**NOTE DE SYNTHÈSE TECHNIQUE**

**DE PROJET**

Création d’une application pour le suivi des étudiants de l’IUT

Sommaire

BAVIER Romain

HANOCQ Alexandre

MOOTINAL Gerardo

2011/2012

Afin d’appliquer les méthodologies et les notions enseignées en licence professionnelle à l’iut de Lens, nous devions réaliser en équipe de 3 personnes, un projet informatique d’une durée de six semaines. Ce projet avait pour but de mettre en pratique nos connaissances.

Pour comprendre la démarche que nous avons utilisée pour mener ce projet à son terme, notre rapport se structure de la façon suivante :

Dans une première partie, nous présenterons les objectifs du projet ainsi que le cahier des charges.

Dans une seconde partie, nous expliquerons plus en détail la démarche que nous avons utilisée ainsi que le coté technique pour mener à bien le projet et les problèmes que nous avons rencontrés.

Le projet réalisé consistait à réaliser une application permettant d’avoir un suivi scolaire et professionnel de chaque étudiant de l’IUT.

# LE PROJET

* 1. Elaboration du cahier des charges

Il fallait qu’elle soit sécurisée et surtout facile à maintenir. (Je sais pas quoi dire d’autre ??)

* 1. Analyse des Besoins

Suite à l’élaboration du cahier des charges, nous avons pu réaliser le MCD (modèle conceptuel de données), que nous avons complété en fonction des ajouts de fonctionnalités dites par notre tuteur de projet, Mr Hémery. Tout cela, pour répondre aux besoins de chacun.

MCD

* 1. Choix technologique

# Les technologies utilisées pour ce projet furent imposée dans le sujet du projet. Ces technologies étaient J2EE6 associé au Framework (Kit de composants logiciels structurels, qui servent à créer les fondations ainsi que les grandes lignes de tout ou d’une partie d'un logiciel, c'est-à-dire l’architecture du projet) JSF 2.

Pour ce qui est de la persistance de nos données, nous avons choisit eclipseLink, Framework open source de mapping objet-relationnel, fournissant une plateforme puissante et flexible permettant de stocker des objets Java dans une base de données relationnelle. Il est un dérivé de TopLink de la société Oracle.

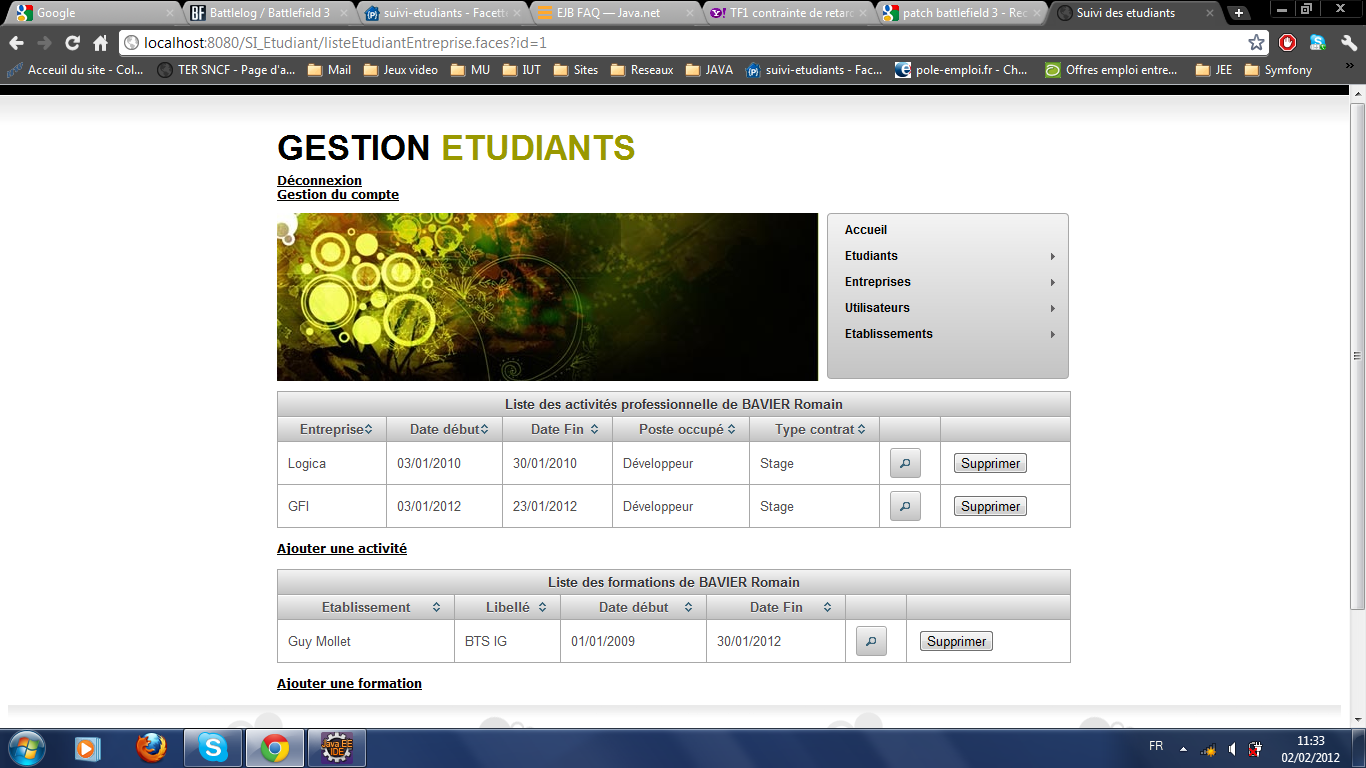
En ce qui concerne la base de données, nous avons utilisées dans un premier temps derby pour des raisons de simplicité, et dans un second temps nous somme passé sur le SGSBD POSTGREY.

