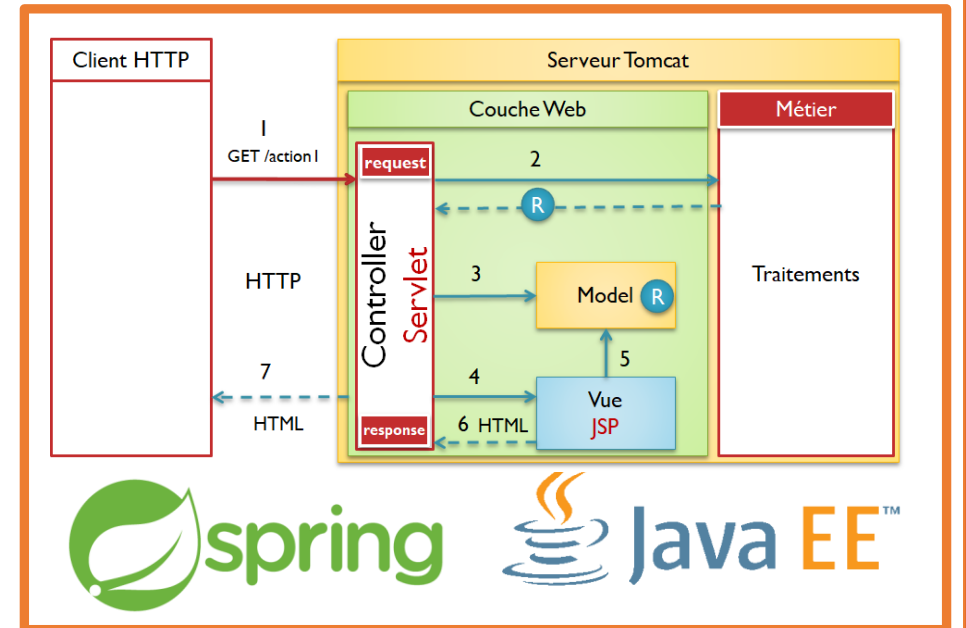
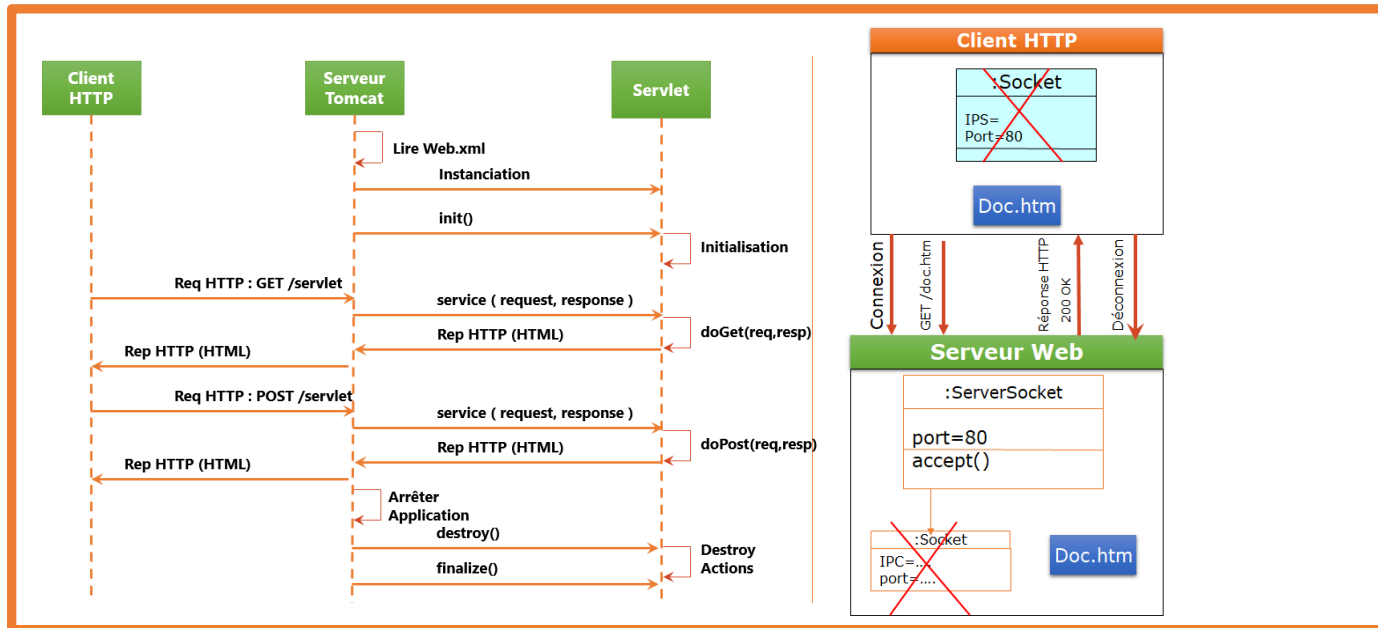


Développement Web JEE : http, Servlet, JSP, JSTL, Spring MVC



Mohamed Youssfi

Laboratoire Signaux Systèmes Distribués et Intelligence Artificielle (SSDIA)

ENSET, Université Hassan II Casablanca, Maroc

Email : med@youssfi.net

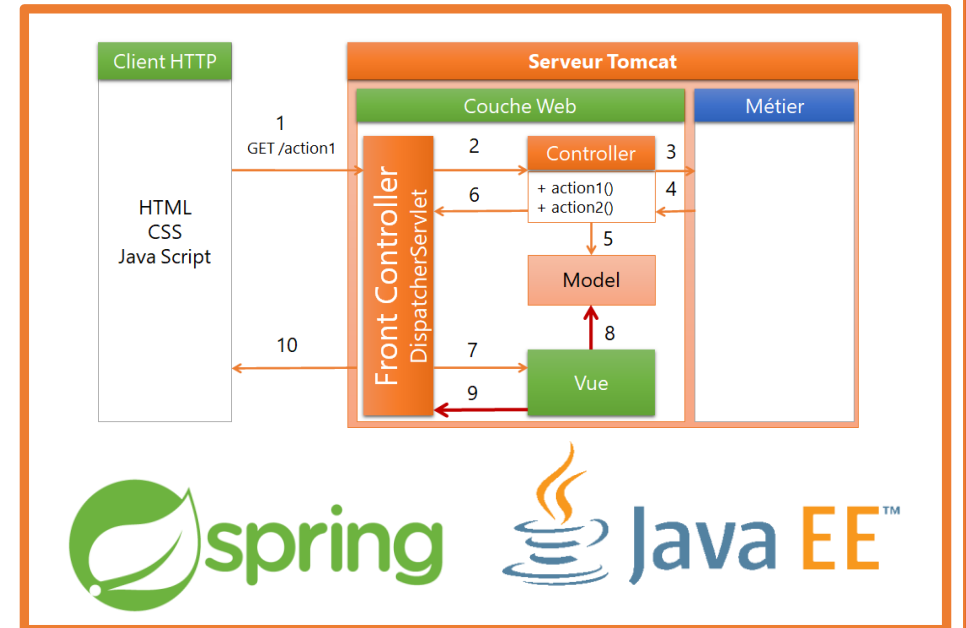
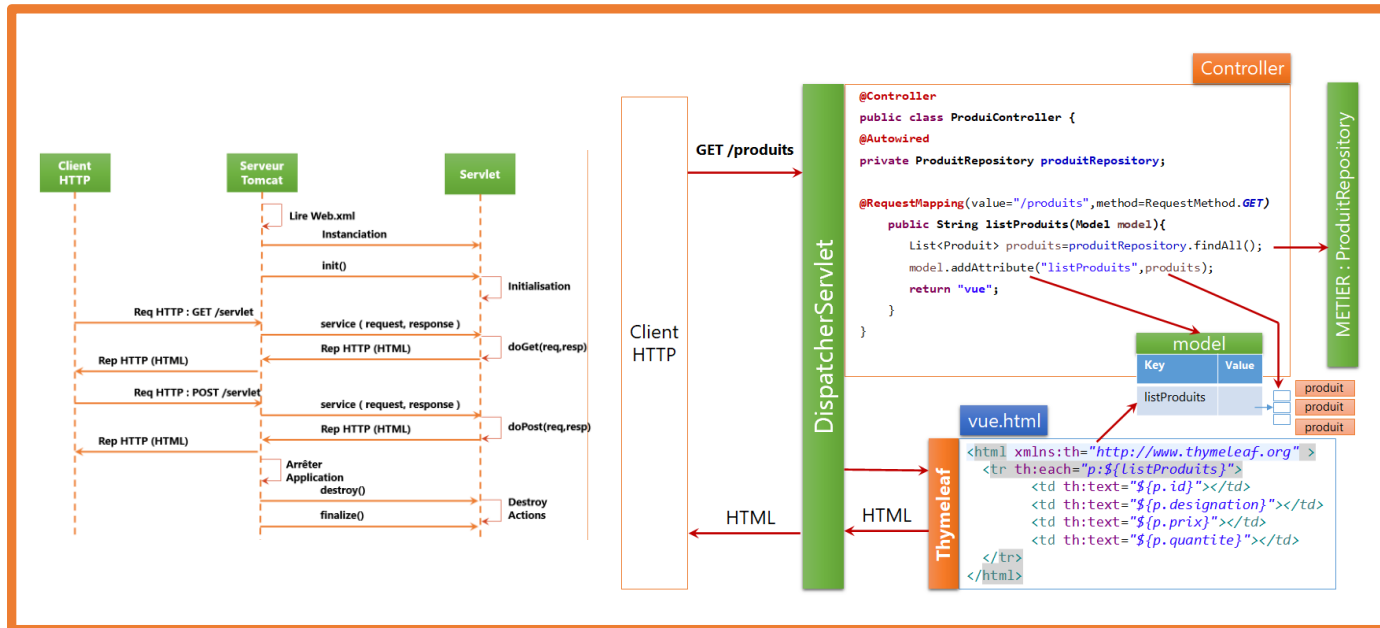
Supports de cours : <http://fr.slideshare.net/mohamedyoussfi9>

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

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



Développement Web JEE : Spring MVC Server Side



Mohamed Youssfi

Laboratoire Signaux Systèmes Distribués et Intelligence Artificielle (SSDIA)

ENSET, Université Hassan II Casablanca, Maroc

Email : med@youssfi.net

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

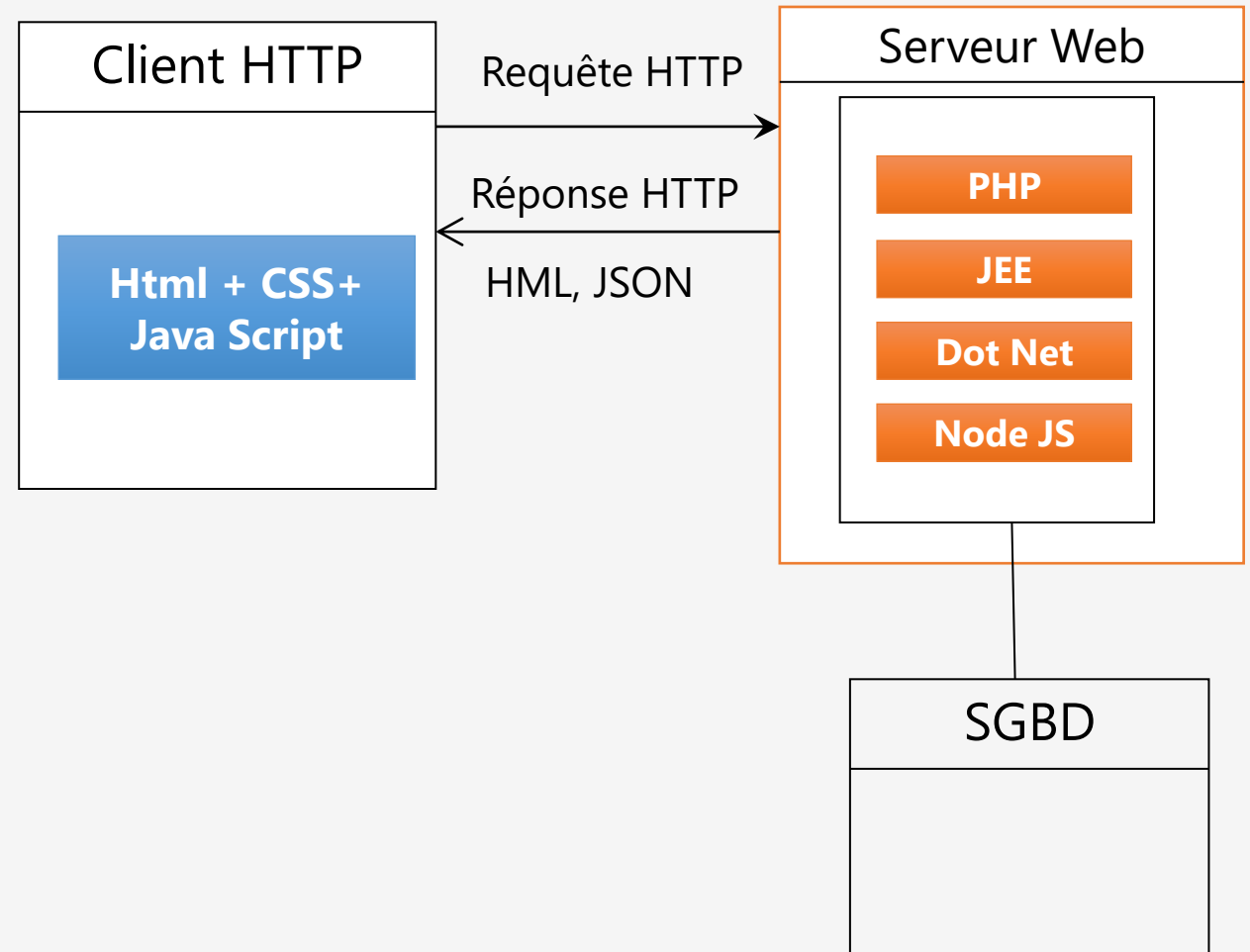
Chaîne vidéo : <http://youtube.com/mohamedYoussfi>

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



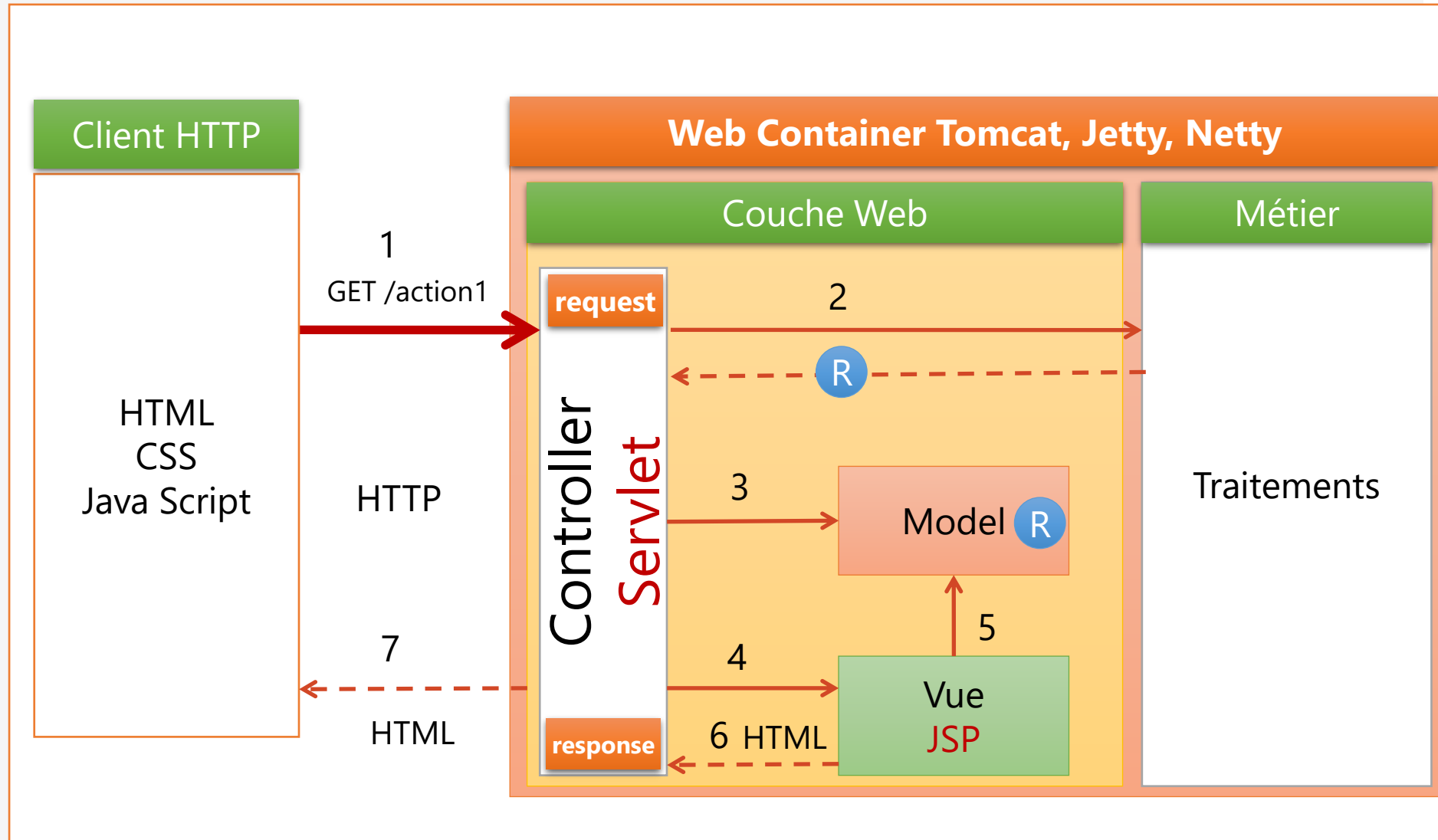
Architecture Web

- Un client web (Browser) communique avec le serveur web (Apache) en utilisant le protocole HTTP
- Une application web se compose de deux parties:
 - La partie **Backend** : S'occupe des traitements effectués coté serveur :
 - Technologies utilisées : PHP, JEE, .Net, Node JS
 - La partie **Frontend** : S'occupe de la présentations des IHM coté Client :
 - Langages utilisés : HTML, CSS, Java Script
- La communication entre la partie Frontend et la partie backend se fait en utilisant le protocole HTTP



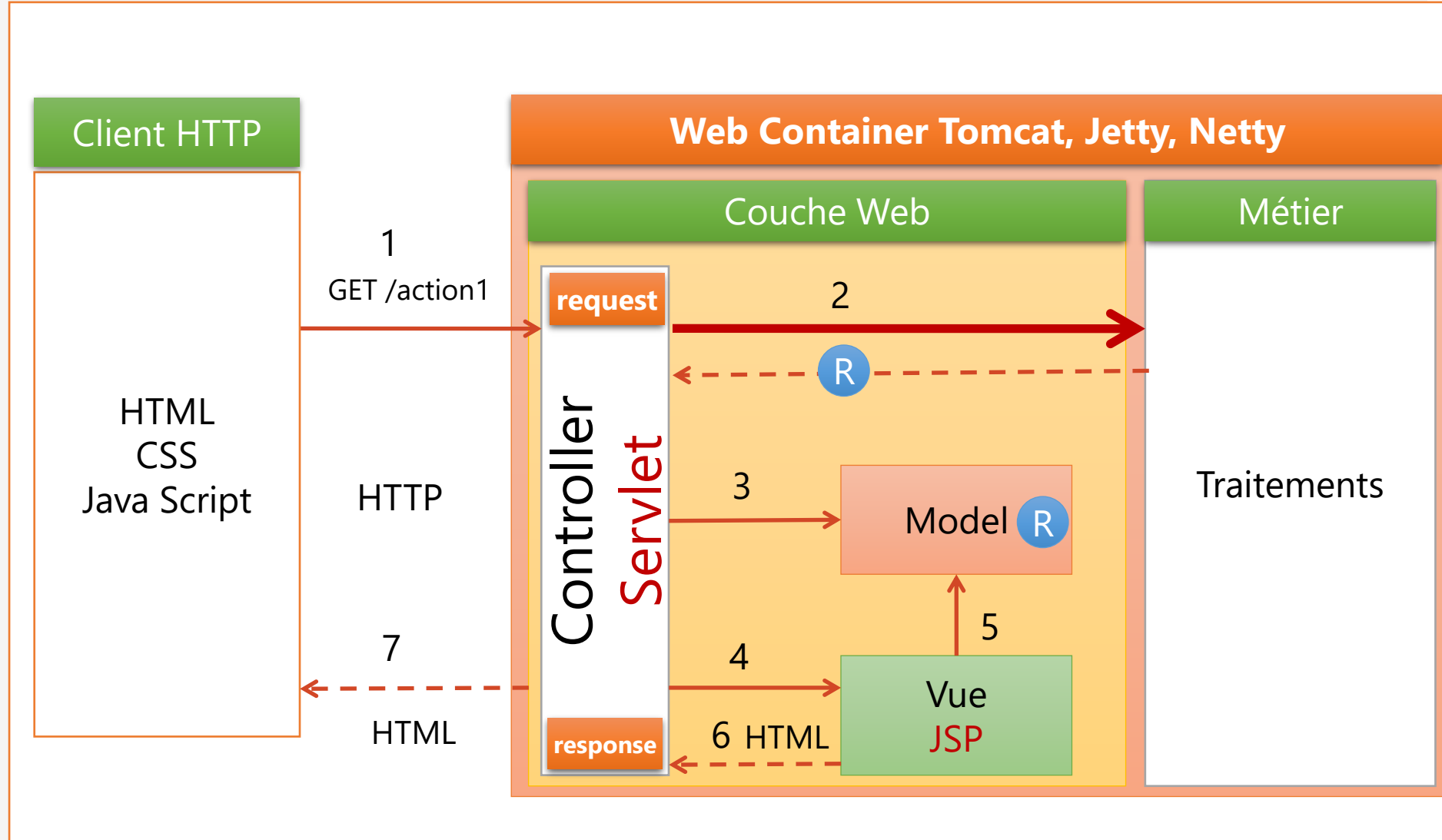
Architecture Web JEE : HTTP, Servlet, JSP, MVC

1 – Le client envoie une requête HTTP de type GET ou POST vers le contrôleur représenté par un composant Web JEE : **SERVLET**. Pour lire les données de la requête HTTP le contrôleur utilise l'objet **request** de type **HttpServletRequest**



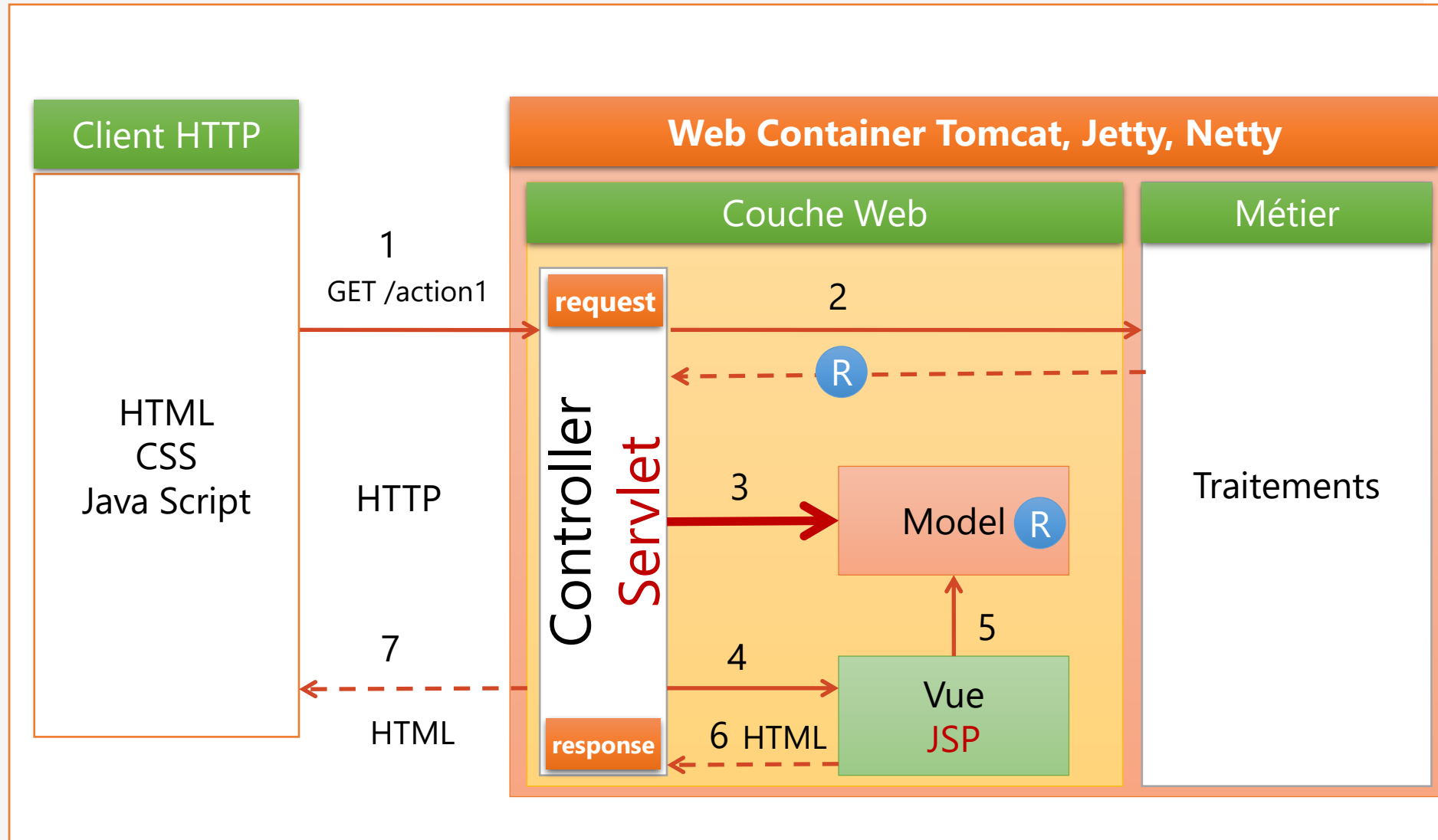
Architecture Web JEE

2 – Le contrôleur fait appel à la couche métier pour effectuer les traitements et récupère les résultats R



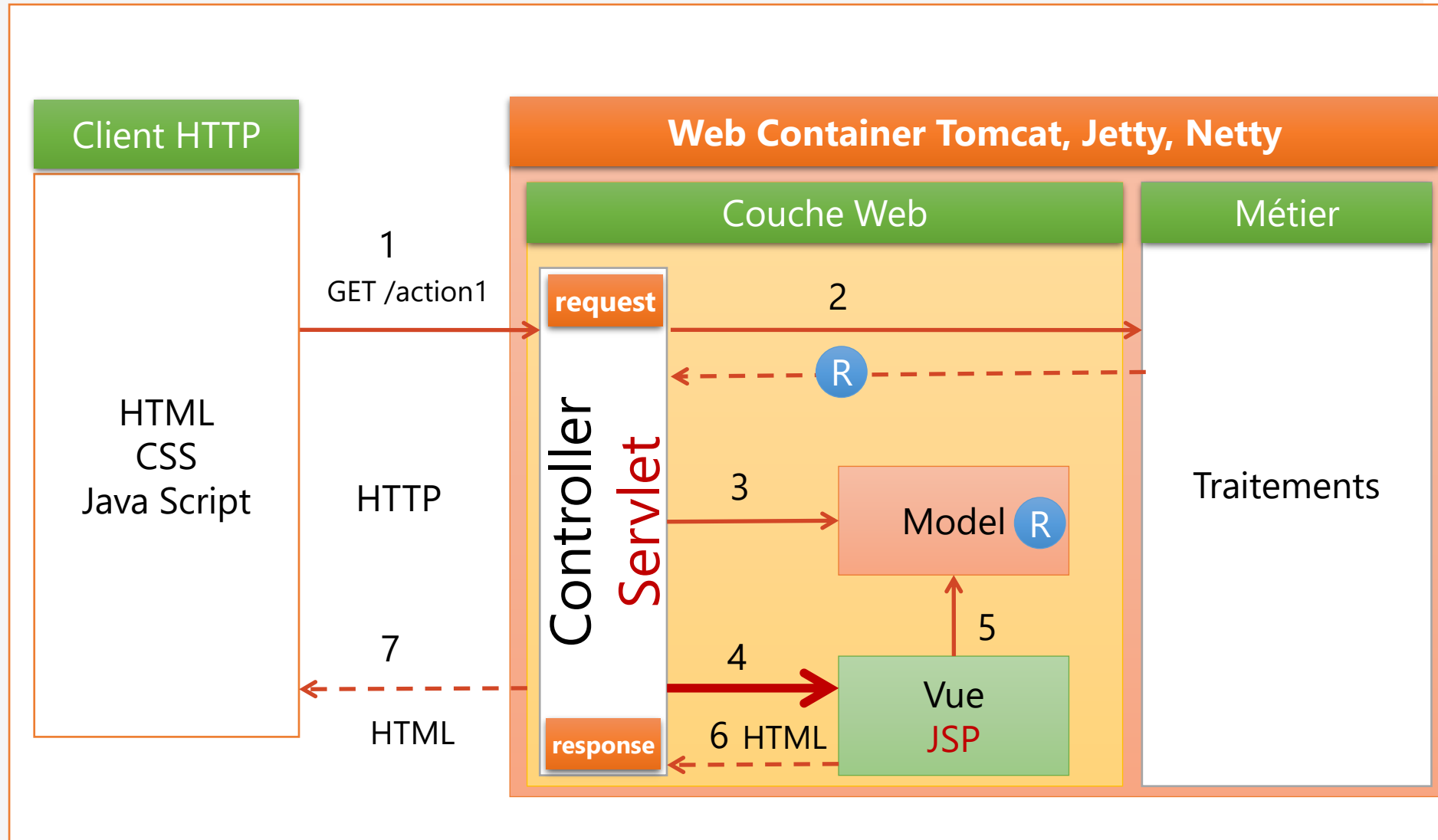
Architecture Web JEE

3 – Le contrôleur Stocke le résultat R dans le modèle M. Le modèle est généralement un objet qui permet de stocker toutes les données qui seront affichées dans la vue. Généralement, le contrôleur stocke le modèle dans l'objet request ou session.



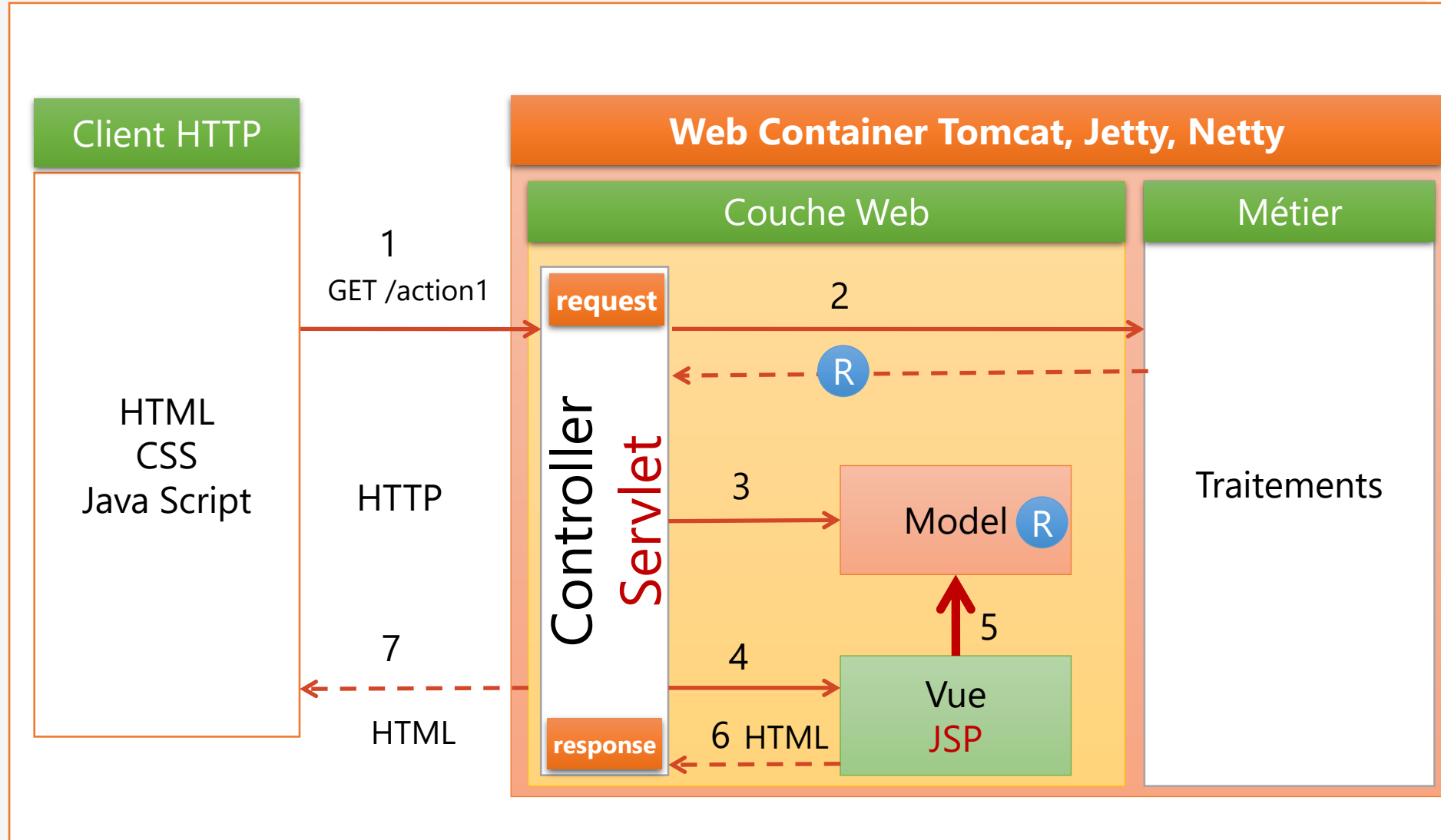
Architecture Web JEE

4 – Le contrôleur fait appel à la vue JSP (Java Server Pages) en lui transmettant les mêmes objets request et response. Cette opération s'appelle Forwarding.



