

Philippe TANGUY

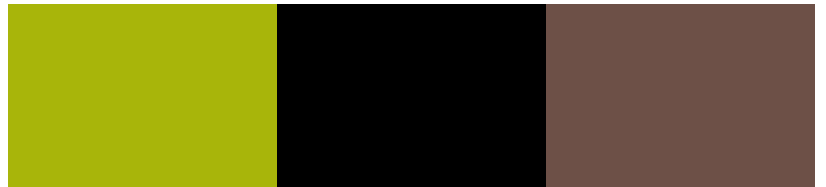
Département LUSI - Télécom Bretagne

*philippe.tanguy@telecom-bretagne.eu*

Mayte SEGARRA

Département INFORMATIQUE - Télécom Bretagne

*mt.segarra@telecom-bretagne.eu*



## **Architecture des SI**

### **Fil rouge : projet de développement**

**UV FIP INF 210**

Année scolaire 2015-2016



Ce document est diffusé sous licence Creative Commons BY NC SA.

<http://creativecommons.org/licenses/by-nc-sa/3.0/fr/>

**Vous êtes autorisé à :**

- **Partager** — copier, distribuer et communiquer le matériel par tous moyens et sous tous formats
- **Adapter** — remixer, transformer et créer à partir du matériel

**selon les conditions suivantes :**

- **Paternité** — Vous devez citer le nom de l'auteur original de la manière indiquée par l'auteur de l'oeuvre ou le titulaire des droits qui vous confère cette autorisation (mais pas d'une manière qui suggérerait qu'ils vous soutiennent ou approuvent votre utilisation de l'oeuvre).
- **Pas d'Utilisation Commerciale** — Vous n'êtes pas autorisé à faire un usage commercial de cette Oeuvre, tout ou partie du matériel la composant.
- **Partage dans les Mêmes Conditions** — Dans le cas où vous effectuez un remix, que vous transformez, ou créez à partir du matériel composant l'Oeuvre originale, vous devez diffuser l'Oeuvre modifiée dans les même conditions, c'est à dire avec la même licence avec laquelle l'Oeuvre originale a été diffusée.



# Sommaire

<b>1</b>	<b>Présentation</b>	<b>5</b>
1.1	Notes préliminaires	5
1.2	Déroulement du projet	5
1.3	Cahier des charges : <i>gestion de cabinet de recrutement</i>	6
1.4	Livrables attendus	6
<b>2</b>	<b>Cahier des charges</b>	<b>7</b>
2.1	Introduction	7
2.1.1	Objet du document	7
2.2	Les objectifs du produit	7
2.2.1	Définition du produit	7
2.2.2	Contexte économique du produit	7
2.3	Exigences sur le produit	7
2.3.1	Exigences fonctionnelles	7
2.3.2	Exigences non fonctionnelles	14
2.3.3	Exigences concernant le développement du produit	15
2.4	Synthèse des exigences	16
2.4.1	Hierarchisation des capacités fonctionnelles	16
<b>3</b>	<b>Fiche tâches/temps</b>	<b>19</b>
3.1	Contenu	19
3.1.1	Exemple de fiche tâches/temps :	20
<b>4</b>	<b>Partie 1 : conception des données de l'application</b>	<b>21</b>
4.1	Déroulement	21
4.2	Introduction	21
4.2.1	L'application	21
4.2.2	L'environnement de travail – les outils	22
4.2.3	Mise en route de l'ébauche de l'application	22
4.2.4	Architecture de l'application	23
4.2.5	Organisation des projets de développement dans Eclipse	25
4.3	Génération de la base de données	25
4.3.1	Schéma conceptuel ( <i>diagramme de classes</i> ) des données de l'application	26
4.3.2	Travail à réaliser	26