Enfin le serveur d’application utilisé est glassfish.

* 1. Développement
     1. La base de données

Code BD.

* + 1. Structure générale de l’application



*Page activité d'un étudiant*

* + 1. La notion des droits et interface d’administration

L’accès à notre application se passe à travers un champ de connexion de notre application. ….

* + 1. Couche Métier

1.4.4.1 Les Entités

Les entités permettent d'encapsuler les données d'une occurrence d'une ou plusieurs tables. Ce sont de simples POJO (Plain Old Java Object). Un POJO est une classe Java qui n'implémente aucune interface particulière ni n'hérite d'aucune classe mère spécifique.

Un bean entité doit obligatoirement avoir un constructeur sans argument et la classe du bean doit obligatoirement être marquée avec l'annotation @javax.persistence.Entity. Cette annotation possède un attribut optionnel nommé name qui permet de préciser le nom de l'entité dans les requêtes. Par défaut, ce nom est celui de la classe de l'entité.

De plus, elle doit posséder au moins une propriété déclarer comme clé primaire avec l'annotation @Id, et implémenter *Serializable* processus de sauvegarde d'un objet complet, d'où il pourra être restauré à tout moment.

Ce bean est composé de propriétés qui seront mappés sur les champs de la table de la base de données sous jacente. Chaque propriété encapsule les données d'un champ d'une table. Ces propriétés sont utilisables au travers de simple accesseurs (getter/setter).

Ci-dessous, voici l’entité Etudiant.

package entity;  
  
@Entity  
@Table(name = "ETUDIANT")  
public class Etudiant implements Serializable {  
  
 private static final long serialVersionUID = 1L;  
  
 private long id;  
 private String nom;  
 private String prenom;  
 private Adresse adresse = new Adresse();  
 private Contact contact = new Contact();  
 private Set<EtudiantFormation> lesFormations = new HashSet<EtudiantFormation>();  
 private Set<EtudiantEntreprise> lesEntreprises = new HashSet<EtudiantEntreprise>();  
  
 public Etudiant() {  
 super();  
  
