

# JPA + Spring web MVC

---

Java Persistence API (JPA) assure la synchronisation entre un graphe d'objets Java et des tables dans une base de données de telle sorte que toute modification dans les objets est répercutée automatiquement en base et vice versa. Spring web MVC quant à lui, repose sur le pattern Model View Controller (MVC), et est un framework web qui permet de passer facilement du monde du web (orienté HTML) au monde Java. Il est donc tentant de bâtir une application composée des trois couches du modèle MVC et d'y insérer des instructions JPA. On a ainsi une application web complète puisqu'elle assure la sauvegarde des données dans une base de données. Cependant, une application ainsi faite va souffrir d'un problème majeure d'évolutivité : elle ne pourra être déployée que sur deux serveurs (le serveur Web plus le serveur de base de données). Or, la tendance actuelle, renforcée par le Cloud Computing, utilise, en plus de ces deux serveurs, un serveur d'application qui exécute le cœur de l'application.

Cet article montre comment on peut utiliser le framework Spring couplé avec JPA pour bâtir une application web modulaire.

## Matériel requis

- Spring 3.0
- Java 1.6
- Eclipse avec le plug-in WTP (ici c'est Eclipse Galileo qui a été utilisé en version JEE developer)
- Apache-Tomcat 5 ou plus.

L'installation de Spring n'est pas requise car les fichiers sources de l'application, récupérables à l'adresse ci-dessous, contiennent aussi toutes les librairies de Spring pour ce projet :

<http://perso.efrei.fr/~charroux/JPA+SpringWebMVC/sources/>

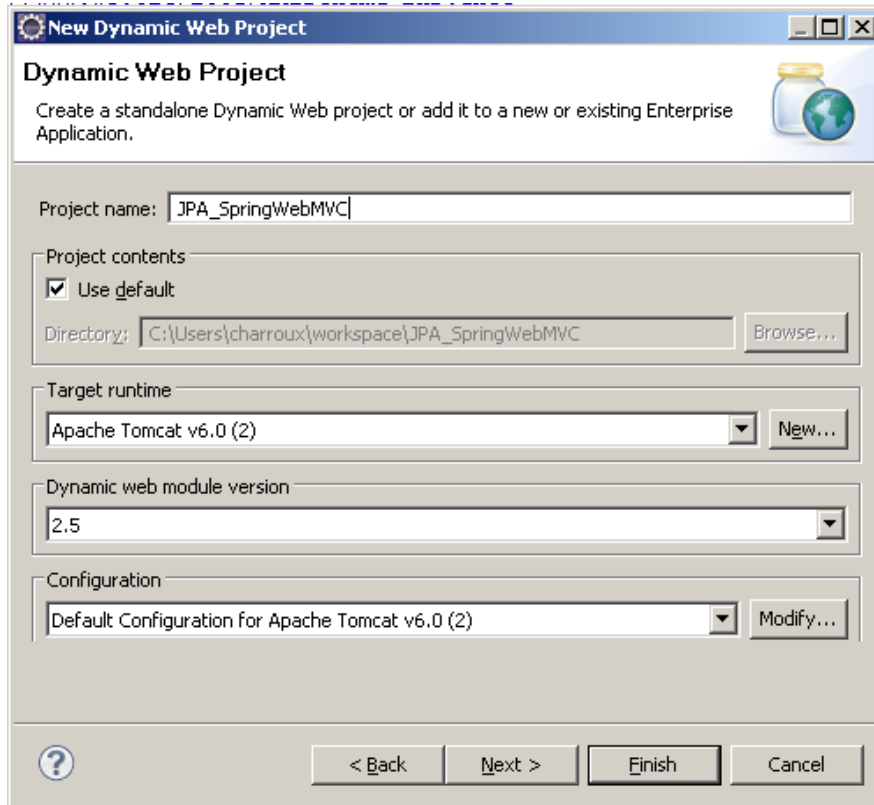
## Création d'un projet côté serveur

Avant tout, il faut télécharger Apache/Tomcat à partir du lien suivant :