<b>4.4</b>	<b>Livraison</b>	<b>28</b>
<b>5</b>	<b>Partie 2 : <i>persistance des données (JPA)</i></b>	<b>29</b>
<b>5.1</b>	<b>Déroulement</b>	<b>29</b>
<b>5.2</b>	<b>Informations préliminaires</b>	<b>30</b>
5.2.1	Configuration d'Eclipse pour l'accès à PostgreSQL	30
5.2.2	Configuration d'Eclipse pour l'accès à GlassFish	30
5.2.3	Configuration de GlassFish pour l'accès à PostgreSQL	30
<b>5.3</b>	<b>Génération des classes entités</b>	<b>31</b>
<b>5.4</b>	<b>Création des DAO</b>	<b>33</b>
5.4.1	Implémentation complète du DAO de la classe entité Entreprise : EntrepriseDAO	33
5.4.2	Création des autres DAO	34
<b>5.5</b>	<b>Procédure de contrôle du bon fonctionnement des DAO</b>	<b>35</b>
<b>5.6</b>	<b>Livraison</b>	<b>35</b>
<b>6</b>	<b>Annexes au document de la partie 2</b>	<b>37</b>
<b>6.1</b>	<b>Gestion correcte des deux extrémités d'une association avec JPA</b>	<b>37</b>
<b>6.2</b>	<b>Complément d'information sur les états d'un bean entité – Incidence sur la suppression</b>	<b>38</b>
<b>6.3</b>	<b>Requêtes JPQL dans les différents DAO</b>	<b>39</b>
<b>7</b>	<b>Partie 3 : <i>services métier et couche présentation</i></b>	<b>43</b>
<b>7.1</b>	<b>Déroulement</b>	<b>43</b>
<b>7.2</b>	<b>Poursuite du développement du composant <i>ServiceEntreprise</i></b>	<b>44</b>
7.2.1	Examen de l'état actuel de la classe	44
7.2.2	Ajout de nouvelles fonctionnalités	44
<b>7.3</b>	<b>Ajout de nouveaux composants</b>	<b>45</b>
7.3.1	Création d'un bean session	45
7.3.2	Référencement du nouveau bean dans la classe ServicesLocator	45
<b>7.4</b>	<b>La couche présentation : l'interface graphique de l'application</b>	<b>45</b>
7.4.1	Examen de l'existant	46
7.4.2	Gestion des formulaires	47
7.4.3	Divers	47
<b>8</b>	<b>Partie 4 : <i>syndication des informations</i></b>	<b>49</b>
<b>8.1</b>	<b>Déroulement</b>	<b>49</b>
<b>8.2</b>	<b>Les technologies mises en œuvre</b>	<b>50</b>
8.2.1	Le format RSS	50
8.2.2	Java API for XML Binding (JAXB)	52
<b>8.3</b>	<b>Travail à réaliser</b>	<b>53</b>
8.3.1	Génération des classes	53
8.3.2	Génération du flux RSS	53



# 1 — Présentation

## 1.1 Notes préliminaires

Le fil rouge du module INF 211 est un projet de développement logiciel qui reprend les enseignements vus dans le module :

⇒ Base de données :

- L'application à développer est une application persistante : les données ne sont pas volatiles, elles persistent dans un SGBDR (PostgreSQL).
- La gestion de la persistance sera confiée au framework de mapping objet/relationnel inclus dans la technologie EJB (JPA).

⇒ Architecture technique :

- L'architecture technique utilisée sera une architecture 3-tier (technologie EJB).
- L'application est une application Web dynamique (le langage utilisé reste toujours le langage Java avec la technologie JSP).
- L'hébergement de l'application se fera au sein d'un serveur d'applications (GlassFish, serveur Open Source JEE 6).

⇒ XML :

- L'application publiera certaines de ses données (syndication) de manière automatique dans un flux RSS. Ce format est basé sur la technologie XML.

Les objectifs du fil rouge de deuxième année sont clairement des objectifs technologiques qui se situent au niveau de la maîtrise d'œuvre. Les outils liés à l'approche gestion de projet vus en première année (cycle en V, analyse des besoins, spécifications, etc.) seront utilisés mais sont supposés maîtrisés.