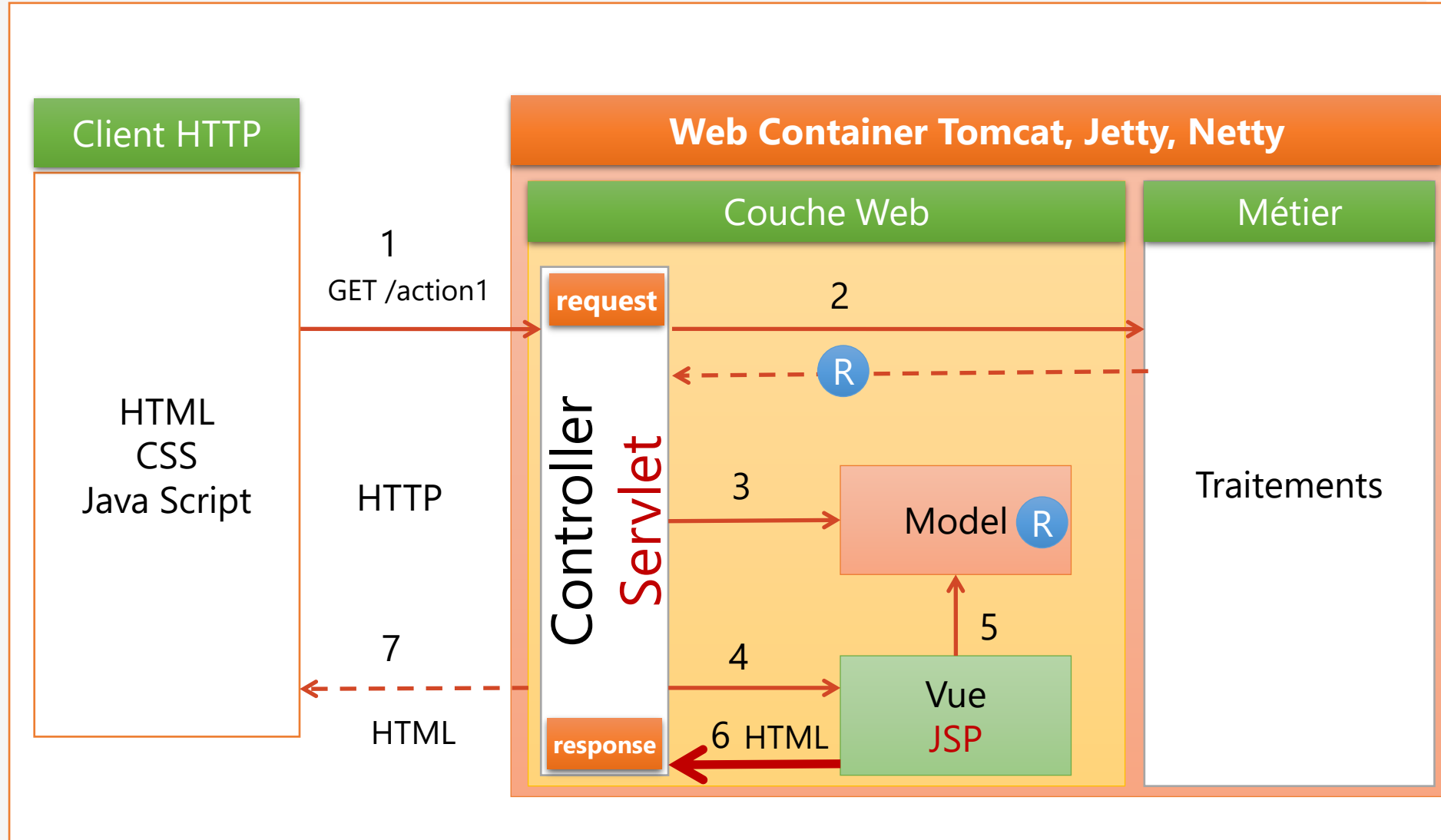
Architecture Web JEE

5 – La vue JSP récupère le résultat à partir du modèle. La vue retrouve le modèle dans l'objet request ou session.



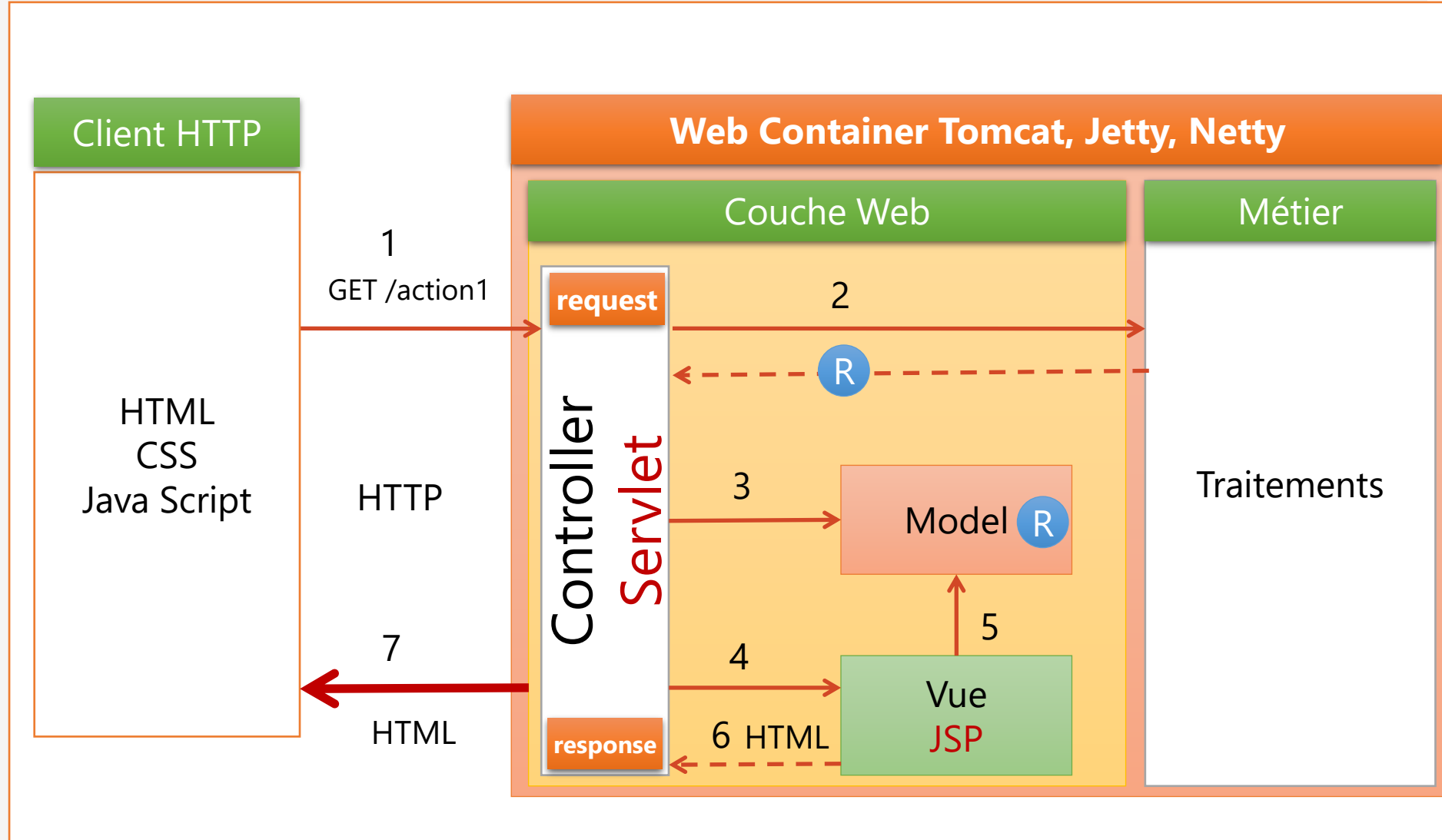
Architecture Web JEE

6 – Le vue JSP génère dynamiquement une page HTML qui contient les résultats du modèle en utilisant l'objet response.



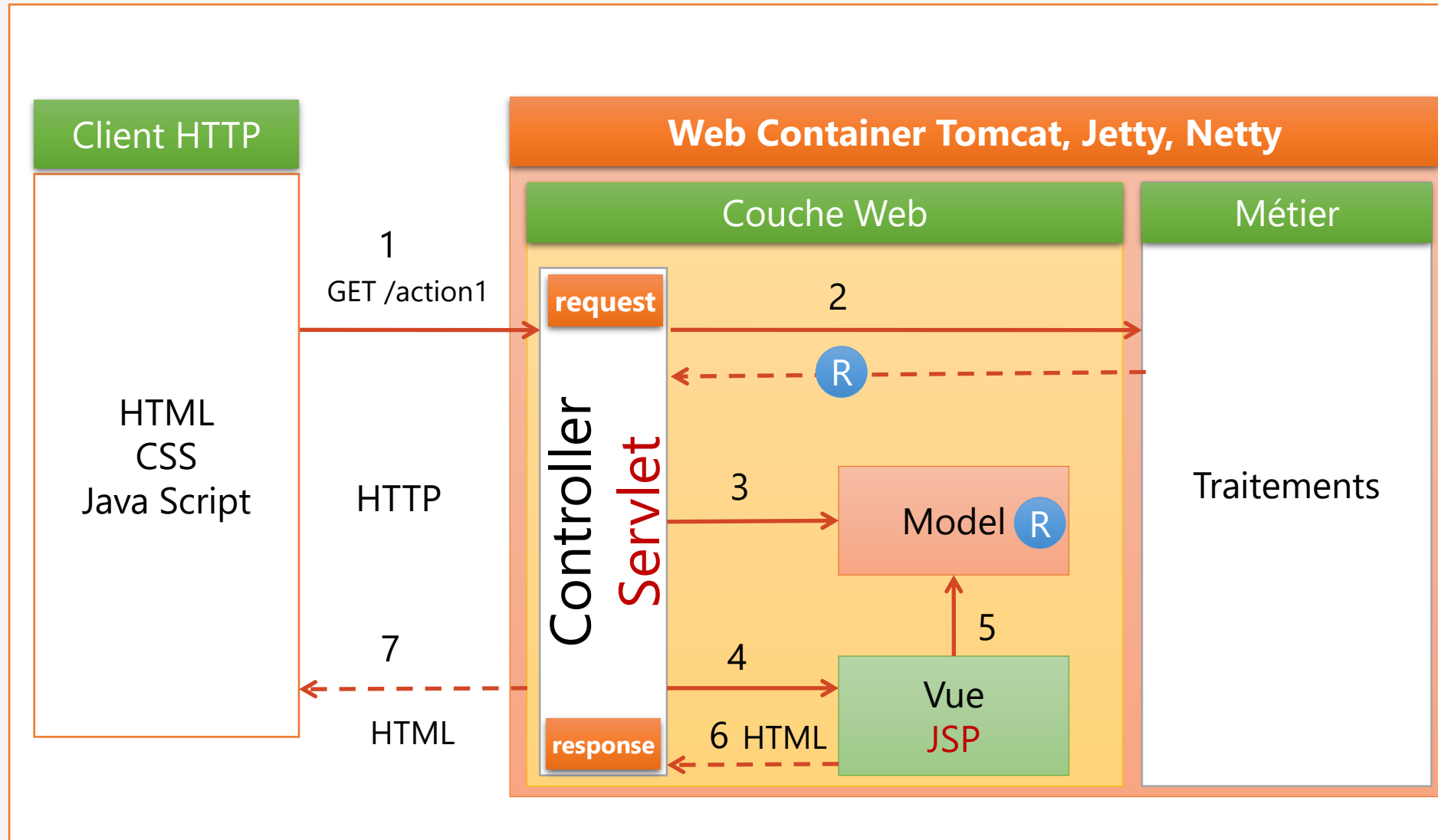
Architecture Web JEE

7 – La page HTML générée est envoyée dans le corps de la réponse HTTP.



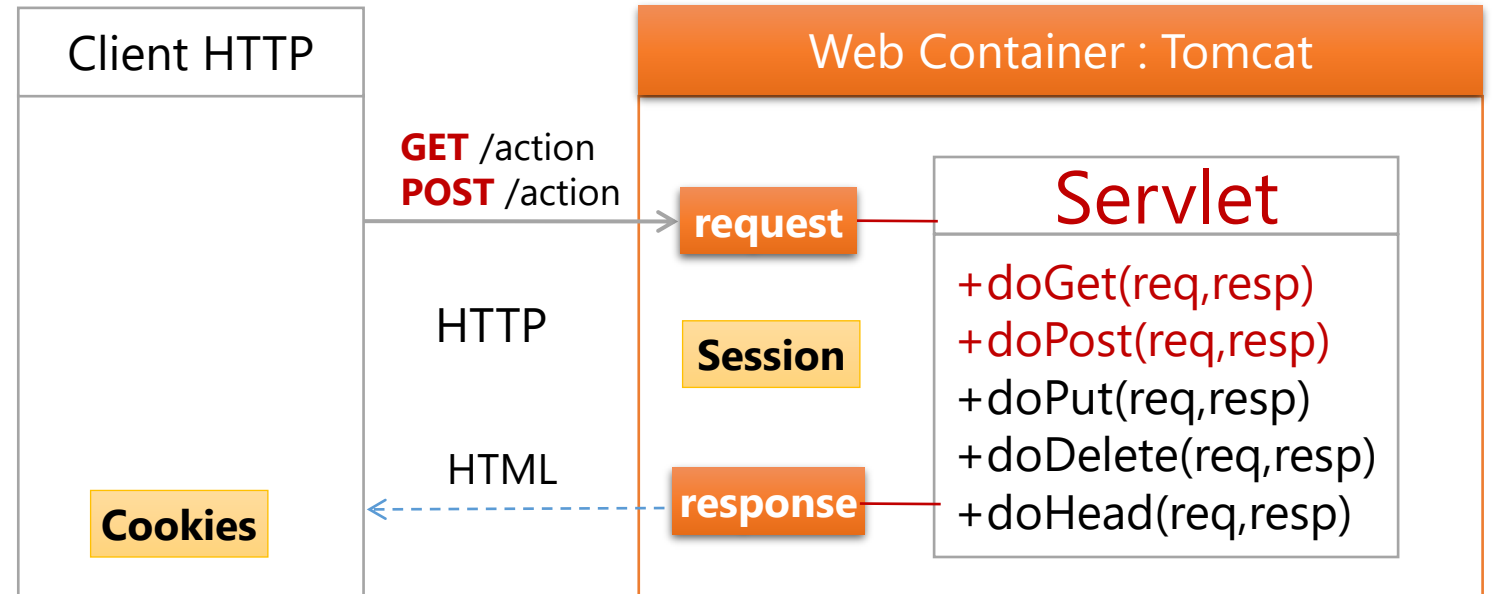
Architecture Web JEE

8 – Le Browser Web affiche le rendu de la page HTML reçue.



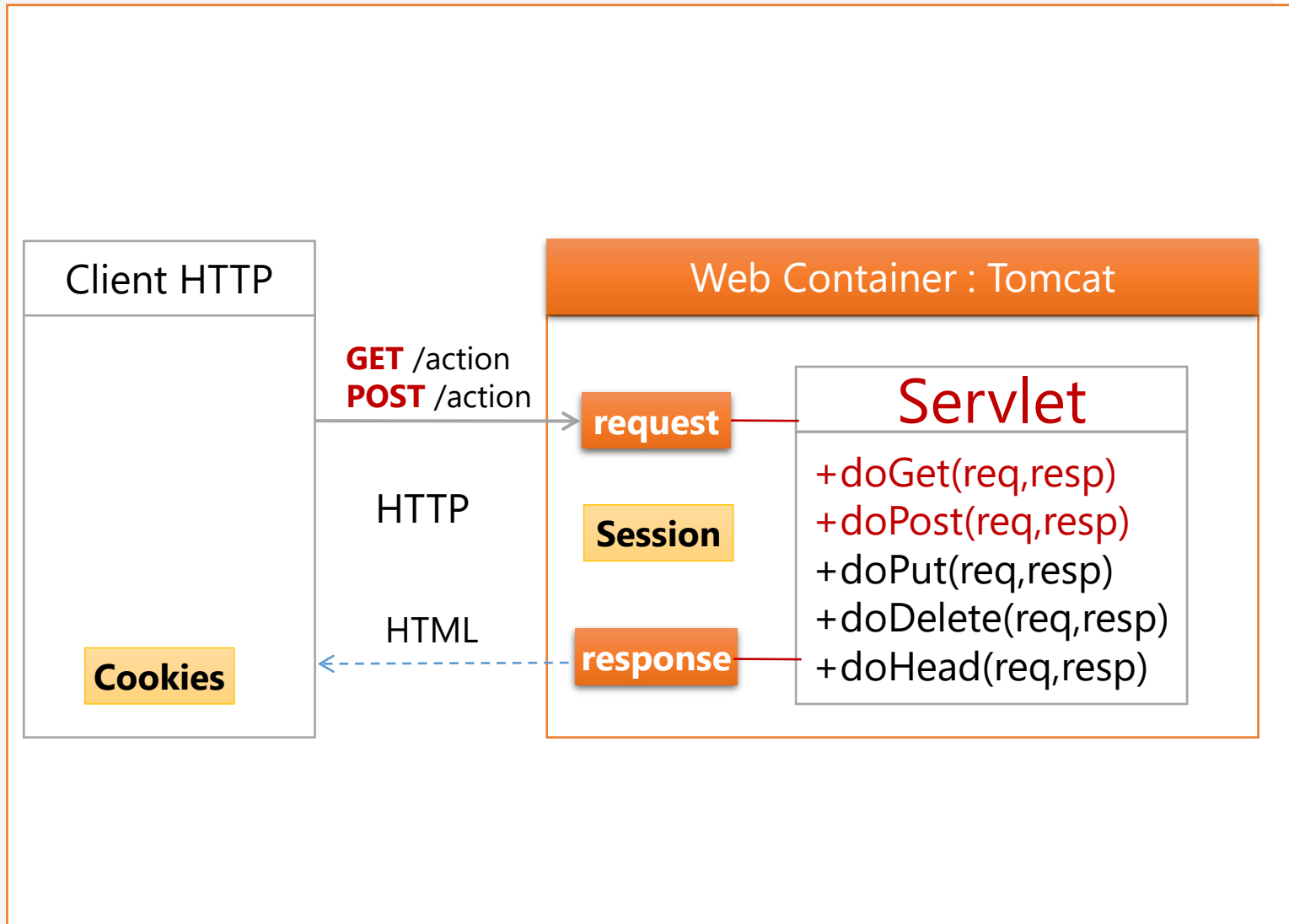
Servlet

- Un Servlet est un composant Web JEE qui permet d'effectuer des traitement du coté du serveur suite à une requête HTTP et envoyer une réponse HTTP.
- Une Servlet est classe Java qui hérite de HttpServlet et qui redéfinit des méthodes comme doGet, doPost, doPut, doDelete, doHead. Et d'autres méthodes qui définissent son cycle de vie.
- La méthode doX est exécutée si une requête HTTP est envoyé par un client http avec la méthode X

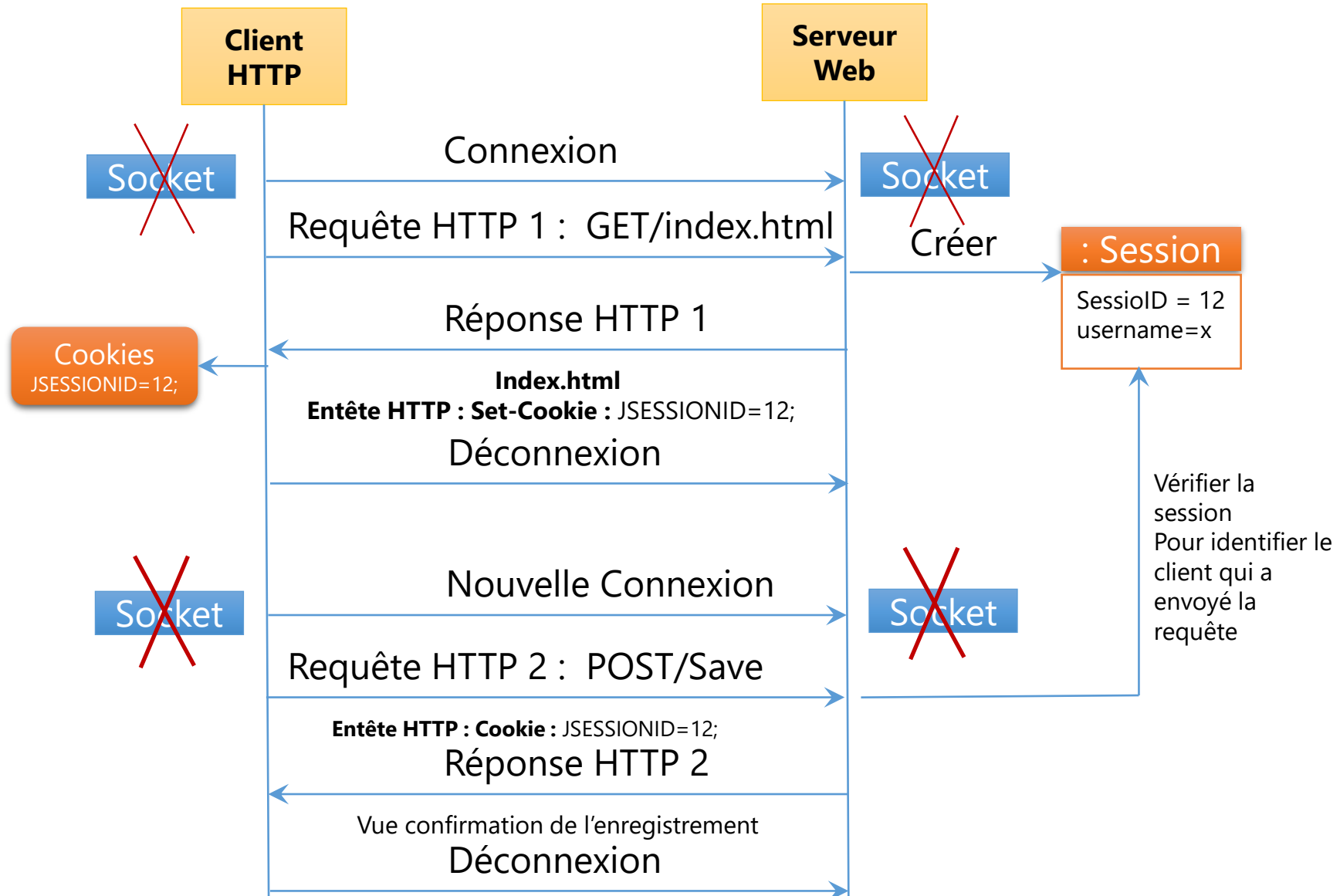


http : Session et Cookies

- Pour lire les données de la requête la servlet utilise l'objet request
- Pour envoyée la réponse HTTP, la servlet utilise l'objet response.
- Pour chaque nouveau client HTTP, le serveur crée un objet Session qui permet de stocker les données relatives au client dans la mémoire du serveur.
- Une session possède un timeout (20 min) au bout duquel, si le client d'envoie pas de reqête HTTP, la session est détruite.
- dans une réponse HTTP, le serveur peut demander au client d'enregistrer des données relatives au client, dans des fichiers de la machine du client. Ces fichiers s'appellent les cookies. Les cookies ont également une durée de vie. Quand cette durée expire, les cookies sont détruit par le navigateur web



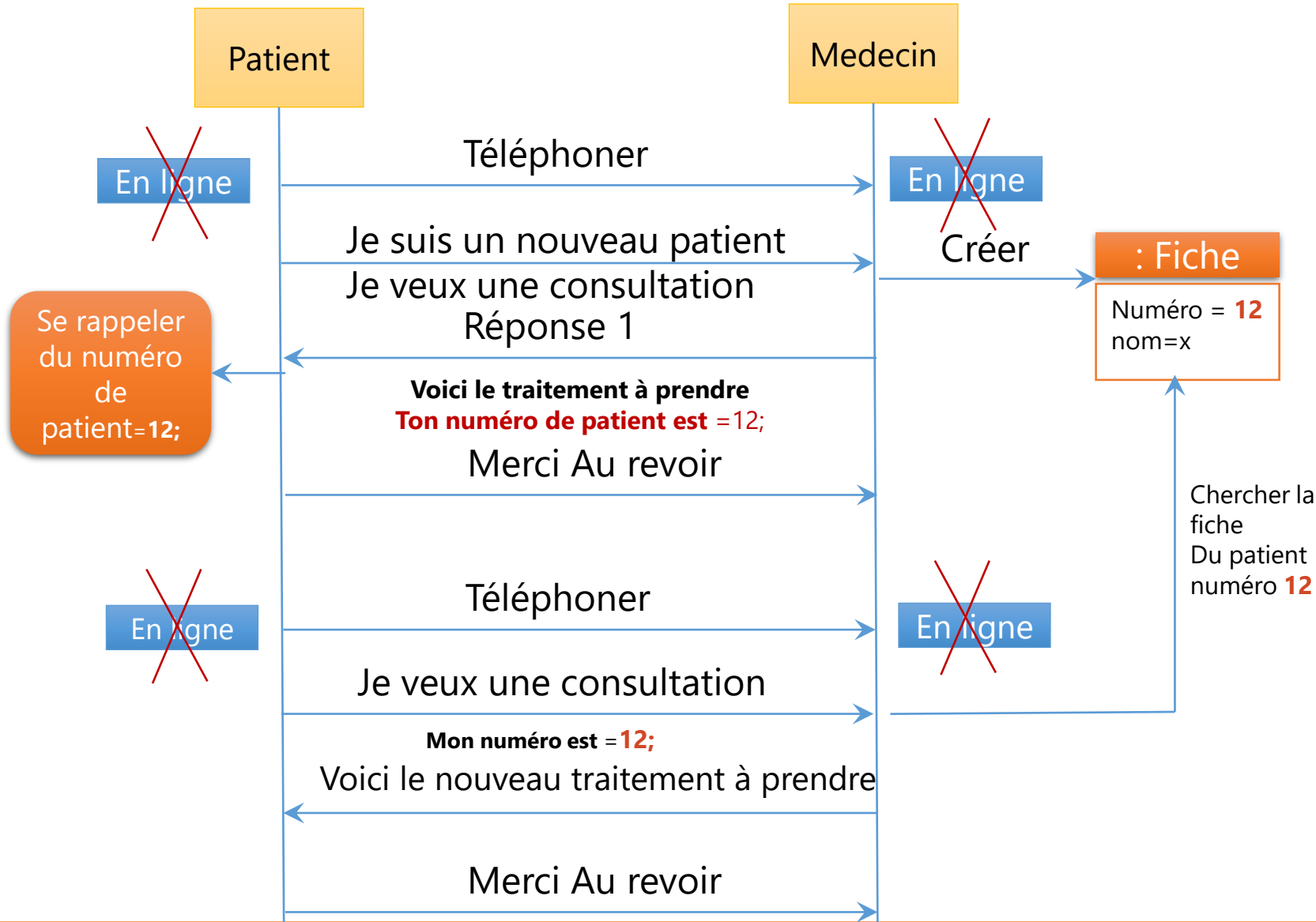
Session et Cookies



Utilisation des sessions et des cookies

- Généralement quant un client HTTP envoie sa première requête, le serveur web crée une session pour ce client.
- Une session est un objet stocké dans la mémoire du serveur qui peut servir pour stocker des informations relatives au client.
- Le serveur attribut un SessionID unique à chaque session.
- Ce SessionID est ensuite envoyé dans la réponse http en sousforme d'un cookie en utilisant l'entête de la réponse HTTP :
- **Set-Cookie** : JSESSIONID=F84DB7B959F76B183DBF05F999FAEE11;
- Ce qui signifie que le serveur demande au client d'enregistrer ce SESSIONID dans un fichier stocké dans la machine du client appelé COOKIE.
- Une fois que le client reçoive la réponse HTTP, la connexion est fermée.
- A chaque fois que le client envoie une requête HTTP vers le serveur, il envoie toujours les données des cookies dont le SESSIONID.
- Les cookies sont envoyés dans la requête HTTP en utilisant une entête COOKIE:
- **Cookie**: JSESSIONID=F84DB7B959F76B183DBF05F999FAEE11
- Grace à cette information, le serveur peut savoir de quel client s'agit-il même s'il s'agit d'une nouvelle connexion.

Session et Cookies > Patient et médecin



Structure d'un Servlet

```
package web; import java.io.IOException; import javax.servlet.*; import javax.servlet.http.*;

public class ControleurServlet extends HttpServlet{

    @Override
    public void init() throws ServletException {
        // Initialisation
        // Exécutée juste après instanciation de la servlet par le serveur Tomcat
    }

    @Override
    protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
        // Traitement effectué si une requête Http est envoyée avec GET
    }

    @Override
    protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException
    {
        // Traitement effectué si une requête Http est envoyée avec POST
    }

    @Override
    public void destroy() {
        // Exécutée juste avant la destruction de la servlet.
        // Au moment de l'arrêt de l'application
    }
}
```

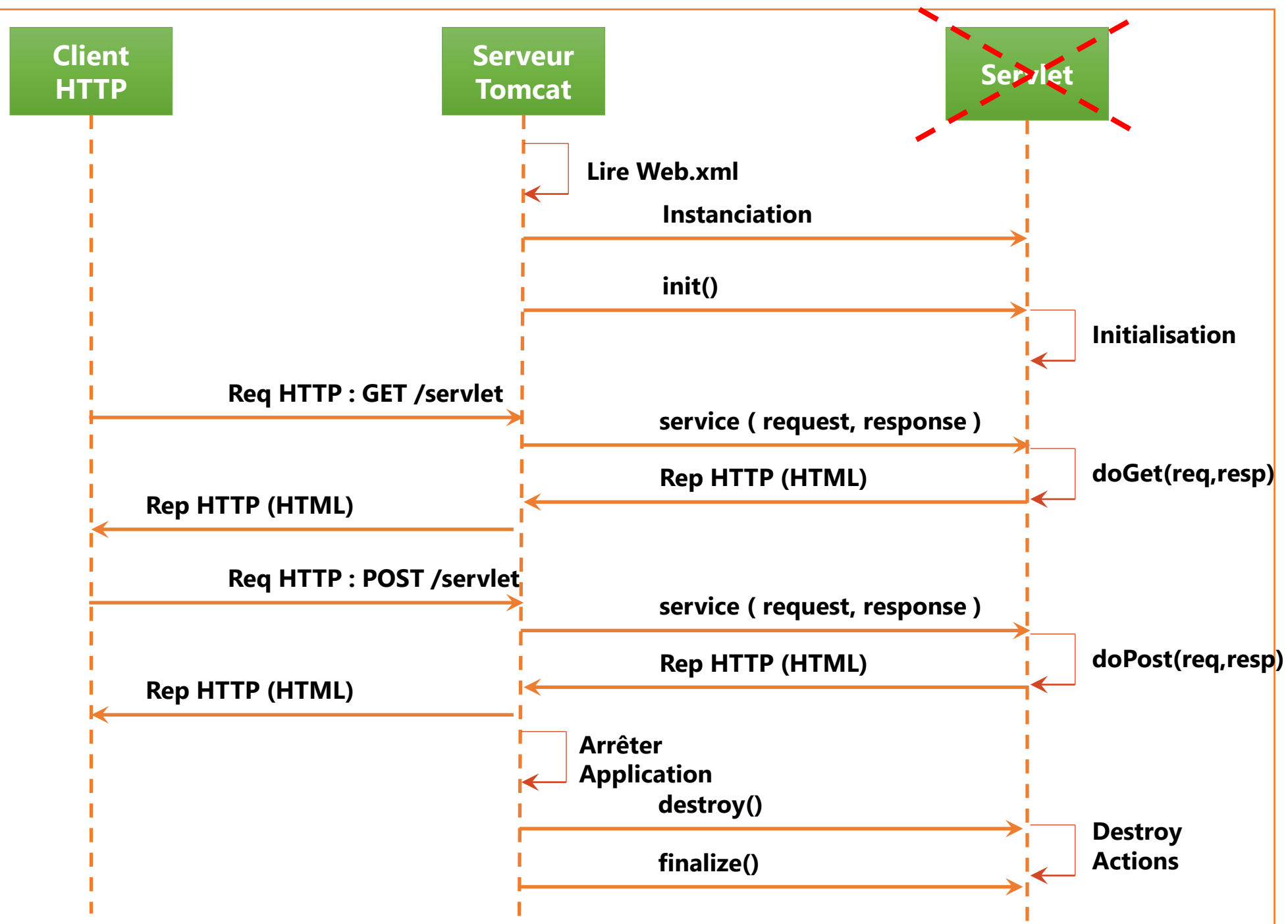
HttpServlet



MyServlet

+doGet(req,resp)
+doPost(req,resp)
+init()
+destroy()

Servlet Life Cycle



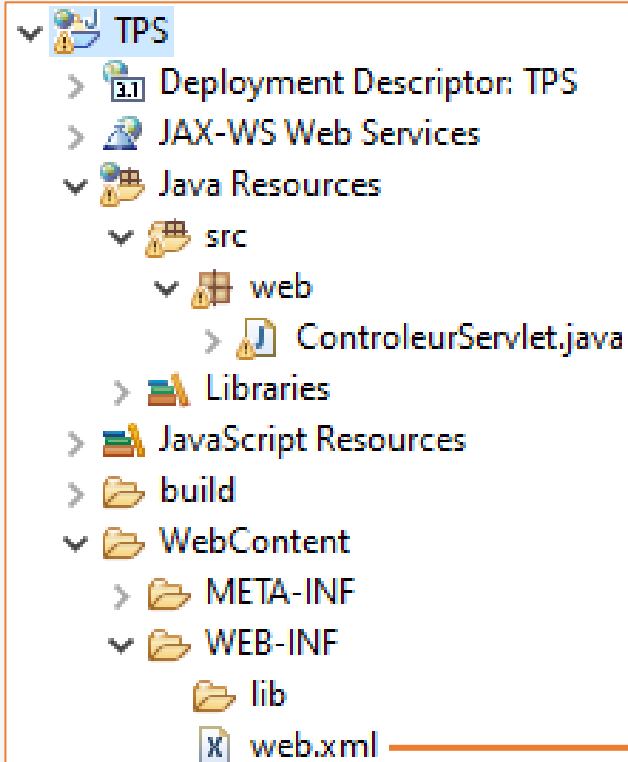
Déployer un Servlet

Pour que le serveur Tomcat reconnaisse une servlet, celle-ci doit être déclarée dans le fichier web.xml qui se trouve dans le dossier WEB-INF.

Le fichier web.xml s'appelle le descripteur de déploiement de Servlet.

Ce descripteur doit déclarer principalement les éléments suivant :

- Le nom attribué à cette servlet
- La classe de la servlet
- Le nom URL à utiliser pour faire appel à cette servlet via le protocole HTTP.