 }  
  
 @OneToMany(mappedBy = "etudiant", cascade = CascadeType.ALL, fetch = FetchType.LAZY, orphanRemoval = true)  
 public Set<EtudiantEntreprise> getLesEntreprises() {  
 return lesEntreprises;  
 }  
  
 public void setLesEntreprises(Set<EtudiantEntreprise> lesEntreprises) {  
 this.lesEntreprises = lesEntreprises;  
 }  
  
 @OneToMany(mappedBy = "etudiant", cascade = CascadeType.ALL, fetch = FetchType.LAZY, orphanRemoval = true)  
 public Set<EtudiantFormation> getLesFormations() {  
 return lesFormations;  
 }  
  
 public void setLesFormations(Set<EtudiantFormation> lesFormations) {  
 this.lesFormations = lesFormations;  
 }  
  
 @Id  
 @GeneratedValue  
 @Column(name = "ETUDIANT\_ID")  
 public long getId() {  
 return id;  
 }  
  
 public void setId(long id) {  
 this.id = id;  
 }  
  
 @Column(length = 70)  
 public String getNom() {  
 return nom;  
 }  
  
 public void setNom(String nom) {  
 this.nom = nom;  
 }  
  
 @Column(length = 70)  
 public String getPrenom() {  
 return prenom;  
 }  
  
 public void setPrenom(String prenom) {  
 this.prenom = prenom;  
 }  
  
 @Embedded  
 public Adresse getAdresse() {  
 return adresse;  
 }  
  
 @Embedded  
 public Contact getContact() {  
 return contact;  
 }  
  
 public void setContact(Contact contact) {  
 this.contact = contact;  
 }  
  
 public void setAdresse(Adresse adresse) {  
 this.adresse = adresse;  
 }  
  
 public String toString() {  
 return nom + " " + prenom;  
 }

Les annotations de type oneToMany permette de récupérer directement l’objet concerné de la BD.

1.4.4.2 La couche DAO

Nous avons utilisé les EJB dit session dans la couche DAO.

Un EJB est une architecture de composants logiciels côté serveur pour la plateforme de développement JEE. Les EJB session permettent  de proposer des services avec ou sans conservation d'état entre les appels.

Contrairement à @Statefull, l’annotation @Stateless en début de classe précise que cet EJB sera sans état : il ne conserve donc pas en mémoire ce qu’il a fait précédemment.

Chaque classe dao correspond à une table de notre base de donnée, et c’est dans ces classe que nous avons écrit les requête permettant d’interagir avec notre base de données comme l’ajout, la sélection, la modification.

package ejb;  
  
import java.util.List;  
  
import javax.ejb.EJBException;  
import javax.ejb.Stateless;  
import javax.persistence.EntityManager;  
import javax.persistence.PersistenceContext;  
  
import entity.Etudiant;  
  
@Stateless  
public class EtudiantEJB {  
 @PersistenceContext(unitName = "si\_etu")  
 private EntityManager em;  
  
 /\*  
 \* Recupere tous les etudiants  
 \*/  
 @SuppressWarnings("unchecked")  
 public List<Etudiant> findAllEtudiants() throws EJBException {  
 List<Etudiant> results = em.createQuery("select e from Etudiant e")  
 .getResultList();  
 return results;  
 }  
  
 /\*  
 \* Cree un etudiant dans la BDD  
 \*/  
 public Etudiant createEtudiant(Etudiant etudiant) throws EJBException {  
 em.persist(etudiant);  
 return etudiant;  
 }  
  
 /\*  
 \* Supprime un etudiant  
 \*/  
 public void removeEtudiant(Etudiant etudiant) throws EJBException {  
 // em.merge(etudiant) => entité doit être détaché du bean sinon cela ne  
 // fonctionne pas  
 em.remove(em.merge(etudiant));  
 }  
  
 /\*  
 \* Met a jour un etudiant  
 \*/  
 public void updateEtudiant(Etudiant etudiant) {  
 em.merge(etudiant);  
 }  
  
 /\*  
 \* Cherche et retourne un etudiant par son identifiant  
 \*/  
 public Etudiant findEtudiantById(Long id) {  
 Etudiant results = (Etudiant) em.find(Etudiant.class, id);  
 return results;  
 }  
  
