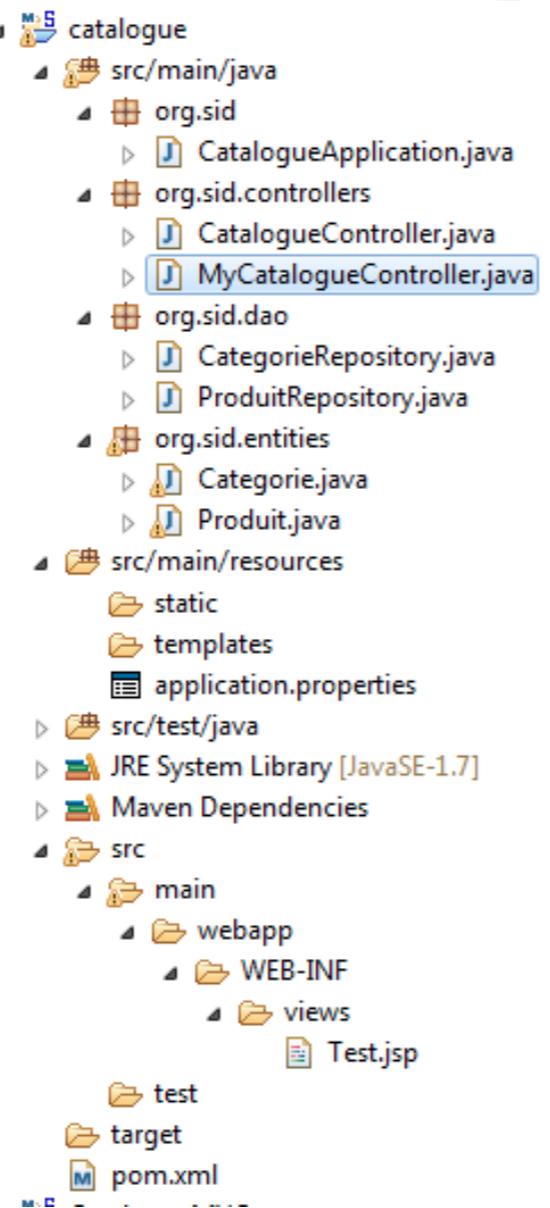
```
@RequestMapping(value="/produitsParCat")
@ResponseBody
public Page<Produit> produitsParCat(Categorie c,int page){
return produitRepository.findByCategorie(c, new PageRequest(page, 3));
}
```

Produits par designation

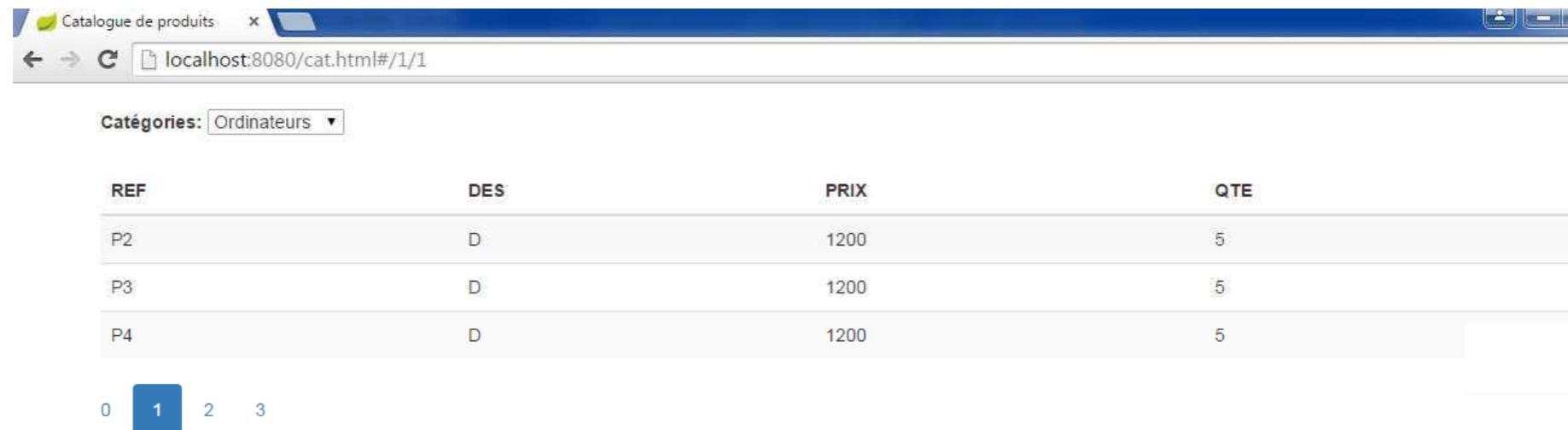


```
@RequestMapping(value="/produitsParMC")
@ResponseBody
public List<Produit> produitsParCat(String mc){
    return produitRepository.findByDesignation(mc);
}
```