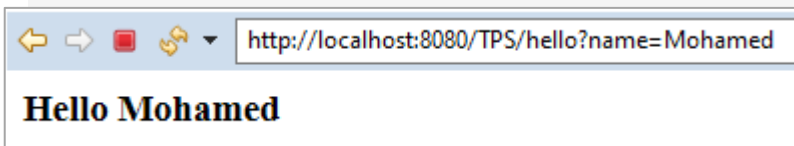


```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns="http://xmlns.jcp.org/xml/ns/javaee"
xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee
http://xmlns.jcp.org/xml/ns/javaee/web-app_3_1.xsd" id="WebApp_ID"
version="3.1">
  <display-name>TPS</display-name>
```

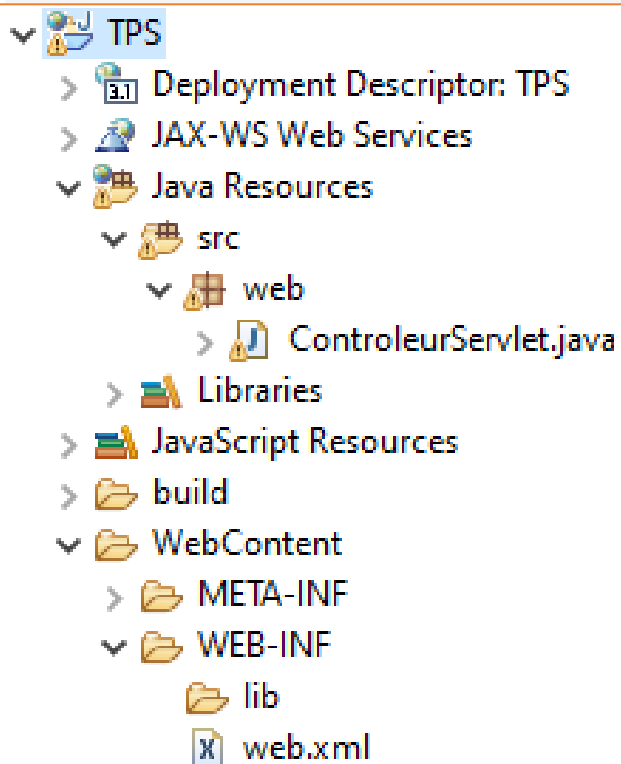
```
  <servlet>
    <servlet-name>cs</servlet-name>
    <servlet-class>web.ControleurServlet</servlet-class>
  </servlet>
```

```
  <servlet-mapping>
    <servlet-name>cs</servlet-name>
    <url-pattern>/hello</url-pattern>
  </servlet-mapping>
```

```
</web-app>
```



Déployer un Servlet : Annotation @WebServlet



- Pour un projet web J2EE, utilisant un module web, version 3.0, le fichier web.xml n'est pas nécessaire.
- Dans ce cas, le déploiement d'une servlet peut se faire en utilisant l'annotation **@WebServlet**:

```
package web;
import java.io.*; import javax.servlet.*;
import javax.servlet.annotation.*;
import javax.servlet.http.*;
@WebServlet(name="cs",urlPatterns={"/hello","*.do"})
public class ControleurServlet extends HttpServlet {

}
```

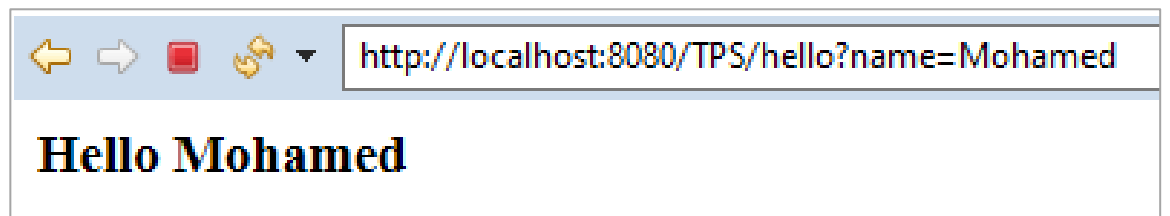
Exemple de Servlet

```
package web;

import java.io.*; import javax.servlet.*; import javax.servlet.http.*;

public class ControleurServlet extends HttpServlet {

    @Override
    protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException,
    IOException {
        response.setContentType("text/html");
        PrintWriter out=response.getWriter();
        String name=request.getParameter("name");
        out.println("<html><head><title>Hello Servlet</title></head>");
        out.println("<body>");
        out.println("<h3>Hello "+name+"</h3>");
        out.println("</body></html>");
    }
}
```



Servlet Vs JSP (Java Server Pages)

Servlet

```
package web; import java.io.*; import javax.servlet.*; import
javax.servlet.http.*;

@WebServlet(name="cs",urlPatterns = {"/hello","*.do"})

public class ControleurServlet extends HttpServlet {

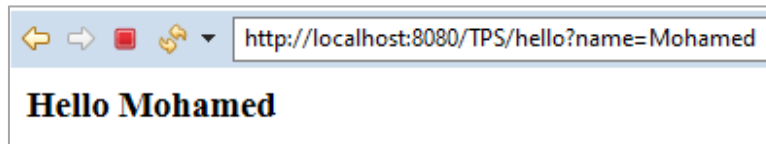
    @Override

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

    response.setContentType("text/html");
    PrintWriter out=response.getWriter();
    String name=request.getParameter("name");
    out.println("<html><head><title>Hello
Servlet</title></head>");
    out.println("<body>");
    out.println("<h3>Hello "+name+"</h3>");
    out.println("</body></html>");

}

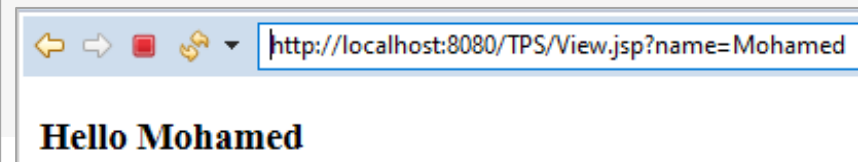
}
```



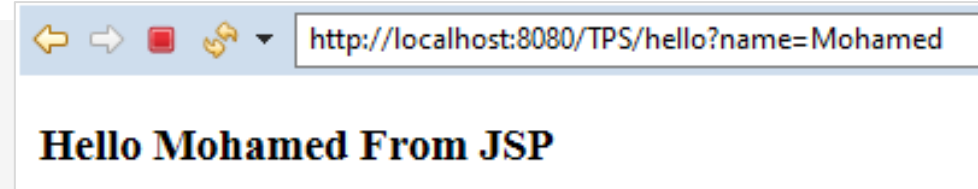
View.jsp

```
<%
String name=request.getParameter("name");
%>
<!DOCTYPE html>
<html>
<head>
    <title>Hello JSP</title>
</head>
<body>
    <h3>Hello <%=name%></h3>
</body>
</html>
```

- Servlet : Classe Java dans laquelle on génère du code HTML
- JSP : Forme d'une page HTML dans laquelle on écrit du code Java
- Quand une JSP est appelée la première fois, elle sera convertie en Servlet par Tomcat.
- Dans une application web JEE, on utilise les deux :
 - Servlet pour jouer le rôle du contrôleur
 - JSP pour jouer le rôle d'une Vue



Communication entre Servlet et JSP : Forward

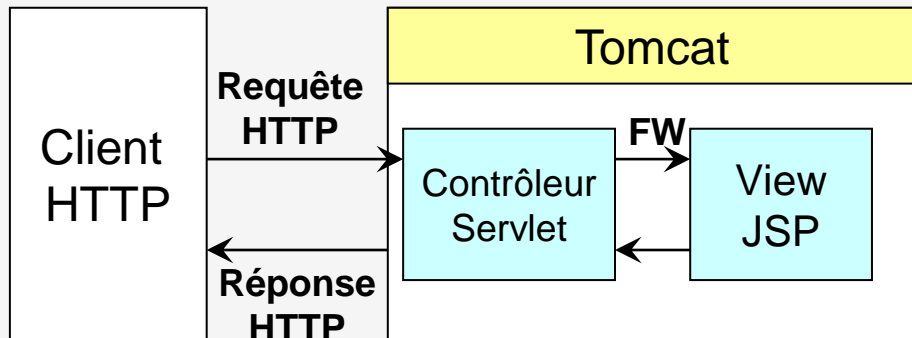


Pour séparer les rôles une servlet peut faire un forward vers une JSP de la manière suivante :

```
@WebServlet(name="cs",urlPatterns = {"/hello","*.do"})
public class ControleurServlet extends HttpServlet {
    @Override
    protected void doGet(HttpServletRequest request, HttpServletResponse
response) throws ServletException, IOException {

    String name=request.getParameter("name");
    request.setAttribute("inputData", name);
    request.getRequestDispatcher("View.jsp").forward(request, response);

} }
```



```
<%
    String name=(String)request.getAttribute("inputData");
%>
<!DOCTYPE html>
<html>
<head>
<title>Hello JSP</title>
</head>
<body>
    <h3>Hello <%=name%> From JSP</h3>
</body>
</html>
```

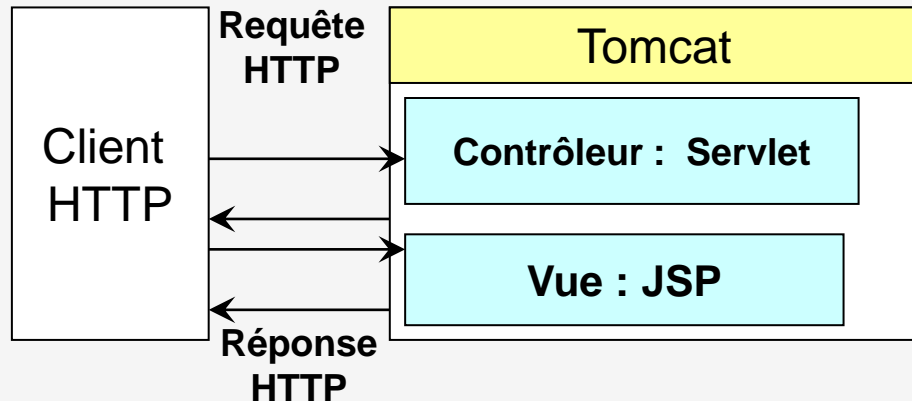
Communication entre Servlet et JSP : Redirection

```
@WebServlet(name="cs",urlPatterns = {"/hello","*.do"})
public class ControleurServlet extends HttpServlet {
    @Override
    protected void doGet(HttpServletRequest request, HttpServletResponse
response) throws ServletException, IOException {

        String name=request.getParameter("name");
        response.sendRedirect("View.jsp?name="+name);
    }
}
```

http://localhost:8080/TPS/View.jsp?name= Mohamed

Hello Mohamed From JSP



```
<%
String name=request.getParameter("name");
%>
<!DOCTYPE html>
<html>
<head>
<title>Hello JSP</title>
</head>
<body>
<h3>Hello <%=name%> From JSP</h3>
</body>
</html>
```



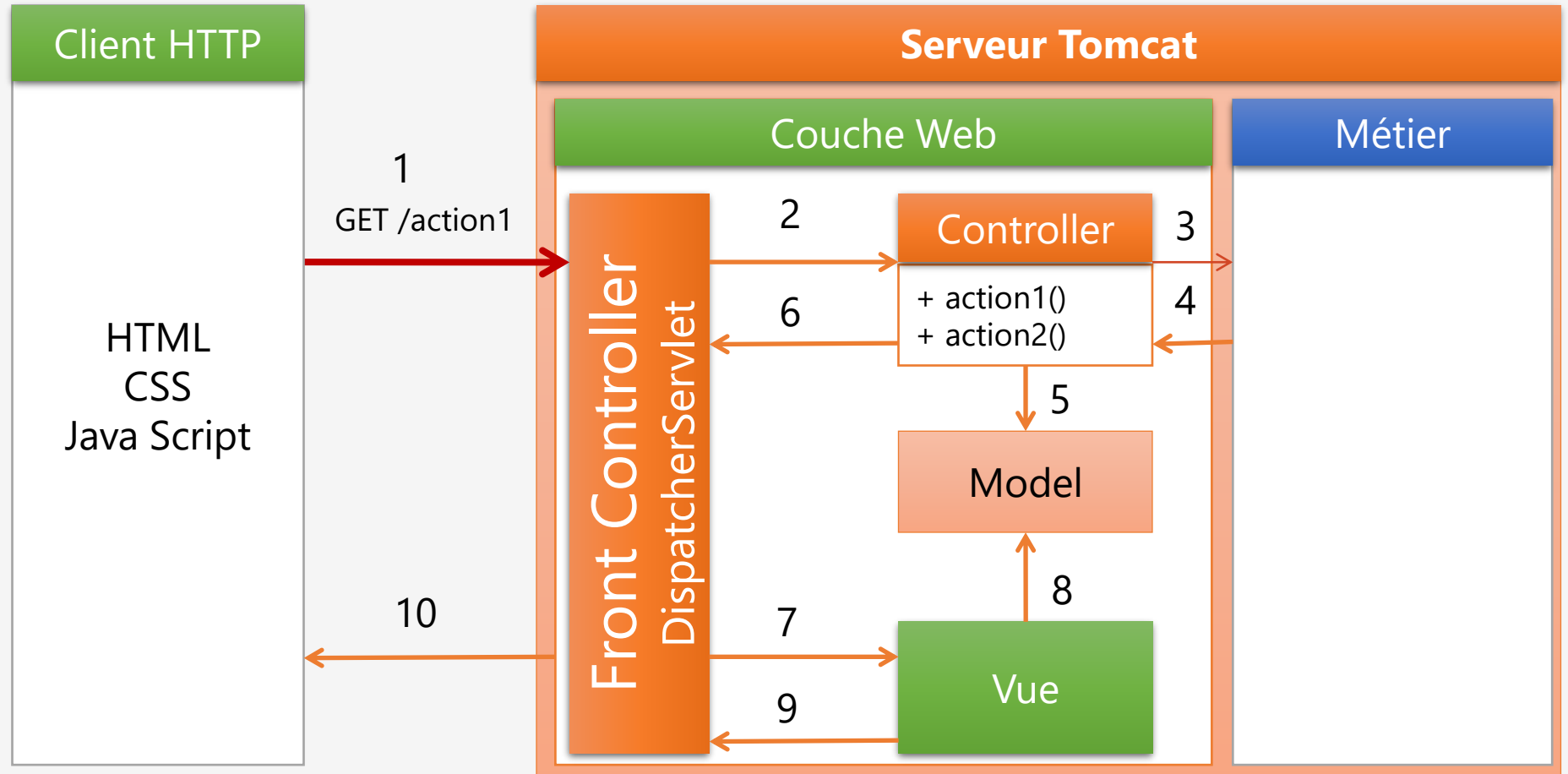

Développement Web JEE

Spring MVC

Architecture Spring MVC

1 –

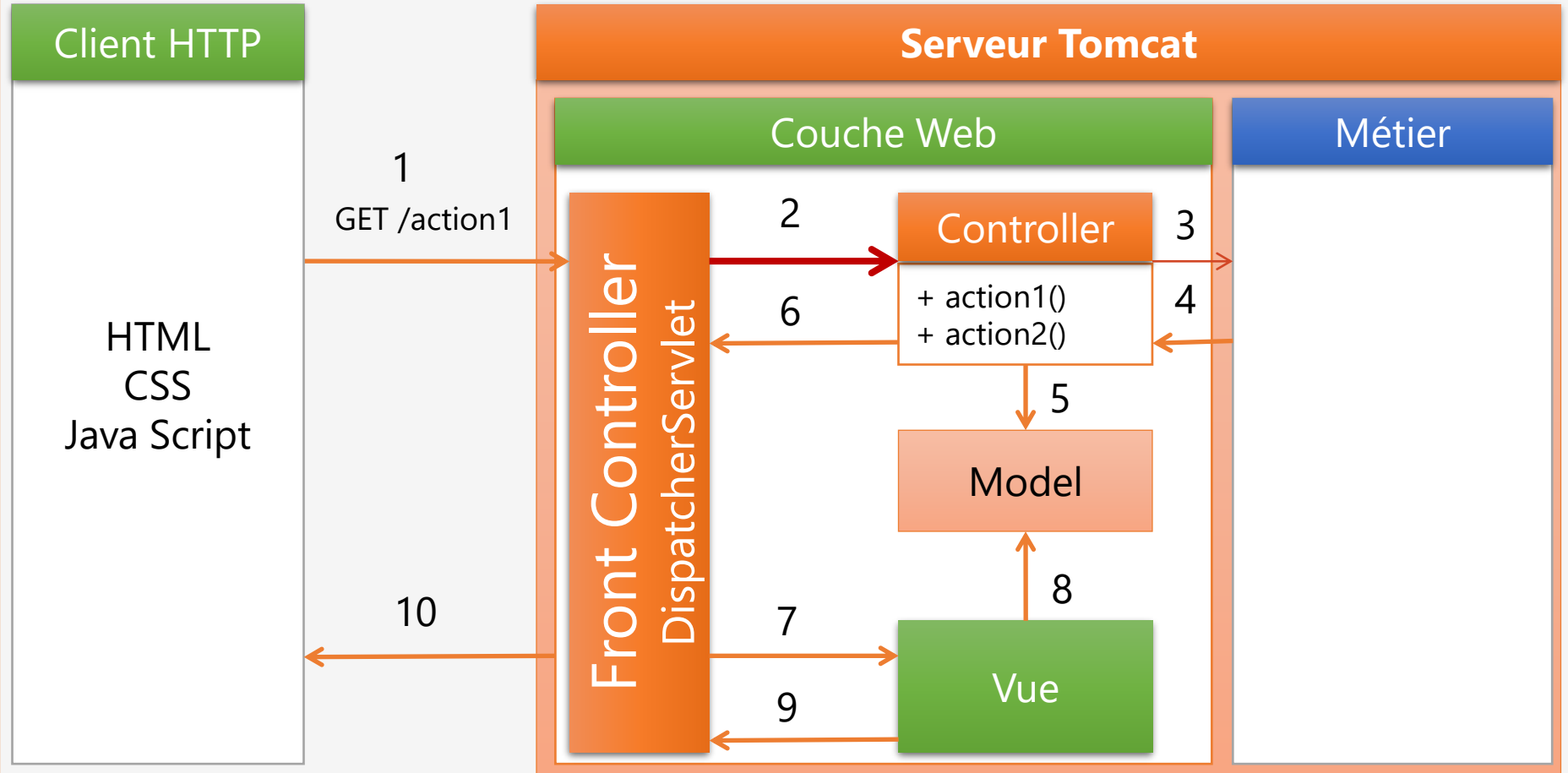
Le client envoie une requête HTTP de type GET ou POST



Architecture Spring MVC

2 -

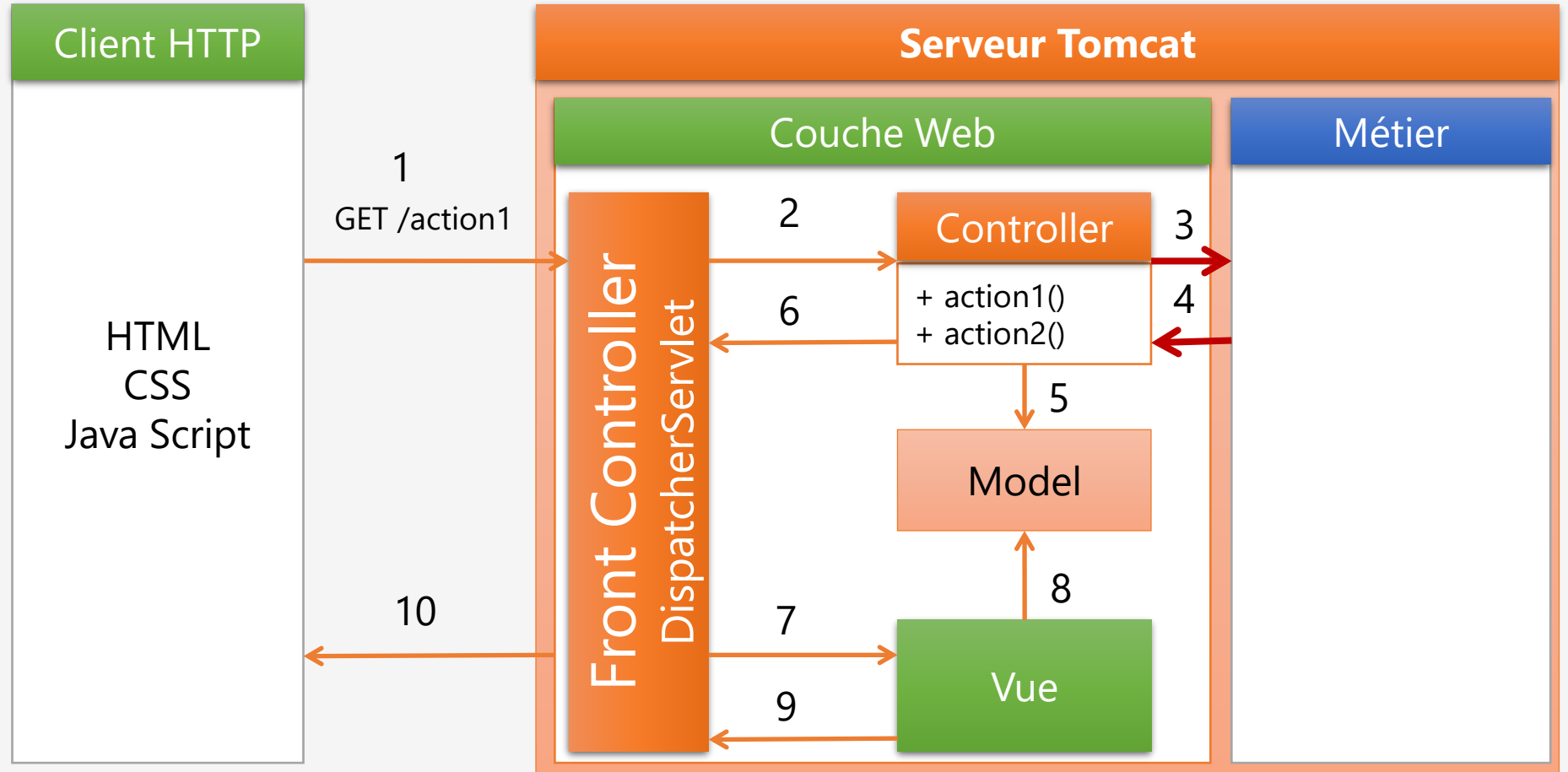
- Toutes les requêtes HTTP sont traitées par un contrôleur frontal fourni par Spring.
- C'est une servlet nommée **DispatcherServlet**.
- Chaque action de l'URL, DispatcherServlet devrait exécuter une opération associée à cette action.
- Cette opération est implémentée dans une classe appelée Controller qui représente un sous contrôleur ou un contrôleur secondaire.



Architecture Spring MVC

3 et 4 –

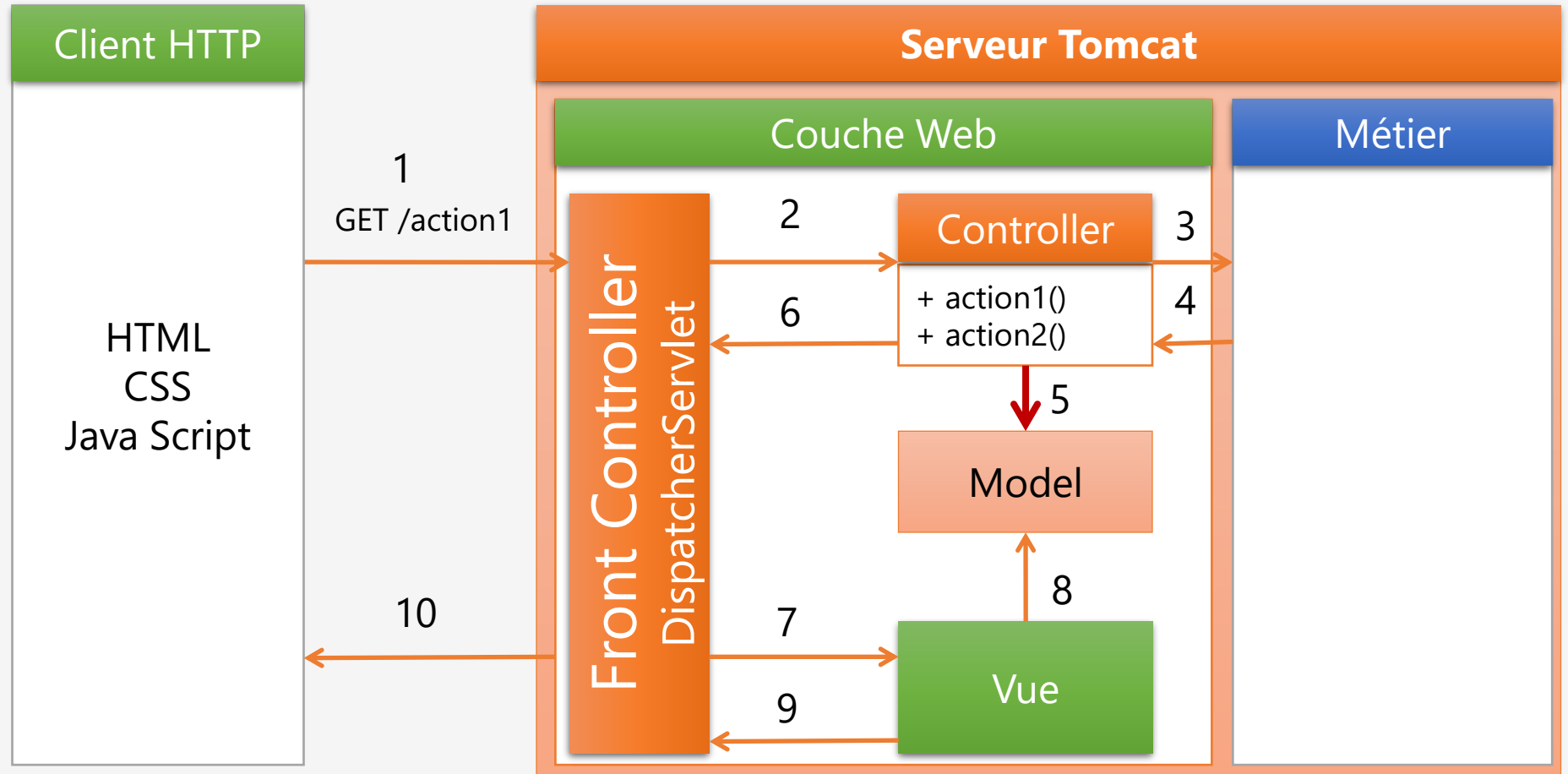
- Le sous contrôleur exécute le traitement associé à l'action en faisant appel à la couche métier et récupère le résultat .



Architecture Spring MVC

5-

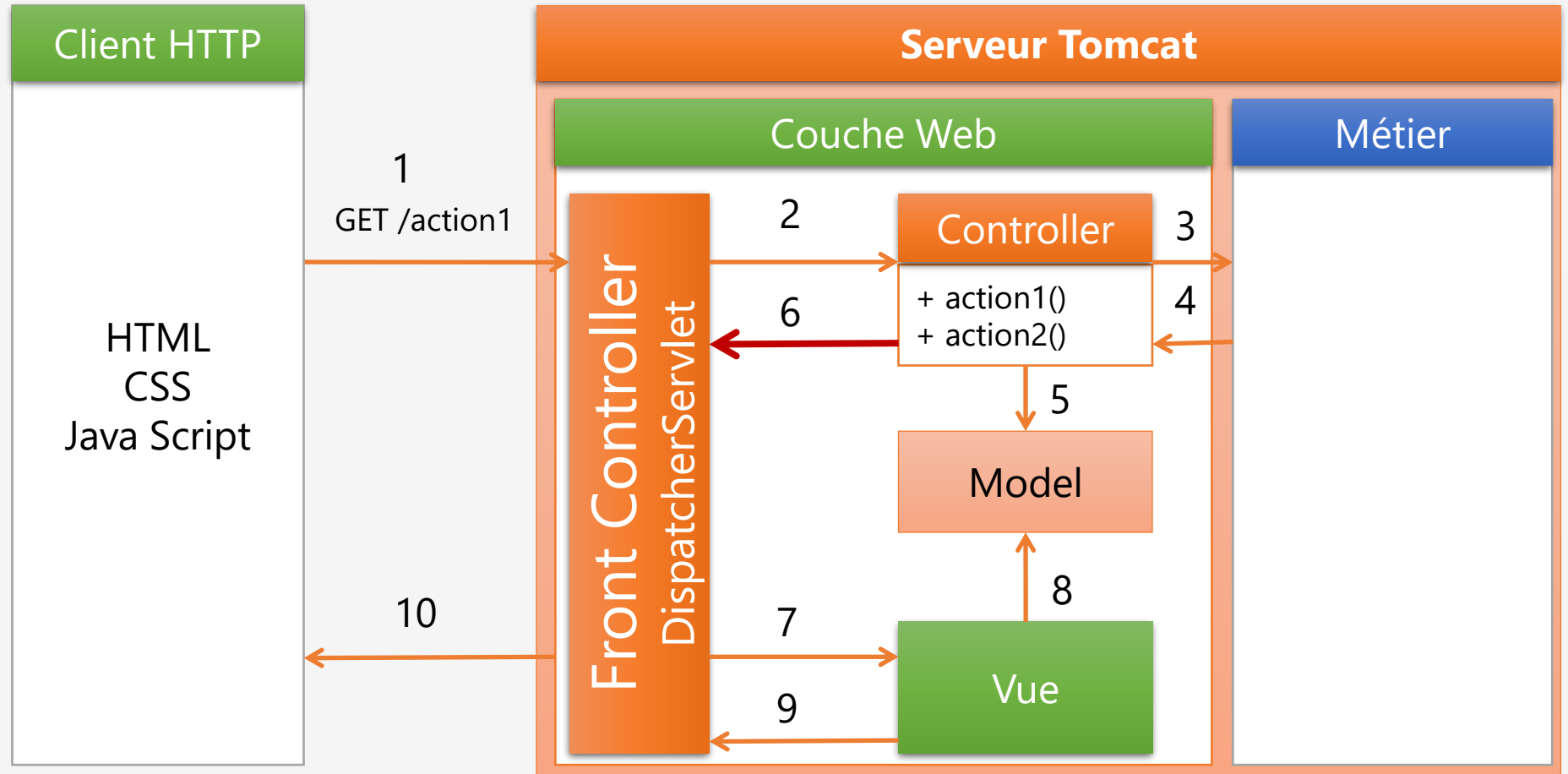
- Le sous contrôleur stocke le résultat dans le modèle fourni par Spring MVC.



Architecture Spring MVC

6-

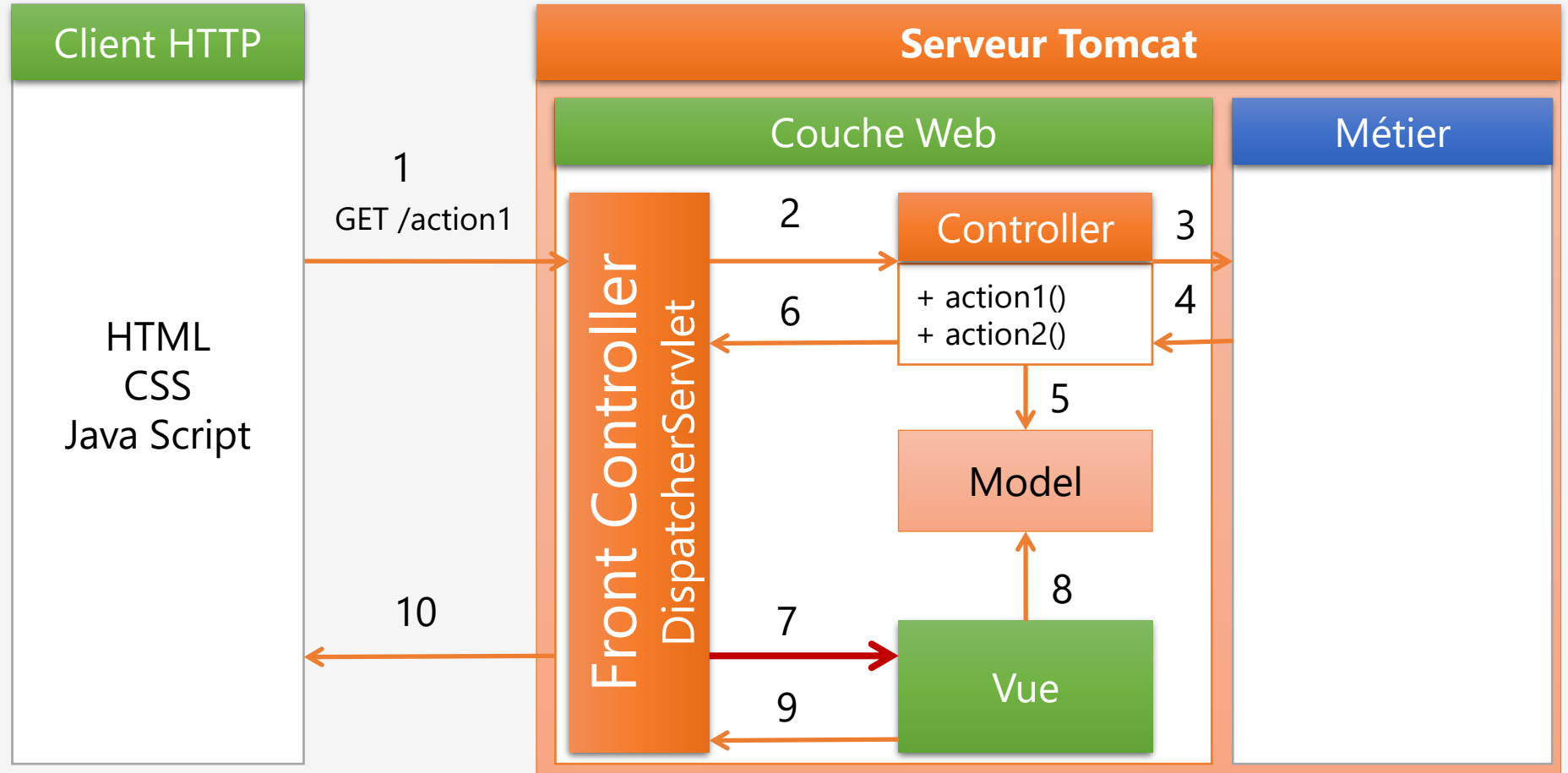
- Le sous contrôleur retourne le nom de la vue et le modèle à DispatcherServlet



Architecture Spring MVC

7-

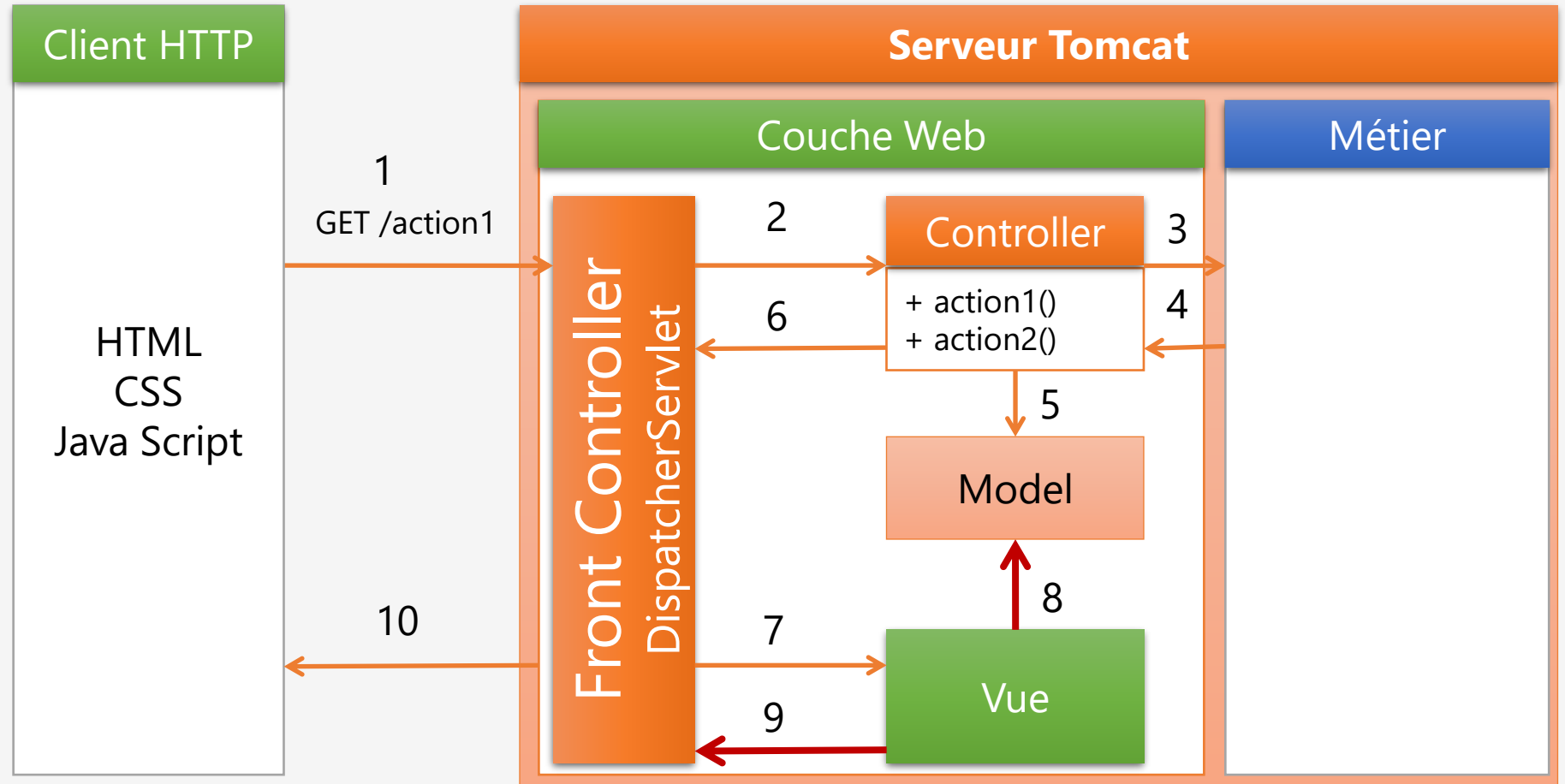
- Le contrôleur frontal DispatcherServlet fait appel à la vue et lui transmet le modèle



Architecture Spring MVC

8 et 9 –

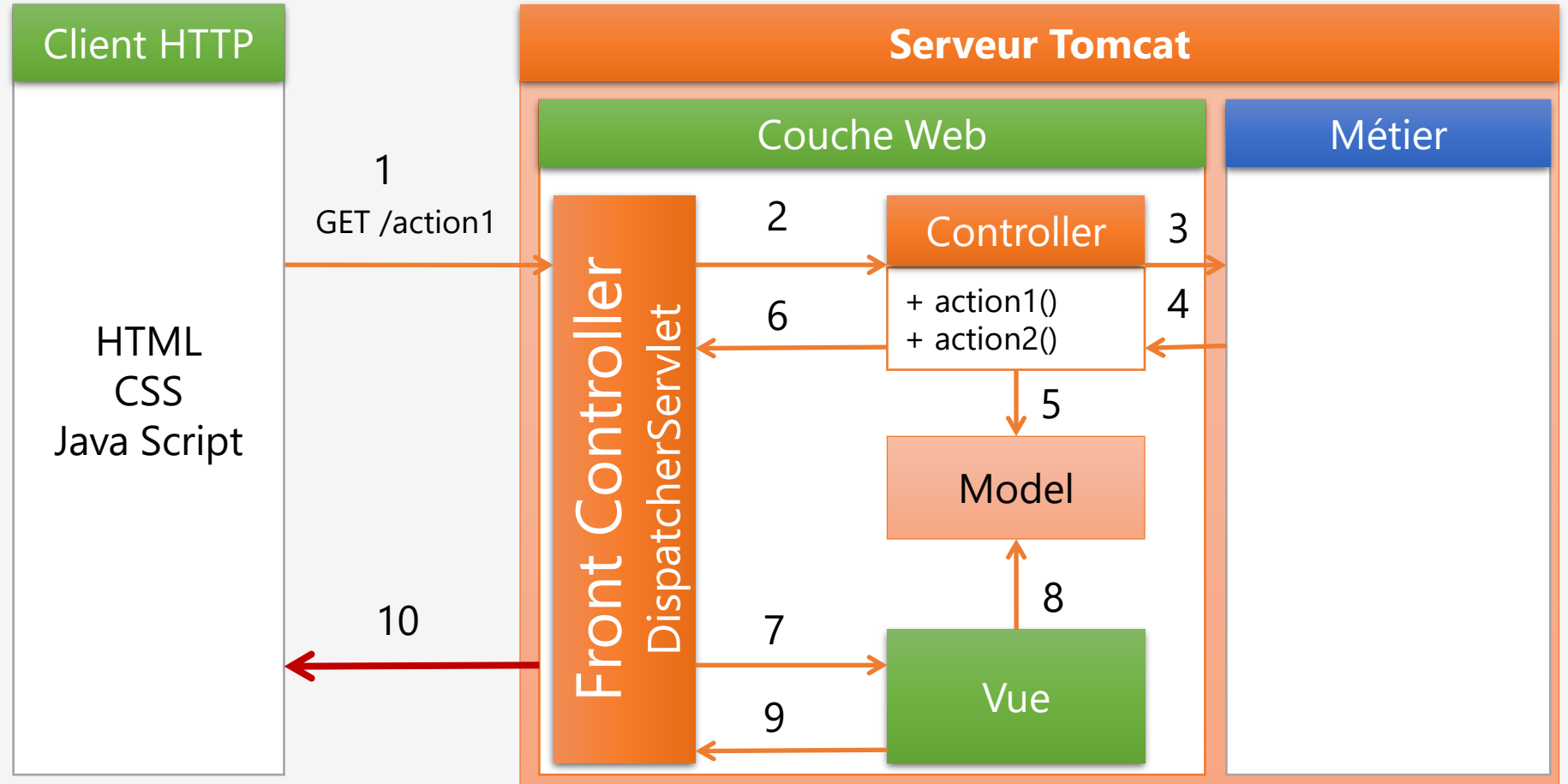
- La vue récupère les résultats à partir du modèle et génère un rendu HTML qui est retourné à DispatcherServlet
- Pour générer du code HTML, on peut utiliser JSP, mais il est déconseillé car il y a mieux : les Moteurs de templates
- Spring MVC offre des moteurs de templates comme **Thymeleaf**, FreeMaker, Mustach, qui permettent de faciliter la génération du code HTML coté serveur



Architecture Spring MVC

10-

- DispatcherServlet envoie la réponse HTTP au client. Cette réponse http contient le code HTML générée par la vue.