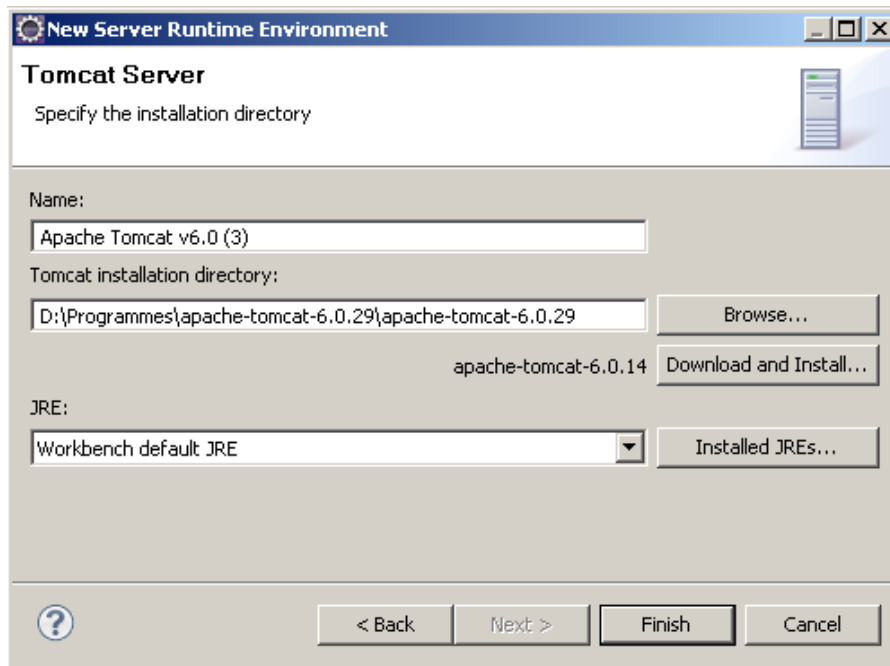
<http://tomcat.apache.org/>

Le téléchargement de la version Core et sa décompression sur votre machine suffit.

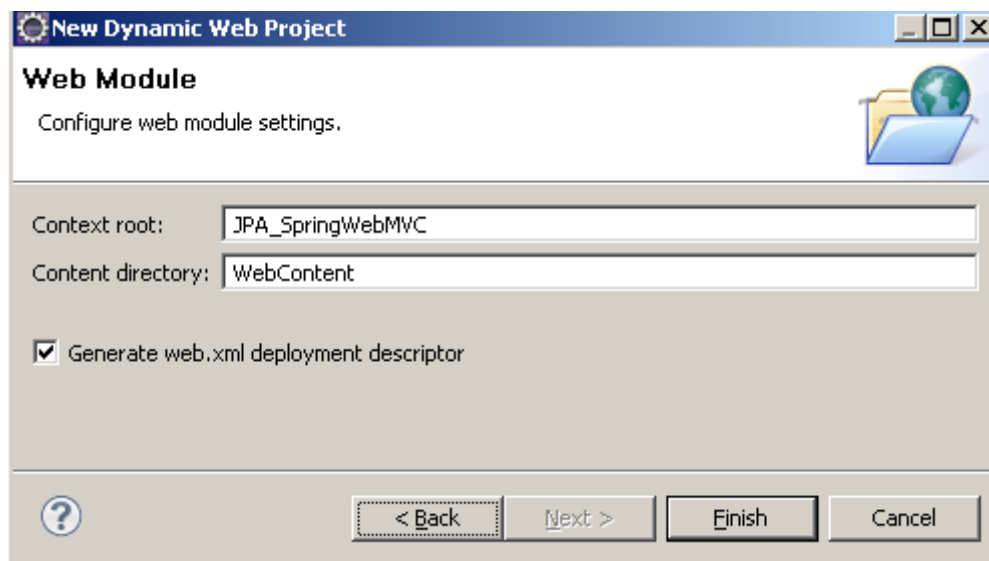
Création d'un projet web de type Dynamic Web Project sous Eclipse :