Structure du projet



Application web avec HTML, AngularJS et BootStrap côté client

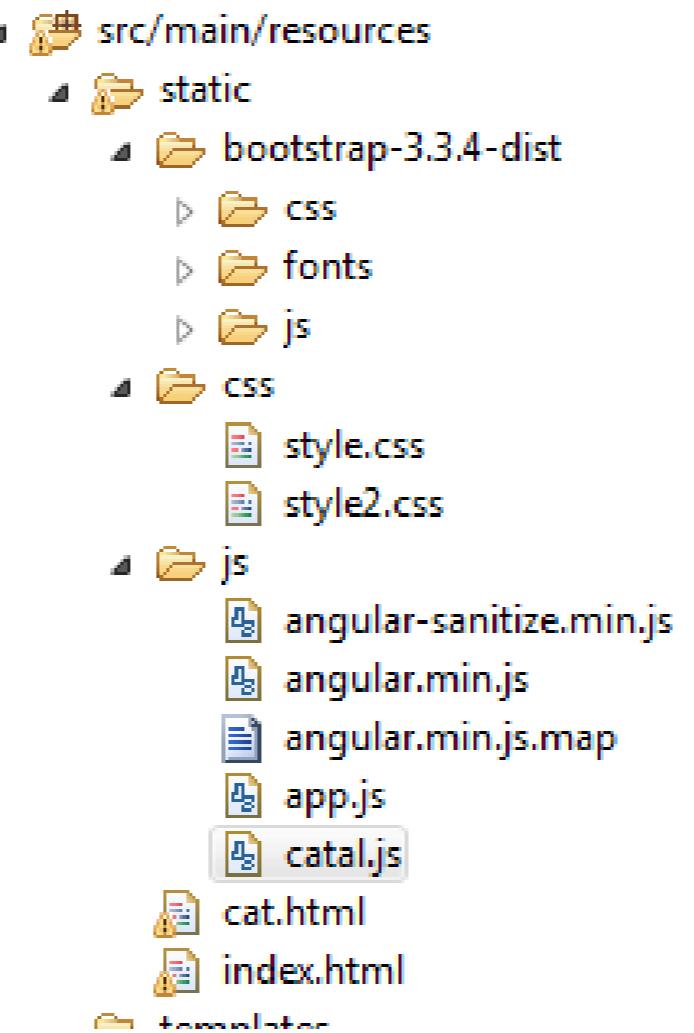


The screenshot shows a web browser window titled "Catalogue de produits". The address bar displays "localhost:8080/cat.html#/1/1". The main content area is a table listing products under the category "Ordinateurs". The table has columns: REF, DES, PRIX, and QTE. There are three rows of data: P2, D, 1200, 5; P3, D, 1200, 5; and P4, D, 1200, 5. Below the table is a pagination control with buttons labeled 0, 1, 2, and 3, where 1 is highlighted.

REF	DES	PRIX	QTE
P2	D	1200	5
P3	D	1200	5
P4	D	1200	5

0 1 2 3

Structure du projet



Catal.js

```
angular.module("catalogue",[])
  .controller("Catcontroller",function($scope,$http,$location){
    $scope.categories=[];
    $scope.produits=[];
    $scope.selectedCategorie=null;
    $scope.pages=new Array();
    $scope.pageCourante=0;
    $scope.chargerCategories=function(){
      $http.get("/allCat")
        .success(function(data){
          $scope.categories=data;
        });
    };
  });
}
```