## 1.2 Déroulement du projet

Le fil rouge se déroule autour de quatre parties principales suivies par la recette de l'application :

### 1. **Partie 1 : conception des données de l'application**

Séance 1 (15 septembre : 1 h 30)

- Mise en place + compréhension du sujet.
- Conception du schéma logique et du script SQL de création des tables de la base de données.

### 2. **Partie 2 : gestion de la persistance de l'application**

Séance 2 (29 septembre : 3 h) et séance 3 (06 octobre : 3h)

- Génération des beans entités et développement des composants DAO

### 3. **Partie 3 : gestion des services métier et de la couche présentation**

Séance 4 (15 octobre : 3 h) et séance 5 (19 octobre : 3 h)

- Développement des composants EJB qui assurent les services métier.
- Le développement de la couche interface utilisateur se fera conjointement.

#### 4. **Partie 4 : gestion de la syndication des informations**

Séance 6 (27 octobre : 3 h)

- Développement du composant qui assure la génération du flux RSS.

#### 5. **Recette de l'application**

Séance 7 (06 novembre : 3 h)

Il y a donc 19 h 30 de travail encadré prévu à l'emploi du temps. Un temps de travail personnel est aussi à prévoir pour la réalisation du projet, voir dans le cahier des charges, le paragraphe *Objectifs de coûts*, page 15.

### 1.3 **Cahier des charges : gestion de cabinet de recrutement**

Le cahier des charges de l'application à développer vous est fourni. Il décrit l'intégralité des fonctionnalités à intégrer dans l'application (certaines sont obligatoires, d'autres optionnelles).

Une démonstration est disponible à l'URL :

[http://srv-labs-006.enst-bretagne.fr/CabinetRecrutement\\_WEB/](http://srv-labs-006.enst-bretagne.fr/CabinetRecrutement_WEB/)

Si certains dysfonctionnements apparaissent ou si l'ergonomie de l'interface vous paraît ambiguë, merci de le signaler.

### 1.4 **Livrables attendus**

Les livrables seront à déposer sur Moodle au format PDF pour les documents écrits et au sein d'une archive (zip, rar, 7z, etc.) pour la partie développement. La liste des livrables est décrite dans le cahier des charges, partie *objectifs de délais*, page 15.

Note : les livrables serviront à l'évaluation du fil rouge qui compte pour 30 % de la note finale de l'UV FIP INF 210.



## 2 — Cahier des charges

### 2.1 Introduction

#### 2.1.1 Objet du document

Ce document décrit les services que doit rendre l'application « cabinet de recrutement » et les exigences qu'elle doit satisfaire.

### 2.2 Les objectifs du produit

#### 2.2.1 Définition du produit

Un cabinet de recrutement est une structure qui se trouve à l'interface entre des entreprises et des chercheurs d'emploi. Les entreprises y déposent leurs offres et les chercheurs d'emploi y déposent leur candidature. Le cabinet de recrutement assure leur mise en relation.

Le système d'information à développer s'adresse donc à ces deux types d'acteur.

La version de base de ce système sera déployée sur un serveur (application Web), l'application sera accessible par l'intermédiaire d'un navigateur Web (*Firefox*, *Internet Explorer*, etc.).

Le système publie par ailleurs les informations concernant les offres d'emploi dans un flux RSS, les informations publiées par ce canal peuvent être lues par n'importe quel lecteur de flux (agrégateur RSS).

#### 2.2.2 Contexte économique du produit

Il n'y a pas de contexte économique associé au produit, celui-ci est développé dans un contexte pédagogique.

### 2.3 Exigences sur le produit

#### 2.3.1 Exigences fonctionnelles

##### Description des fonctionnalités

Les différentes fonctionnalités sont organisées par groupes (fonctionnalités liées aux entreprises, aux offres d'emploi, etc.) et sont, pour chacune d'entre-elles, marquées avec un niveau d'importance :

- Vitale : fonctionnalités obligatoires qui doivent être présentes dans la livraison finale.
- Importante : fonctionnalités obligatoires qui doivent être présentes la livraison finale mais qui feront l'objet d'un deuxième incrément.
- Mineure : fonctionnalités optionnelles.