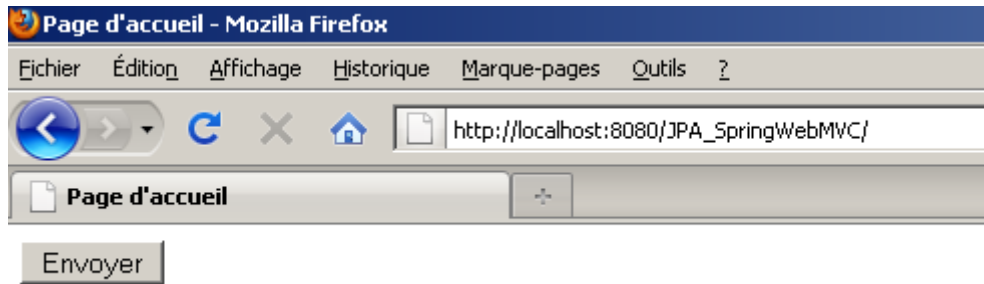
Où, Target Runtime a été initialisée avec Apache Tomcat en cliquant sur New et en sélectionnant le répertoire où est installé Apache Tomcat sur votre machine :



Vous pouvez alors naviguer de fenêtre en fenêtre en cliquant sur suivant jusqu'à parvenir à la fenêtre ci-dessous qui vous permet de saisir le « Context root ».

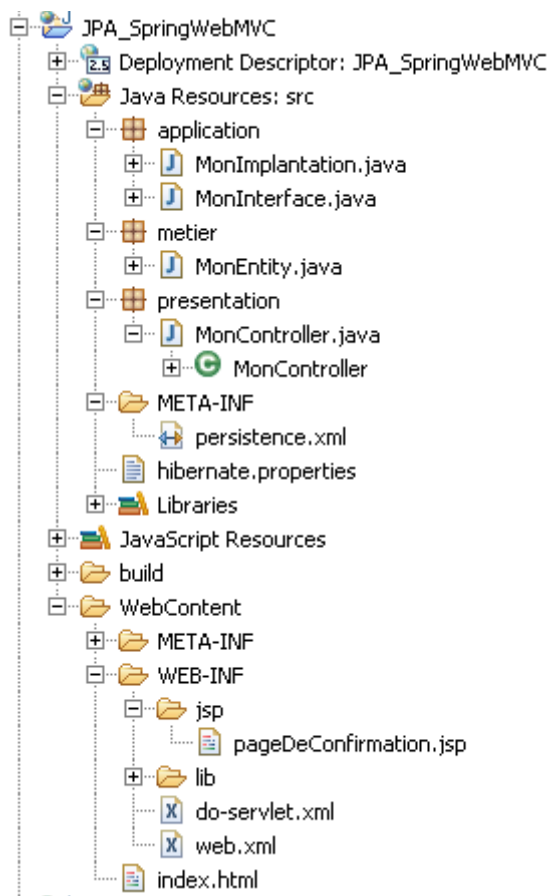


Ce contexte fera partie de l'URL pour accéder à votre serveur à partir d'un navigateur comme dans :



## Ajout des fichiers sources et des librairies

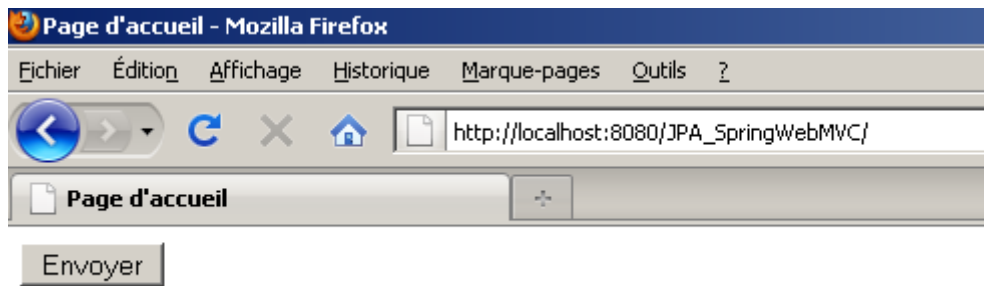
Décompressez le fichier téléchargé (voir le début de cet article) dans le répertoire du projet. Mettez à jour Eclipse via un F5 sur le projet. Celui-ci doit à présent se présenter comme suit :