Catal.js

```
$scope.chargerProduits=function(){
    $http.get("/produitsParCat?page="+$scope.pageCourante+"&idCategorie
        ="+$scope.selectedCategorie)
        .success(function(data){
            $scope.produits=data;
            $scope.pages=new Array(data.totalPages);
        });
};

$scope.chargerCategories();
$scope.gotoURL=function(){
    $scope.pageCourante=0;
    $location.path("/"+$scope.selectedCategorie);
};
$scope.gotoPage=function(page){
    $scope.pageCourante=page;
    $location.path("/"+$scope.selectedCategorie+"/"+page);
};
```

Catal.js

```
$scope.$watch(  
    function(){return $location.path();},  
    function(newPath){  
        //console.log(newPath);  
        var tabPath=newPath.split("/");  
        $scope.pageCourante=0;  
        if(tabPath.length==2){  
            $scope.selectedCategorie=tabPath[1];  
            $scope.chargerProduits();  
        }  
        else if(tabPath.length==3){  
            $scope.selectedCategorie=tabPath[1];  
            $scope.pageCourante=tabPath[2];  
            $scope.chargerProduits();  
        };  
    });  
});
```

Cat.html

```
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>Catalogue de produits</title>
<link rel="stylesheet" href="bootstrap-3.3.4-dist/css/bootstrap.min.css"
/>
<link rel="stylesheet"
href="bootstrap-3.3.4-dist/css/bootstrap-theme.min.css" />
<link rel="stylesheet" href="css/style2.css" />
</head>
<body ng-app="catalogue" ng-controller="Catcontroller">
<div class="container spacer">
<form>
<label>Catégories:</label> <select ng-
model="selectedCategorie" ng-change="gotoURL()">
<option ng-repeat="cat in categories" value="{{cat.idCategorie}}">
{{cat.nomCategorie}}</option>
</select>
</form>
</div>
```

Cat.html

```
<div class="container">
  <table class="table table-striped table-hover spacer">
    <thead>
      <tr>
        <th>REF</th>
        <th>DES</th>
        <th>PRIX</th>
        <th>QTE</th>
      </tr>
    </thead>
    <tbody>
      <tr ng-repeat="p in produits.content">
        <td>{{p.reference}}</td>
        <td>{{p.designation}}</td>
        <td>{{p.prix}}</td>
        <td>{{p.quantite}}</td>
      </tr>
    </tbody>
  </table>
</div>
```

Cat.html

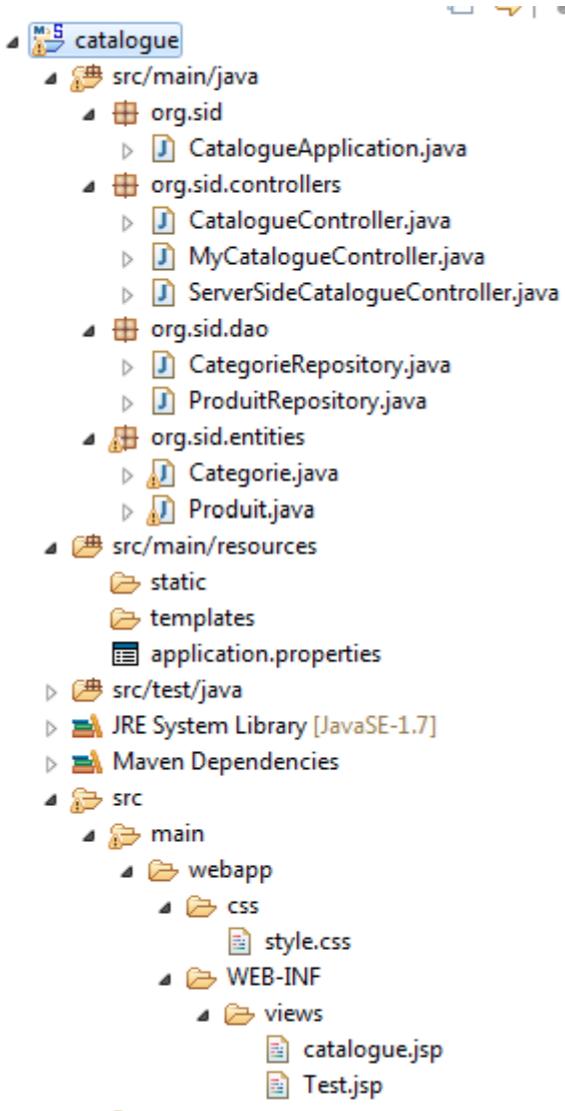
```
<ul ng-show="produits.totalPages>1" class="container nav nav-pills">
  <li class="clickable" ng-repeat="p in pages track by $index" ng-
  class="{active:$index==pageCourante}" >
    <a ng-click="gotoPage($index)">{{$index}}</a>
  </li>
</ul>
<script type="text/javascript" src="js/angular.min.js"></script>
<script type="text/javascript" src="js/catal.js"></script>
</body>
</html>
```