Client
HTTP

GET /produits

DispatcherServlet

```
@Controller
public class ProduitController {
    @Autowired
    private ProduitRepository produitRepository;

    @RequestMapping(value="/produits",method=RequestMethod.GET)
    public String listProduits(Model model){
        List<Produit> produits=produitRepository.findAll();
        model.addAttribute("listProduits",produits);
        return "vue";
    }
}
```

Controller

METIER : ProduitRepository

model

Key	Value
listProduits	

produit
produit
produit

vue.html

Thymeleaf

```
<html xmlns:th="http://www.thymeleaf.org" >
  <tr th:each="p:${listProduits}">
    <td th:text="${p.id}"></td>
    <td th:text="${p.designation}"></td>
    <td th:text="${p.prix}"></td>
    <td th:text="${p.quantite}"></td>
  </tr>
</html>
```

HTML

HTML



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 :

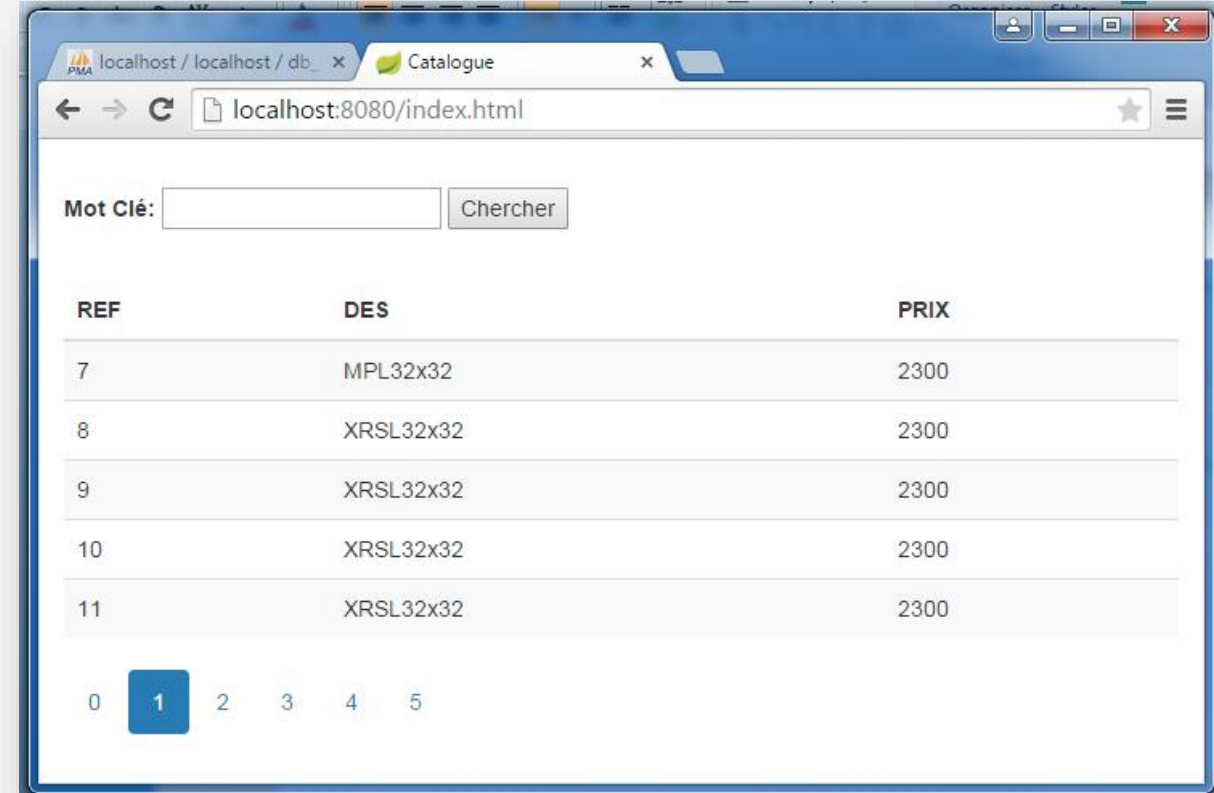
- Sa référence de type Long
- Sa désignation de type String
- Son prix

L'application doit permettre de :

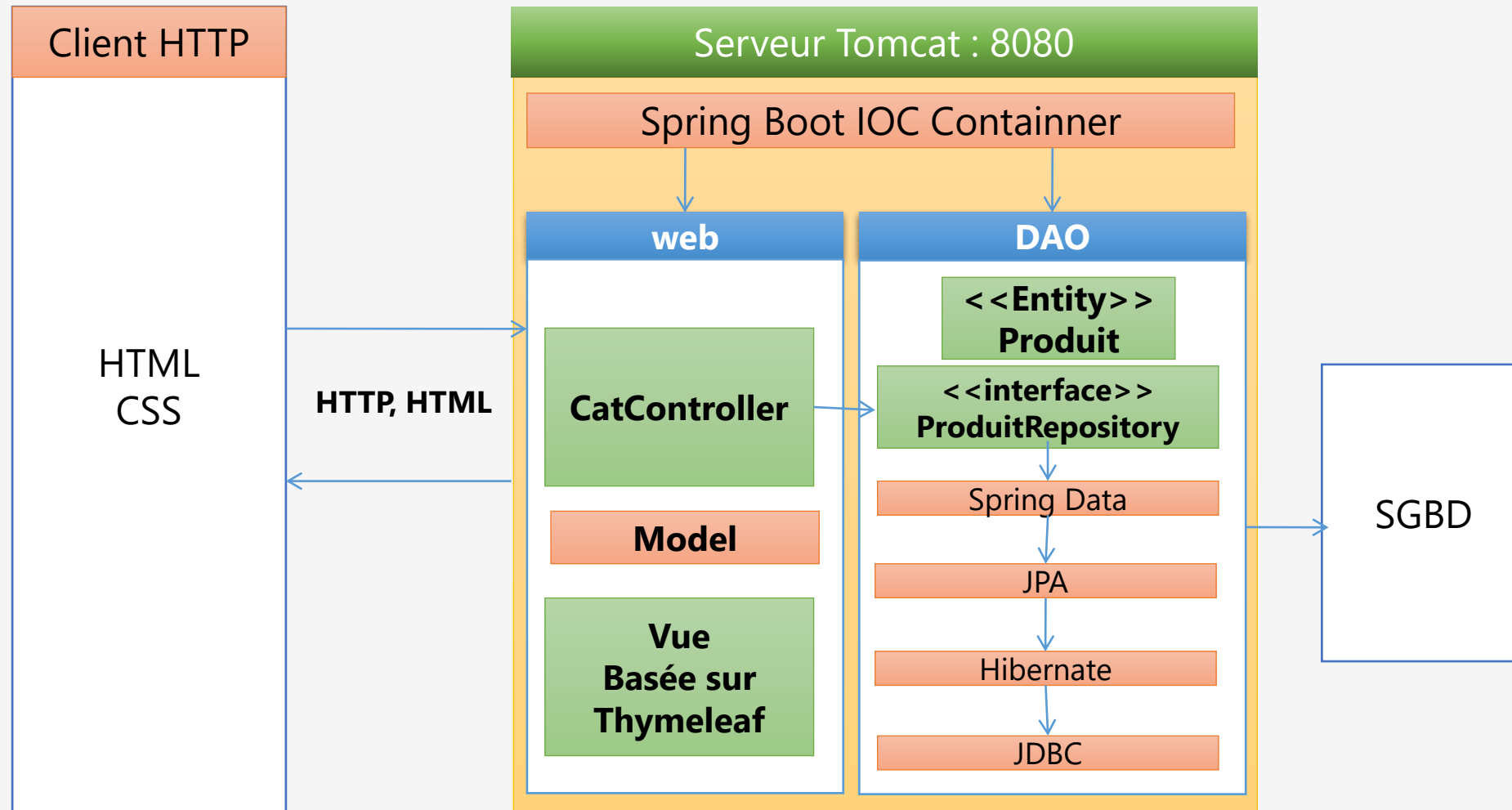
- Ajouter de nouveaux produits
- Consulter les produits
- Chercher les produits par mot clé
- Consulter un produit
- Mettre à jour un produit
- Supprimer un produit

Les données sont stockées dans une base de données MySQL

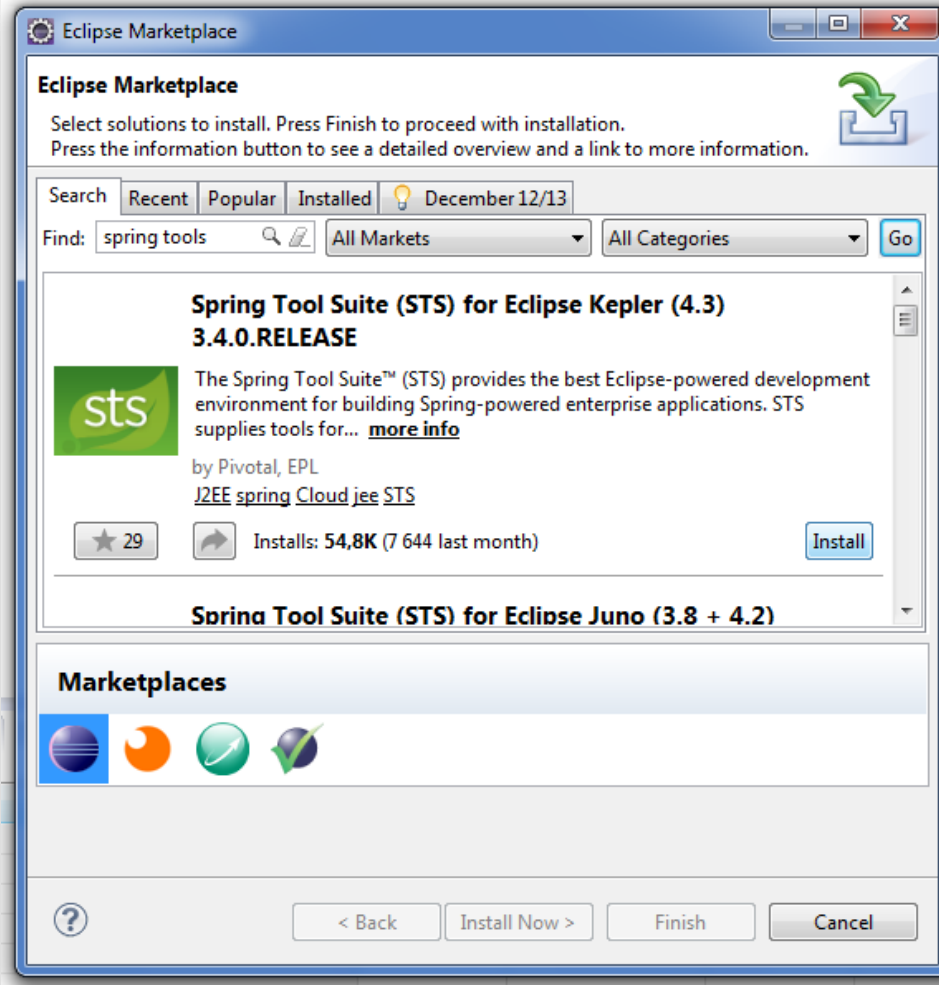
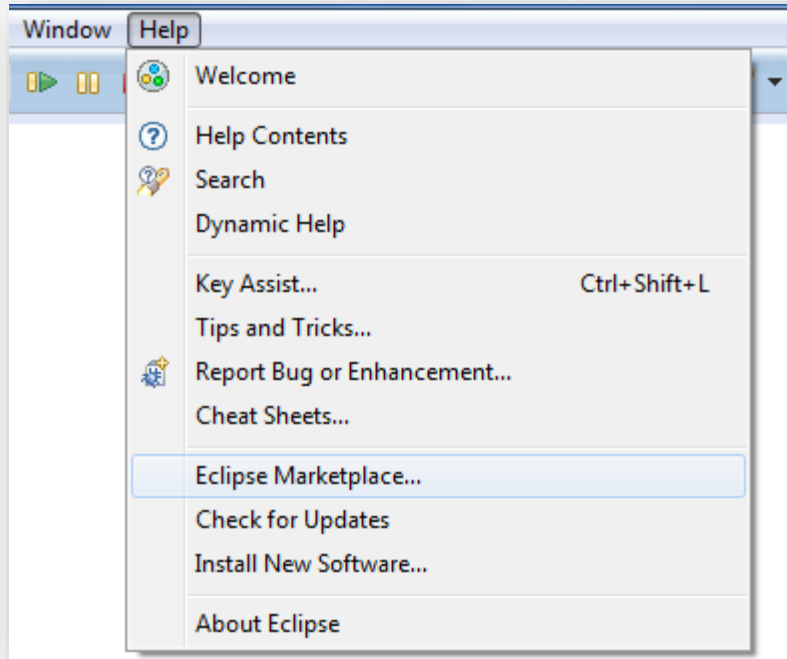
La couche web respecte MVC coté serveur.



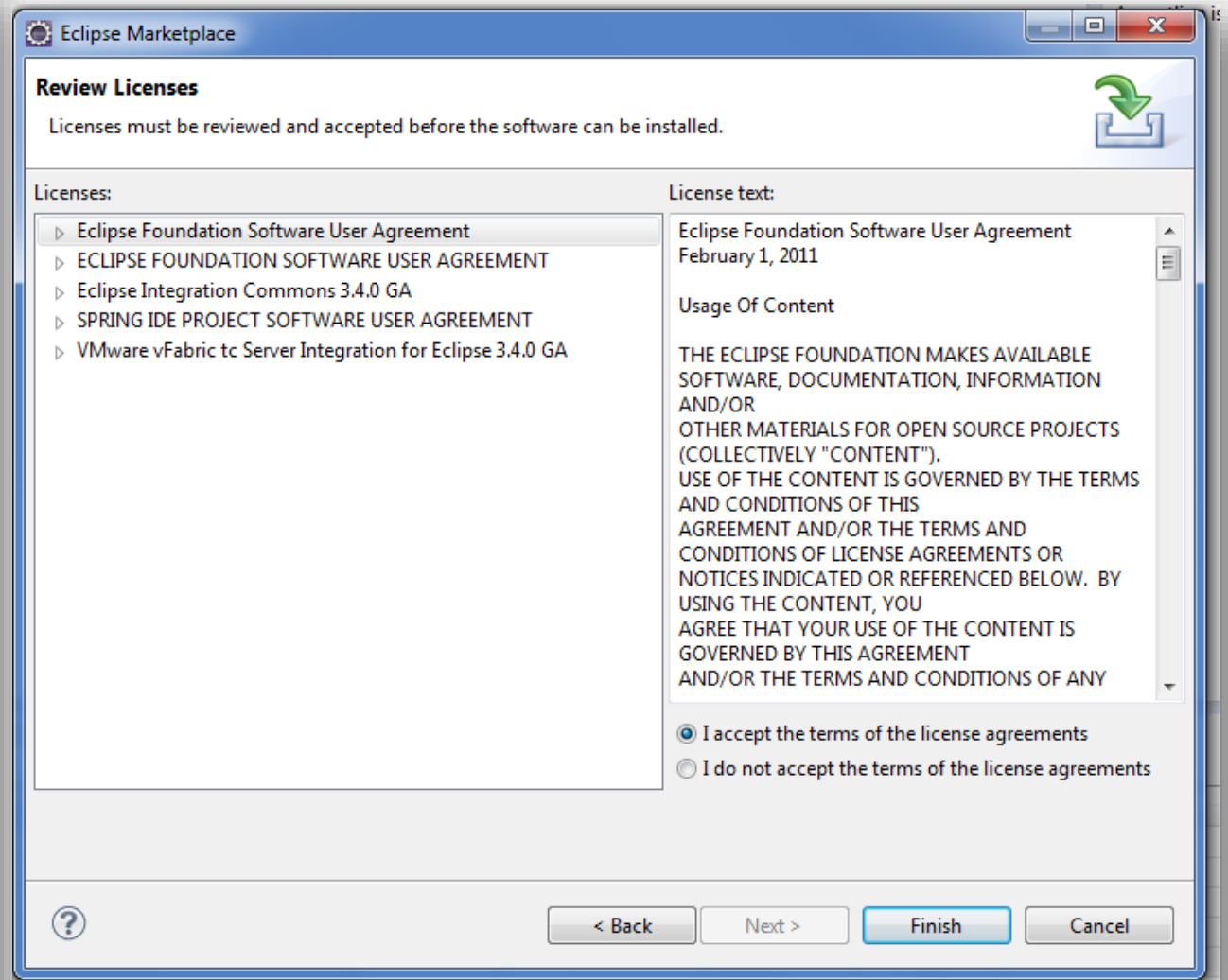
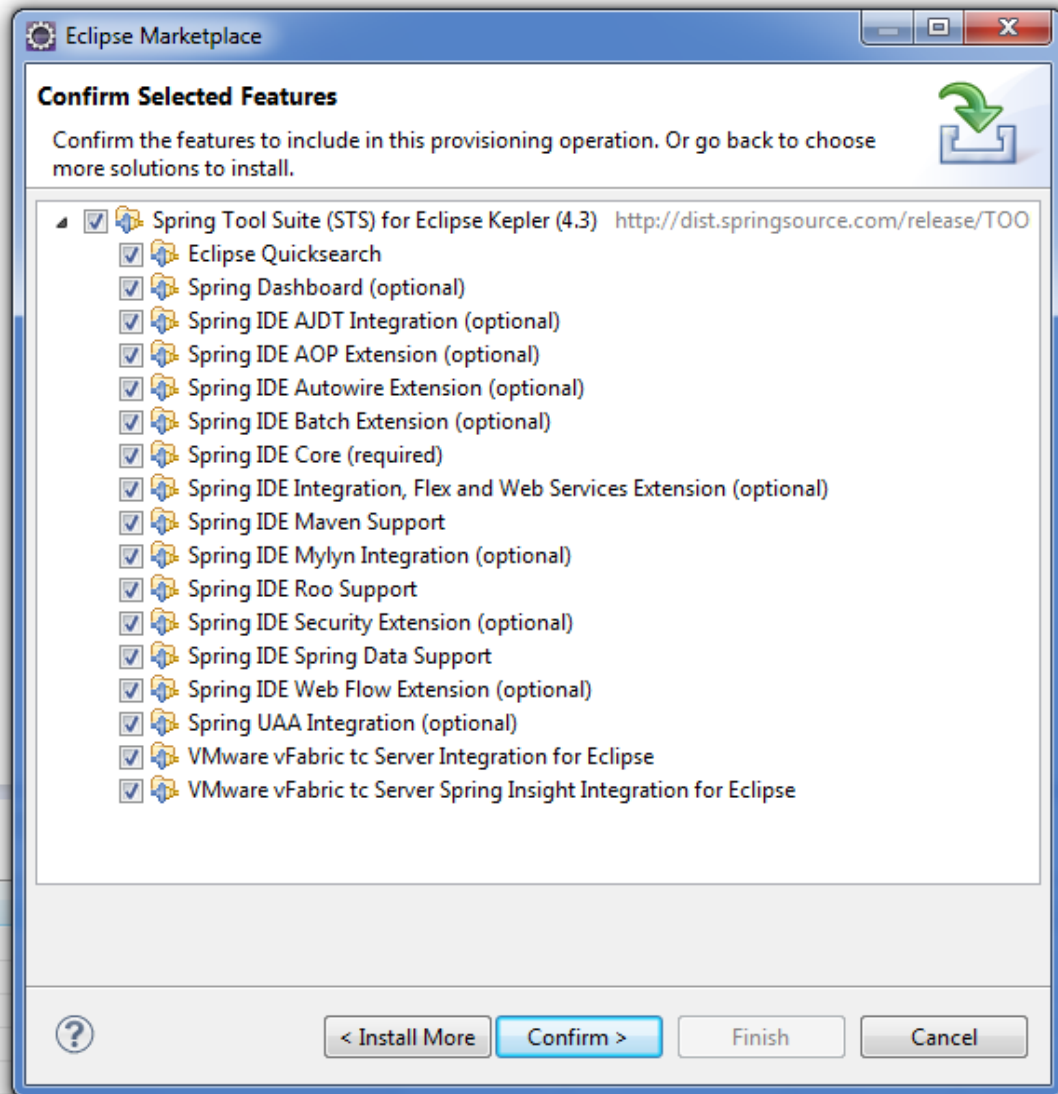
Spring MVC avec Thymeleaf



Installation du plugin : spring tools pour eclipse



Installation du plugin : spring tools pour eclipse



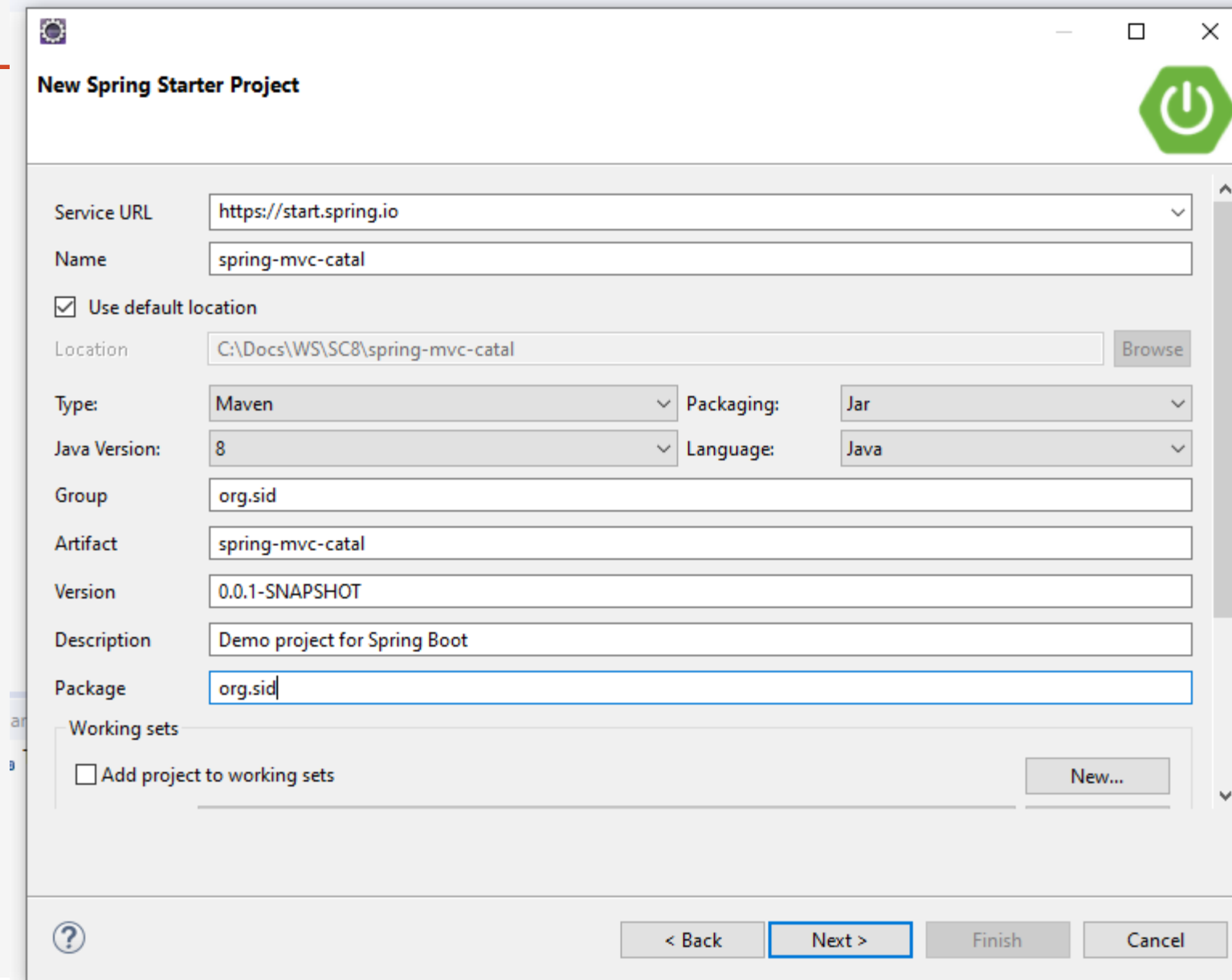
Spring Boot

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 fichiers 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)

Création d'un projet Spring Starter

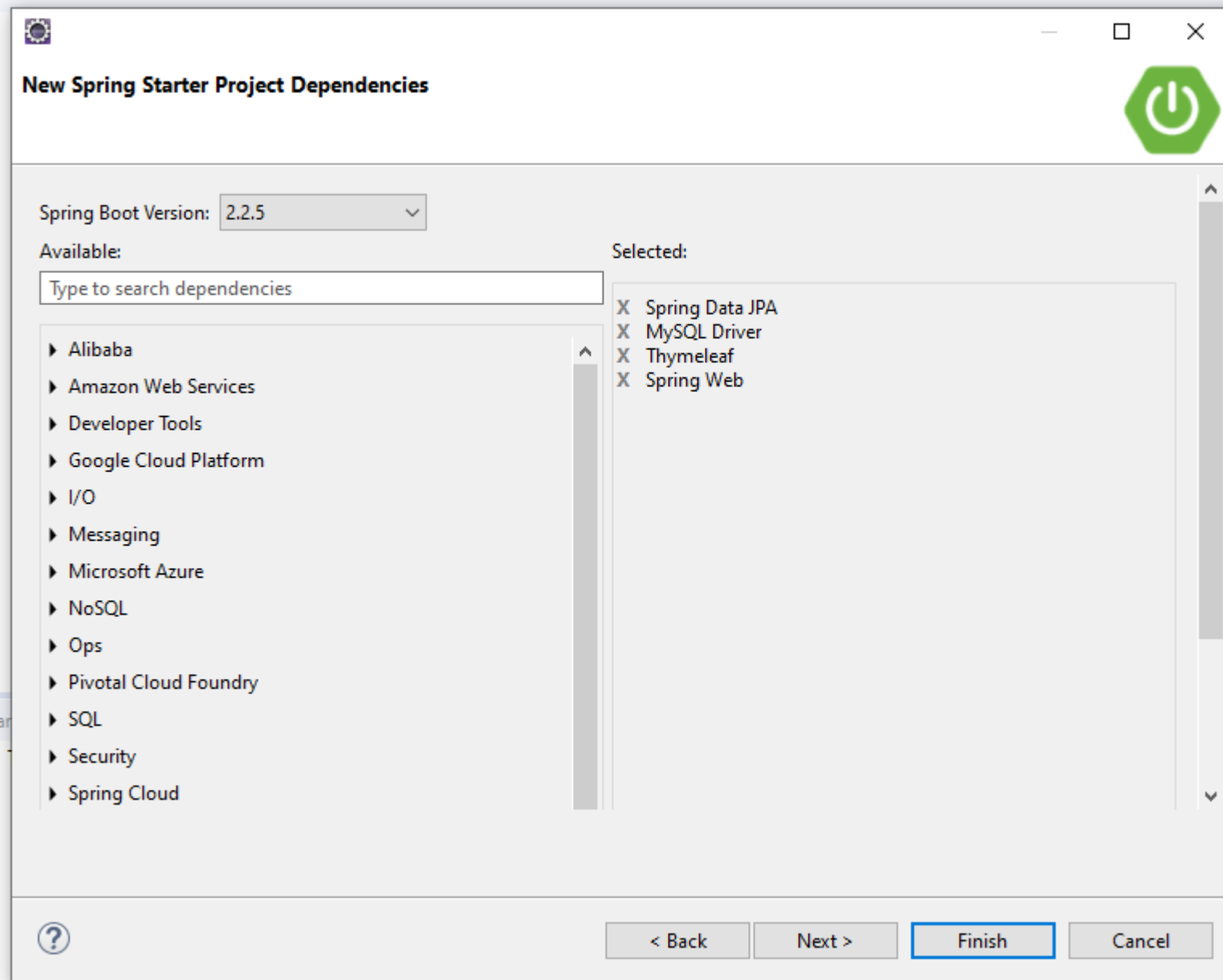


The screenshot shows the 'New Spring Starter Project' dialog box in the Eclipse IDE. The dialog has a title bar with a question mark icon, a maximize button, and a close button. A green power button icon is located in the top right corner of the dialog area. The main content area contains several fields and options for configuring a new project:

- Service URL:** A text field containing 'https://start.spring.io'.
- Name:** A text field containing 'spring-mvc-catal'.
- Use default location:** A checked checkbox.
- Location:** A text field containing 'C:\Docs\WS\SC8\spring-mvc-catal' and a 'Browse' button.
- Type:** A dropdown menu set to 'Maven'.
- Packaging:** A dropdown menu set to 'Jar'.
- Java Version:** A dropdown menu set to '8'.
- Language:** A dropdown menu set to 'Java'.
- Group:** A text field containing 'org.sid'.
- Artifact:** A text field containing 'spring-mvc-catal'.
- Version:** A text field containing '0.0.1-SNAPSHOT'.
- Description:** A text field containing 'Demo project for Spring Boot'.
- Package:** A text field containing 'org.sid'.
- Working sets:** A section with an unchecked checkbox 'Add project to working sets' and a 'New...' button.

At the bottom of the dialog, there is a row of buttons: a question mark icon, '< Back', 'Next >' (highlighted with a blue border), 'Finish', and 'Cancel'.

Sélectionner les dépendances maven



- Web,
- JPA,
- MySQL
- Thymeleaf

Dépendances maven Spring boot

La dépendance Web

- Permet d'ajouter au classpath du projet une version de tomcat embarquée comme conteneur web de l'application.

La dépendance JPA :

- Ajouter au classpath du projet :
 - Les dépendances JPA
 - Les dépendances Hibernate
 - Les dépendance Spring Data qui permet de faciliter à l'extrême l'accès aux données de l'application.

