13 mai
2016

INFO Répartie
Projet Ergosum

LEO LETOURNEUR – LOIC GERLAND

POLYTECH

Table des matières

Ps : Le projet a été modifié sur Eclipse.

Architecture de Spring

Spring est un framework de développement Java basé sur la notion de conteneur léger en opposition aux serveurs d'applications Java EE. Une application utilisant Spring est le plus souvent structurée en trois couches (MVC) :

- **La couche présentation** : interface homme machine
- **La couche service** : interface métier et traitement métier
- **La couche persistance** : Accès aux données, recherche et persistance des objets en base de données.

Configuration

Maven :
Outil de construction de projets qui facilite et automatise certaines tâches d'un projet Java :

- Automatiser certaines tâches : compilation, tests unitaires et déploiement des applications.
- Gérer des dépendances de bibliothèques nécessaires au projet.
- Générer des documentations.

Pom (Project Object Model) :
C'est un fichier XML qui contient une description détaillée du projet, avec des informations concernant le versionning et la gestion des configurations, les dépendances, les ressources de l'application, les tests... C'est sur ce fichier que Maven s'appuie pour télécharger les dépendances d'un projet.

Création des classes métier en utilisant Hibernate :
Configuration, classe métier, couche hibernate :

Avant tout chose, si l'on utilise un projet avec Maven, il faut ajouter la dépendance à Hibernate dans le pom.xml. Ensuite il faut installer installer Hibernate dans le projet et le configurer avec la base de données et les fichiers de configuration souhaités (**image 1**). Pour la connecter la base de données, il faut créer le driver et l'associer à hibernate (**image 2**). Enfin, lorsque hibernate est bien configurer, on peut alors générer les classes à partir du menu « Hibernate Code Generation » (**image 3**).

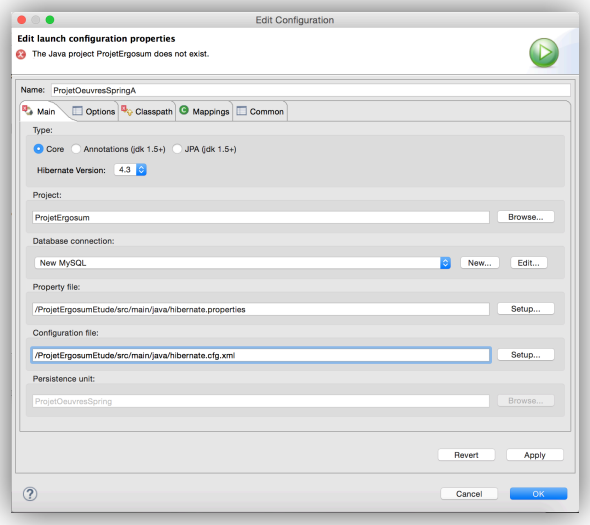


Image 1

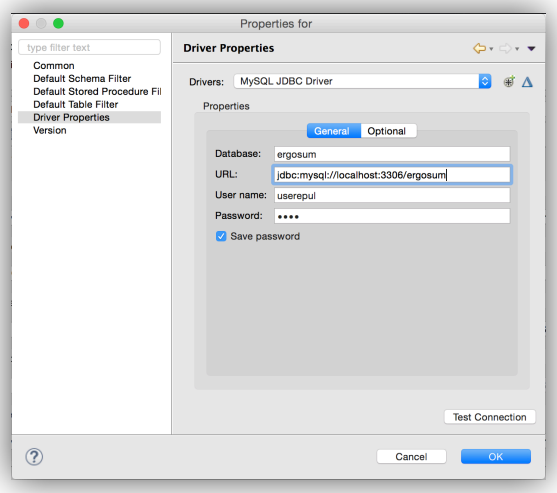


Image 2

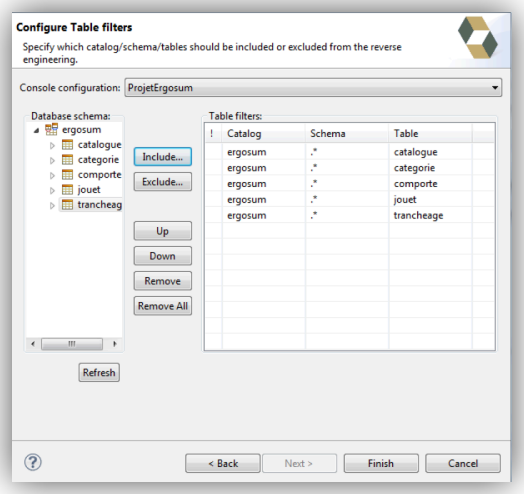


Image 3

Les classes métier sont alors générées. Il y a ici deux possibilités, soit les classes comporte des annotations de mapping, soit les classes sont associées à des fichiers hbm.xml qui comportent les annotations.