MVC SERVER SIDE

- JSP
- JSTL

Structure du projet



Catalogue.jsp

Catégorie: Ordinateurs OK

REF	DES	PRIX	QTE	DISPO	PHOTO
A	HP	700.0	5	false	
E	E	1200.0	5	false	
P1	D	1200.0	5	false	

Page 0 [Page 1](#) [Page 2](#) [Page 3](#)

Le contrôleur

```
package org.sid.controllers;

import org.sid.dao.CategorieRepository; import org.sid.dao.ProduceRepository;
import org.sid.entities.Categorie; import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.data.domain.PageRequest; import org.springframework.stereotype.Controller;
import org.springframework.ui.Model; import org.springframework.web.bind.annotation.ModelAttribute;
import org.springframework.web.bind.annotation.RequestMapping;

@Controller
public class ServerSideCatalogueController {
    @Autowired
    private CategorieRepository categorieRepository;
    @Autowired
    private ProduceRepository produitRepository;
```

Le contrôleur

```
@RequestMapping("/catalogue")
public String catalogue(Model model){
    model.addAttribute("categories", categorieRepository.findAll());
    model.addAttribute("categorie", new Categorie());
    return "catalogue";
}

@RequestMapping("/produits")
public String produits(@ModelAttribute Categorie categorie, Model
    model, int page){
    model.addAttribute("categories", categorieRepository.findAll());
    model.addAttribute("pageProduits", produitRepository.findByCategorie
        (categorie, new PageRequest(page, 3)));
    return "catalogue";
}

}
```

catalogue.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>
<title>Catalogue de produits</title>
<link rel="stylesheet" type="text/css"
      href="<%=request.getContextPath()%>/css/style.css"/>
</head>
<body>
<div>
    <f:form modelAttribute="categorie" action="produits?page=0">
        <table>
            <tr>
                <td>Catégories:</td>
                <td><f:select path="idCategorie" items="${categories}"
itemLabel="nomCategorie"
itemValue="idCategorie"></f:select></td>
                <td><input type="submit" value="OK"/></td>
            </tr>
        </table>
    </f:form>
</div>
```

catalogue.jsp

```
<div>
    <div>
        <table class="table1">
            <tr>
                <th>REF</th><th>DES</th><th>PRIX</th><th>QTE</th> <th>DISPO</th>
            <th>PHOTO</th>
            </tr>
            <c:forEach items="${pageProduits.getContent()}" var="p">
                <tr>
                    <td>${p.designation }</td>
                    <td>${p.prix }</td>
                    <td>${p.quantite }</td>
                    <td>${p.disponible }</td>
                    <td>${p.photo }</td>
                </tr>
            </c:forEach>
        </table>
    </div>
```

catalogue.jsp