Où on voit que la page HTML d'accueil de l'application est la page index.html. Elle contient un formulaire qui contient un simple bouton :

```
<html>
<form action="do/appelService" method=POST>
    <input type="submit" name="Cliquer ici pour lancer le service"/>
</form>
</body>
</html>
```

Qui une fois affiché dans le navigateur du client ressemblera à ce qui suit :



Le formulaire contient un champ action (`action="do/appelService"`) qui indique quel code Java va être exécuté sur le serveur quand les données du formulaire auront été postées. Ce code Java est contenu dans la classe `MonController` faisant partie du package `presentation`.

```
@Controller
public class MonController {

    @RequestMapping("/appelService")
    public ModelAndView appelApplication() {
        monInterface.createEntity(); // appel à l'application
        ModelAndView mav = new ModelAndView();
        mav.setViewName("pageDeConfirmation"); // affiche
pageDeConfirmation.jsp
        mav.addObject("titre", "Message de réponse :");// variable
titre dans pageDeConfirmation.jsp
        mav.addObject("message", "Entity créée !");// variable message
dans pageDeConfirmation.jsp
        return mav;
    }
}
```

`MonController` est annoté avec l'annotation `@Controller` Spring. C'est en effet un Controller au sens MVC du terme, c'est-à-dire le code Java à la norme Spring qui assure le lien entre une page web à afficher (la View), des données issues du formulaire HTML (le Model), et l'application située dans le package « application ». A noter que l'application est tellement simple qu'elle n'utilise pas de Model au sens MVC. La vue à afficher une fois que le contrôleur est terminé est précisée par l'instruction

```
mav.setViewName("pageDeConfirmation ");
```

où `pageDeConfirmation` est une page JSP qui est placée dans le répertoire `jsp`. Le lien avec l'application est réalisé par l'appel à la méthode `monInterface.createEntity();`. Le code correspondant est situé dans le package « application ». Il est important de remarquer que c'est la partie totalement réutilisable de l'application (elle peut être réutilisée dans une application qui n'est pas orientée web par exemple mais orientée services). Ce code applicatif fait appel au code métier du package du même nom qui contient les classes persistantes JPA synchronisées avec la base de donnée.

En résumé, l'architecture de l'application est composée des couches suivantes :

View
Model
Controller
Application
Metier
Services

Où la couche services offre la persistance ainsi que les transactions grâce à JPA.

## Configuration de l'application

Le répertoire META-INF contient le fichier persistence.xml qui déclare principalement les classes persistantes, ainsi que la configuration de l'accès à la base de données.

Le fichier de définition des composants pour Spring est le fichier do-servlet.xml. Il contient essentiellement la définition du Controller :