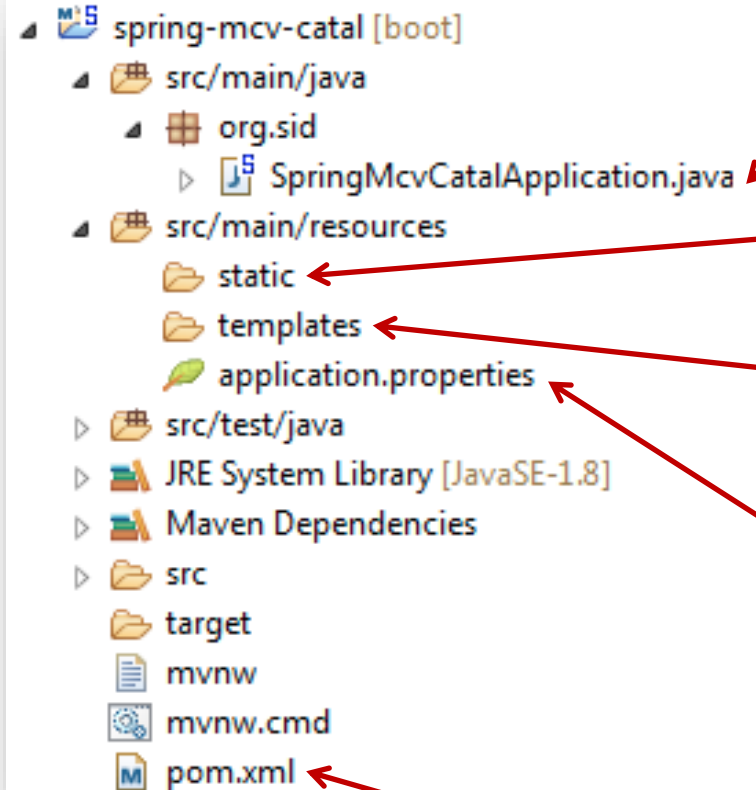
La dépendance MySQL :

- permet d'ajouter au classpath du projet le pilote JDBC MySQL

La dépendance Tymeleaf :

- permet d'ajouter au class path du projet une moteur de template thymeleaf qui permet de faciliter la création des vues et d'éviter de travailler avec JSP et JSTL.

Structure du projet



Application Spring Boot

Ressources statiques de l'application web : HTML, CSS, Java Script, images, etc...

Les vues de l'application qui seront interprétées côté serveur par le moteur de vue Thymeleaf

Fichier de configuration de l'application Spring Boot. Fichier lu au démarrage de Spring

Fichier de dépendances Maven

Application Spring Boot

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

Contenu de pom.xml

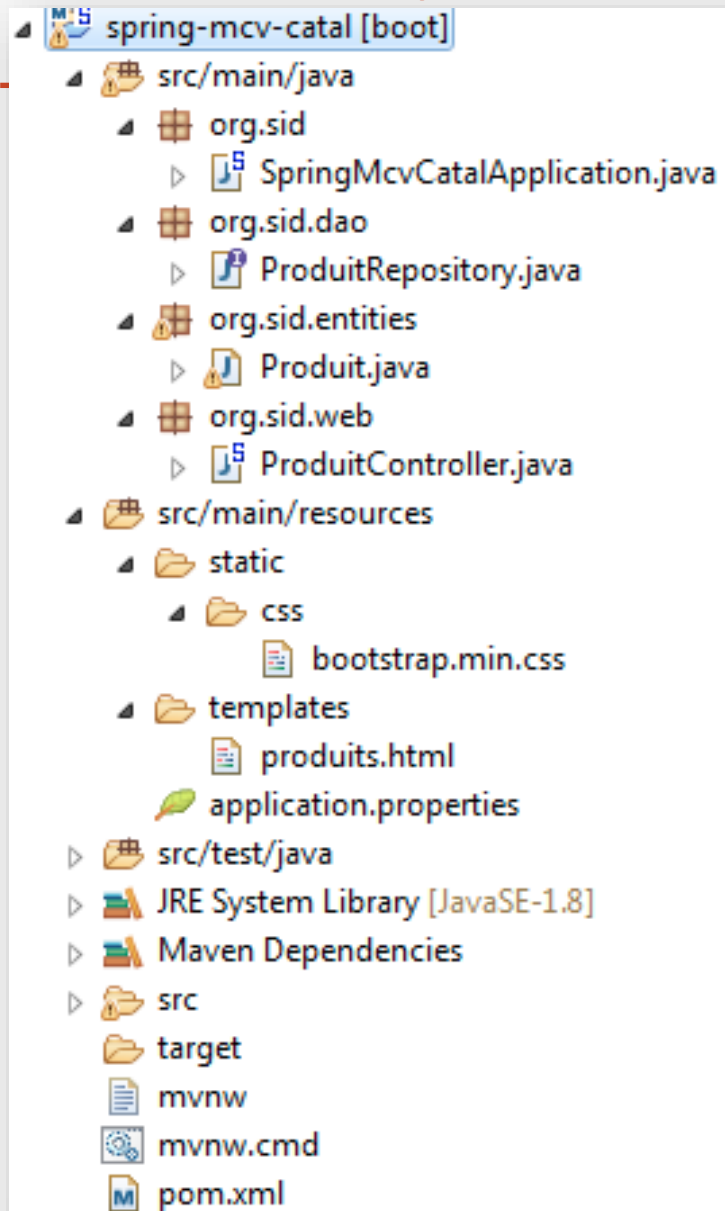
```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
    http://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <groupId>org.sid</groupId>
  <artifactId>catalogue-mvc</artifactId>
  <version>0.0.1-SNAPSHOT</version>
  <packaging>jar</packaging>
  <name>spring-mcv-catal</name>
  <description>Demo project for Spring Boot</description>
  <parent>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-parent</artifactId>
    <version>1.3.0.RELEASE</version>
    <relativePath/> <!-- lookup parent from repository -->
  </parent>
  <properties>
    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
    <java.version>1.8</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-thymeleaf</artifactId>
  </dependency>
  <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-web</artifactId>
  </dependency>
  <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>
```

Apeçu des dépendances maven

Maven Dependencies		
spring-boot-starter-data-jpa-1.3.0.RELEASE.jar	- C:\Users\y	mysql-connector-java-5.1.37.jar
spring-boot-starter-1.3.0.RELEASE.jar	- C:\Users\y	spring-boot-starter-test-1.3.0.RELEASE.jar
spring-boot-1.3.0.RELEASE.jar	- C:\Users\y	junit-4.12.jar
spring-boot-autoconfigure-1.3.0.RELEASE.jar	- C:\Users\y	mockito-core-1.10.19.jar
spring-boot-starter-logging-1.3.0.RELEASE.jar	- C:\Users\y	objenesis-2.1.jar
logback-classic-1.1.3.jar	- C:\Users\y	hamcrest-core-1.3.jar
logback-core-1.1.3.jar	- C:\Users\y	hamcrest-library-1.3.jar
jul-to-slf4j-1.7.13.jar	- C:\Users\y	spring-core-4.2.3.RELEASE.jar
log4j-over-slf4j-1.7.13.jar	- C:\Users\y	spring-test-4.2.3.RELEASE.jar
snakeyaml-1.16.jar	- C:\Users\y	
spring-boot-starter-aop-1.3.0.RELEASE.jar	- C:\Users\y	
spring-aop-4.2.3.RELEASE.jar	- C:\Users\y	
aopalliance-1.0.jar	- C:\Users\y	
aspectjweaver-1.8.7.jar	- C:\Users\y	
spring-boot-starter-jdbc-1.3.0.RELEASE.jar	- C:\Users\y	
tomcat-jdbc-8.0.28.jar	- C:\Users\y	
tomcat-juli-8.0.28.jar	- C:\Users\y	
spring-jdbc-4.2.3.RELEASE.jar	- C:\Users\y	
hibernate-entitymanager-4.3.11.Final.jar	- C:\Users\y	
jboss-logging-3.3.0.Final.jar	- C:\Users\y	
jboss-logging-annotations-1.2.0.Beta1.jar	- C:\Users\y	
hibernate-core-4.3.11.Final.jar	- C:\Users\y	
antlr-2.7.7.jar	- C:\Users\y	
jandex-1.1.0.Final.jar	- C:\Users\y	
dom4j-1.6.1.jar	- C:\Users\y	
xml-apis-1.0.b2.jar	- C:\Users\y	
hibernate-commons-annotations-4.0.5.Final.jar	- C:\Users\y	
hibernate-jpa-2.1-api-1.0.0.Final.jar	- C:\Users\y	
javassist-3.18.1-GA.jar	- C:\Users\y	
javax.transaction-api-1.2.jar	- C:\Users\y	
spring-data-jpa-1.9.1.RELEASE.jar	- C:\Users\y	
spring-data-commons-1.11.1.RELEASE.jar	- C:\Users\y	
spring-orm-4.2.3.RELEASE.jar	- C:\Users\y	
spring-context-4.2.3.RELEASE.jar	- C:\Users\y	
spring-tx-4.2.3.RELEASE.jar	- C:\Users\y	
spring-beans-4.2.3.RELEASE.jar	- C:\Users\y	
slf4j-api-1.7.13.jar	- C:\Users\y	
jcl-over-slf4j-1.7.13.jar	- C:\Users\y	
spring-aspects-4.2.3.RELEASE.jar	- C:\Users\y	
spring-boot-starter-thymeleaf-1.3.0.RELEASE.jar	- C:\Users\y	
thymeleaf-spring4-2.1.4.RELEASE.jar	- C:\Users\y	
thymeleaf-2.1.4.RELEASE.jar	- C:\Users\y	
ognl-3.0.8.jar	- C:\Users\y	
unbescape-1.1.0.RELEASE.jar	- C:\Users\y	
thymeleaf-layout-dialect-1.3.1.jar	- C:\Users\y	
groovy-2.4.4.jar	- C:\Users\y	
spring-boot-starter-web-1.3.0.RELEASE.jar	- C:\Users\y	
spring-boot-starter-tomcat-1.3.0.RELEASE.jar	- C:\Users\y	
tomcat-embed-core-8.0.28.jar	- C:\Users\y	
tomcat-embed-el-8.0.28.jar	- C:\Users\y	
tomcat-embed-logging-juli-8.0.28.jar	- C:\Users\y	
tomcat-embed-websocket-8.0.28.jar	- C:\Users\y	
spring-boot-starter-validation-1.3.0.RELEASE.jar	- C:\Users\y	
hibernate-validator-5.2.2.Final.jar	- C:\Users\y	
validation-api-1.1.0.Final.jar	- C:\Users\y	
classmate-1.1.0.jar	- C:\Users\y	
jackson-databind-2.6.3.jar	- C:\Users\y	
jackson-annotations-2.6.3.jar	- C:\Users\y	
jackson-core-2.6.3.jar	- C:\Users\y	
spring-web-4.2.3.RELEASE.jar	- C:\Users\y	
spring-webmvc-4.2.3.RELEASE.jar	- C:\Users\y	
spring-expression-4.2.3.RELEASE.jar	- C:\Users\y	

Structure du projet



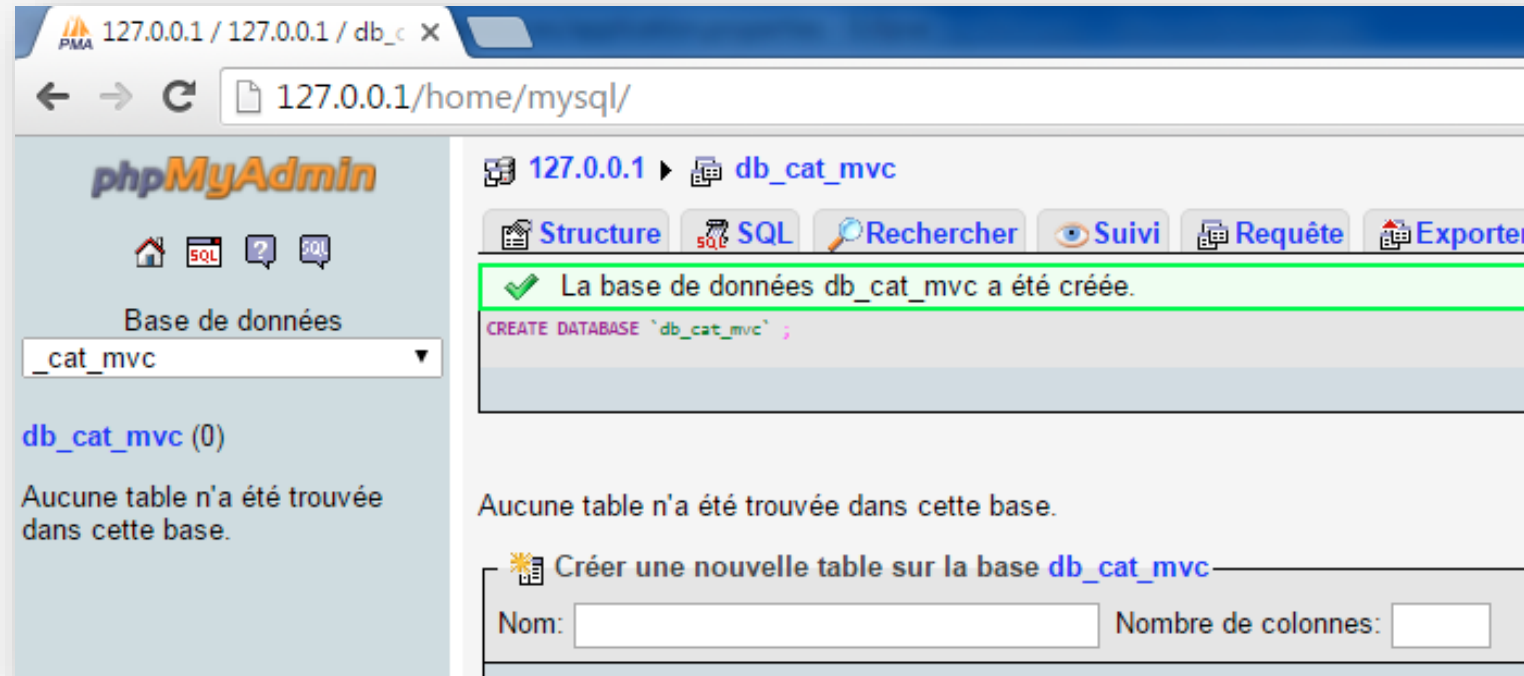
Entité Produit

```
package org.sid.entities;
import java.io.Serializable; import javax.persistence.Entity;
import javax.persistence.GeneratedValue; import javax.persistence.Id;
import javax.validation.constraints.DecimalMin;
import javax.validation.constraints.NotNull;
import javax.validation.constraints.Size;
@Entity
public class Produit implements Serializable {
    @Id @GeneratedValue
    private Long idProduit;
    @NotNull
    @Size(min=5,max=12)
    private String designation;
    @DecimalMin(value="100")
    private double prix;
    // getters et setters
    // Constructeur par défaut
    // Constructeur avec params
}
```

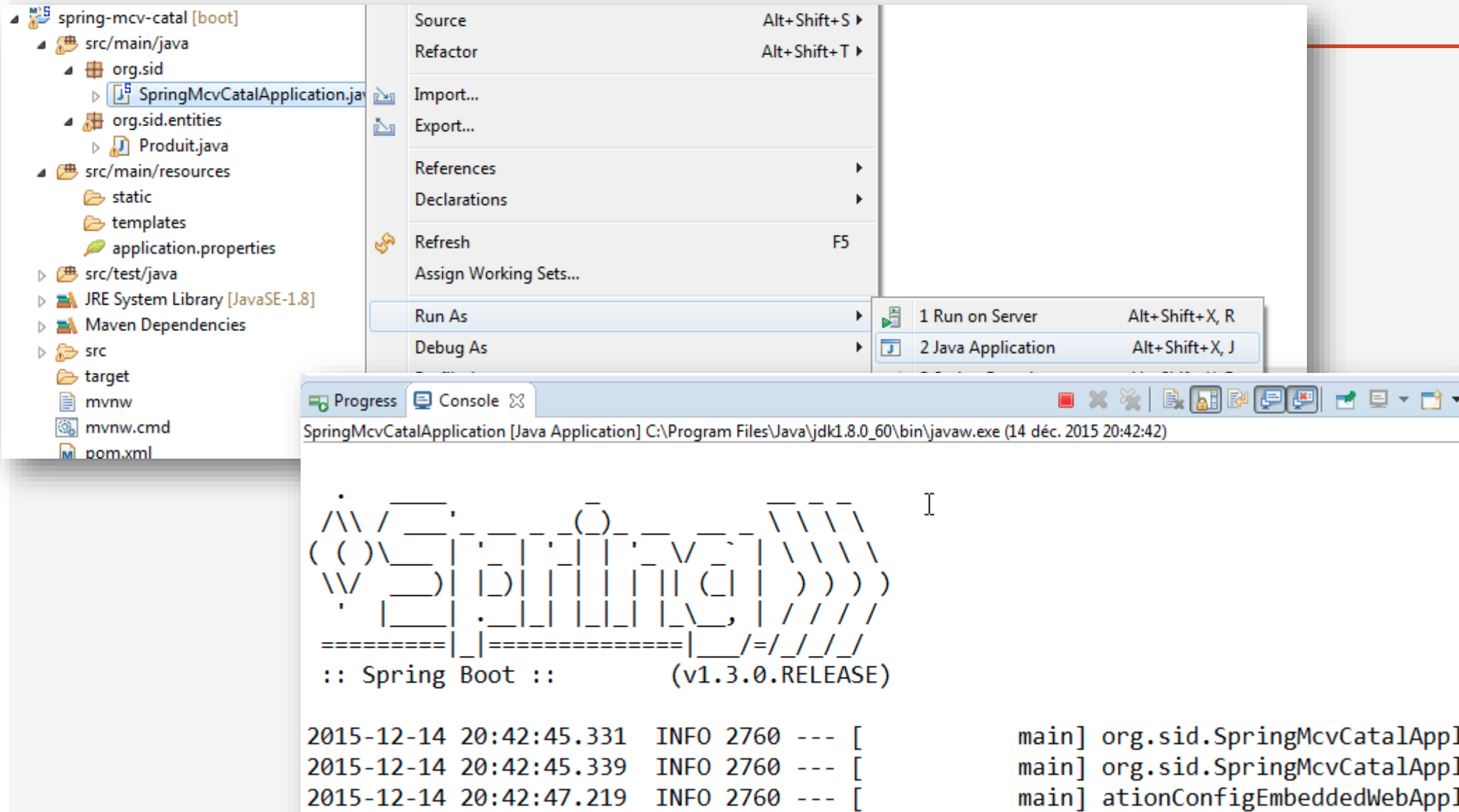
Déployer le data source : application.properties

```
spring.datasource.url = jdbc:mysql://localhost:3306/db_cat_mvc
spring.datasource.username = root
spring.datasource.password =
spring.datasource.driverClassName = com.mysql.jdbc.Driver
spring.jpa.show-sql = true
spring.jpa.hibernate.ddl-auto = update
spring.jpa.properties.hibernate.dialect = org.hibernate.dialect.MySQL5Dialect
```

Créer la base de données



Démarrer l'application Spring Boot



Vérifier si la table produits a été bien générée

```
Progress Console
SpringMvcCatalApplication [Java Application] C:\Program Files\Java\jdk1.8.0_60\bin\javaw.exe (14 déc. 2015 20:42:42)
TranslatorFactory : HHH000397: using ASTQueryTranslatorFactory
2ddl.SchemaUpdate : HHH000228: Running hbm2ddl schema update
2ddl.SchemaUpdate : HHH000102: Fetching database metadata
2ddl.SchemaUpdate : HHH000396: Updating schema
ata : HHH000262: Table not found: produit
ata : HHH000262: Table not found: produit
ata : HHH000262: Table not found: produit
2ddl.SchemaUpdate : HHH000232: Schema update complete
```

127.0.0.1 / 127.0.0.1 / db_cat_mvc

127.0.0.1/home/mysql/

phpMyAdmin
















Base de données
_cat_mvc (1)






db_cat_mvc (1)

produit

127.0.0.1 > db_cat_mvc > produit

Afficher Structure SQL Rechercher Suivi Insérer Exporter Importer Opérations Vider Supprimer



	Colonne	Type	Interclassement	Attributs	Null	Défaut	Extra	Action
<input type="checkbox"/>	id_produit	bigint(20)			Non	Aucun	AUTO_INCREMENT	    
<input type="checkbox"/>	designation	varchar(255)	latin1_swedish_ci		Oui	NULL		    
<input type="checkbox"/>	prix	double			Non	Aucun		    

Tout cocher / Tout décocher Pour la sélection :     

Version imprimable Gestion des relations Suggérer des optimisations quant à la structure de la table Suivre la table

Ajouter 1 colonne(s) ☐ En fin de table ☐ En début de table ☐ Après id_produit Exécuter

Index: ?

Action	Nom de l'index	Type	Unique	Compressé	Colonne	Cardinalité	Interclassement	Null	Commentaire
 	PRIMARY	BTREE	Oui	Non	id_produit	0	A		

Couche DAO avec Spring data








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

Tester la couche DAO

```
package org.sid;
import java.util.List; import org.sid.dao.ProduitRepository;
import org.sid.entities.Produit; import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.context.ApplicationContext;
@SpringBootApplication
public class SpringMvcCatalApplication {
    public static void main(String[] args) {
        ApplicationContext ctx=SpringApplication.run(SpringMvcCatalApplication.class, args);
        ProduitRepository dao=ctx.getBean(ProduitRepository.class);
        // Ajouter quelques produits
        dao.save(new Produit("TR342",540));          dao.save(new Produit("HR 4378",540));
        dao.save(new Produit("AXL 123",540));
        // Consulter tous les produits
        System.out.println("-----");
        List<Produit> prods=dao.findAll();
        for(Produit p:prods){    System.out.println(p.getDesignation()+"--"+p.getPrix());    }
        // Consulter un produit
        System.out.println("-----");
        Produit p=dao.findOne(2L);
        System.out.println(p.getDesignation()+"--"+p.getPrix());
    }
}
```


Exécution

```
2015-12-11 21:01:05.992 INFO 5510 [main] org.hibernate
Hibernate: insert into produit (designation, prix) values (?, ?)
Hibernate: insert into produit (designation, prix) values (?, ?)
Hibernate: insert into produit (designation, prix) values (?, ?)
-----
Hibernate: select produit0_.id_produit as id_produ1_0_, produit0_.de
TR342--540.0
HR 4378--540.0
AXL 123--540.0
-----
Hibernate: select produit0_.id_produit as id_produ1_0_0_, produit0_
HR 4378--540.0
```

			id_produit	designation	prix
<input type="checkbox"/>			1	TR342	540
<input type="checkbox"/>			2	HR 4378	540
<input type="checkbox"/>			3	AXL 123	540

Ajouter des méthodes à l'interface JpaRepository

Une méthode pour consulter les produits sachant la désignation

Une méthode qui permet de retourner une page de produits ont la désignation contient un mot clé:

```
package org.sid.dao;
import java.util.List;
import org.sid.entities.Produit;
import org.springframework.data.domain.Page;
import org.springframework.data.domain.Pageable;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.data.jpa.repository.Query;
import org.springframework.data.repository.query.Param;

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

Tester les méthodes

```
System.out.println("-----");
Page<Produit> pageProduits=dao.chercherProduits("%R%", new PageRequest(0, 2));
System.out.println("Page numéro:"+pageProduits.getNumber());
System.out.println("Nombre de produits:"+pageProduits.getNumberOfElements());
System.out.println("Page numéro:"+pageProduits.getTotalPages());
System.out.println("Total produits:"+pageProduits.getTotalElements());
for(Produit pr:pageProduits.getContent()){
System.out.println(pr.getDesignation());
}
```

```
Page numéro:0
Nombre de produits:2
Page numéro:3
Total produits:6
TR342
HR 4378
```

Couche Web

Insert title here x

localhost:8080/chercher?mc=H

Mot Clé:

ID	DES	PRIX
2	HLX	54300.0
4	HP870	3400.0
6	HLX	54300.0
8	HP870	3400.0
10	HLX	54300.0

0 1 2

Saisie d'un produit x

localhost:8080/form

Désignation:

Prix:

Saisie d'un produit x

localhost:8080/saveProduit

Désignation:

la taille doit être entre 5 et 12

Prix:

doit être supérieur ou égal à 100

Saisie d'un produit x

localhost:8080/saveProduit

Confirmation

ID: 50

Désignation: prod78

Prix: 400.0

Contrôleur

```
package org.sid.web;
import javax.validation.Valid;
import org.sid.dao.ProduitRepository;
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.ui.Model;
import org.springframework.validation.BindingResult;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;
import org.springframework.web.bind.annotation.RequestParam;

@Controller
public class ProduitController {
    @Autowired
    private ProduitRepository produitRepository;
```

Contrôleur

```
@RequestMapping(value="/chercher",method=RequestMethod.GET)
public String chercher(Model model,
    @RequestParam(name="mc",defaultValue="")String motCle,
    @RequestParam(name="page",defaultValue="0")int page){

    Page<Produit> pageProduits=
        produitRepository.chercherProduits("%"+motCle+"%", new PageRequest(page, 5));
    model.addAttribute("pageProduit", pageProduits);
    model.addAttribute("pageCourante", page);
    model.addAttribute("mc",motCle);
    int[] pages=new int[pageProduits.getTotalPages()];
    for(int i=0;i<pages.length;i++) pages[i]=i;
        model.addAttribute("pages", pages);
    return "produits";
}
```

Contrôleur

```
@RequestMapping(value="/form")
public String formProduit(Model model){
    model.addAttribute("produit", new Produit());
    return "formProduit";
}

@RequestMapping(value="/saveProduit",method=RequestMethod.POST)
public String save(Model model,@Valid Produit p, BindingResult
bindingResult){
    if(bindingResult.hasErrors()){
        return "formProduit";
    }
    produitRepository.save(p);
    model.addAttribute("produit", p);
    return "confirmation";
}
}
```

Vue : produits.html

```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org">
<head>
<meta charset="utf-8"/>
<title>Insert title here</title>
<link rel="stylesheet" type="text/css" href="../../static/css/bootstrap.min.css"
th:href="@{css/bootstrap.min.css}"/>
<link rel="stylesheet" type="text/css" href="../../static/css/myStyle.css"
th:href="@{css/myStyle.css}"/>
</head>
<body>
<div class="container spacer">
  <form action="chercher?page=0" method="GET">
    <label>Mot Clé:</label>
    <input type="text" name="mc" th:value="${mc}"/>
    <input type="submit" value="Chercher"/>
  </form>
</div>
```


Vue : formProduit.html

```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org">
<head>
<meta charset="utf-8"/>
<title>Saisie d'un produit</title>
  <link rel="stylesheet" type="text/css" href="../../static/css/bootstrap.min.css"
  th:href="@{css/bootstrap.min.css}"/>
  <link rel="stylesheet" type="text/css" href="../../static/css/myStyle.css"
  th:href="@{css/myStyle.css}"/>
</head>
```

Vue : formProduit.html

```
<body>
  <div class="col-md-6 col-sm-6 col-xs-12">
    <form th:action="saveProduit" method="post">
      <div th:object="{product}">
        <div class="form-group">
          <label class="control-label">Désignation:</label>
          <input class="form-control" type="text" name="designation" th:value="{designation}"/>
          <span class="error" th:errors="{designation}"></span>
        </div>
        <div class="form-group">
          <label class="control-label">Prix:</label>
          <input class="form-control" type="text" name="prix" th:value="{prix}"/>
          <span class="error" th:errors="{prix}"></span>
        </div>
        <div>
          <input class="btn btn-primary" type="submit" value="Save"/>
        </div>
      </div>
    </form>
  </div>
</body>
</html>
```

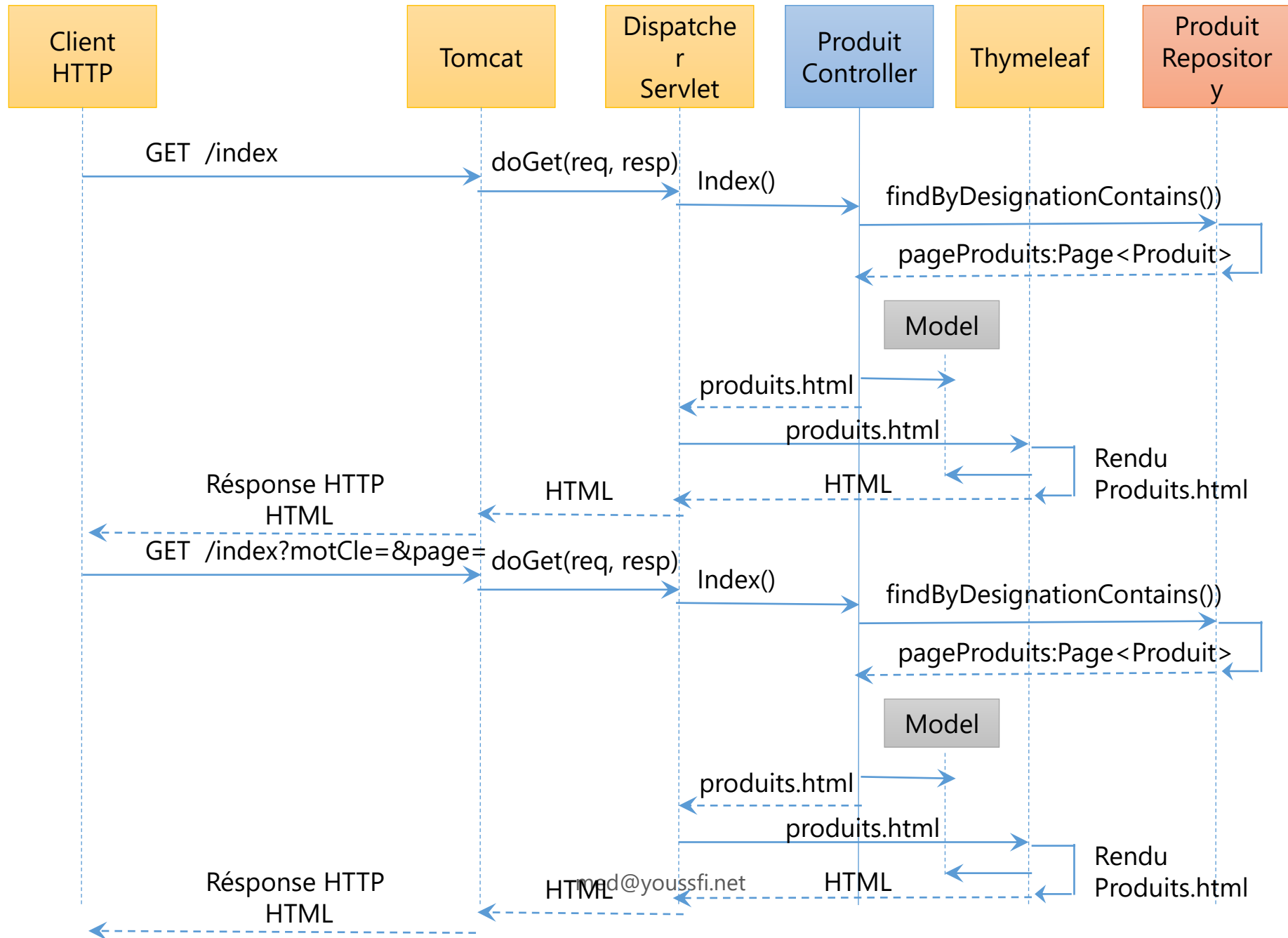
Vue :

confirmation.html

```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org">
<head>
<meta charset="utf-8"/>
<title>Saisie d'un produit</title>
  <link rel="stylesheet" type="text/css" href="../../static/css/bootstrap.min.css"
  th:href="@{css/bootstrap.min.css}"/>
  <link rel="stylesheet" type="text/css" href="../../static/css/myStyle.css"
  th:href="@{css/myStyle.css}"/>
</head>
```

Vue : confirmation.html

```
<body>
<div class="col-md-6 col-sm-6 col-xs-12">
<div class="panel panel-info spacer">
  <div class="panel-heading">Confirmation </div>
  <div class="panel-body">
    <div class="form-group">
      <label class="control-label">ID:</label>
      <label class="control-label" th:text="${produit.idProduit}"></label>
    </div>
    <div class="form-group">
      <label class="control-label">Désignation:</label>
      <label class="control-label" th:text="${produit.designation}"></label>
    </div>
    <div class="form-group">
      <label class="control-label">Prix:</label>
      <label class="control-label" th:text="${produit.prix}"></label>
    </div>
  </div>
</div>
</div>
</body>
</html>
```





VUES DE L'APPLICATION BASÉES SUR LE MOTEUR DE TEMPLATE TYMELEAF

Utilisation des Layouts : Templates

Généralement toutes les page d'une application web partagent le même contenu html (Header, footer, menus, etc..)

Pour éviter des faire des copies coller dans toutes les pages, il est important de définir une page Template qui définit

- Toutes les parties fixes de toutes les pages (header, footer, menus, etc...)
- Déclarer les sections qui changeront de contenu en fonction de chaque page.

Exemple de template: layout.html

```
<!DOCTYPE html>
<html
  xmlns:th="http://www.thymeleaf.org"
  xmlns:layout="http://www.ultraq.net.nz/thymeleaf/layout">
<head>
  <meta charset="utf-8"/>
  <title>Gestion des produits</title>
  <link rel="stylesheet" type="text/css" href="../../static/css/bootstrap.min.css"
    th:href="@{/css/bootstrap.min.css}" />
  <link rel="stylesheet" type="text/css" href="../../static/css/style.css"
    th:href="@{/css/style.css}" />
</head>
<body>
  <header> Header FIXE </header>
  <section layout:fragment="content">

</section>
  <footer class="navbar-fixed-bottom">
    </footer>FOOTER FIXE
</body>
</html>
```

HEADER FIXE

CONTENT VARIABLE

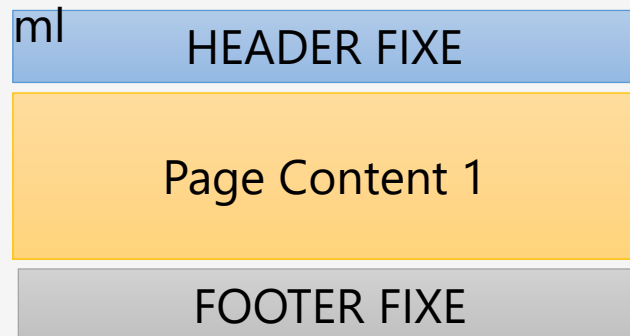
FOOTER FIXE

Utilisation du layout dans une page

page1.html

```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org"
      xmlns:layout="http://www.ultraq.net.nz/thymeleaf/layout"
      layout:decorator="layout" >
  <body>
    <div layout:fragment="content">
      Page 1 Content
    </div>
  </body>
</html>
```

layout.html



Thymeleaf Layout Dialect

Maven Dependency

Pour Spring Boot 2.x

```
<dependency>  
    <groupId>nz.net.ultraq.thymeleaf</groupId>  
    <artifactId>thymeleaf-layout-dialect</artifactId>  
</dependency>
```

Suite de l'application