```
<div>
    <c:if test="${ not empty pageProduits and
pageProduits.getTotalPages()>=2}">
        <c:forEach begin="0" end="${pageProduits.getTotalPages() -1}"
var="p">
            <span class="autrePage">
                <c:if test="${pageProduits.getNumber()!=p }">
                    <a
                        href="produits?page=${p}&idCategorie=${categorie.idCategorie}>Page ${p
}=</a>
                </c:if>
                <c:if test="${pageProduits.getNumber()==p }">
                    Page ${p }
                </c:if>
            </span>
        </c:forEach>
    </c:if>
</div>
</div>
</body>
</html>
```

style.css

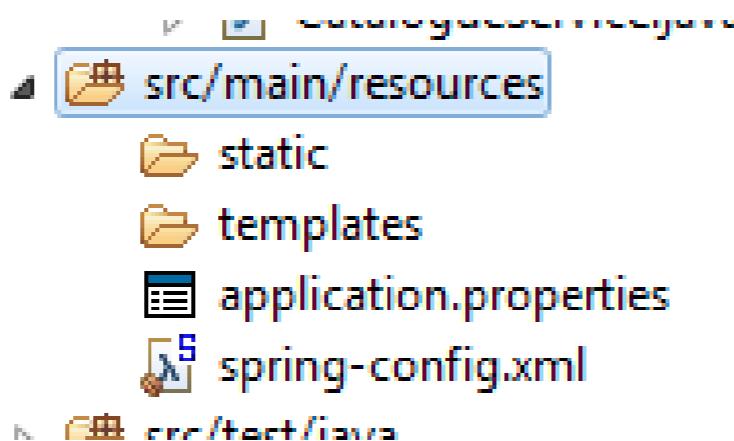
```
body{  
    font-family: cursive;  
    font-size: 12px;  
}  
  
div{  
    border: 1px dotted gray;  
    padding: 5px;  
    margin: 5px;  
}  
  
.table1 th{  
    border: 1px dotted gray;  
    padding: 5px;  
    margin: 5px;  
    background: pink;  
}  
  
.table1 td{  
    border: 1px dotted gray;  
    padding: 5px;  
    margin: 5px;  
    background: white;  
}
```

Web Service SOAP

```
package org.sid.services;
import java.util.List; import javax.jws.WebMethod; import
javax.jws.WebParam; import javax.jws.WebService; import
org.sid.dao.CategorieRepository;
import org.sid.dao.ProduitRepository; import org.sid.entities.Categorie;
import org.sid.entities.Produit; import
org.springframework.beans.factory.annotation.Autowired; import
org.springframework.stereotype.Component;
@Component
@WebService
public class CatalogueService {
@Autowired
    private ProduitRepository produitRepository;
@Autowired
    private CategorieRepository categorieRepository;
@WebMethod
    public List<Categorie> allCategories(){
        return categorieRepository.findAll();
    }
@WebMethod
    public List<Produit> produits(@WebParam(name="idCat")Long idCat){
        Categorie c=new Categorie(); c.setIdCategorie(idCat);
        return produitRepository.findByCategorie(c, null).getContent();
    }
}
```

Déployer le web service avec spring

```
<?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 class="org.springframework.remoting.jaxws.SimpleJaxWsServiceExporter">
        <property name="baseAddress"
            value="http://0.0.0.0:8787/services/"></property>
    </bean>
</beans>
```



L'application Spring Boot

```
package org.sid;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.EnableAutoConfiguration;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.context.annotation.ComponentScan;
import org.springframework.context.annotation.ImportResource;
@SpringBootApplication
@ComponentScan
@ImportResource("classpath:spring-config.xml")
@EnableAutoConfiguration
public class CatalogueApplication {
    public static void main(String[] args) {
        SpringApplication.run(CatalogueApplication.class, args);
    }
}
```

Tester le web service



This XML file does not appear to have any style information associated with it. The document tree is shown below.

```
<!-- Published by JAX-WS RI at http://jax-ws.dev.java.net. RI's version is JAX-WS RI 2.2.4-b01. -->
<!-- Generated by JAX-WS RI at http://jax-ws.dev.java.net. RI's version is JAX-WS RI 2.2.4-b01. -->
<?xml version="1.0" encoding="UTF-8"?>
<definitions xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd"
    xmlns:wsp="http://www.w3.org/ns/ws-policy" xmlns:wsp1_2="http://schemas.xmlsoap.org/ws/2004/09/policy"
    xmlns:wsam="http://www.w3.org/2007/05/addressing/metadata" xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
    xmlns:tns="http://services.sid.org/" xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.xmlsoap.org/wsdl/"
    targetNamespace="http://services.sid.org/" name="CatalogueServiceService">
    <types>
        <xsd:schema>
            <xsd:import namespace="http://services.sid.org/" schemaLocation="http://localhost:8787/services/?xsd=1"/>
        </xsd:schema>
    </types>
    <message name="produits">...</message>
    <message name="produitsResponse">...</message>
    <message name="allCategories">...</message>
    <message name="allCategoriesResponse">...</message>
    <portType name="CatalogueService">...</portType>
    <binding name="CatalogueServicePortBinding" type="tns:CatalogueService">...</binding>
    <service name="CatalogueServiceService">
        <port name="CatalogueServicePort" binding="tns:CatalogueServicePortBinding">
            <soap:address location="http://localhost:8787/services/">
        </port>
    </service>
</definitions>
```

Consulter les catégories

