INTRODUCTION:

Java EE permet d’ajouter a java et java SE de départ de nombreuses bibliothèques qui vont donner accès à de nombreuses fonctionnalités puissantes, utiles et faciles à utiliser. l’objectif majeur de java entreprise edition et le development d’applications web robustes distribuées, déployées et exécutées sur un serveur d’applications(Openclassrooms).

Maven, géré par Apache software foundation, est un outil de gestion de projets java et surtout l’Entreprise Edition. Son objectif est d’assurer le bon ordre de production d’un projet.

AVANT PROPOS:

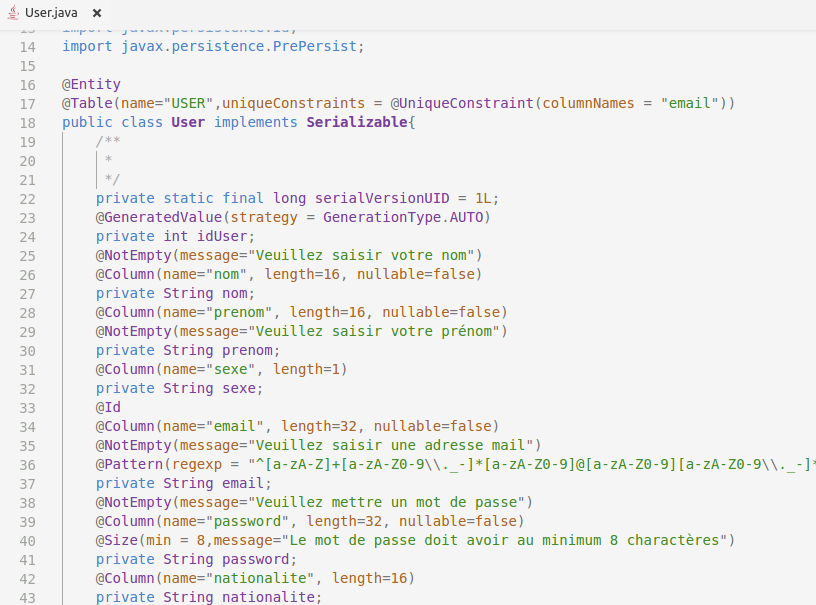
Durant le cours de programmation avancée IPS-M2, chaque étudiant doit produire une petite application web en respectant les contraintes données. L’application permettera de créer une base de données contentant des departement avec ses lieux et chaque lieu aura des monuments dont on associera des célébrités. Les données doivent être présentées sous formes de pages web visualisables et faciles à gérer. Une petite remarque est le fait que j’ai pas fait quelques fonction sur tous les pojos, par example, la validation des formulaire n’est disponible que dans le formulaire d’inscription des membres. Donc pour le faire sur le formulaire d’ajout de département, il faudra juste suivre la même procédure.

CONFIGURATION GENERALE:

Le modèle est constitué de differents pojos(classes classiques de java), ces derniers sont mappés à l’aide des annotations de Hibernate, le mapping va definir la méthode de l’enregistrement des instances dans la base de données de manière automatique en se servant de l’Entity Manager. La base de données utilisée dans le projet est Mysql. Les vues sont des Jsp qui génèrent des pages html en utilisant les données du modèle. La partie contrôleur est gérée par Spring qui mappera les url est executera la function correspondante. La partie authentifacation et sécurité(interception) est prise en charge avec Struts. Toute les dependences sont mentionnées dans le pom.xml du projet qui télecharge automatiquement depuis internet ce qu’il faut pour le bon fonctionnement des libraries. Il faut ajouter les dependences de Maven dans les propriétés du projet.

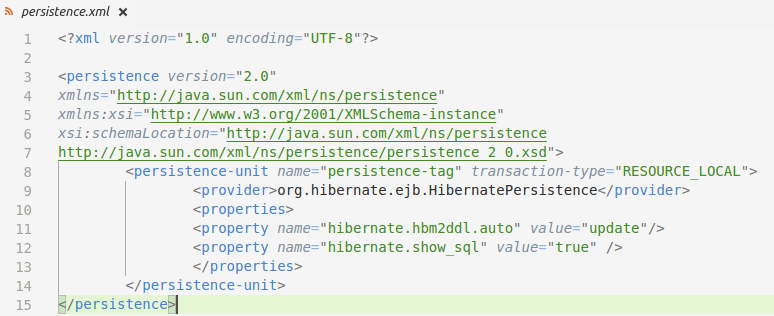
PARTIE MODELE:

voici à quoi ressemble un pojo de notre projet:

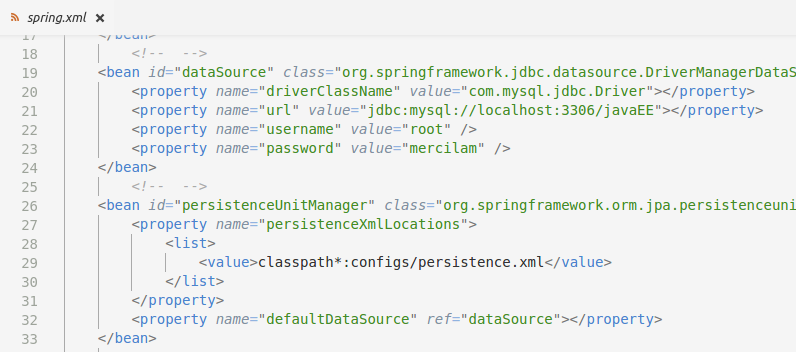


Differentes annotations nous sont offerte et nous permettent de manipuler le pojo comme on le souhaite(sous quel nom l’attribut va etre enregistré dans la base de données, les contraintes associées à cet attribut, la longueur minimale ou maximale, …). par example, l’annotation **@Pattern** nous offre la possibilité d’utiliser les expression regulières pour verfier la syntaxe des attributs avant de les ajouter dans la base de données, il existe aussi l’annotation **@Email** qui verifie que l’attribut est sous forme d’une email.

Voici la configuration simple de hibernate qui permet de prendre en charge la persistence:



est cette configuration est appelée dans la configuration de Spring comme suit, Et aussi il faut passer les identifiants de la base de données, le nom de la base de donnée et le jdbc.Driver (Mysql, Oracle ...):

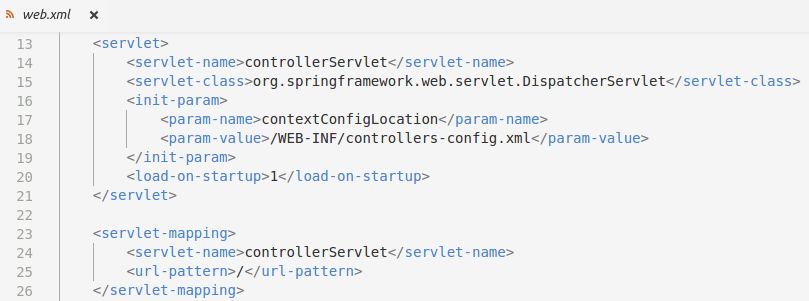
PARTIE CONTOLLEUR:

Le contrôleur est en charge de mapper les action a partir de leurs URL et executer une fonction correspondante. Pour mettre en place un controlleur dans Spring, il faut au minimum definr une classe Controlleur qui va etre annoté avec **@Controller** importé de org.springframework.stereotype .

Il faudrau aussi configurer quelques points :

* La balise <mvc:ressources/> permet de definir un mapping pour les ressources : les css, js, image …
* Dans la classe org.springframework.web.servlet.view.**InternalResourceViewResolver,** il faudra configurer la maniere d’interpretation des return des controlleurs.
* La balise <context:component-scan/> definie le package ou se trouve les contrôlleurs.
* Voici la configuration associée en code :

une fois cette configuration faite, il faudra l’inclure dans le fichier web.xml qui se trouve dans le WAB-INF de webapp/ de la maniere suivante :

Le controlleur Spring propose plusieurs fonctionalités pour couvrir les besoin coté web :

* Le contrôleur instancie l’interface du Dao et avec l’annotation **@AutoWired** il permet de remplire l’instance avec les bons paramètres pour prévenir les NullPointerException.
* l’annotation **@RequestMapping** permet de dire que la fonction qui suit est celle qui s’execute une fois son attribut « value » est passé en url. Son attribut « params » declare les parametres attendus passés à l’url mappée, ses paramètres pourront être récupérés à l’aide de l’annotation **@RequestParam** dans les arguments de la fonction.
* Les méthodes éxecutées retourne des chaines de charactères qui à l’aide de la configuration précédante retrouve la bonne page jsp correspondante.
* La classe Model et grace a sa méthode addModelAttribute nous permet de passer des objets a nos page jsp, le contenu de ces objets peut être afficher dans la page web grace à l’interpolation en mettant l’objet dans **${monObjet}** . Il permet aussi de mettre en place un binding entre les champs html avec une instance d’un pojos en utilisant des taglibs
* à l’aide de **@Valid** et la classe org.springframework.validation.**BindingResult** on peut verifier les champs remplis dans les formulaire qu’ils correspondent bien aux spécifications mentionnées dans les pojos et aucune contrainte n’est violée avant de passer à la partie sauvegarde dans la base de données
* plusieurs controlleurs peuvent être déclarés dans le package si une organisation s’impose.

PARTIE VUE :

La partie vue est gérée par les pages jsp qui elles aussi proposent de nombreuses fonctionalités permettant de venir au besoin coté web. En important des taglibs Struts ou Spring par example, on peut etendre les balises html en trouvant de nouveaux attributs.