```
cata_mvc_1 [boot] [devtools]
├── Spring Elements
├── src/main/java
│   ├── org.sid
│   │   ├── CataMvc1Application.java
│   │   ├── org.sid.dao
│   │   │   ├── ProduitRepository.java
│   │   ├── org.sid.entities
│   │   │   ├── Produit.java
│   │   ├── org.sid.security
│   │   │   ├── SecurityConfig.java
│   │   ├── org.sid.web
│   │   │   ├── ProduitController.java
│   ├── src/main/resources
│   │   ├── static
│   │   │   ├── css
│   │   │   │   ├── bootstrap.min.css
│   │   │   │   └── style.css
│   │   └── templates
│   │       ├── Confirmation.html
│   │       ├── EditProduit.html
│   │       ├── FormProduit.html
│   │       ├── layout.html
│   │       ├── login.html
│   │       └── produits.html
│   └── application.properties
├── src/test/java
├── JRE System Library [JavaSE-1.8]
├── Maven Dependencies
├── src
├── target
│   ├── mvnw
│   ├── mvnw.cmd
│   └── pom.xml
```

Authentification

User Name:

admin

Pass Word:

.....

Login

localhost:8080/produits/index?mc=HP

Chercher Produits admin

Mot Clé: HP Chercher

Liste des produits

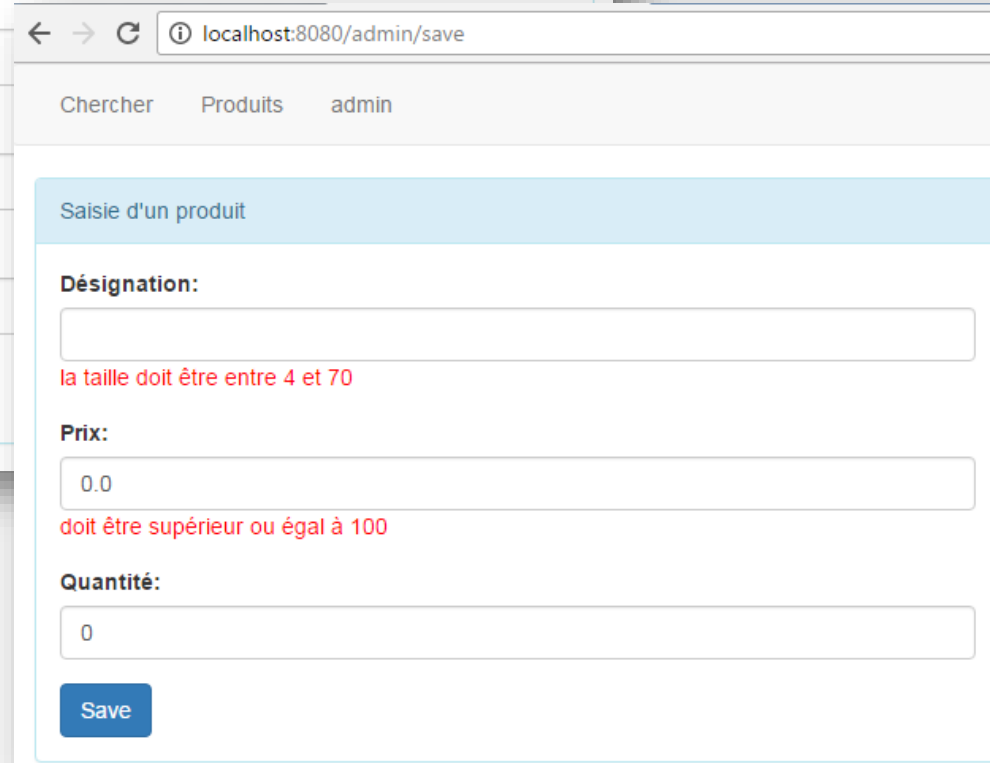
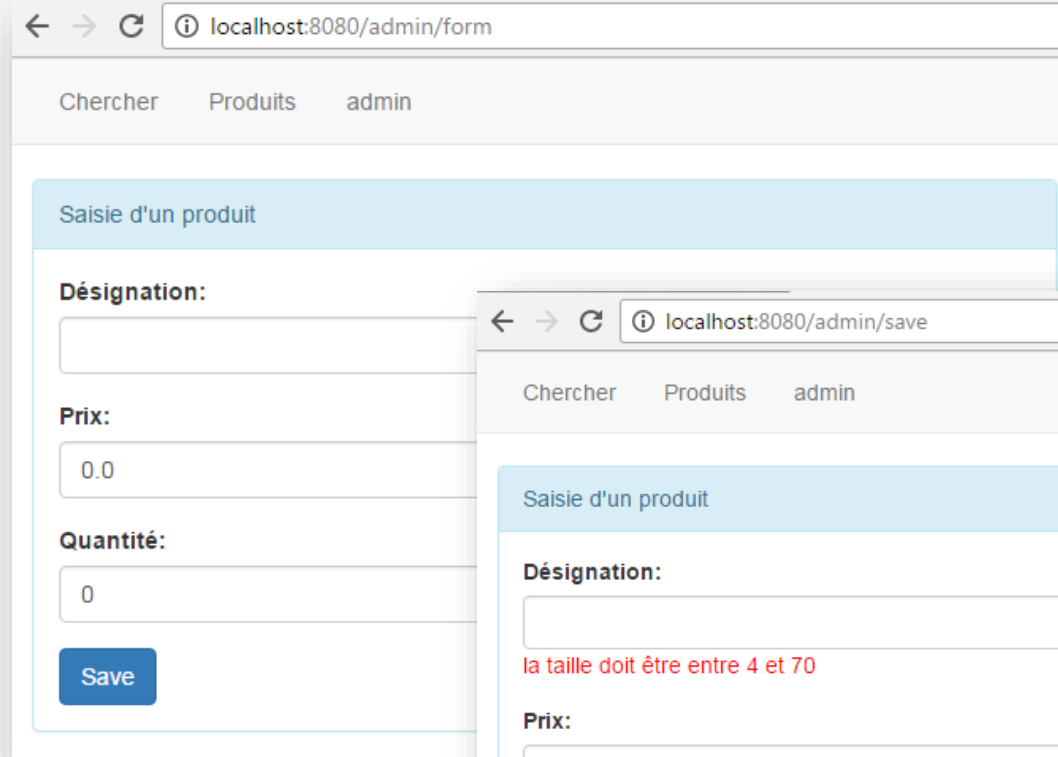
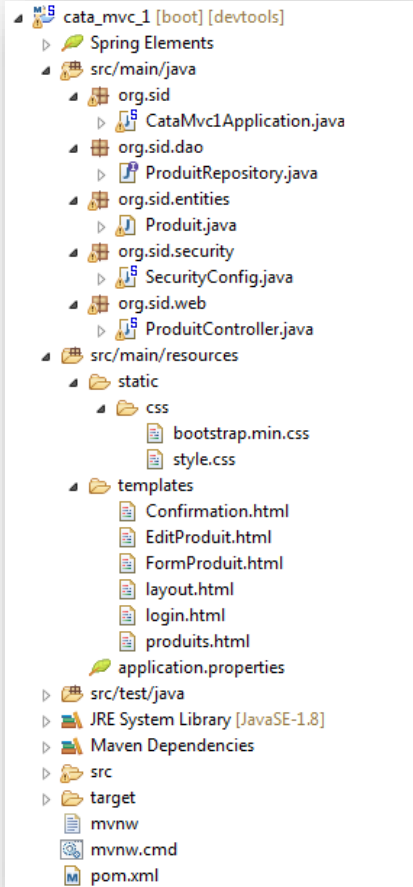
ID	Désignation	Prix	Quantité		
2	Imprimante HP 548	300.0	14	Edit	Delete
3	Ordinateur HP 87	200.0	23	Edit	Delete
6	Ordinateur HP 87 9000	800.0	23	Edit	Delete
8	Imprimante HP 548	300.0	14	Edit	Delete

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24

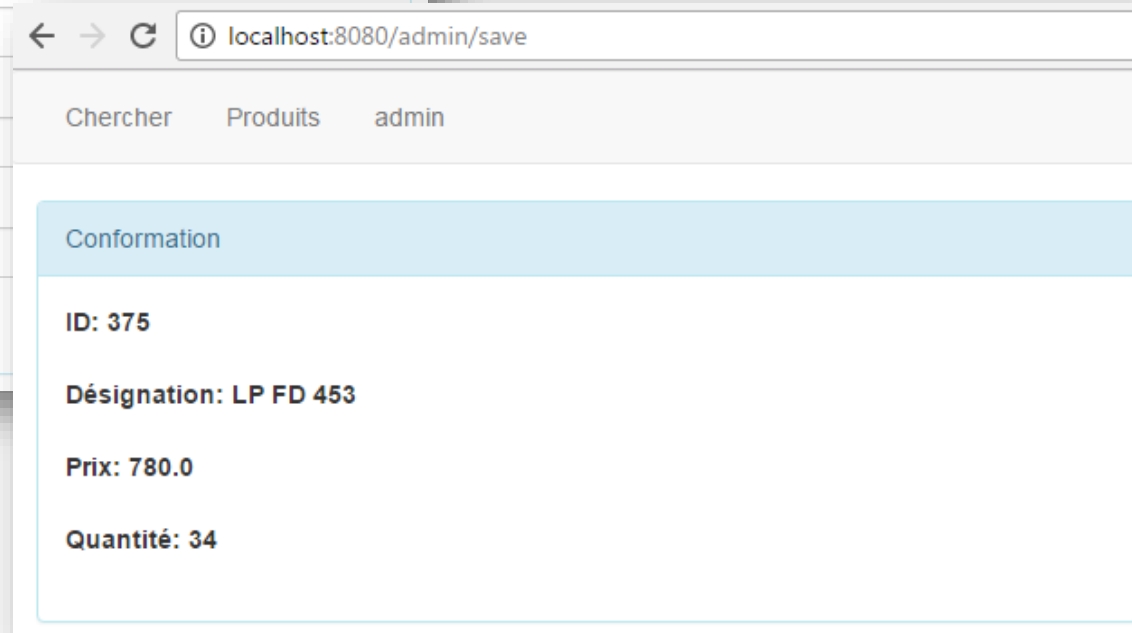
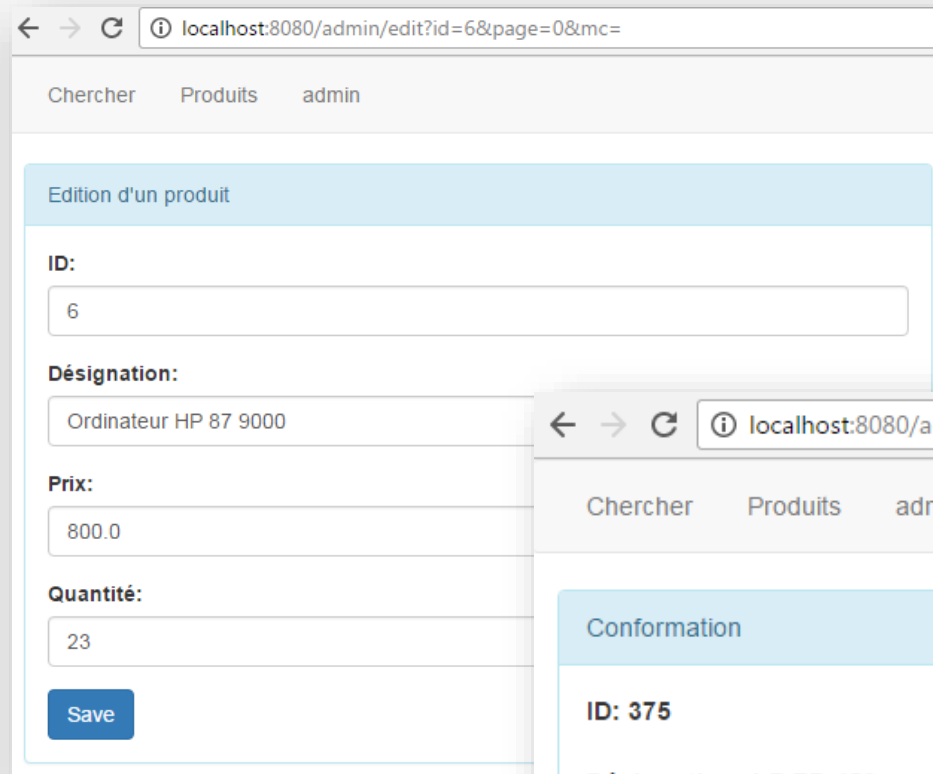
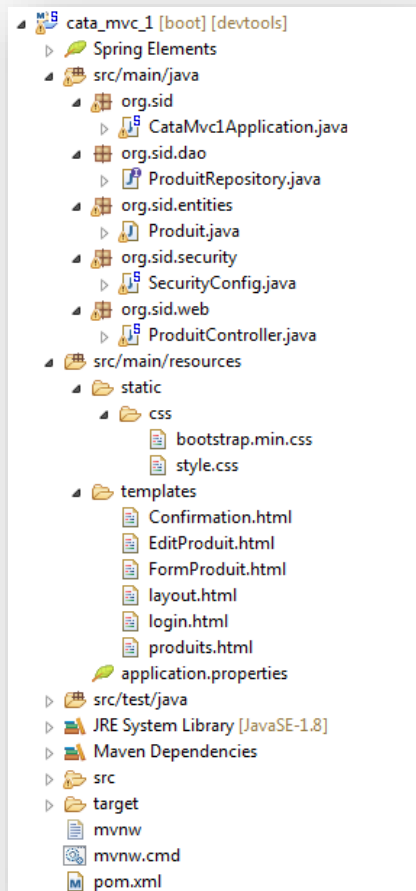
25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47

48 49 50 51 52 53 54 55 56 57 58

Suite de l'application



Suite de l'application



Entité Produit

```
package org.sid.entities;

import java.io.Serializable; import javax.persistence.Entity;
import javax.persistence.GeneratedValue; import javax.persistence.Id;
import javax.validation.constraints.DecimalMin;import javax.validation.constraints.NotNull;
import javax.validation.constraints.Size;

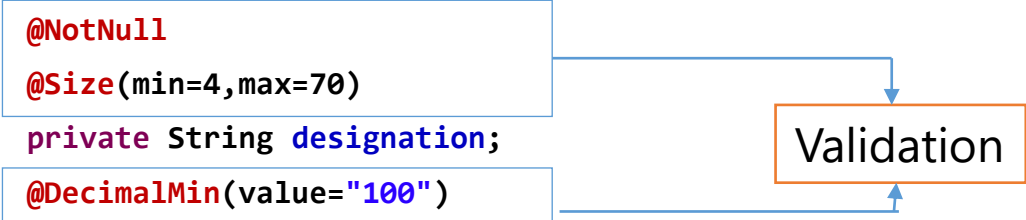
@Entity
public class Produit implements Serializable {
    @Id @GeneratedValue
    private Long id;

    @NotNull
    @Size(min=4,max=70)
    private String designation;

    @DecimalMin(value="100")
    private double prix;
    private int quantite;

    // Constructeurs
    public Produit() {}
    public Produit(String designation, double prix, int quantite) {
        this.designation = designation; this.prix = prix; this.quantite = quantite;
    }

    // Getters et Setters
}
```



```
graph LR
    A["@NotNull  
@Size(min=4,max=70)"] --> C[Validation]
    B["@DecimalMin(value=100)"] --> C
```

Interface ProduitRepository

```
package org.sid.dao;

import org.sid.entities.Produit;
import org.springframework.data.domain.Page;
import org.springframework.data.domain.Pageable;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.data.jpa.repository.Query;
import org.springframework.data.repository.query.Param;

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


Déployer le data source : application.properties

```
#security.user.name=admin
```

```
#security.user.password=123
```

```
spring.datasource.url = jdbc:mysql://localhost:3306/db_cat_mvc_1
```

```
spring.datasource.username = root
```

```
spring.datasource.password =
```

```
spring.datasource.driverClassName = com.mysql.jdbc.Driver
```

```
spring.jpa.show-sql = true
```

```
spring.jpa.hibernate.ddl-auto = update
```

Application Spring Boot

```
package org.sid;

import org.sid.dao.ProduitRepository; import org.sid.entities.Produit;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

import org.springframework.context.ApplicationContext;

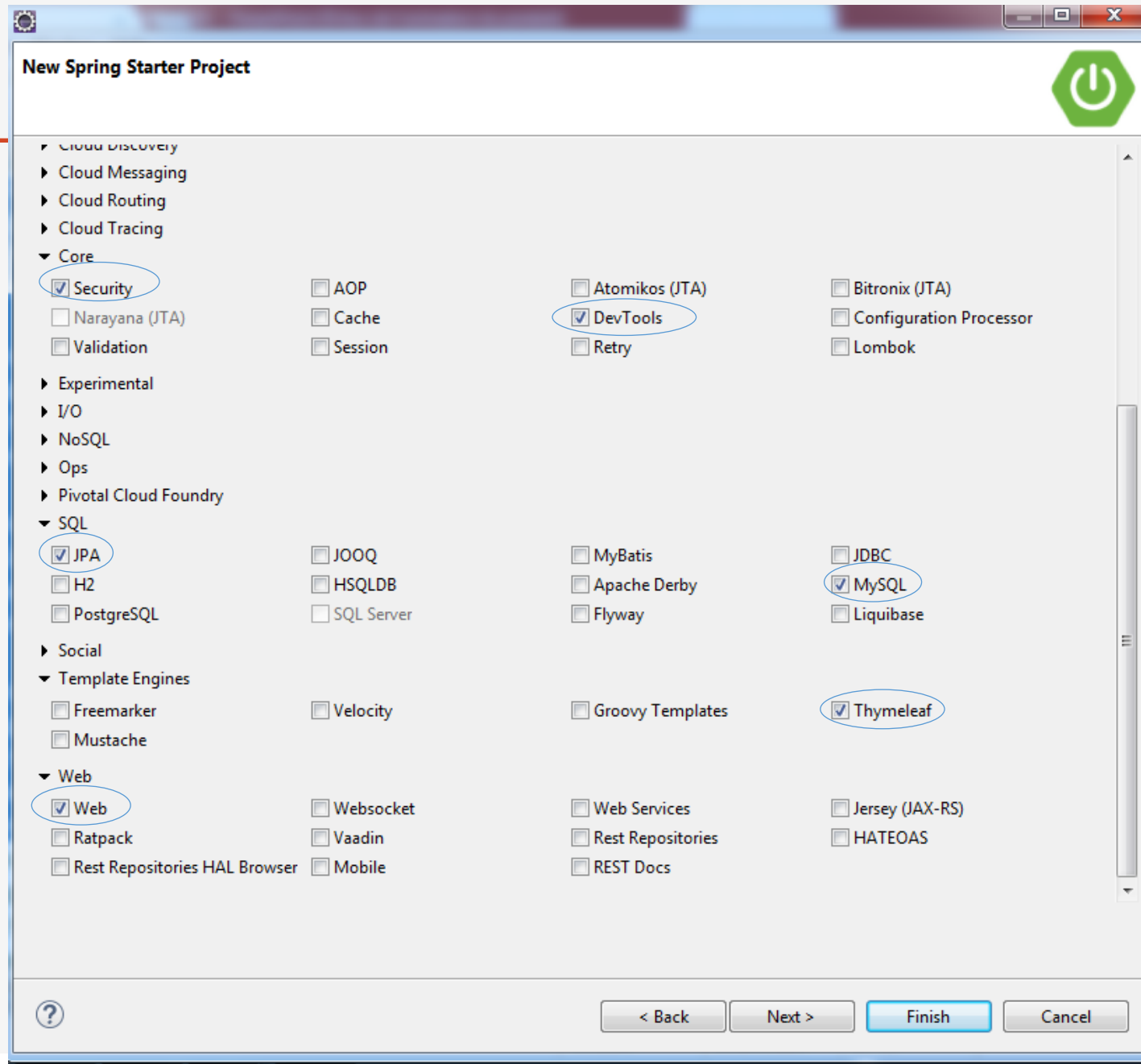
@SpringBootApplication

public class CataMvc1Application {

    public static void main(String[] args) {

        ApplicationContext ctx=SpringApplication.run(CataMvc1Application.class, args);
```

Dependencies Maven



The image shows a 'New Spring Starter Project' dialog box with a green power icon in the top right corner. The left sidebar contains a tree view with categories: Cloud Discovery, Cloud Messaging, Cloud Routing, Cloud Tracing, Core, Experimental, I/O, NoSQL, Ops, Pivotal Cloud Foundry, SQL, Social, Template Engines, and Web. The main area displays a grid of dependency checkboxes. Several dependencies are selected and circled in blue: Security, JPA, Web, DevTools, MySQL, and Thymeleaf. The bottom of the dialog features a help icon, and navigation buttons: '< Back', 'Next >', 'Finish' (highlighted in blue), and 'Cancel'.

New Spring Starter Project

Cloud Discovery
Cloud Messaging
Cloud Routing
Cloud Tracing
Core
Experimental
I/O
NoSQL
Ops
Pivotal Cloud Foundry
SQL
Social
Template Engines
Web

☒ Security
☐ Narayana (JTA)
☐ Validation

☐ AOP
☐ Cache
☐ Session

☐ Atomikos (JTA)
☒ DevTools
☐ Retry

☐ Bitronix (JTA)
☐ Configuration Processor
☐ Lombok

☒ JPA
☐ H2
☐ PostgreSQL

☐ JOOQ
☐ HSQLDB
☐ SQL Server

☐ MyBatis
☐ Apache Derby
☐ Flyway

☐ JDBC
☒ MySQL
☐ Liquibase

☐ Freemarker
☐ Mustache

☐ Velocity

☐ Groovy Templates

☒ Thymeleaf

☒ Web
☐ Ratpack
☐ Rest Repositories HAL Browser

☐ Websocket
☐ Vaadin
☐ Mobile

☐ Web Services
☐ Rest Repositories
☐ REST Docs

☐ Jersey (JAX-RS)
☐ HATEOAS

? < Back Next > Finish Cancel

Dependencies Maven

```
<dependency>
```

```
  <groupId>org.springframework.boot</groupId>
```

```
  <artifactId>spring-boot-starter-data-jpa</artifactId>
```

```
</dependency>
```

```
<dependency>
```

```
  <groupId>org.springframework.boot</groupId>
```

```
  <artifactId>spring-boot-starter-thymeleaf</artifactId>
```

```
</dependency>
```

```
<dependency>
```

```
  <groupId>org.springframework.boot</groupId>
```

```
  <artifactId>spring-boot-starter-web</artifactId>
```

```
</dependency>
```

ProduitController

```
package org.sid.web;

import javax.validation.Valid;

import org.sid.dao.ProduitRepository;

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.ui.Model;

import org.springframework.validation.BindingResult;
```

ProduitController : Chercher les produits

```
@Controller
```

```
public class ProduitController {
```

```
@Autowired
```

```
    private ProduitRepository produitRepository;
```

```
@RequestMapping(value="/produits/index")
```

```
public String produits(Model model,
```

```
    @RequestParam(name="page",defaultValue="0")int page,
```

```
    @RequestParam(name="size",defaultValue="4")int size,
```

```
    @RequestParam(name="mc",defaultValue="")String mc){
```

```
    //List<Produit> produits=produitRepository.findAll();
```

med@youssfj.net

Vue : produits.html

```
<html xmlns:th="http://www.thymeleaf.org"
```

```
xmlns:layout="http://www.ultraq.net.nz/thymeleaf/layout"
```

```
layout:decorator="template1">
```

Formulaire de recherche

```
<div layout:fragment="content">
```

```
<div class="container">
```

```
<form th:action="@{/user/index}" method="get">
```

Tableau des produits

```
<label>Mot Clé:</label> <input type="text" name="motCle" th:value="${motCle}"/>
```

```
<button class="btn btn-primary">Chercher</button>
```

```
</form>
```

```
</div>
```

```
<div class="con
```

```
<div class="container">
```

```
<ul class="nav nav-pills">
```

```
<li th:class="${pageCourante}==${status.index}?'active': ''"
```

```
th:each="pa,status:${pages}" >
```

```
<a th:href="@{/user/index(page=${status.index},size=${size},motCle=${motCle})}"
```

```
th:text="${status.index}"></a>
```

```
</li>
```

```
</ul>
```

```
</div>
```

Pagination

```
<table class="table">
```

Suppression d'un produit

ProduitController.java

```
@RequestMapping(value="/admin/delete",method=RequestMethod.GET)
```

```
public String delete(Long id,String mc,int page){
```

```
    produitRepository.delete(id);
```

```
    return "redirect:/produits/index?page="+page+"&mc="+mc;
```

```
}
```

produits.html

```
<a th:href="@{/admin/delete(id=${p.id})}">  
    Delete  
</a>
```


Saisie et Ajout d'un produit avec Valisation

```
@RequestMapping(value="/admin/form",method=RequestMethod.GET)
```

ProduitController.java

```
public String form(Model model){  
  
    model.addAttribute("produit", new Produit());  
  
    return "FormProduit";  
  
}
```

```
@RequestMapping(value="/admin/save",method=RequestMethod.POST)
```

@Entity

Produit.java

```
public class Produit implements Serializable{  
    @Id @GeneratedValue  
    private Long id;  
  
    @NotNull  
    @Size(min=4,max=70)  
    private String designation;  
  
    @DecimalMin(value="100")  
    private double prix;  
    private int quantite;  
  
}
```

```
<form th:action="@{/admin/save}" method="post">  
    <label class="control-label">Désignation:</label>  
    <input type="text" name="designation" th:value="${produit.designation}"/>  
    <span th:errors="${produit.designation}" ></span>  
  
    <label>Prix:</label>  
    <input type="text" name="prix" th:value="${produit.prix}"/>  
    <span th:errors="${produit.prix}" ></span>  
  
    <label>Quantité:</label>  
    <input type="text" name="quantite" th:value="${produit.quantite}"/>  
    <span th:errors="${produit.quantite}" ></span>  
  
    <button type="submit">Save</button>  
</form>
```

FormProduit.html

Confirmation.html

FormProduit.html

```
<div layout:fragment="content">
  <div class="col-md-6 col-md-offset-3 col-xs-12">
    <div class="panel panel-primary">
      <div class="panel-heading">Confirmation</div>
      <div class="panel-body">
        <div class="form-group">
          <label class="control-label">ID:</label>
          <label class="control-label" th:inline="text">[[${produit.id}]]</label>
        </div>

        <div class="form-group">
          <label class="control-label">Désignation:</label>
          <label class="control-label" th:inline="text">[[${produit.designation}]]</label>
        </div>

        <div class="form-group">
          <label class="control-label">Pix:</label>
          <label class="control-label" th:inline="text">[[${produit.prix}]]</label>
        </div>

        <div class="form-group">
          <label class="control-label">Quantité:</label>
          <label class="control-label" th:inline="text">[[${produit.quantite}]]</label>
        </div>
      </div>
    </div>
  </div>
</div>
```

Edition d'un produit

ProduitController.java

```
@RequestMapping(value="/admin/edit",method=RequestMethod.GET)
```

```
public String edit(Model model,Long id){
```

```
    Produit p=produitRepository.findOne(id);
```

```
<form th:action="@{/admin/save}" method="post">
    <label class="control-label">ID:</label>
    <input type="text" name="id" th:value="${produit.id}"/>
    <span th:errors="${produit.id}" ></span>

    <label class="control-label">Désignation:</label>
    <input type="text" name="designation" th:value="${produit.designation}"/>
    <span th:errors="${produit.designation}" ></span>

    <label>Prix:</label>
    <input type="text" name="prix" th:value="${produit.quantite}"/>
    <span th:errors="${produit.prix}" ></span>

    <label>Quantité:</label>
    <input type="text" name="quantite" th:value="${produit.quantite}"/>
    <span th:errors="${produit.quantite}" ></span>

    <button type="submit">Save</button>
</form>
```

EditProduit.html

Action Par défaut

```
@RequestMapping(value="/")  
  
public String index(){  
  
    return "redirect:/produits/index";  
  
}
```

Action Pour le formulaire d'authentification

```
@RequestMapping(value="/login")
```

```
public String login(){
```

```
    return "login";
```

```
}
```

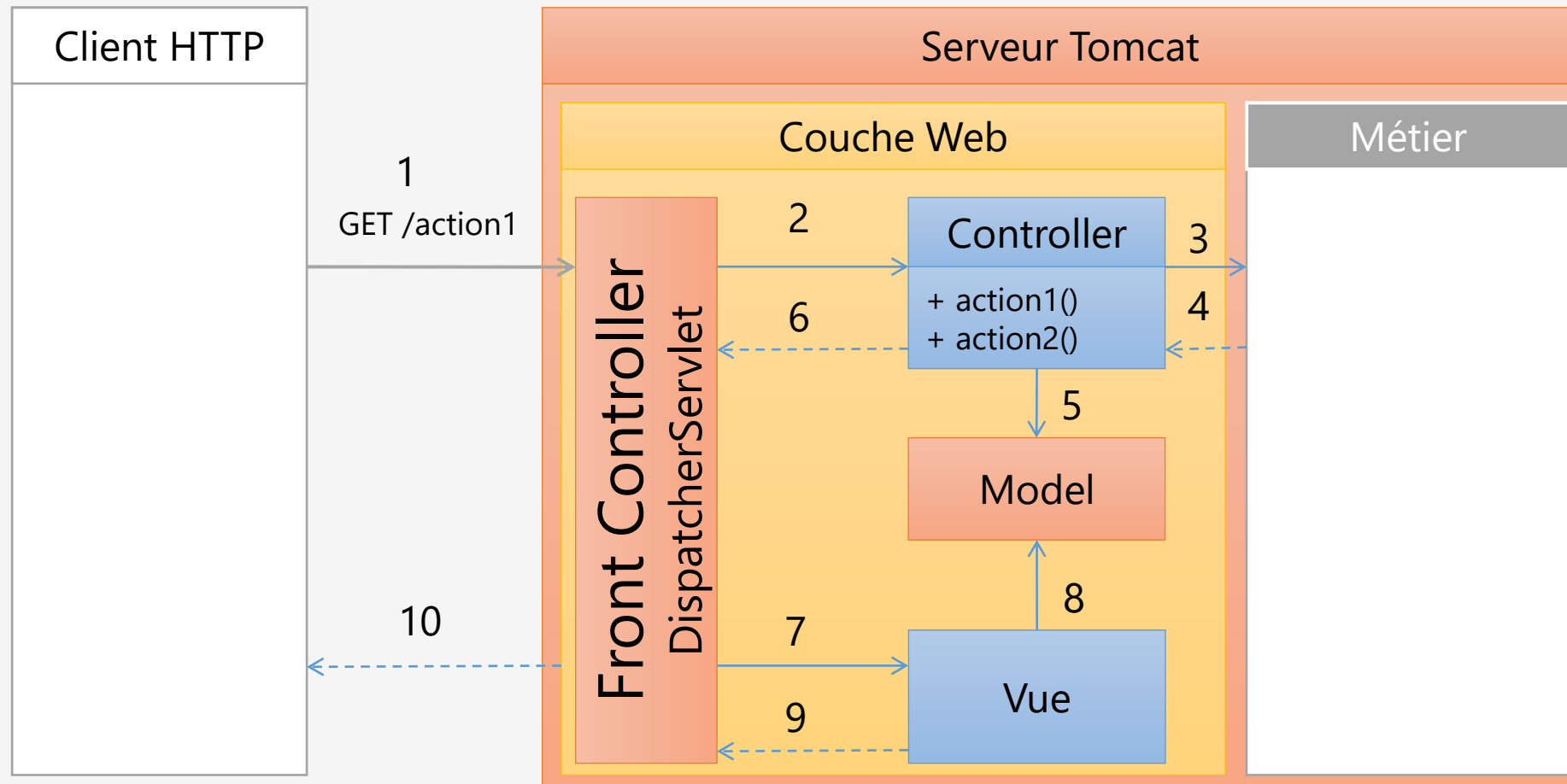
```
}
```



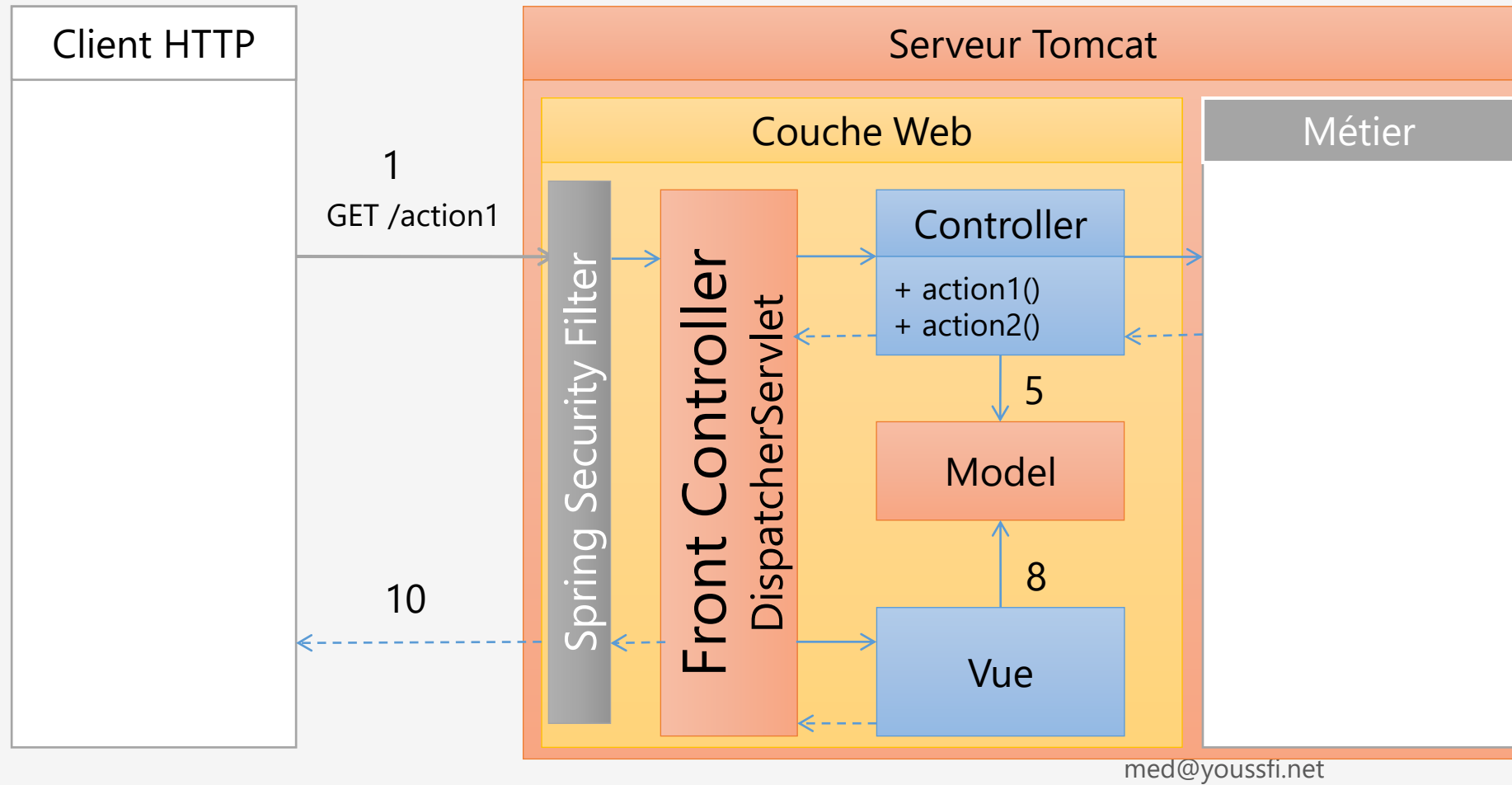
SPRING SECURITY

Architecture Spring MVC

1 – Le client envoie une requête HTTP de type GET ou POST



Spring Security



Configuration de Spring Security

Spring Security est un module de Spring qui permet de sécuriser les applications Web.

Spring Security configure des filtres (springSecurityFilterChain) qui permet d'intercepter les requêtes HTTP et de vérifier si l'utilisateur authentifié dispose des droits d'accès à la ressource demandée.