Annotations

```java
public class Adherent implements Serializable {
    private static final long serialVersionUID = 1L;

    @Id
    @GeneratedValue(strategy=GenerationType.AUTO)
    @Column(name="id_adherent", insertable = false, updatable = false)
    private int idAdherent;

    @Column(name="nom_adherent")
    private String nomAdherent;
```

Hbm.xml

```xml
<class name="hibernate.metier.Jouet" table="jouet" catalog="ergosum">
    <id name="numero" type="string">
        <column name="NUMERO" length="15" />
        <generator class="assigned" />
    </id>
    <many-to-one name="categorie" class="hibernate.metier.Categorie" fetch="select">
        <column name="CODECATEG" length="15" not-null="true" />
    </many-to-one>
```

Il est alors possible de créer des services très épurés comme ci-dessous :

```java
public Object find(Class classe, int id) {
    Object object = null;

    EntityTransaction transaction = startTransaction();
    transaction.begin();

    object = entitymanager.find(classe, id);
    entitymanager.close();
    emf.close();

    return object;
}
```

Fichier hibernate.cfg.xml

En remplacement ou en complément au fichier hibernate.properties, le fichier hibernate.cfg.xml permet de définir les propriétés concernant la connexion à la base de données. Les propriétés sont alors définies par un tag <property>. Le nom de la propriété est définie grâce à l'attribut « name » et sa valeur est fournie dans le corps du tag. Lors de la génération des classes, le fichier est rempli automatiquement avec l'adresse des fichiers de mapping (<mapping …>).

```xml
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE hibernate-configuration PUBLIC
"-//Hibernate/Hibernate Configuration DTD 3.0//EN"
"http://hibernate.sourceforge.net/hibernate-configuration-3.0.dtd">
<hibernate-configuration>
    <session-factory name="SessionFactory">
        <property name="hibernate.connection.driver_class">com.mysql.jdbc.Driver</property>
        <property name="hibernate.connection.password">epul</property>
        <property name="hibernate.connection.url">jdbc:mysql://localhost:3306/ergosum</property>
        <property name="hibernate.connection.username">userepul</property>
        <property name="hibernate.dialect">org.hibernate.dialect.MySQLDialect</property>
        <property name="hibernate.search.autoregister_listeners">false</property>

        <mapping resource="hibernate/metier/Comporte.hbm.xml" />
        <mapping resource="hibernate/metier/Jouet.hbm.xml" />
        <mapping resource="hibernate/metier/Categorie.hbm.xml" />
        <mapping resource="hibernate/metier/Catalogue.hbm.xml" />
        <mapping resource="hibernate/metier/Trancheage.hbm.xml" />
    </session-factory>
</hibernate-configuration>
```

La couche service avec la classe service :

Quelle est son rôle :

Le but de la couche de service est de définir les fonctions qui permettent d'interagir avec la base de données. Pour cela, elle s'appuie sur les classes métier générées par Hibernate et sur une classe qui va instancier une session afin de dialoguer avec la base de données. Généralement on définit deux fonctions dans cette classe qui vont permettre l'ouverture et la fermeture de la session. Par exemple pour ouvrir une session :

```java
public static Session currentSession() throws ServiceHibernateException {
    Session s = null;
    try {
        s = (Session) session.get();
        // Open a new Session, if this Thread has none yet
        if (s == null) {
            s = sessionFactory.openSession();
            session.set(s);
        }
    } catch (HibernateException ex) {
        throw new ServiceHibernateException(
                "Impossible d'accéder à la SessionFactory: "
                        + ex.getMessage(), ex);
    }
    return s;
}
```

Quel est le code qui lit le fichier hibernate.cfg.xml :

```
// on lit la configuration du fichier hibernate.cfg.xml
sessionFactory = new Configuration().configure("/com/ergosum/persistance/hibernate.cfg.xml").buildSessionFactory();
```

La ligne de code ci-dessus est celle qui permet de configurer le sessionFactory à partir de la configuration présente dans le fichier hibernate.cfg.xml. Ainsi à partir de ce sessionFactory on peut instancier des sessions afin de communiquer avec la base de données.