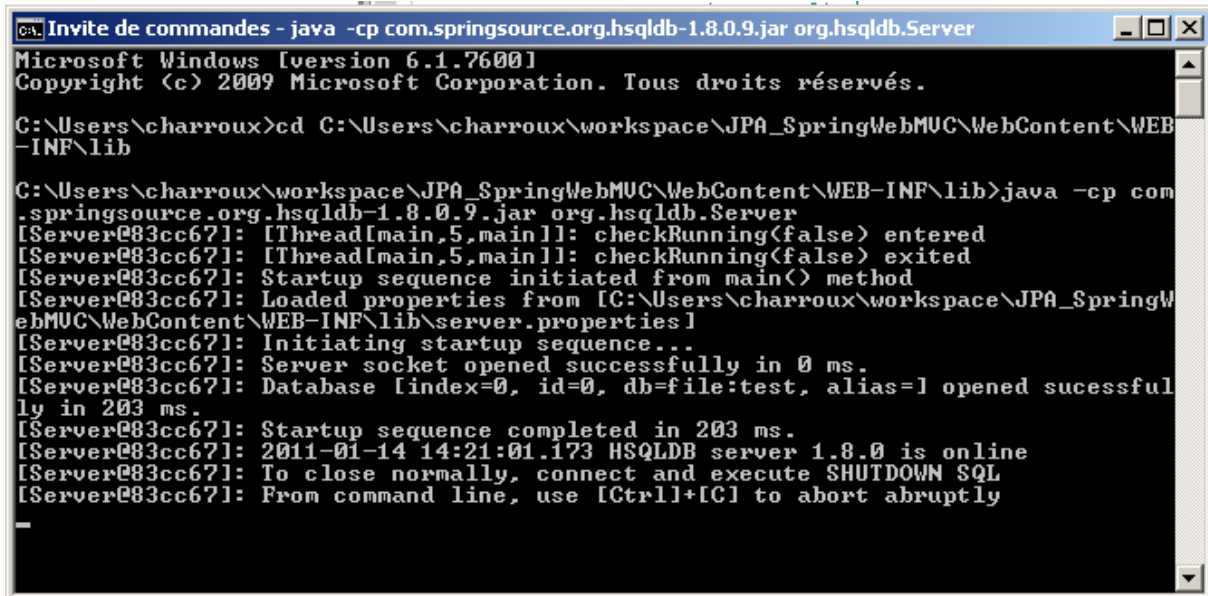
```
<bean id="appelServiceId" class="presentation.MonController">
  <property name="monInterface">
    <ref bean="monImplantation"/>
  </property>
</bean>
```

Où l'identifiant Spring *monImplantation*, identifie le composant de la couche application :

```
<bean id="monImplantation" class="application.MonImplantation">
  <constructor-arg ref="entityManagerFactory"/>
</bean>
```

## Lancement de l'application

Ici, c'est la base de données HSQLDB qui est utilisée. Elle se lance de la façon suivante à partir du répertoire où sont stockées les librairies :

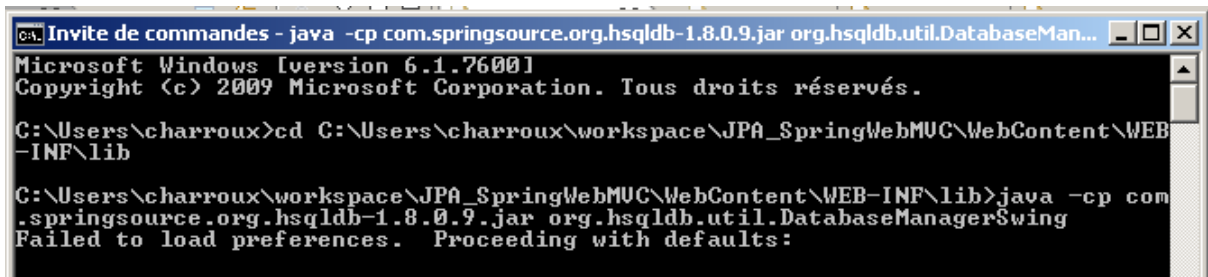


```
CA> Invite de commandes - java -cp com.springsource.org.hsqldb-1.8.0.9.jar org.hsqldb.Server
Microsoft Windows [version 6.1.7600]
Copyright (c) 2009 Microsoft Corporation. Tous droits réservés.

C:\Users\charroux>cd C:\Users\charroux\workspace\JPA_SpringWebMUC\WebContent\WEB-INF\lib

C:\Users\charroux\workspace\JPA_SpringWebMUC\WebContent\WEB-INF\lib>java -cp com.springsource.org.hsqldb-1.8.0.9.jar org.hsqldb.Server
[Server@83ccc67]: [Thread[main,5,main]]: checkRunning(false) entered
[Server@83ccc67]: [Thread[main,5,main]]: checkRunning(false) exited
[Server@83ccc67]: Startup sequence initiated from main() method
[Server@83ccc67]: Loaded properties from [C:\Users\charroux\workspace\JPA_SpringWebMUC\WebContent\WEB-INF\lib\server.properties]
[Server@83ccc67]: Initiating startup sequence...
[Server@83ccc67]: Server socket opened successfully in 0 ms.
[Server@83ccc67]: Database [index=0, id=0, db=file:test, alias=] opened successfully in 203 ms.
[Server@83ccc67]: Startup sequence completed in 203 ms.
[Server@83ccc67]: 2011-01-14 14:21:01.173 HSQLDB server 1.8.0 is online
[Server@83ccc67]: To close normally, connect and execute SHUTDOWN SQL
[Server@83ccc67]: From command line, use [Ctrl]+[C] to abort abruptly
```