Les actions du contrôleur ne seront invoquées que si l'utilisateur authentifié dispose de l'un des rôles attribués à l'action

Configuration de Spring Security

Dans cet exemple , nous supposons que nous avons trois tables dans la base de données : users, roles et users_roles et que les mot de passes sont cryptés en format MD5.

roles

role
USER
ADMIN

users

username	password	active
user	ee11cbb19052e40b07aac0ca060c23ee	1
admin	21232f297a57a5a743894a0e4a801fc3	1

users_roles

username	role
admin	ADMIN
admin	USER
user	USER

Dépendance Maven

```
<dependency>  
  
  <groupId>org.springframework.boot</groupId>  
  
  <artifactId>spring-boot-starter-security</artifactId>  
  
</dependency>
```

inMemoryAuthentication

```
package org.sid.sec;
```

```
@Configuration
```

```
@EnableWebSecurity
```

```
public class SecurityConfig extends WebSecurityConfigurerAdapter {
```

```
@Override
```

```
protected void configure(AuthenticationManagerBuilder auth) throws Exception {
```

```
    /* Pour le cas ou les utilisateurs sont connues et fixes d'une manière statique */
```

med@yousfi.net

```
    auth.inMemoryAuthentication().withUser("admin").password("1234").roles("ADMIN", "USER");
```

SecurityConfig.java : JDBC Authentication

@Configuration

@EnableWebSecurity

```
public class SecurityConfig extends WebSecurityConfigurerAdapter {
```

@Autowired

```
private DataSource dataSource;
```

@Override

```
protected void configure(AuthenticationManagerBuilder auth) throws Exception {
```

```
/* Pour le cas ou les utilisateurs sont stockés dans une base la même base de  
données de l'application */
```

med@youssfi.net

```
auth.jdbcAuthentication()
```

SecurityConfig.java

```
@Override
```

```
protected void configure(HttpSecurity http) throws Exception {
```

```
    /* Indiquer à SpringSecurity que l'authentification passe par un formulaire  
    d'authentification avec username et password */
```

```
    http.formLogin().loginPage("/login");
```

```
    /* Toutes les requêtes HTTP avec URL /user/* nécessitent d'être authentifié avec  
    un utilisateur ayant le rôle USER */
```

```
    http.authorizeRequests().antMatchers("/user/*").hasRole("USER");
```

```
    /* Toutes les requêtes HTTP avec URL /admin/* nécessitent d'être authentifié  
    avec un Utilisateur ayant le rôle ADMIN*/
```

med@yousfi.net

```
    http.authorizeRequests().antMatchers("/admin/*").hasRole("ADMIN");
```

login.html

```
<!DOCTYPE html>

<html xmlns:th="http://www.thymeleaf.org" >

<head>

<meta charset="utf-8"/>

<title>Produits</title>

<link rel="stylesheet" type="text/css"

href="../../static/css/bootstrap.min.css"

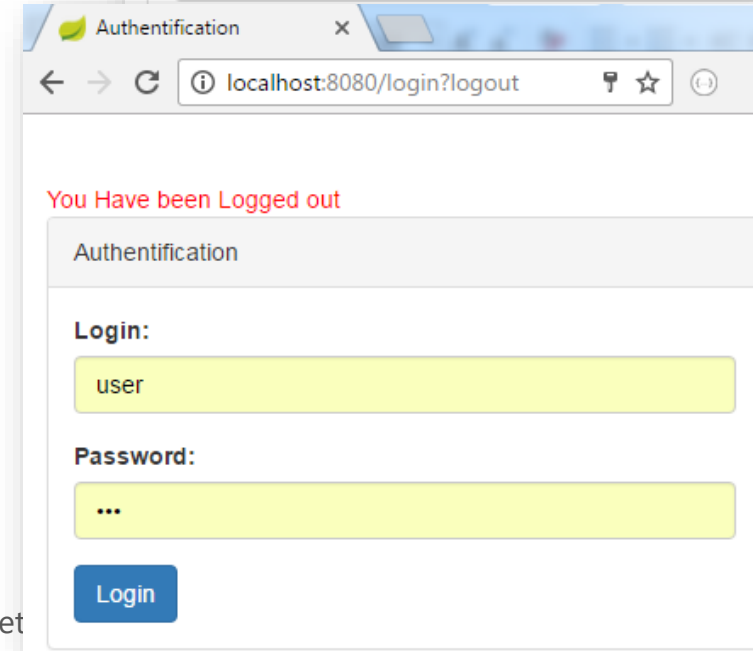
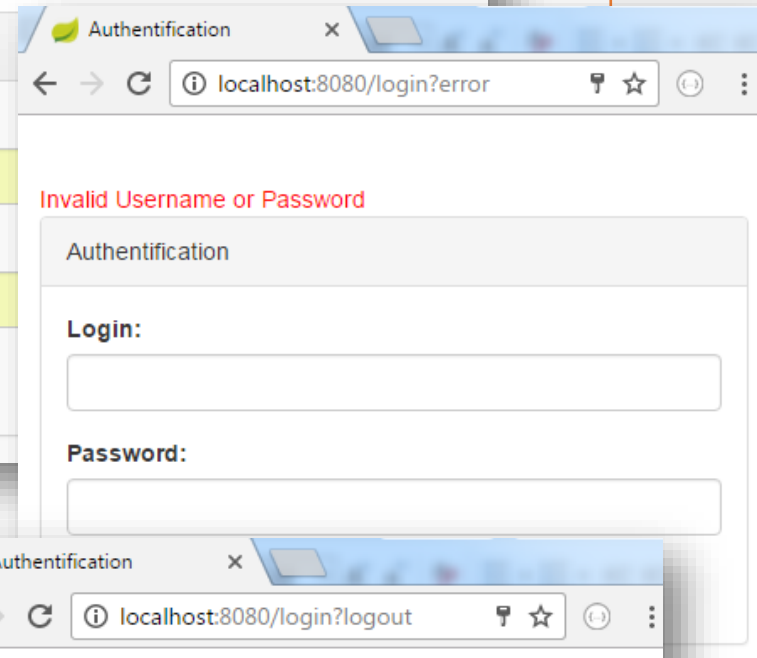
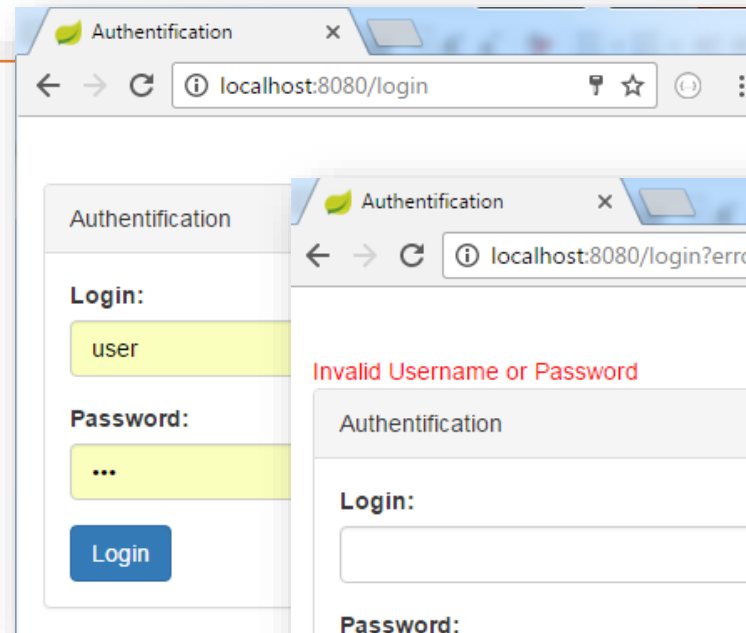
th:href="@{/css/bootstrap.min.css}" />

<link rel="stylesheet" type="text/css"

href="../../static/css/style.css"

th:href="@{/css/style.css}" />

</head>
```



login.html

```
<div class="panel panel-default">

    <div class="panel-heading">Authentication</div>

    <div class="panel-body">

        <form th:action="@{login}" method="post">

            <div class="form-group">

                <label class="control-label">User Name:</label>

                <input class="form-control" type="text" name="username"/>

            </div>

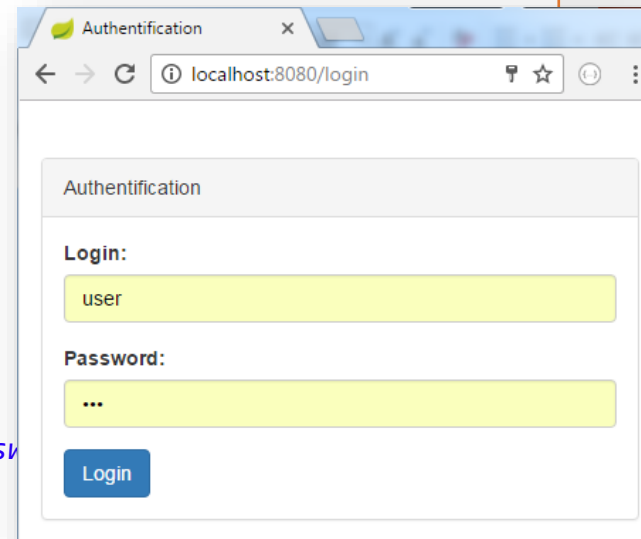
            <div class="form-group">

                <label class="control-label">Pass Word:</label>

                <input class="form-control" type="password" name="password"/>

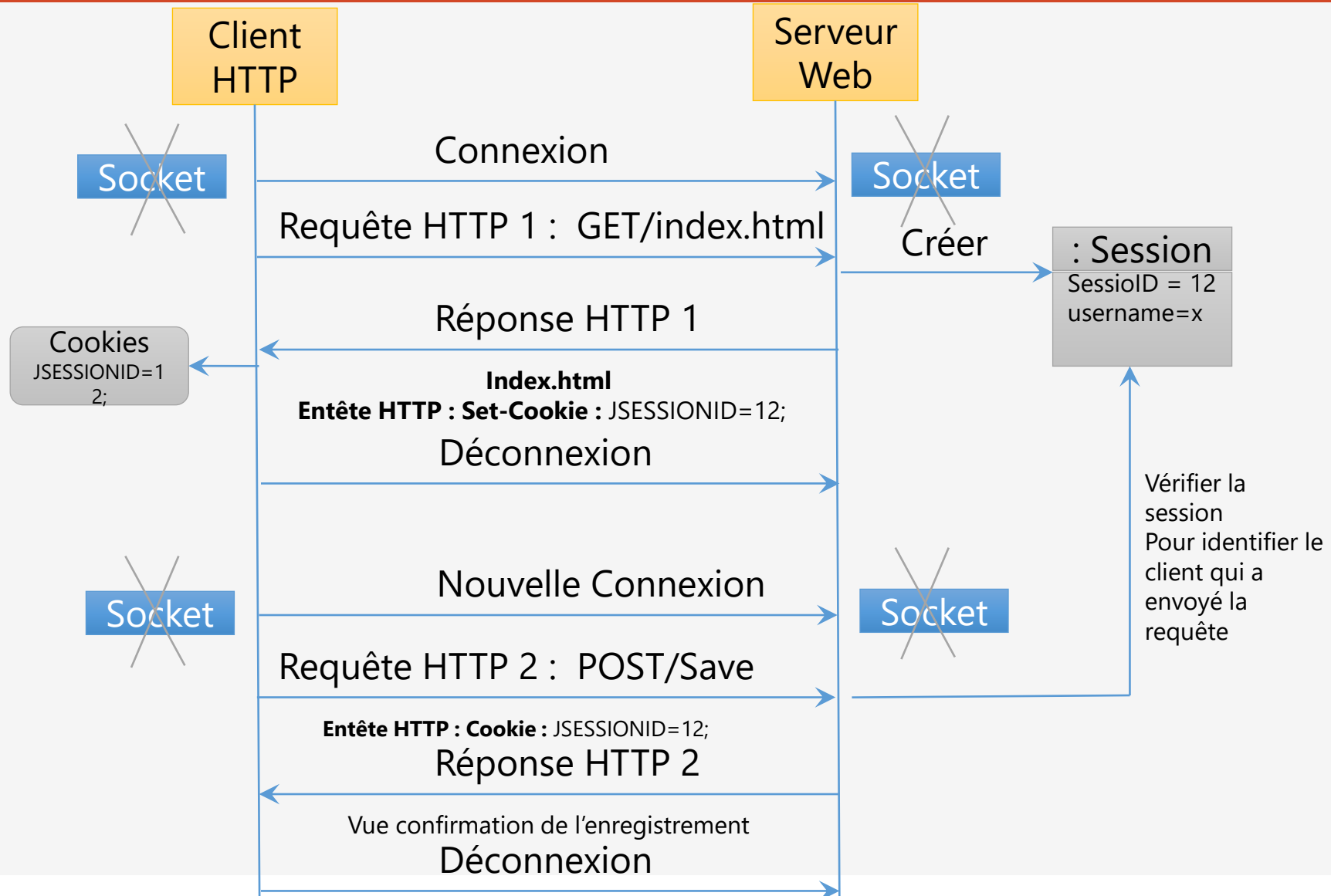
                med@youssfi.net

            </div>
```



The screenshot shows a web browser window with the title 'Authentication' and the address bar displaying 'localhost:8080/login'. The form is titled 'Authentication' and contains two input fields: 'Login:' with the value 'user' and 'Password:' with a masked value '...'. A blue 'Login' button is at the bottom.

Session et Cookies



Utilisation des sessions et des cookies

Généralement quant un client HTTP envoie sa première requête, le serveur web crée une session pour ce client.

Une session est un objet stocké dans la mémoire du serveur qui peut servir pour stocker des informations relatives au client.

Le serveur attribut un SessionID unique à chaque session.

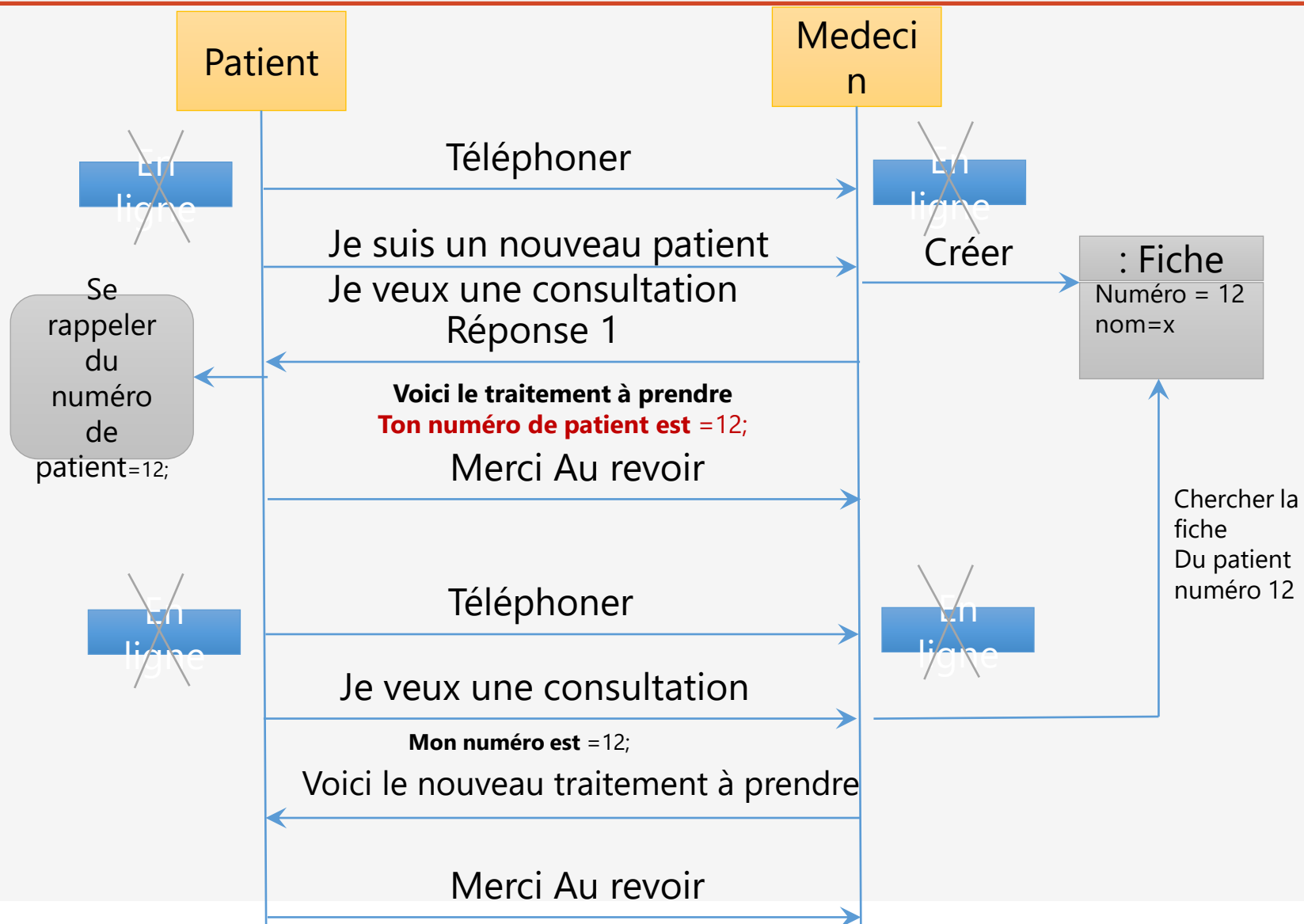
Ce SessionID est ensuite envoyé dans la réponse http en sousforme d'un cookie en utilisant l'entête de la réponse HTTP :

- **Set-Cookie** : JSESSIONID=F84DB7B959F76B183DBF05F999FAEE11;

Ce qui signifie que le serveur demande au client d'enregistrer ce SESSIONID dans un fichier stocké dans la machine du client appelé COOKIE.

Une fois que le client reçoit la réponse HTTP, la connexion est fermée.

Session et Cookies > Patient et médecin



CSRF : Cross Site Request Forgery

C'est une attaque par falsification de requête inter-sites

Force le navigateur d'une victime authentifiée à envoyer une requête http, comprenant le cookie de session de la victime ainsi que toute autre information automatiquement incluse, à une application web vulnérable.

Ceci permet à l'attaquant de forcer le navigateur de la victime à générer des requêtes,

l'application vulnérable considérant alors qu'elles émanent légitimement de la victime.

Une attaque CSRF va exécuter du code malveillant dans une application Web au travers de la session d'un utilisateur connecté.

CSRF

Par défaut Spring Security permet de protéger le site contre ce genre d'attaque.

Dans chaque formulaire de votre application Spring Security ajoute un champ caché contenant un jeton :

```
<input type="hidden" name="_csrf" value="61390c32-08f8-4c85-b55a-  
fe20bc7f0913" />
```

Pour chaque requête envoyé, Spring Sec vérifie si ce jeton existe dans la requête client.

Avec cette procédure, on est sûre que la requête vient vraiment du client authentifié et non pas d'une requête falsifiée.

Code Complet

layout.html

```
<!DOCTYPE html >

<html

xmlns:th="http://www.thymeleaf.org"

xmlns:layout="http://www.ultraq.net.nz/thymeleaf/layout">

<head>

<meta charset="utf-8"/>

<title>Insert title here</title>

<link rel="stylesheet" type="text/css" href="../../static/css/bootstrap.min.css"

th:href="@{/css/bootstrap.min.css}" />

<link rel="stylesheet" type="text/css" href="../../static/css/style.css"

th:href="@{/css/style.css}" />

</head>
```

layout.html

```
</header>
```

```
<section layout:fragment="content">
```

```
</section>
```

```
<footer class="navbar-fixed-bottom">
```

```
<hr/>
```

```
<div class="container">
```

```
<small>
```

Copyright @2016

med@youssfi.net

```
</small>
```


produits.html

```
<!DOCTYPE html>

<html

xmlns:th="http://www.thymeLeaf.org"

xmlns:layout="http://www.ultraq.net.nz/thymeLeaf/layout"

    layout:decorator="layout" >

<head>

<meta charset="utf-8"/>

<title>Produits</title>

    <link rel="stylesheet" type="text/css" href="../static/css/bootstrap.min.css"

th:href="@{/css/bootstrap.min.css}" />

    <link rel="stylesheet" type="text/css" href="../static/css/style.css" med@yousfi.net
```

produits.html

```
<div class="container">

  <h3>Liste des produits</h3>

  <table class="table table-striped">

    <thead>

      <tr>

        <th>ID</th><th>Désignation</th><th>Prix</th><th>Quantité</th>

      </tr>

    </thead>

    <tbody>

      <tr th:each="p:${listProduits}">

        <td th:text="${p.id}"></td>
```

```
<div class="container">

  <ul class="nav nav-pills">

    <li th:class="${pageCourante}==${status.index}?'active':''"

      th:each="p,status:${pages}" >

        <a th:href="@{index(page=${status.index},mc=${mc})}" th:text="${status.index}"></a>

      </li>

    </ul>

  </div>

</section>

</body>

</html>
```

FormProduit.html

```
<!DOCTYPE html>

<html xmlns:th="http://www.thymeleaf.org"
xmlns:layout="http://www.ultraq.net.nz/thymeleaf/layout"

    layout:decorator="layout" >

<head>

<meta charset="utf-8"/>

<title>Produits</title>

    <link rel="stylesheet" type="text/css"    href="../../static/css/bootstrap.min.css"

    th:href="@{/css/bootstrap.min.css}" />

    <link rel="stylesheet" type="text/css"    href="../../static/css/style.css"

    th:href="@{/css/style.css}" />

med@youssfi.net

</head>
```


confirmation.html

```
<!DOCTYPE html>

<html xmlns:th="http://www.thymeleaf.org"
xmlns:layout="http://www.ultraq.net.nz/thymeleaf/layout"

    layout:decorator="layout" >

<head>

<meta charset="utf-8"/>

<title>Produits</title>

    <link rel="stylesheet" type="text/css"

        href="../static/css/bootstrap.min.css"

        th:href="@{/css/bootstrap.min.css}" />

</head>

<body>
```

```
<div class="form-group">

    <label class="control-label">Prix:</label>

    <label class="control-label" th:text="${produit.prix}"></label>

</div>

<div class="form-group">

    <label class="control-label">Quantité:</label>

    <label class="control-label" th:text="${produit.quantite}"></label>

</div>

</div>

</div>
```

EditProduit.html

```
<!DOCTYPE html>

<html xmlns:th="http://www.thymeleaf.org"
xmlns:layout="http://www.ultraq.net.nz/thymeleaf/layout"

    layout:decorator="layout" >

<head>

<meta charset="utf-8"/>

<title>Produits</title>

    <link rel="stylesheet" type="text/css"

        href="../static/css/bootstrap.min.css"

        th:href="@{/css/bootstrap.min.css}" />

</head>

<body>
```


Projet Développement Web JEE

On souhaite créer une application qui permet de gérer des comptes bancaire.

- Chaque compte est défini un code, un solde et une date de création
- Un compte courant est un compte qui possède en plus un découvert
- Un compte épargne est un compte qui possède en plus un taux d'intérêt.
- Chaque compte appartient à un client.
- Chaque client est défini par son code et son nom
- Chaque compte peut subir plusieurs opérations.
- Il existe deux types d'opérations : Versement et Retrait
- Une opération est définie par un numéro, une date et un montant.

Exigences fonctionnelles

L'application doit permettre de :

- Gérer des clients :

- Ajouter un client
- Consulter tous les clients
- Consulter les clients dont le nom contient un mot clé

- Gérer les comptes :

- Ajouter un compte
- Consulter un compte

- Gérer les opérations :

- Effectuer un versement d'un montant dans un compte
- Effectuer un retrait d'un montant dans un compte
- Effectuer un virement d'un montant d'un compte vers un autre
- Consulter les opérations d'un compte page par page

Les opérations nécessitent une opération d'authentification

Exigences Techniques

Les données sont stockées dans une base de données MySQL

L'application se compose de trois couches :

- La couche DAO qui est basée sur Spring Data, JPA, Hibernate et JDBC.
- La couche Métier
- La couche Web basée sur MVC coté Serveur en utilisant Thymeleaf.

La sécurité est basée sur Spring Security

Travail demandé :

Etablir une architecture technique du projet

Etablir un diagramme de classes qui montre les entités, la couche DAO et la couche métier.

Créer un projet SpringBoot qui contient les éléments suivants :

- Les entités
- La couche DAO (Interfaces Spring data)
- La couche métier (Interfaces et implémentations)
- La couche web :
 - Les contrôleurs Spring MVC
 - Les Vue basée sur Thymeleaf

Sécuriser l'application en utilisant un système d'authentification basé sur Spring Security

Architecture Technique

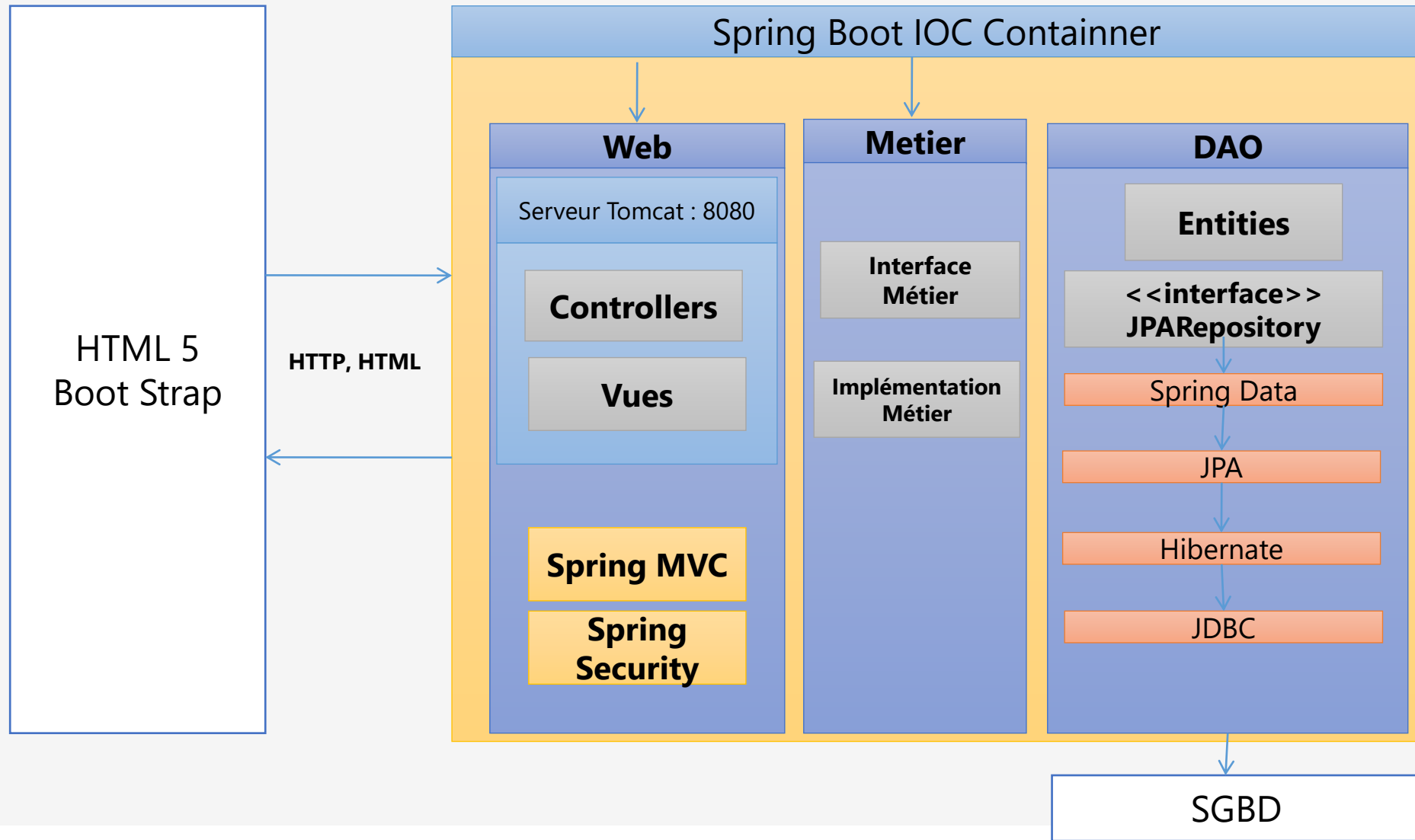
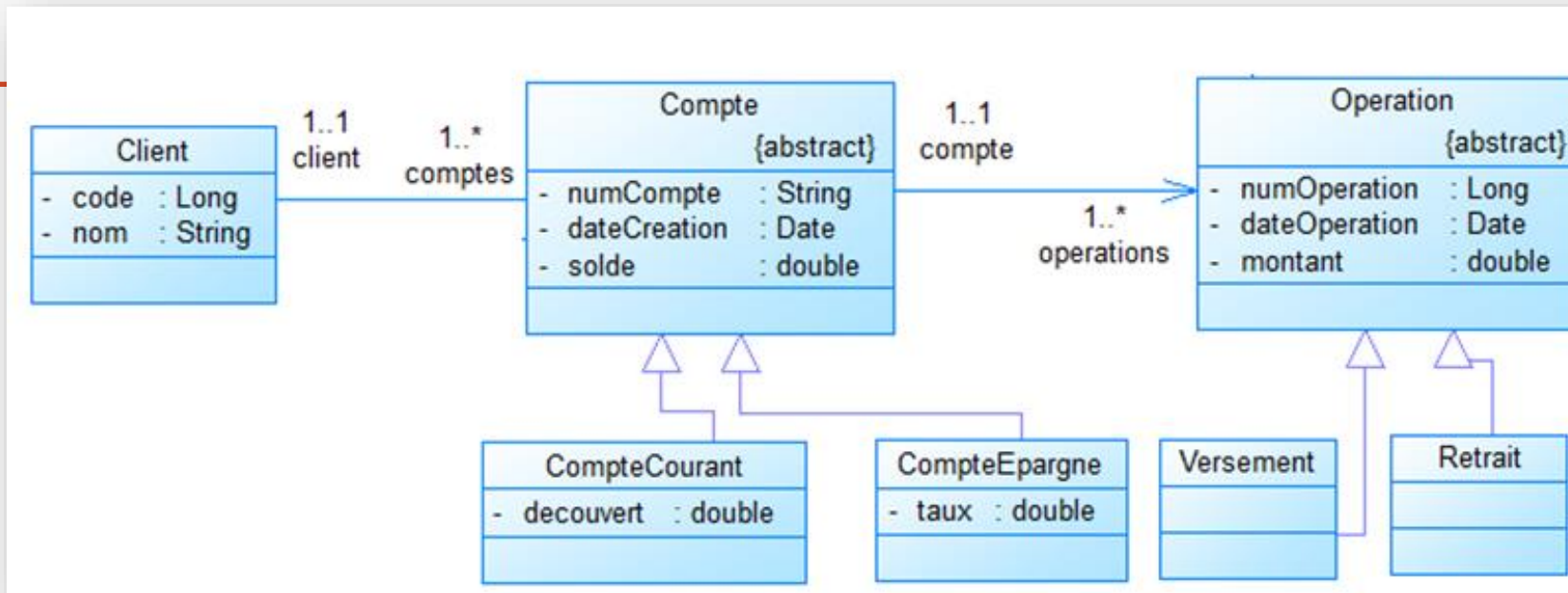


Diagramme de classes des entités et MLDR



MLRD : En utilisant la stratégie Single Table pour l'héritage

- **T_CLIENTS** (`CODE_CLI`, `NOM_CLI`)
- **T_COMPTE** (`NUM_CPTE`, `TYPE_PTE`, `DATE_CR`, `SOLDE`, `DEC`, `TAUX`, `#CODE_CLI`)
- **T_OPERATIONS** (`NUM_OP`, `TYPE_OP`, `DATE_OP`, `MONTANT`, `#NUM_CPTE`)

Aperçu d'une vue de l'application web

•
•

The screenshot shows a web browser window with the address bar displaying `localhost:8080/comptes/consulterCompte?codeCompte=c1`. The application has a navigation bar with 'Operations', 'Comptes', and 'Client' tabs, and a user profile 'admin' with a 'Logout' link. The main content is divided into three sections:

- Consultation d'un compte:** A form with a text input for 'Code Cpte' containing 'c1' and an 'OK' button.
- Informations sur le compte:** A box displaying account details: 'Client: Hassan', 'Code Cpte: c1', 'Solde: 90333.0', 'Date création: 2016-12-28 20:37:20.0', 'Type: CompteCourant', and 'Découvert: 800.0'.
- Opérations sur le compte:** A section for performing transactions. It includes a 'Compte : c1' label, radio buttons for 'Versement' (selected), 'Retrait', and 'Virement', a 'Montant : 0' input field, and a 'Save' button.
- Liste des opérations:** A table showing a list of transactions.

Num	Type	Date:	Montant
33	Retrait	2016-12-31 14:09:15.0	8000.0
32	Versement	2016-12-31 14:09:08.0	9000.0
31	Versement	2016-12-31 14:09:01.0	0.0
29	Retrait	2016-12-31 12:55:21.0	800.0
27	Retrait	2016-12-31 12:54:57.0	90.0

At the bottom of the application, there is a footer with 'Copyright @2016' and a taskbar showing various system icons and the date/time '14:10 31/12/2016'.



COUCHE DAO

Entité JPA Client

@Entity

```
public class Client implements Serializable{
```

```
    @Id @GeneratedValue
```

```
    private Long code;
```

```
    private String nom;
```

```
    @OneToMany(mappedBy="client", fetch=FetchType  
.LAZY)
```

```
    private Collection<Compte> comptes;
```

```
// Générer un constructeur sans param
```

```
public Client() { super(); }
```

Entité JPA Compte

@Entity

```
@Inheritance(strategy=InheritanceType.SINGLE_TABLE)
```

```
@DiscriminatorColumn( name="TYPE_CPTE",  
discriminatorType=DiscriminatorType.STRING,length=2)
```

```
public abstract class Compte implements Serializable {
```

```
    @Id
```

```
    private String code;
```

```
    private double solde;
```

```
    private Date dateCreation;
```

```
    @ManyToOne
```

```
    @JoinColumn(name="CODE_CLI")
```

```
    private Client client;
```

```
    @OneToMany(mappedBy="compte")
```

```
    private Collection<Operation> operations;
```

```
// Générer un constructeur sans param
```

```
public Compte() { super(); }
```

```
// Générer un constructeur avec params
```

```
public Compte(String code, double solde, Date  
dateCreation, Client client) {
```

```
    super(); this.code = code; this.solde = solde;
```

```
this.dateCreation = dateCreation; this.client = client;  
}
```

```
//Générer les getters et setters
```

```
}
```

Entité JPA CompteCourant

```
@Entity

@DiscriminatorValue("CC")

public class CompteCourant extends Compte {

    private double decouvert;

    // Constructeur sans param

    public CompteCourant() { super(); }

    // Constructeur avec params

    public CompteCourant(String code, double
solde, Date dateCreation, Client client,
double decouvert ) {
```

```
    super(code, solde, dateCreation, client);
```

Entité JPA CompteEpargne

```
@Entity
@DiscriminatorValue("CE")
public class CompteEpargne extends Compte {

    private double taux;

    // Constructeur sans param
    public CompteEpargne() { super(); }
    // Constructeur avec params
    public CompteEpargne (String code, double solde, Date
dateCreation, Client client, double taux) {
        super(code, solde, dateCreation, client);
        this.taux = taux;
    }

    // Getters et Setters

}
```

Entité Operation

```
@Entity
@Inheritance(strategy=InheritanceType.SINGLE_TABLE)
@DiscriminatorColumn(name="TYPE",length=2)
public abstract class Operation {
    @Id @GeneratedValue
    private Long numero;
    private Date dateOperation;
    private double montant;
    @ManyToOne
    @JoinColumn(name="CODE_CPTE")
    private Compte compte;
    // Constructeur sans param
    public Operation() { super(); }
    // Constructeur avec params
    public Operation(Date dateOperation, double montant,
Compte compte) {
        super();
        this.dateOperation = dateOperation;
        this.montant = montant;
        this.compte = compte;
    }
    // Getters et Setters
}
```

Entité Versement

```
@Entity
@DiscriminatorValue("V")
public class Versement extends Operation{
    public Versement() { super();}
    public Versement(Date dateOperation,
double montant, Compte compte) {
        super(dateOperation, montant, compte);
    }
}
```

Entité Retrait

```
@Entity
@DiscriminatorValue("R")
public class Retrait extends Operation{
    public Retrait() { super();}
    public Retrait(Date dateOperation, double
montant, Compte compte) {
        super(dateOperation, montant, compte);
    }
}
```

Interfaces DAO basées sur Spring Data

```
import org.springframework.data.jpa.repository.JpaRepository;

import ma.emsi.entities.Client;

public interface ClientRepository extends JpaRepository<Client, Long> {

}
```

```
import org.springframework.data.jpa.repository.JpaRepository;
import ma.emsi.entities.Compte;
public interface CompteRepository extends JpaRepository<Compte, String>{

}
```

```
import org.springframework.data.jpa.repository.JpaRepository;
import ma.emsi.entities.Operation;
public interface OperationRepository extends JpaRepository<Operation, Long>{

}
```

test de la couche DAO

```
@SpringBootApplication
```

```
public class TpJpaApplication {
```

```
public static void main(String[] args) {
```

```
ApplicationContext ctx= SpringApplication.run(TpJpaApplication.class, args);
```

```
ClientRepository clientRepository= ctx.getBean(ClientRepository.class);
```

```
/* * Ajouter 3 clients */
```

```
clientRepository.save(new Client("Hassan")); clientRepository.save(new Client("AAAA"));
```

```
clientRepository.save(new Client("CCCCC"));
```

```
CompteRepository compteRepository= ctx.getBean(CompteRepository.class);
```

```
/* * consulter un client sachant son code */
```

code	nom
1	Hassan
2	AAAA
3	CCCCC

type_cpte	code	date_creation	solde	decouvert	taux	code_cli
CC	c1	2016-11-25 15:44:09	9000	800	NULL	1
CE	c2	2016-11-25 15:44:09	7800	NULL	5	1

type	numero	date_operation	montant	code_cpte
V	1	2016-11-25 15:44:09	6000	c1
V	2	2016-11-25 15:44:09	7000	c1
R	3	2016-11-25 15:44:09	5400	c1

TP Spring MVC, JPA, Hibernate et Spring Data

On souhaite créer une application Web JEE qui permet de gérer les taxes relatives à des sociétés. Chaque Taxe concerne une entreprise. Dans l'application on traite deux types des taxes : TVA (Taxe sur la Valeur Ajoutée) ou IR (Impôt sur le Revenu). Une taxe est définie par son numéro (auto-incrémenté), le titre de la taxe, sa date, le montant de la taxe. Une entreprise est définie par son code, son nom, sa raison sociale et son email.