 @SuppressWarnings("unchecked")  
 public List<Etudiant> findAllEtudiantsByEnt(Long ent) throws EJBException {  
 List<Etudiant> results = em  
 .createQuery(  
 "select DISTINCT e from Etudiant e, EtudiantEntreprise ent where e.id=ent.id.etudiantId and ent.id.entrepriseId=:ent")  
 .setParameter("ent", ent).getResultList();  
 return results;  
 }  
}

En clair, en utilisant l’annotation [PersistenceContext](http://java.sun.com/javaee/5/docs/api/javax/persistence/PersistenceContext.html), l'EJB récupère une instance de l'interface [EntityManager](http://java.sun.com/javaee/5/docs/api/javax/persistence/EntityManager.html). Celle-ci va permettre les opérations de persistance sur les entités, tels que create, select, update et delete. Cette instance est préparée par le serveur d'application à partir du fichier persistence.xml.

* + 1. La couche contrôle

Cette couche fait une sorte de pont entre la couche vue et la couche DAO.

Nous ne l’avons pas configuré dans le fichier managedBean.xml, mais choisie de mettre des annotations directement dans un fichier JAVA.

En ce qui concerne les scopes utilisés, nous avons utilisé pour pratiquement pour tout les beans des conversationScoped.

En effet, ceux-ci permettent de garder plus longtemps en mémoire un bean ainsi que les valeurs inscrites. Il est donc possible d’ouvrir une conversation dans une page, d’y instancier des objets, de les récupérer dans une autre page, et de fermer la conversation

Un autre avantage, est que les conversation scoped sont compatible avec ajax.

Pour effectuer cela, on ajoute @ConversationScoped dans le backing bean et on injecte une variable de type Conversation. Puis on appelle conversation.begin() et conversation.end() pour contrôler la création et la suppression de l'instance du bean.

package bean;  
  
import java.io.Serializable;  
//named : propre à conversationScoped, ne surtout pas utiliser managedBean   
@Named(value = "etudiantBean")  
@ConversationScoped  
public class EtudiantBean implements Serializable {  
 private static final long serialVersionUID = 1L;  
  
 @Inject  
 Conversation conversation;  
  
 @EJB  
 private EtudiantEJB etudiantEJB;  
  
 private Etudiant etudiant = new Etudiant();  
  
 private List<Etudiant> etudiants = new ArrayList<Etudiant>();  
  
 private HashMap<Long, Boolean> checked = new HashMap<Long, Boolean>();  
  
 /\*  
 \* on instancie pas, c'est l'etudiant que l'on récupere à partir du jsf, il  
 \* servira pour recupérer l'étudiant que l'on souhaite modifier (voir jsf  
 \* modifEtudiant)  
 \*/  
 private Etudiant editEtudiant;  
  
 private Etudiant selectedEtudiant;  
  
 @PostConstruct  
 public void init() {  
 if (conversation.isTransient()) {  
 conversation.begin();  
 }  
 try {  
 // si l'on récupére ent dans l'url, alors la liste d'étudiant  
 // affichée est la liste d'étudiant de lentreprise  
 if (this.getPassedParameter() != null) {  
 Long ent = Long.parseLong(this.getPassedParameter());  
  
 etudiants = etudiantEJB.findAllEtudiantsByEnt(ent);  
 } else {  
 etudiants = etudiantEJB.findAllEtudiants();  
 }  
  
 } catch (EJBException e) {  
  
 Redirection.erreurXhtml();  
 }  
  
 }  
  
 public String ajout() {  
 // Ajout  
 this.etudiantEJB.createEtudiant(etudiant);  
 // Fin  
 conversation.end();  
  
 /\*  
 \* si on est en session, actualisation de la liste d'étudiant et de  
 \* l'étudiant etudiants = etudiantEJB.findAllEtudiants(); etudiant =new  
 \* Etudiant();  
 \*/  
 Redirection.listeEtudiants();  
 return "listEtudiant";  
 }  
  