La couche métier

Rôle de la classe GestionErgosum :

La classe GestionErgosum permet d'exprimer les fonctions du contrôleur en requêtes que le ServiceHibernate va exécuter sur la base de données. Prenons comme exemple la méthode ajouter(Jouet unJouet) qui est appelée dans la fonction sauverJouet() du contrôleur. Le code a été commenté pour répondre à la question.

```
public void ajouter(Jouet unJouet) throws HibernateException, ServiceHibernateException {
    Transaction tx = null;
    try {
        //Récupération ou création de la SessionFactory qui exécute les requête
        session = ServiceHibernate.currentSession();
        //Début de la transaction
        tx = session.beginTransaction();
        //On sauvegarde le nouveau jouet
        session.save(unJouet);
        //On confirme la sauvegarde et termine la transaction
        tx.commit();
        //On termine la session
        ServiceHibernate.closeSession();
    }
    catch (ServiceHibernateException ex) {
        throw new ServiceHibernateException("Erreur de service Hibernate: " + ex.getMessage(), ex);
    } catch (HibernateException ex) {
        //Si une erreur survient, on fait un rollback
        if (tx != null) {
            tx.rollback();
        }
        throw new MonException("Erreur  Hibernate: ", ex.getMessage());
    }
    ServiceHibernate.closeSession();
}
```

Les requêtes, dans quel langage sont-elles écrites :

Les requêtes sont écrites en HQL (Hibernate Query Language). Hibernate propose son propre langage HQL dans le but d'offrir un langage d'interrogation commun à toutes les bases de données. Son intérêt est d'être indépendant de la base de données sous jacente : la requête SQL sera générée par Hibernate à partir du HQL en fonction de la base de données précisée via un dialect. Exemple : SELECT t FROM Trancheage AS t

Donnez un exemple en SQL et le traduire avec HQL:

Exemple : Nombre de maison dans un pays (Plusieurs maisons peuvent être à la même adresse)

SQL	HQL
SELECT COUNT (m.*) FROM maison m JOIN adresse a ON m.idAdresse=a.id JOIN pays p ON a.idPays=p.id GROUP BY p.id	SELECT COUNT(m.id) FROM maison as m JOIN m.adresse as a JOIN a.pays as p GROUP BY p.id

La couche Contrôle

Etude de sa mise en place :

Le contrôleur permet d'intercepter la requête et retourner la vue appropriée. Les méthodes du contrôleur sont appelées via des URL de la forme *.htm. Contrairement à l'écriture d'une servlet, il n'y a pas besoin d'écrire de boucle pour le dispatching des actions. Spring se charge de résoudre les URL et d'appeler la bonne méthode du contrôleur. Pour mettre en place ce contrôleur, nous devons créer une classe qui étend la classe « MultiActionController » et créer des méthodes pour chaque page du site. Pour qu'une méthode corresponde à une page, nous utilisons des annotations qui indiqueront quelle fonction utiliser suivant la requête.

```
@RequestMapping(value = "ProjetErgosum", method = RequestMethod.GET)
public ModelAndView home(Locale locale, Model model) {
    logger.info("Welcome home! The client locale is {}.", locale);
```

Dans l'exemple ci-dessus, la fonction home est appelé pour l'URL :
« localhost:8080/ProjetErgosum »

Rôle des fichiers :

En plus de la classe contrôleur, des fichiers de configuration sont nécessaires pour indiquer à Spring vers quelles classes rediriger les requêtes. Le fichier le plus important est le « Dispatcher-Servlet.xml » car c'est le point d'entrée de l'application. C'est lui qui effectue le mapping de l'application et distribue les requêtes aux servlets correspondantes. Un deuxième fichier qui peut être utile est « Servlet-Context.xml », il n'est pas nécessaire mais défini les beans appelés à partir des contrôleurs de l'application.