Une présentation synthétique de ces niveaux d'importance est faite dans la partie 2.4 page 16.

### Fonctionnalités liées aux entreprises

Nom :	Référencer une nouvelle entreprise
Description	Le référencement de l'entreprise dans le système lui permet par la suite de se connecter et de saisir ultérieurement des offres d'emploi.
Utilisateur autorisé à accéder à la fonctionnalité	Pour des fins de simplification, tout utilisateur du système peut référencer une nouvelle entreprise.
Entrées	Les données à saisir pour caractériser une entreprise sont : <ul style="list-style-type: none"> <li>• le nom ;</li> <li>• le descriptif de l'entreprise ;</li> <li>• l'adresse postale. Pour simplifier, on considère qu'une entreprise n'occupe qu'un seul emplacement géographique.</li> </ul>
Sorties	Une fois l'entreprise référencée, le système présente les données précédemment saisies et un identifiant (de la forme 'ENT_ + numéro de l'entreprise') qui permettra ensuite la connexion sur le système. Comme il n'y a pas d'exigences de sécurité, la gestion de l'authentification avec un mot de passe n'est pas nécessaire. Une fois connectée avec son identifiant, l'entreprise a accès aux informations et aux fonctionnalités qui lui sont spécifiques.
Importance	Vitale

Nom :	Lister les entreprises référencées
Description	Affichage de la liste des entreprises.
Utilisateur autorisé à accéder à la fonctionnalité	Accès à tout utilisateur.
Entrées	-
Sorties	Pour chaque entreprise, la liste fait apparaître les informations suivantes : identifiant de l'entreprise, nom, adresse postale et nombre d'offres d'emploi référencées par entreprise.
Importance	Vitale

Nom :	Afficher les informations d'une entreprise
Description	Affichage des informations spécifiques à une entreprise.
Utilisateur autorisé à accéder à la fonctionnalité	Accès à tout utilisateur.
Entrées	-
Sorties	Affichage de l'ensemble des données liées à l'entreprise : <ul style="list-style-type: none"> <li>• l'identifiant (de la forme 'ENT_ + numéro de l'entreprise') ;</li> <li>• le nom ;</li> <li>• le descriptif de l'entreprise ;</li> <li>• l'adresse postale.</li> </ul>
Importance	Vitale

Nom :	Lister les offres d'emploi spécifiques à une entreprise
Description	Une fois connectée, l'entreprise a accès à la liste des offres d'emploi qu'elle a déposées. La présentation de cette liste permet l'accès aux fonctionnalités de mise à jour et de suppression des offres d'emploi.
Utilisateur autorisé à accéder à la fonctionnalité	L'entreprise.
Entrées	-
Sorties	Pour chaque offre d'emploi, la liste fait apparaître les informations suivantes : numéro de l'offre, titre (avec un lien pour l'affichage complet de l'offre), nom de l'entreprise, niveau de qualification requis et date de dépôt et, par offre, la liste des candidatures potentielles avec un lien vers l'affichage complet. En plus de ces informations, un lien permettra l'accès vers la mise à jour de l'offre et sa suppression.
Importance	Vitale



Nom :	Mettre à jour les informations d'une entreprise
Description	Une fois connectée, l'entreprise a la possibilité de mettre à jour les informations qui ont été saisies lors de son référencement.
Utilisateur autorisé à accéder à la fonctionnalité	Seule l'entreprise peut modifier ses propres informations.
Entrées	Présentation des données de l'entreprise (nom, descriptif de l'entreprise et adresse postale) qui peuvent être modifiées. L'identifiant de l'entreprise n'est pas modifiable.
Sorties	Une fois les informations modifiées, le système les présente pour vérification.
Importance	Mineure