 // procédure pemettant de supprimer les etudiants selectionnés de la liste  
 // checked  
 public void supprimer() {  
 for (Etudiant unEtudiant : etudiants) {  
 if (checked.get(unEtudiant.getId())) {  
 etudiantEJB.removeEtudiant(unEtudiant);  
 }  
 }  
 conversation.end();  
 /\*  
 \* si on est en session, actualisation de la liste d'étudiant  
 \* etudiants=etudiantEJB.findAllEtudiants();  
 \*/  
  
 Redirection.listeEtudiants();  
 }  
  
 // fonction permettant de modifier un etudiant => retourne list (voir  
 // face-config.xml)  
 public String modifier() {  
 etudiantEJB.updateEtudiant(editEtudiant);  
 conversation.end();  
 return "listEtudiant";  
 }  
  
 // fonction retournant edit (voir face-config.xml)  
 public String edit() {  
 return "edit";  
 }  
  
 public void creerListeEtudiantsPDF() {  
 // Appeler la procédure pour creer mon PDF d'etudiants  
 PDF.CreerListeEtudiantsPDF(etudiants, "liste\_etudiants");  
 }  
  
 public String getPassedParameter() {  
 FacesContext facesContext = FacesContext.getCurrentInstance();  
 String parametreId = (String) facesContext.getExternalContext()  
 .getRequestParameterMap().get("ent");  
 return parametreId;  
 }  
  
 /\*  
 \* Getters & Setters  
 \*/

* + 1. La couche vue

Cette couche correspond à l’affichage de notre site. Nous avons donc utilisé dans un premier temps les balises facelets du framework JSF 2.0 (langage EL).

*Facelets* est une technologie de présentation pour le développement d'applications web en *Java*. Une page *JSP* est transformée en une *Servlet* qui possède un cycle de vie différent de celui de *JSF*, ce qui peut être source de confusion et de problèmes. A l'inverse, *Facelets* est spécifiquement développé pour*JSF* et est plus performant et léger.

*Facelets* est basé sur *xml*, c'est pour cette raison que les vues sous *facelets* sont des pages *xhtml* et qu'elles doivent impérativement respecter la structure d'un document *xml*.

Une fois le projet bien avancer, nous avons intégrer la librairie PrimeFaces à notre projet. Cette librairie additionnelle offre de nouveaux composants plus riches a JSF qui s’avère très limité. Les points forts de cette librairie sont surtout sa simplicité à mettre en place, la richesse de ses composants et le fait qu’elle supporte l’AJAX.

Ci-dessous voici une vue primefaces.

* + 1. Découpage des tâches.

Romain : Structure du site générale + etudiant, user, entreprise, etudiant\_entreprise + les droits + gestion du compte + test unitaire

Gerardo : entreprise formation, etudiant formation établissement.

Alex : Base de données + design du site + pdf

Pour mettre en commun l’avancé de chacun, nous avons utilisé SVN, installer le plugin subclipse à eclipse permettant d’intéragir directement avec le serveur svn depuis l’interface d’eclipse. Cela nous à permis de gagner du temps.

* + 1. Problèmes rencontrés.

Romain : Je n’est pas particulièrement eut de problème étant donné que j’avais déjà pratiqué le JAVA, la persistance, le beanbinding JAVA durant mon BTS Informatique de gestion et eut quelques cours à l’IUT portant sur J2EE6 et le Framework JSF. De plus, ces 2 technologies étant très utilisées dans le domaine de l’informatique, la documentation assez conséquente sur internet ont permis de mieux assimilé la chose et de résoudre les problèmes que j’ai put rencontrés lors du développement.

pb au niveau du mapping des entités. En effet lorsque l’on ajoute une formation celle – ci est bien ajouté à la BD, cependant celle-ci devrait aussi s’ajouté à la liste formation de établissement. Or cela ne fonctionne pas. Ce qui fait que lorsque j’effectue une requête de type etablissement.getLesFormations(), celle-ci ne s’actualise pas. La solution aurait été de refaire une requête dans la la bd met ce traitement fais un peu « lourd » à mon gout, les entités crée permettant la navigabilité.

+ assimilation de l’étendue des scope.

Pb avec SVN, et classpath…