La couche Présentation

Quelles sont les technologies utilisées :

La couche présentation est surtout constituée de vues, de ressources comme les images et de fichiers JavaScripts. Cet ensemble d'éléments va définir le design du site internet. Les éléments importants de cette couche sont les vues. Elles sont basées sur la technologie JSP (JavaServer Pages) qui permet de créer dynamiquement des pages HTML à partir de balises personnalisées. Pour rendre ces pages plus dynamiques et apporter des fonctionnalités supplémentaires, on utilise des fichiers JavaScripts dans lesquels nous pouvons écrire des fonctions afin de rajouter de traitements coté client.

Les ressources images, le css :

Les images et le css sont deux ressources qui définissent exclusivement l'aspect visuel des vues. Dans le projet Ergosum, il y a beaucoup d'images qui représentent plusieurs éléments du site comme des icônes, des boutons ou encore le background. Pour parfaire, la mise en page du site et l'insertion des images, on utilise des fichiers css (Cascading Style Sheets) pour définir spécifiquement l'aspect des éléments.

Le rôle de la couche JSTL :

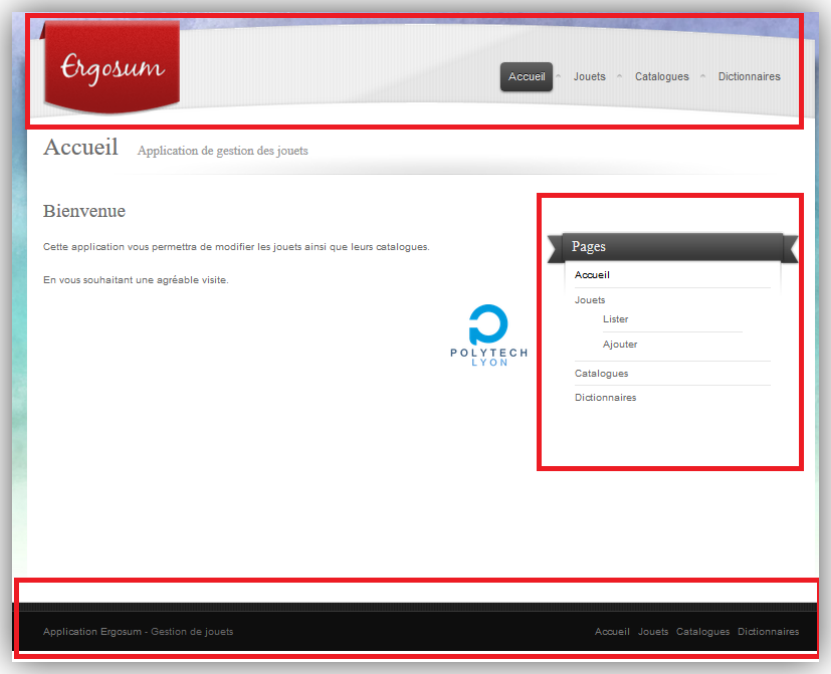
JSTL est l'acronyme de Java server page Standard Tag Library. C'est une librairie qui propose des tags personnalisés afin d'ajouter des fonctionnalités au développement des pages JSP. Il permet d'utiliser des balises XML et ainsi ajouter du code dans les pages. Cela permet d'effectuer divers traitements sur les objets comme des tests ou des boucles pour, par exemple, afficher des listes. Dans ergosum, une balise JSTL est utilisé pour afficher la liste des jouets comme on peut le voir sur le code ci-dessous :

```
<c:forEach items="${mesJouets}" var="item">
    <tr>
        <td><input type="checkbox" name="id" value="${item.numero}"></td>
        <td><a href="modifierJouet.htm?id=${item.numero}">Modifier</a></td>
        <td>${item.numero}</td>
        <td>${item.libelle}</td>
        <td>${item.categorie.codecateg}</td>
        <td>${item.trancheage.codetranche}</td>
    </tr>
</c:forEach>
```

Modification de traitements supplémentaires

Point 1

Layout



Toutes les pages du site ont une mise en forme commune qui est composé d'un header, d'une barre de navigation latérale sur la droite et d'un footer.