Nom :	Supprimer une entreprise
Description	Le déréférencement d'une entreprise s'opère par sa suppression. La suppression de l'entreprise implique la suppression en cascade des éventuelles offres d'emploi associées ainsi que des messages envoyés et reçus liés à celles-ci.
Utilisateur autorisé à accéder à la fonctionnalité	Seule l'entreprise peut décider de sa suppression.
Entrées	-
Sorties	Message d'acquiescement indiquant que l'entreprise a bien été effacée.
Importance	Mineure

Nom :	Envoyer un message à un candidat pour une offre d'emploi donnée
Description	Dans le contexte d'une offre d'emploi donnée, une entreprise peut envoyer un message à un candidat dont le profil est susceptible de correspondre à l'offre (voir fonctionnalités d'indexation). L'accès à la fonctionnalité d'envoi de message est fait à partir de la fonctionnalité d'affichage des informations de l'offre d'emploi si l'utilisateur connecté est l'entreprise propriétaire de l'offre.
Utilisateur autorisé à accéder à la fonctionnalité	L'entreprise.
Entrées	Un message se caractérise par : <ul style="list-style-type: none"> <li>• Un candidat destinataire (sélectionné automatiquement à partir de l'offre d'emploi).</li> <li>• Le corps du message.</li> <li>• Une date de dépôt du message renseignée automatiquement.</li> </ul>
Sorties	Une fois le message envoyé, le système affiche les différents éléments du message : numéro du message, offre d'emploi concernée, nom du candidat destinataire, date de dépôt et corps du message.
Importance	Mineure

Nom :	Lister les messages reçus des candidats pour une offre d'emploi de l'entreprise
Description	Historique des messages reçus (demandes d'entretien, etc.) classés par date.
Utilisateur autorisé à accéder à la fonctionnalité	L'entreprise
Entrées	-
Sorties	Pour chaque message, la liste fait apparaître : le numéro du message, le candidat expéditeur, l'offre d'emploi concernée, la date d'envoi et le corps du message.
Importance	Mineure

Nom :	Lister les messages envoyés à des candidats pour une offre d'emploi de l'entreprise
Description	Historique des messages envoyés (proposition de rendez-vous, etc.) classés par date.
Utilisateur autorisé à accéder à la fonctionnalité	L'entreprise
Entrées	-
Sorties	Pour chaque message, la liste fait apparaître : le numéro du message, le candidat destinataire, l'offre d'emploi concernée, la date d'envoi et le corps du message.
Importance	Mineure

### Fonctionnalités liées aux offres d'emploi

<b>Nom :</b>	<b>Référencer une nouvelle offre d'emploi</b>
<b>Description</b>	Chaque entreprise peut référencer une nouvelle offre d'emploi qu'elle publie sur le système.
<b>Utilisateur autorisé à accéder à la fonctionnalité</b>	L'entreprise
<b>Entrées</b>	Les données à saisir pour caractériser une offre d'emploi sont : <ul style="list-style-type: none"> <li>• le titre de l'offre ;</li> <li>• le descriptif de la mission ;</li> <li>• le profil recherché par l'entreprise ;</li> <li>• Un ou plusieurs secteurs d'activité ainsi qu'un niveau de qualification exigé : voir fonctionnalités d'indexation.</li> </ul>
<b>Sorties</b>	Une fois l'offre référencée, le système affiche les différents éléments : numéro de l'offre, titre, descriptif, profil recherché, niveau de qualification, liste des secteurs d'activité et date de dépôt.
<b>Importance</b>	Vitale

<b>Nom :</b>	<b>Lister des offres d'emploi</b>
<b>Description</b>	Affichage de la liste de toutes les offres d'emploi disponibles dans le système. Attention : une entreprise accède à la liste des offres d'emploi dont elle est propriétaire. Les deux fonctionnalités sont similaires mais la seconde offre plus de fonctionnalité : voir lister les offres d'emploi d'une entreprise.
<b>Utilisateur autorisé à accéder à la fonctionnalité</b>	Accès à tout utilisateur.
<b>Entrées</b>	-
<b>Sorties</b>	Pour chaque offre d'emploi, la liste fait apparaître les informations suivantes : numéro de l'offre, titre, nom de l'entreprise qui a déposé l'offre, niveau de qualification requis et date de dépôt.
<b>Importance</b>	Vitale