```
<SOAP-ENV:Envelope xmlns:SOAP-  
ENV="http://schemas.xmlsoap.org/soap/envelope/">  
  <SOAP-ENV:Header/>  
  <SOAP-ENV:Body>  
    <allCategories xmlns="http://services.sid.org/" />  
  </SOAP-ENV:Body>  
</SOAP-ENV:Envelope>  
  
<?xml version="1.0" ?>  
  <S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">  
    <S:Body>  
      <ns2:allCategoriesResponse xmlns:ns2="http://services.sid.org/">  
        <return>  
          <idCategorie>1</idCategorie>  
          <nomCategorie>Ordinateurs</nomCategorie>  
        </return>  
        <return>  
          <idCategorie>2</idCategorie>  
          <nomCategorie>Imprimantes</nomCategorie>  
        </return>  
      </ns2:allCategoriesResponse>  
    </S:Body>  
</S:Envelope>
```

Consulter les produits d'une catégorie

```
<SO AP-ENV:Envelope xmlns:SO AP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
  <SOAP-ENV:Header/>
  <SOAP-ENV:Body>
    <produits xmlns="http://services.sid.org/">
      <idCat xmlns="">2</idCat>
    </produits>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>

<?xml version="1.0"?>
<S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
  <S:Body>
    <ns2:produitsResponse xmlns:ns2="http://services.sid.org/">
      <return>
        <categorie>
          <idCategorie>2</idCategorie> <nomCategorie>Imprimantes</nomCategorie>
        </categorie>
        <designation>IMP</designation> <disponible>false</disponible>
        <prix>1200.0</prix> <quantite>5</quantite>
        <reference>B</reference>
      </return>
      <return>
        <categorie>
          <idCategorie>2</idCategorie> <nomCategorie>Imprimantes</nomCategorie>
        </categorie>
        <designation>C</designation> <disponible>false</disponible>
        <prix>1200.0</prix> <quantite>5</quantite>
        <reference>C</reference>
      </return>
    </ns2:produitsResponse>
  </S:Body>
</S:Envelope>
```

SERVICE RMI

Interface Remote

```
package org.sid.rmi;

import java.rmi.Remote;
import java.rmi.RemoteException;
import java.util.List;
import org.sid.entities.Categorie;
import org.sid.entities.Produit;

public interface CatalogueRemote extends Remote {
    public List<Categorie> listCategories()throws RemoteException;
    public List<Produit> produitsParCategories(Long idCat) throws RemoteException;
}
```

Implémentation du service RMI

```
package org.sid.rmi;

import java.rmi.RemoteException; import java.util.List;
import org.sid.dao.CategorieRepository; import org.sid.dao.ProduitRepository;
import org.sid.entities.Categorie; import org.sid.entities.Produit;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Component;