L'application doit permettre de :

- Gérer les entreprises (ajouter, éditer, mettre à jour, consulter, supprimer, chercher) :
- Gérer les taxes (ajouter, éditer, mettre à jour, consulter, supprimer, chercher)

Les exigences techniques de l'application sont :

Les données sont stockées dans une base de données MySQL

L'application se compose de deux couches:

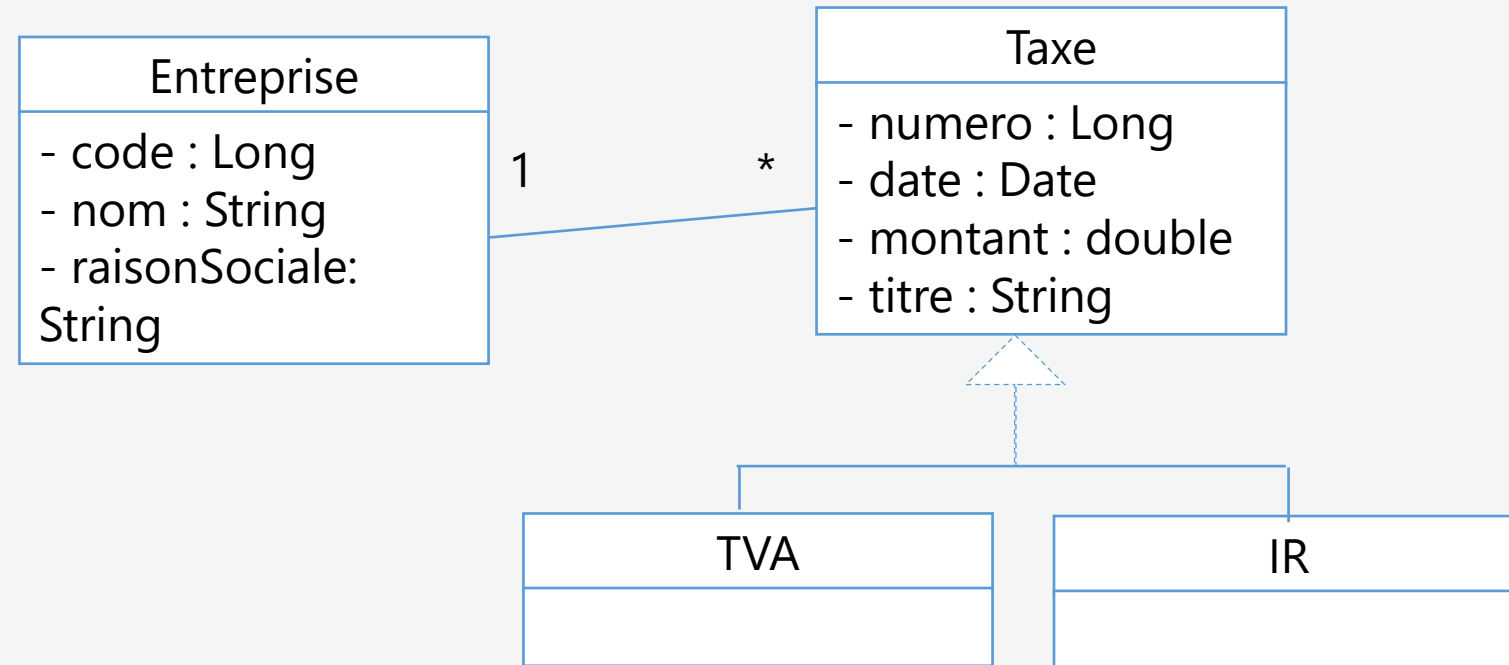
- La couche DAO qui est basée sur Spring Data, JPA, Hibernate et JDBC.
- La couche Web basée sur MVC coté Serveur en utilisant Thymeleaf.

L'inversion de contrôle est basée sur Spring IOC.

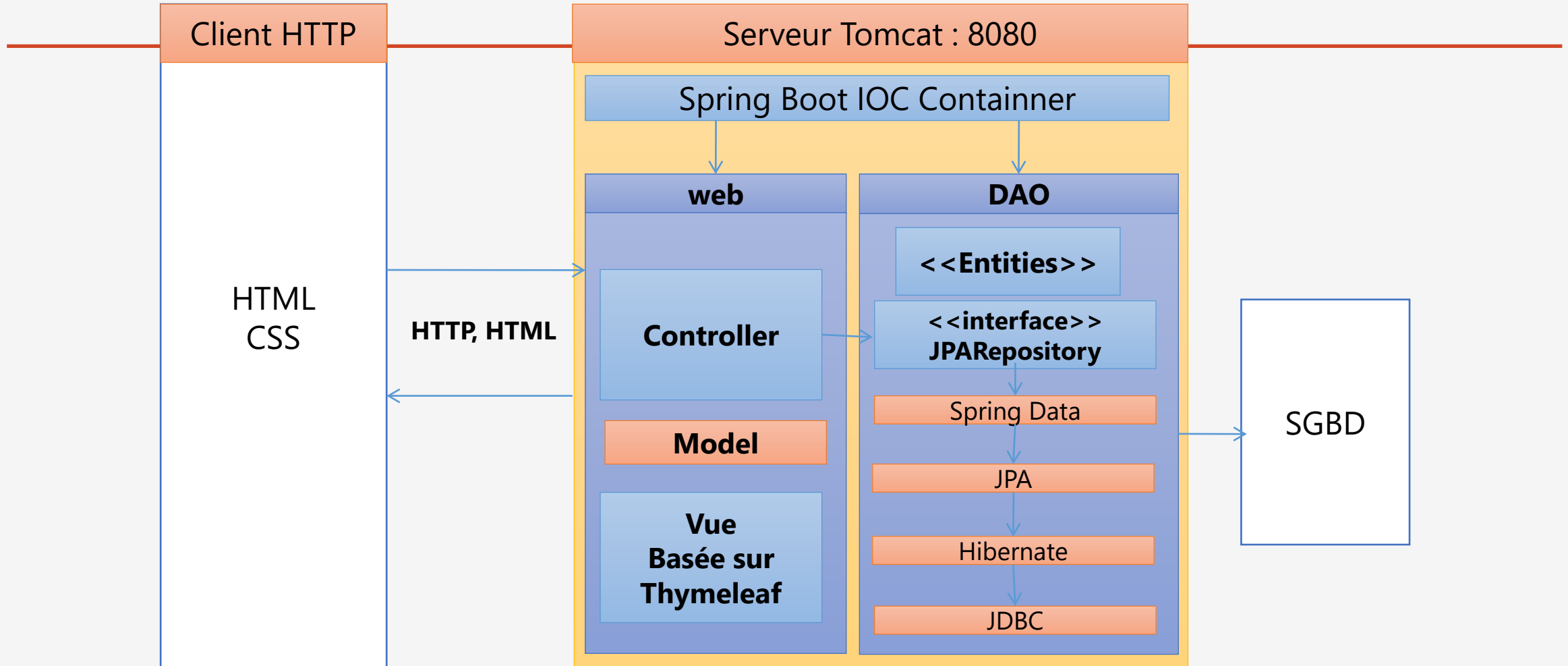
Travail demandé

1. Créer un projet Spring Starter avec les dépendances JPA, MySQL, Web et Thymeleaf.
2. Couche DAO
 - a) Créer les entités JPA
 - b) Créer les interfaces JPA Repository basées sur Spring Data
 - c) Tester quelques opérations de la couche DAO
3. Créer une application Web JEE basée sur Spring MVC qui permet de chercher les entreprises dont le nom contient un mot clé saisi dans une zone de texte avec un lien qui permet de consulter les taxes d'une entreprise.

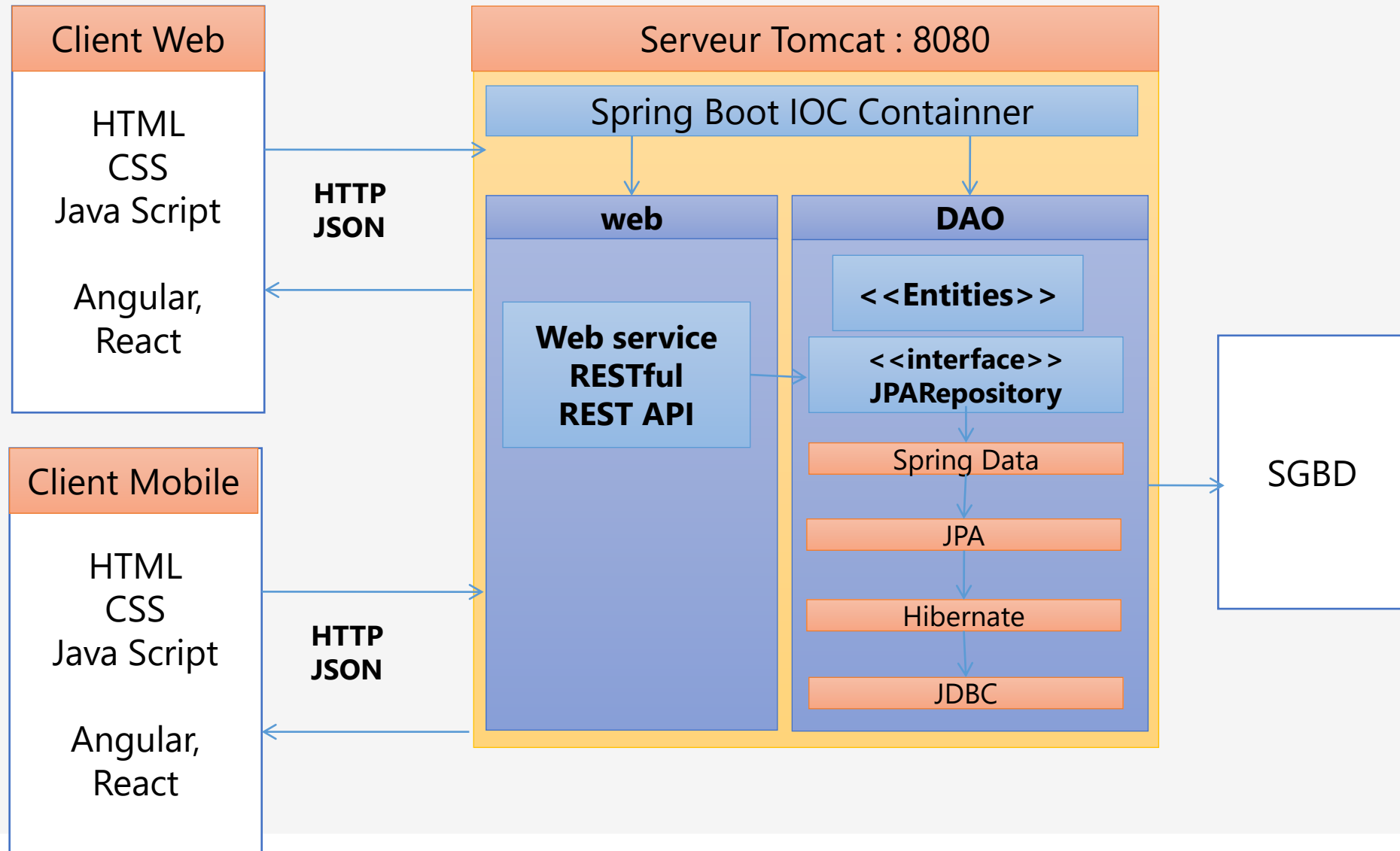
Diagramme de classes



Architecture Technique



Application web Restful côté Client avec Spring Côté backend et Angular côté Frontend



Web services

Les Web services sont des composants web qui contiennent des opérations qui permettent d'effectuer des traitements suite à des requêtes HTTP

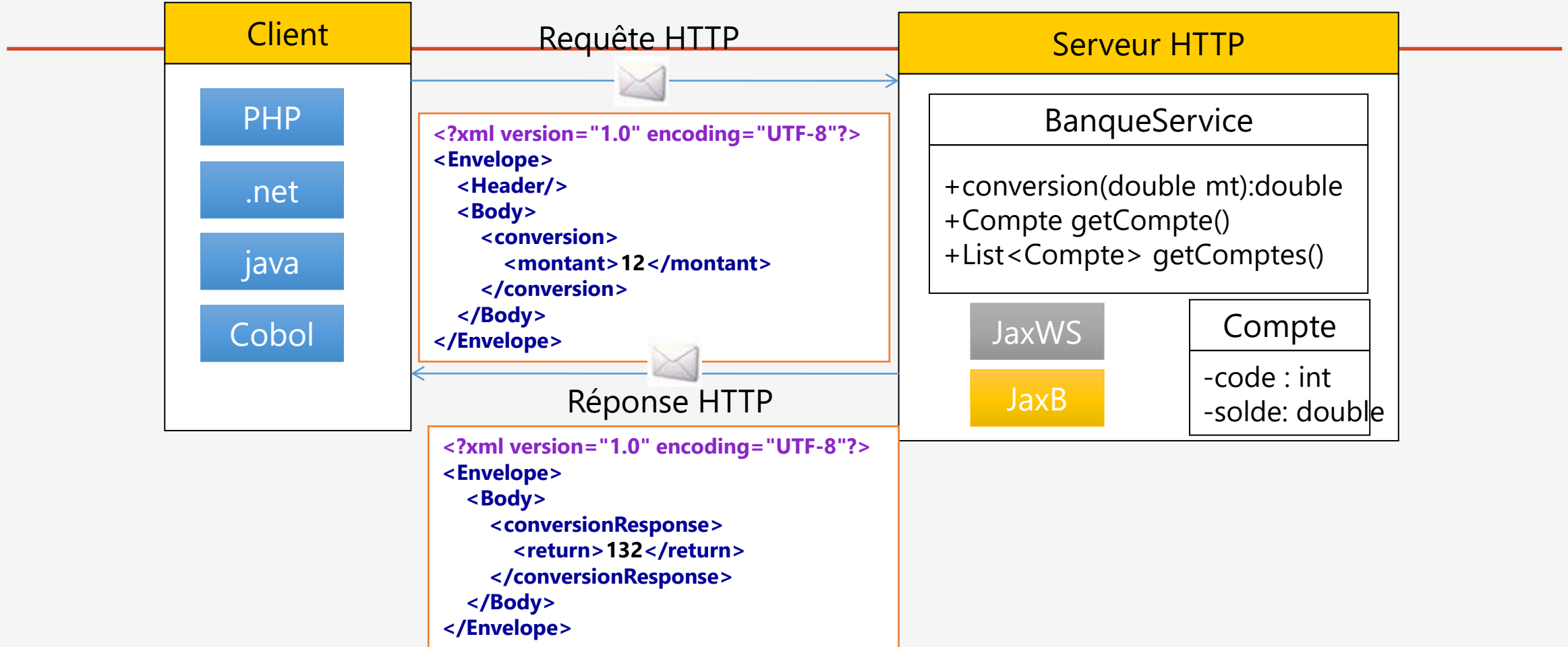
Les opérations du web service retournent au client HTTP des résultats à différents formats :

- XML
- JSON

Il existe deux types de Web services :

- Web services basés sur le protocole SOAP (Simple Object Access Protocol) qui permet de faire dialoguer les applications distribuées en échangeant des messages XML.
- Web Services RESTful qui permettent d'effectuer des traitements suite à une simple requête HTTP en retournant le résultat au format demandé par le client http (xml, json, etc.)

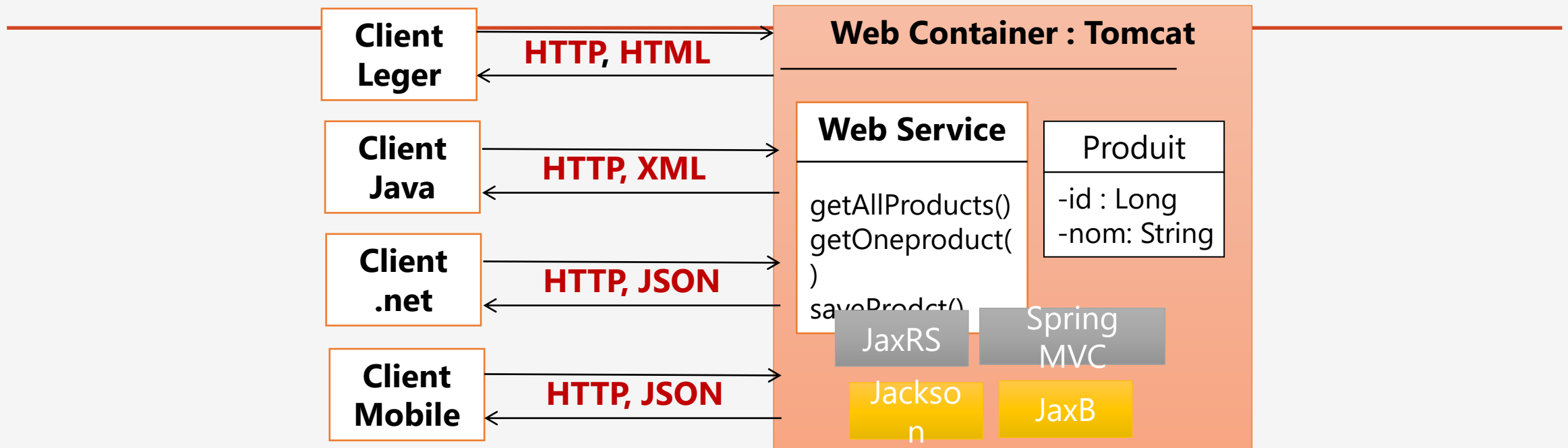
Web Services SOAP



Pour qu'un client puisse consommer un web service basé sur SOAP, il a besoin de connaître d'abord sa description

Cette description est faite grâce au WSDL (Web Service Description Language)

REST: Principe de base



GET <http://localhost:8080/produits> Pour Consulter tous les produits

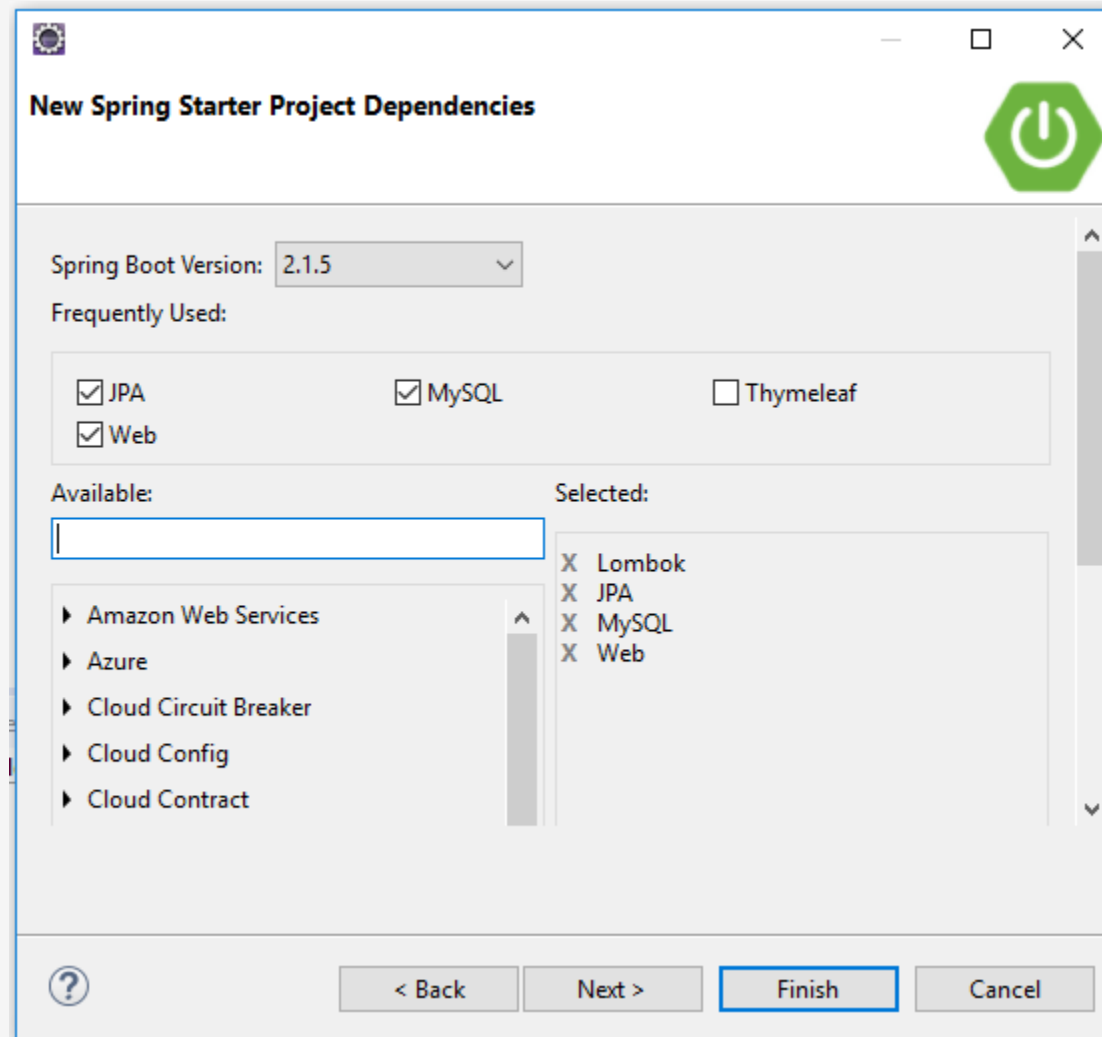
POST <http://localhost:8080/produits> Pour Ajouter un produit

GET <http://localhost:8080/produits/6> Pour Consulter Le produit dont ID=5

PUT <http://localhost:8080/produits/6> Pour Mettre à jour Le produit dont ID=5

DELETE <http://localhost:8080/produits/6> Pour Supprimer Le produit dont ID=5

Projet Spring Boot



The image shows a 'New Spring Starter Project Dependencies' dialog box. It features a title bar with a question mark icon, standard window controls, and a green power button icon. The main content area is divided into sections for version selection, frequently used dependencies, and available/selected dependencies. The 'Spring Boot Version' is set to 2.1.5. Under 'Frequently Used', JPA, Web, and MySQL are checked, while Thymeleaf is unchecked. The 'Available' list includes Amazon Web Services, Azure, Cloud Circuit Breaker, Cloud Config, and Cloud Contract. The 'Selected' list includes Lombok, JPA, MySQL, and Web. At the bottom, there are buttons for '< Back', 'Next >', 'Finish' (highlighted with a blue border), and 'Cancel', along with a help icon.

New Spring Starter Project Dependencies

Spring Boot Version: 2.1.5

Frequently Used:

☒ JPA ☒ MySQL ☐ Thymeleaf
☒ Web

Available:

Selected:

- ▶ Amazon Web Services
- ▶ Azure
- ▶ Cloud Circuit Breaker
- ▶ Cloud Config
- ▶ Cloud Contract

- X Lombok
- X JPA
- X MySQL
- X Web

? < Back Next > Finish Cancel

Entité JPA Compte

```
package org.sid.entities;

import java.io.Serializable;

import java.util.Date;

import javax.persistence.*;

import lombok.AllArgsConstructor;

import lombok.Data;

import lombok.NoArgsConstructor;

@Entity

@Data @AllArgsConstructor @NoArgsConstructor

public class Compte implements Serializable {

    @Id @GeneratedValue

    private Long code;

    private double solde;

    private Date dateCreation;
}
```


ORM avec Spring Data

```
package org.sid.dao;
```

```
import org.sid.entities.Compte;
```

```
import org.springframework.data.jpa.repository.JpaRepository;
```

```
public interface CompteRepository extends JpaRepository<Compte, Long> {
```

```
}
```

Web Service Restful avec Spring Rest Controller

Spring MVC offre des annotations qui permettent de créer facilement des web services Restful, sans avoir besoin d'utiliser JAXRS.

Les annotations principales suivantes peuvent être utilisées pour ce fait:

- `@RestController` : Pour annoter le web service
- `@RequestMapping` : Pour associer une requête HTTP quelconque
- `@GetMapping` : Pour associer une requête HTTP de type GET
- `@PostMapping` : Pour associer une requête HTTP de type POST
- `@PutMapping` : Pour associer une requête HTTP de type PUT
- `@DeleteMapping` : Pour associer une requête HTTP de type DELETE
- `@RequestBody` : Pour associer le corps de la requête HTTP à un paramètre d'une méthode du web service
- `@PathVariable` : Pour associer un paramètre de path URL un paramètre d'une méthode du web service
- `@RequestParam` : Pour associer un Query param à un paramètre d'une méthode du web service

Web Service Restful avec Spring RestController

```
package org.sid.web;

import java.util.List; import org.sid.dao.CompteRepository;

import org.sid.entities.Compte; import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.web.bind.annotation.*;

@RestController

@RequestMapping(value="/banque")

public class BanqueRestServiceRC {

    @Autowired

    private CompteRepository compteRepository;
```

Web Service Restful avec Spring RestController

```
@PostMapping("/comptes")
```

```
public Compte save(@RequestBody Compte cp) {
```

```
    compteRepository.save(cp);
```

```
    return cp;
```

```
}
```

```
@PutMapping("/comptes/{code}")
```

```
public Compte update(@PathVariable(value="code")Long code,Compte cp) {
```

```
    cp.setCode(code);
```

```
    compteRepository.save(cp);
```

```
    return cp;
```

Application Spring Boot

```
package org.sid;

import java.util.Date; import org.sid.dao.CompteRepository;

import org.sid.entities.Compte; import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.boot.CommandLineRunner; import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication

public class SpringJerseyApplication implements CommandLineRunner {

    @Autowired

    private CompteRepository compteRepository;

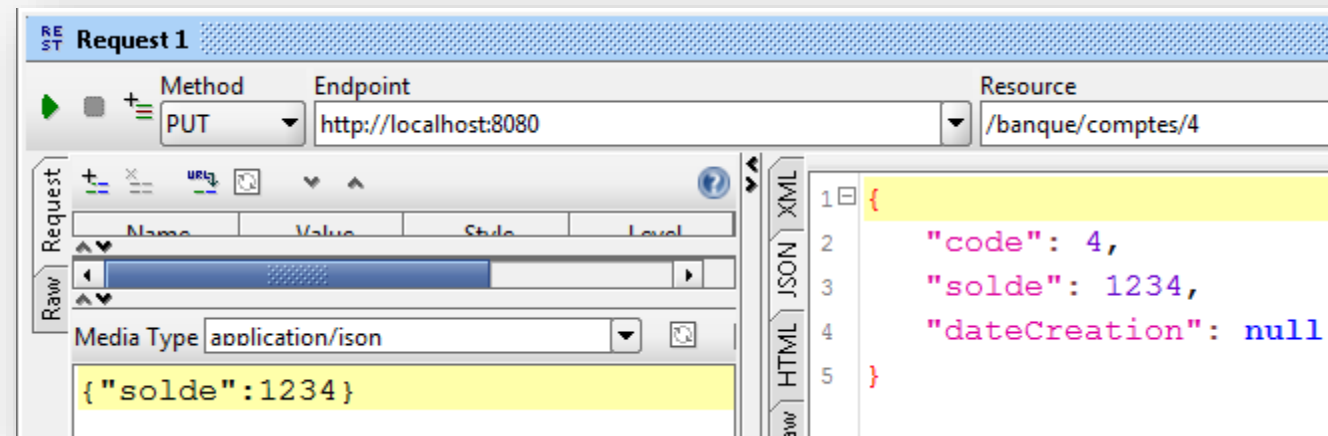
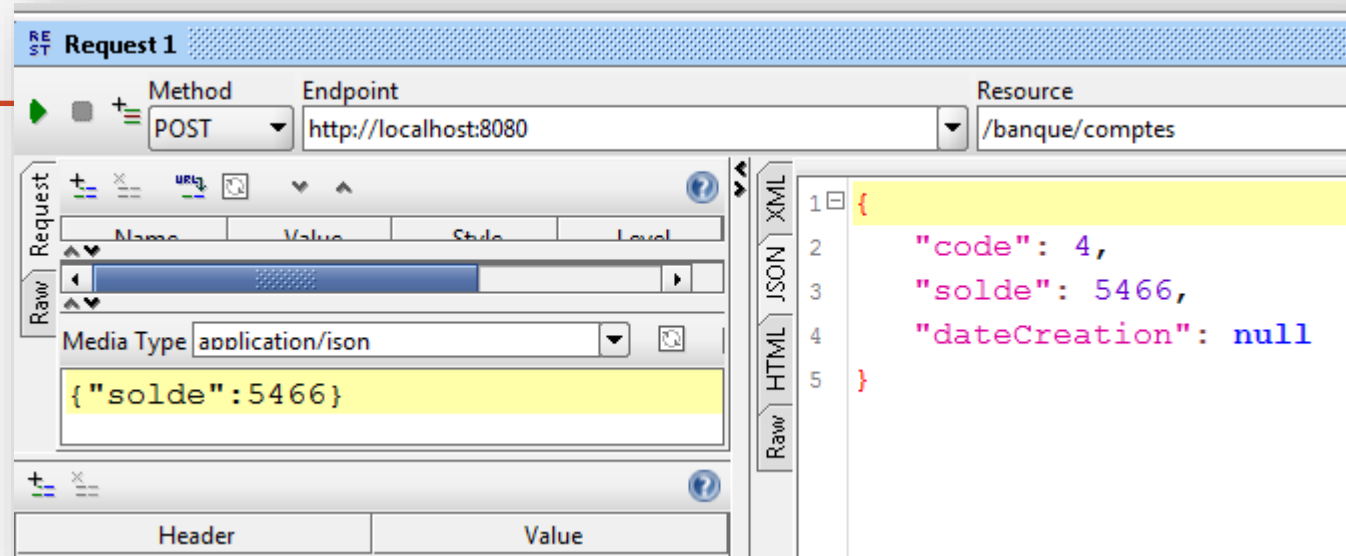
    public static void main(String[] args) {
        SpringApplication.run(SpringJerseyApplication.class, args);
    }
}
```

Tester L'API REST

```
localhost:8080/banque/comptes
[
  {
    "code": 1,
    "solde": 4492.983404535533,
    "dateCreation": 1514030824857
  },
  {
    "code": 2,
    "solde": 126.6795242423402,
    "dateCreation": 1514030824975
  },
  {
    "code": 3,
    "solde": 1302.8399033231146,
    "dateCreation": 1514030824980
  }
]
```

```
localhost:8080/banque/comptes/2
{
  "code": 2,
  "solde": 126.6795242423402,
  "dateCreation": 1514030824975
}
```

Tester L'API REST



Web Service Restful avec Spring Data Rest

Spring Data rest offre un moyen très rapide qui permet d'exposer une API Rest.

Il suffit d'utiliser l'annotation `@RespositoryRestResource` dans l'interface Spring Data pour que toutes les méthodes de cette interfaces soient accessibles via une API Rest générée automatiquement.

```
<dependency>  
  <groupId>org.springframework.boot</groupId>  
  <artifactId>spring-boot-starter-data-rest</artifactId>  
</dependency>
```


Web Service Restful avec Spring Data Rest

```
package org.sid.dao;
```

```
import org.sid.entities.Compte;
```

```
import org.springframework.data.jpa.repository.JpaRepository;
```

```
import org.springframework.data.rest.core.annotation.RepositoryRestResource;
```

```
@RepositoryRestResource
```

```
public interface CompteRepository extends JpaRepository<Compte, Long> {
```

```
}
```

Test de l'API REST basée sur Spring Data rest



```
{
  "_embedded": {
    "comptes": [
      {
        "solde": 2708.003123246959,
        "dateCreation": "2017-12-23T12:50:09.525+0000",
        "_links": {
          "self": {
            "href": "http://localhost:8080/comptes/1"
          },
          "compte": {
            "href": "http://localhost:8080/comptes/1"
          }
        }
      },
      {
        "solde": 4994.377966622107,
        "dateCreation": "2017-12-23T12:50:09.687+0000",
        "_links": {
          "self": {
            "href": "http://localhost:8080/comptes/2"
          },
          "compte": {
            "href": "http://localhost:8080/comptes/2"
          }
        }
      }
    ]
  }
}
```

Test de l'API REST basée sur Spring Data rest

```
{
  "solde": 2708.003123246959,
  "dateCreation": "2017-12-23T12:50:09.525+0000",
  "_links": {
    "self": {
      "href": "http://localhost:8080/comptes/1"
    },
    "compte": {
      "href": "http://localhost:8080/comptes/1"
    }
  }
}
```

Test de l'API REST basée sur Spring Data rest



The screenshot shows a web browser window with the address bar displaying `localhost:8080/comptes?page=0,size=2&sort=solde,desc`. The main content area displays a JSON response from a REST API, which is a list of two account objects. The JSON is color-coded and has a tree view on the left side.

```
{
  "_embedded": {
    "comptes": [
      {
        "solde": 4994.377966622107,
        "dateCreation": "2017-12-23T12:50:09.687+0000",
        "_links": {
          "self": {
            "href": "http://localhost:8080/comptes/2"
          },
          "compte": {
            "href": "http://localhost:8080/comptes/2"
          }
        }
      },
      {
        "solde": 4171.701628368106,
        "dateCreation": "2017-12-23T12:50:09.694+0000",
        "_links": {
          "self": {
            "href": "http://localhost:8080/comptes/3"
          },
          "compte": {
            "href": "http://localhost:8080/comptes/3"
          }
        }
      }
    ]
  }
}
```