<b>Nom :</b>	<b>Afficher les informations d'une offre d'emploi</b>
<b>Description</b>	Affichage des informations spécifiques à une offre d'emploi
<b>Utilisateur autorisé à accéder à la fonctionnalité</b>	L'accès est possible à tout utilisateur. Des fonctionnalités supplémentaires sont affichées si un utilisateur (entreprise ou candidat) est connecté.
<b>Entrées</b>	-
<b>Sorties</b>	Affichage de l'ensemble des données liées à l'offre d'emploi : <ul style="list-style-type: none"> <li>• le numéro de l'offre ;</li> <li>• le titre ;</li> <li>• la description de l'entreprise propriétaire de l'offre ;</li> <li>• le descriptif de la mission ;</li> <li>• le profil recherché ;</li> <li>• le lieu de la mission (pour simplifier c'est l'adresse de l'entreprise) ;</li> <li>• le niveau de qualification exigé ;</li> <li>• la liste des secteurs d'activité ;</li> <li>• la date de dépôt.</li> </ul> Optionnel (les fonctionnalités d'envoi de messages ont une importance mineure) : <ul style="list-style-type: none"> <li>• Si l'utilisateur connecté est un candidat et si l'offre d'emploi correspond au profil de celui-ci, une fonctionnalité de demande d'entretien (envoi de message) est disponible.</li> <li>• Si l'utilisateur connecté est une entreprise et qu'un ou plusieurs candidats ont un profil compatible, la liste de ceux-ci est présentée afin de faire une proposition de rendez-vous (envoi de message).</li> </ul>
<b>Importance</b>	Vitale

<b>Nom :</b>	<b>Mettre à jour les informations d'une offre d'emploi</b>
<b>Description</b>	Une fois connectée, l'entreprise a la possibilité de mettre à jour les informations de ses offres d'emploi.
<b>Utilisateur autorisé à accéder à la fonctionnalité</b>	Seule l'entreprise propriétaire peut modifier les informations de l'offre d'emploi.
<b>Entrées</b>	Présentation des données de l'offre (numéro, titre, descriptif, profil recherché, niveau de qualification et secteurs d'activité) qui peuvent être modifiées. L'identifiant de l'entreprise n'est pas modifiable.
<b>Sorties</b>	Une fois les informations modifiées, le système les présente pour vérification.
<b>Importance</b>	Mineure

<b>Nom :</b>	<b>Supprimer une offre d'emploi</b>
<b>Description</b>	Une offre d'emploi résolue s'opère dans le système par sa suppression, l'offre n'est ainsi plus disponible. La suppression d'une offre implique la suppression en cascade des éventuels messages envoyés et reçus dépendants de celle-ci.
<b>Utilisateur autorisé à accéder à la fonctionnalité</b>	Seule l'entreprise peut décider de la suppression d'une offre d'emploi dont elle est propriétaire.
<b>Entrées</b>	-
<b>Sorties</b>	Message d'acquittement indiquant que l'entreprise a bien été effacée.
<b>Importance</b>	Mineure