Un utilitaire pour inspecter la base peut aussi être démarré :

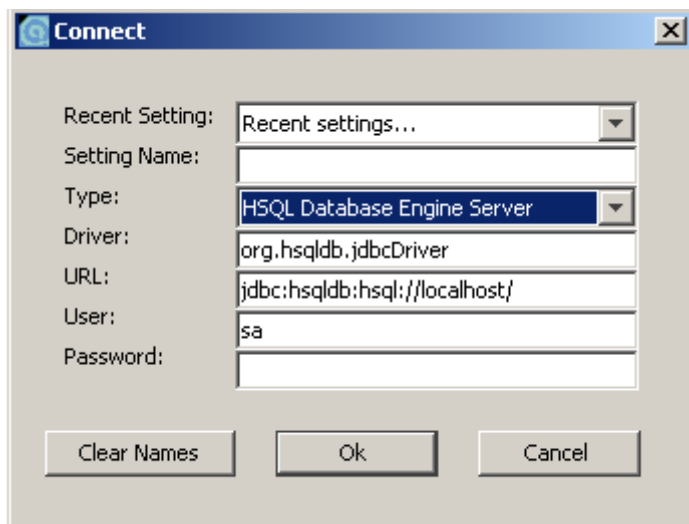


```
CA> Invite de commandes - java -cp com.springsource.org.hsqldb-1.8.0.9.jar org.hsqldb.util.DatabaseMan...
Microsoft Windows [version 6.1.7600]
Copyright (c) 2009 Microsoft Corporation. Tous droits réservés.

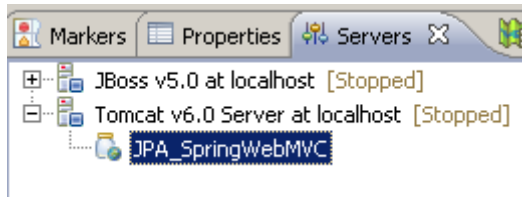
C:\Users\charroux>cd C:\Users\charroux\workspace\JPA_SpringWebMUC\WebContent\WEB-INF\lib

C:\Users\charroux\workspace\JPA_SpringWebMUC\WebContent\WEB-INF\lib>java -cp com.springsource.org.hsqldb-1.8.0.9.jar org.hsqldb.util.DatabaseManagerSwing
Failed to load preferences. Proceeding with defaults:
```

Attention ! Pour accéder à la base, il faut s'y connecter via le serveur :



Le pilotage d'Apache-Tomcat se fait dans l'onglet serveur (accessible via Windows-> Show view -> Other... -> Servers) :



Où il faut ajouter Tomcat, et ne pas oublier d'y insérer le projet (clic droit sur Tomcat -> menu Add and remove...). Il ne reste plus qu'à démarrer le serveur, ouvrir un navigateur et y saisir l'adresse du serveur (voir ci-dessus).

## Conclusion

Spring Web MVC est un framework qui permet de passer du monde du Web au monde Java. Couplé avec JPA, on a tous les moyens de réaliser une application complète qui va depuis l'interface utilisateur jusqu'à la sauvegarde des données en base de données. Cependant, pour que l'application puisse évoluer et être réutilisée, il ne faut pas placer de code applicatif directement dans les Controller MVC mais se contenter d'invoquer ce code. Viennent alors s'ajouter 3 couches (application, métier et services) aux 3 couches MVC.

La limite de ce système en couches est atteinte quand l'application se contente d'exposer dans des formulaires Web les données de la base de données. Dans ce cas, toutes ces couches sont superflues et on aura recours à une architecture différente comme REST. Mais c'est une autre histoire...