@Component
public class CatalogueRmiService implements CatalogueRemote {
    @Autowired private ProduitRepository produitRepository;
    @Autowired private CategorieRepository categorieRepository;
    @Override
    public List<Categorie> listCategories() throws RemoteException {
        return categorieRepository.findAll();
    }
    @Override
    public List<Produit> produitsParCategories(Long idCat)
        throws RemoteException { return produitRepository.findByCategorie(new
        Categorie(idCat), null).getContent(); }
}
```

Déploiement du service RMI

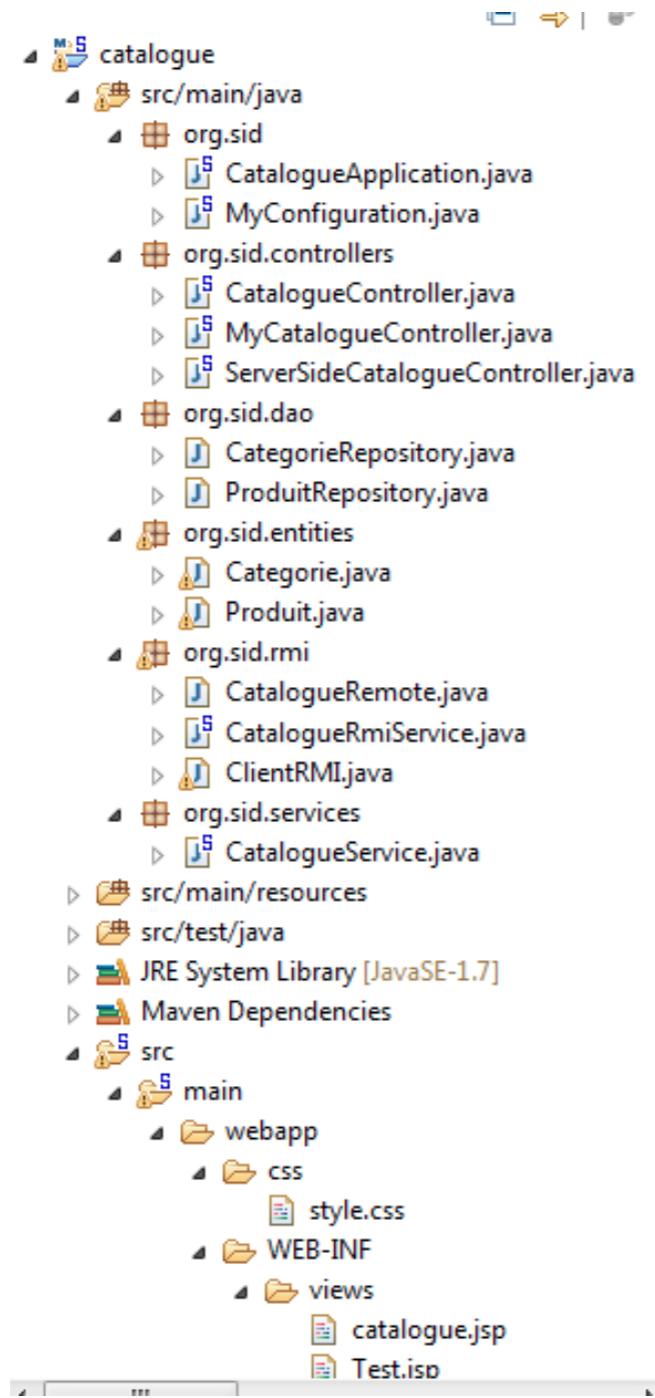
```
package org.sid;
import org.sid.rmi.CatalogueRemote; import
org.springframework.beans.factory.annotation.Autowired; import
org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.remoting.jaxws.SimpleJaxWsServiceExporter;
import org.springframework.remoting.rmi.RmiServiceExporter;
@Configuration
public class MyConfiguration {
    @Autowired
    CatalogueRemote catalogueRemote;
    @Bean
    public SimpleJaxWsServiceExporter jaxWSExporter(){
        return new SimpleJaxWsServiceExporter();
    }
    @Bean
    public RmiServiceExporter rmiExporter(){
        RmiServiceExporter rmiExporter=new RmiServiceExporter();
        rmiExporter.setService(catalogueRemote);
        rmiExporter.setServiceInterface(CatalogueRemote.class);
        rmiExporter.setServiceName("CAT");
    }
    return rmiExporter;
}
```

Application Spring Boot

```
package org.sid;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.EnableAutoConfiguration;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.context.ApplicationContext;
import org.springframework.context.annotation.ComponentScan;
@SpringBootApplication
@ComponentScan
//@ImportResource("classpath:spring-config.xml")
@EnableAutoConfiguration
public class CatalogueApplication {
    public static void main(String[] args) {
        SpringApplication.run(CatalogueApplication.class, args);
    }
}
```

Client RMI

```
package org.sid.rmi;
import java.rmi.Naming; import java.util.List;
import org.sid.entities.Categorie;
public class ClientRMI {
    public static void main(String[] args) {
        try {
            CatalogueRemote stub=(CatalogueRemote)
                Naming.lookup("rmi://localhost:1099/CAT");
            for(Categorie c:cats){stub.listCategories();
                System.out.println(c.getNomCategorie());
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```



Structure finale du projet