- par Example pour completer la partie vue afin de binder des variables, on trouvera le nouveau attribut de la balise personnallisée de Spring <Spring:form **ModelAttribute=...**> qui pointe vers un nom d’une instance de pojo déjà définie dans le controleur et ajouté dans le modèle avant d’être passé dans la jsp. Ainsi, en utilisant la balise <Spring:input **path=...**> on peut definir un chemin vers les attributs de l’objet du modelAttribute.

- il nous est aussi proposé d’utiliser les conditions et les boucles dans les jsp, ce qui nous permet de faire d’interessantes mainoeuvres en tout simplicité

importer le taglib de la jstl

<%@taglib uri="http://java.sun.com/jsp/jstl/core" prefix="core"%>

Ensuite faire une codition

<core:**if** test=...>

…

…Ici le contenu à afficher si la condition dans le test est verifiée

…

</core:**if**>

Ou faire une boucle par example :

<c:**forEach** items="${AllMonuments}" var="unMonument">

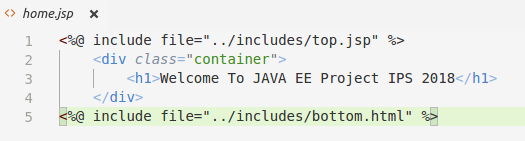
…

…Ici contenu de la boucle

…

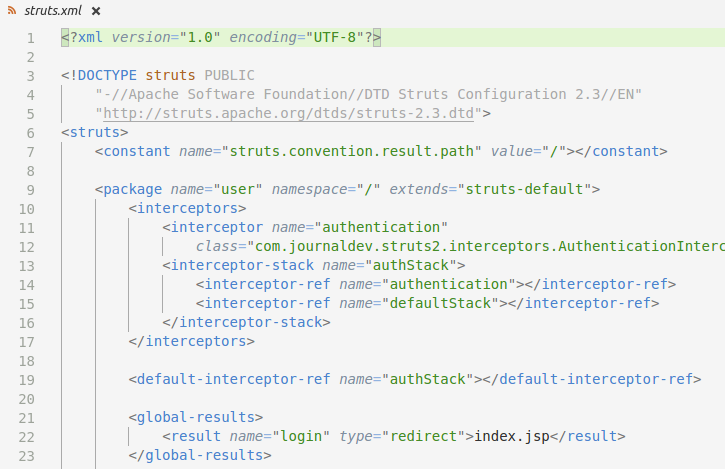
</core:**forEach**>

Une autre fonctionalité très partique dans le but d’organiser le code et eviter les repétitions. Il y’a possibilité de définir des templates qui sont fixe et de les appeler dans nos pages, un simple example, mettre en place un header et un footer et en suite les inclure dans toutes les pages. Il suffit de faire ainsi :

En utilisant les includes les pages deviennent très très simples et facile à modifier.

PARTIE SECURITE ET INTERCEPTION :

Malgré que Spring dispose d’un spring security facile à deployer et très efficace, j’ai préféré utiliser Struts pour intercepter les url et ainsi filtrerle passage aux pages. j’ai téléchargé l’example existant dans [www.journaldev.](http://www.journaldev.org/)com et je l’ai inclus dans le projet en l’adaptant à ce dernier. Voici ce qu’il faut au minimum pour faire marcher les interceptions de Struts :

* un fichier de configuration de struts est ajouter dans le dossier des ressources juste à côté de spring.xml on y’ajoute struts.xml, voici son contenu :
* La partie interceptor utilise celle par défault de struts
* le global-result permet de rediriger la page vers un resultats si le retout de la fonction ne correspond à aucun traitement
* l’example qui suit va utiliser l’option action de struts pour montrer l’action à executer en cas ou l’attribut name est appelé. Une petite remarque, struts fais toujours appel à la fonction intercept avant de continuer sur l’action
* la classe LoginAction a une fonction qui s’appelle execute() qui est notre action à faire, cette fonction retourne une chaine de charactère, et suivant le String retourné, on le map dans les options result de action dans le fichier de configuration.
* Exception dans le projet, la base de données est consulté directement depuis la fonction execute de la classe LoginAction au lieu d’appeller le DAO, car il etait pas possible d’inctancier le dao depuis cette classe(pas ajouté dans la configuration). Voici comment verifier la bonne authentifacation des utilisateurs :
* Dans la fonction intercept() , la session utilisateur est recupérée. Donc si aucun utilisateur n’est connecté, il sera automatiquement redirigé vers une page d’authentification. Si un utilisateur est déjà connecté, son rôle va être verifié. Ici soit il sera connecté en tant que touriste ou bien en tant que voyagiste. Le voyagiste étant avec plus d’accès, si un touriste essaye d’accéder à une page que seul un voyagiste peu consulter, il sera très rapidement informé qu’il n’a pas les droits. Le code suivant en montre comment est implémentée cette verification :
* Aussi pour intercepter les url, il faut aussi ajouter struts au web.xml de la manière suivante :



Voici le diagramme UML imposé et respecté :