Cependant quand nous avons reçu le projet, tout le code de ce layout était dupliqué dans toutes les vues. Cela pose un problème de modularité car si l'on veut changer un des éléments du layout nous devons changer le code dans toutes les vues. Pour remédier à ce problème, nous avons donc mis en place un tag layout. Pour cela, nous utilisons une fonctionnalité du JSP qui nous permet de créer des balises personnalisées. Ainsi, dans un fichier layout.tag nous allons écrire le code du layout en rajoutant une balise spéciale <jsp :doBody> qui va indiquer l'endroit où le code des vues va être insérée.

```
<div id="content-wrap">
    <div id="page-wrap">
        <div class="page-title">
            <h1>${titre}</h1>
            <span>${phrase}</span>
        </div>
        <div id="side-content">
            <jsp:doBody />
        </div>
    </div>
```

Ensuite, il suffit de rajouter ce tag dans le code des vues pour indiquer qu'il faut le remplacer par le code du fichier.

```
<t:layout titre="Accueil" phrase="Application de gestion des jouets">
    <jsp:body>
            <h4>Bienvenue</h4>
```

Root

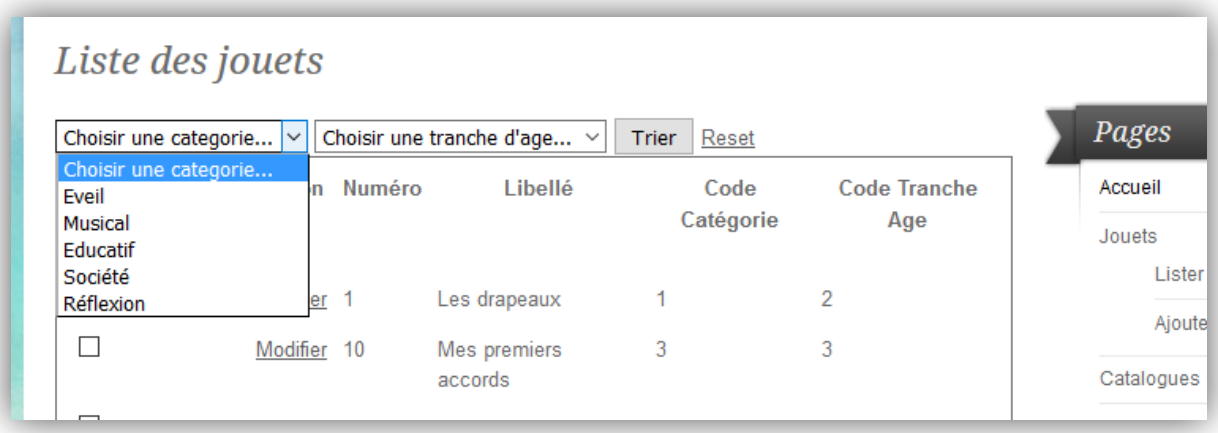
Nous avons mis à jour le contrôleur pour qu'il accepte la route ProjetErgosum/ en temps que chemin d'accueil. Nous avons aussi mis à jour le layout pour que les routes soit correcte depuis n'importe quel URL en ajoutant <%=request.getContextPath()%> devant nos URLs.

Point 2

Pour permettre une présentation des jouets par catégorie ou tranche d'âge nous avons modifié le contrôleur afin qu'il accepte en paramètre l'id de la catégorie ainsi que celui de la tranche d'âge. Ensuite, nous avons modifié la requête du service pour qu'elle trie les jouets en fonction de ces deux nouveaux paramètres.

```
public List<Jouet> listerTousLesJouets(String categorie, String trancheAge) throws HibernateException,
        ServiceHibernateException {
    try {

        boolean first = true;
        session = ServiceHibernate.currentSession();
        // On passe une requête SQL en utilisant les noms des fichiers hbm

        String marequete = "SELECT j  FROM Jouet AS j";

        if(categorie != null) {
            if(first) {
                marequete += " WHERE";
                first = false;
            }

            marequete += " j.categorie = "+categorie;
        }
        if(trancheAge != null) {
            if(first) {
                marequete += " WHERE";
                first = false;
            }
            else {
                marequete += " AND";
            }
            marequete += " j.trancheage = "+trancheAge;
        }
```

Ainsi si l'on précise par exemple un numéro de catégorie, on ajoutera un « WHERE » sur ce numéro afin de garder uniquement les jouets qui ont ce numéro. Coté utilisateur, on a rendu possible cette sélection en ajoutant deux listes déroulantes au-dessus du tableau qui permettent de choisir une catégorie et/ou une tranche d'âge.



Une fois les critères sélectionnés, il suffit de cliquer sur le bouton Trier pour que la page se recharge avec les nouveaux éléments. Nous avons rajouté un bouton Reset qui permet de remettre les tries à zéro.