<b>Nom :</b>	<b>Référencer une nouvelle candidature</b>
<b>Description</b>	Le référencement d'une candidature dans le système lui permet par la suite de se connecter.
<b>Utilisateur autorisé à accéder à la fonctionnalité</b>	Pour des fins de simplification, tout utilisateur du système peut référencer une nouvelle candidature.
<b>Entrées</b>	Un candidat se caractérise par : <ul style="list-style-type: none"> <li>• un numéro de candidature ;</li> <li>• un nom ;</li> <li>• un prénom ;</li> <li>• une date de naissance ;</li> <li>• une adresse postale ;</li> <li>• une adresse électronique ;</li> <li>• une date de dépôt ;</li> <li>• le CV (potentiellement un document au format PDF mais, pour simplifier, un contenu en texte simple suffira) ;</li> <li>• un ou plusieurs secteurs d'activité ainsi qu'un niveau de qualification exigé : voir fonctionnalités d'indexation.</li> </ul>
<b>Sorties</b>	Une fois inscrite, l'entreprise reçoit un identifiant (de la forme 'CAND_ + id') qui lui permet de se connecter sur le système. Les informations qui y seront affichées lui seront spécifiques.
<b>Importance</b>	Vitale

<b>Nom :</b>	<b>Lister les candidatures référencées</b>
<b>Description</b>	Affichage de la liste des candidatures.
<b>Utilisateur autorisé à accéder à la fonctionnalité</b>	Accès à tout utilisateur.
<b>Entrées</b>	-
<b>Sorties</b>	Pour chaque candidature, la liste fait apparaître les informations suivantes : numéro de la candidature, nom, prénom, adresse postale, adresse email, niveau de qualification, date de dépôt.
<b>Importance</b>	Vitale

<b>Nom :</b>	<b>Afficher les informations d'une candidature</b>
<b>Description</b>	Affichage des informations spécifiques à une offre d'emploi.
<b>Utilisateur autorisé à accéder à la fonctionnalité</b>	L'accès est possible à tout utilisateur.
<b>Entrées</b>	-
<b>Sorties</b>	Affichage de l'ensemble des données liées à la candidature : <ul style="list-style-type: none"> <li>• l'identifiant (de la forme 'CAND_ + numéro de la candidature) ;</li> <li>• le nom ;</li> <li>• le prénom ;</li> <li>• la date de naissance ;</li> <li>• l'adresse postale ;</li> <li>• l'adresse email ;</li> <li>• le CV ;</li> <li>• le niveau de qualification possédé ;</li> <li>• la liste des secteurs d'activité ;</li> <li>• la date de dépôt.</li> </ul>
<b>Importance</b>	Vitale

Nom :	Lister les offres d'emploi qui correspondent à une candidature donnée
Description	Une fois connecté, le candidat accède à une liste d'offres d'emploi susceptibles de correspondre à son profil. Cette fonctionnalité est similaire à la liste des offres d'emploi accessible à tout utilisateur (qui en affiche la totalité) mais elle opère une sélection.
Utilisateur autorisé à accéder à la fonctionnalité	Le candidat.
Entrées	-
Sorties	Pour chaque offre d'emploi, la liste fait apparaître les informations suivantes : numéro de l'offre, titre, nom de l'entreprise qui a déposé l'offre, niveau de qualification requis et date de dépôt.
Importance	Vitale

Nom :	Mettre à jour les informations d'une candidature
Description	Une fois connecté, le candidat a la possibilité de mettre à jour les informations qui ont été saisies lors de son référencement.
Utilisateur autorisé à accéder à la fonctionnalité	Le candidat.
Entrées	Présentation des données de la candidature (identifiant, nom, prénom, date de naissance, adresse postale, adresse email, CV, niveau de qualification et secteurs d'activité) qui peuvent être modifiées. L'identifiant de la candidature n'est pas modifiable.
Sorties	Une fois les informations modifiées, le système les présente pour vérification.
Importance	Mineure

Nom :	Supprimer une candidature
Description	Lorsqu'un candidat trouve un emploi, sa candidature doit être supprimée dans le système. La suppression d'une candidature implique la suppression en cascade des éventuels messages envoyés et reçus dépendants de celle-ci.
Utilisateur autorisé à accéder à la fonctionnalité	Le candidat.
Entrées	-
Sorties	Message d'acquittement indiquant que la candidature a bien été effacée.
Importance	Mineure

Nom :	Envoyer un message à une entreprise pour une offre d'emploi donnée
Description	Dans le contexte d'une offre d'emploi donnée, un candidat peut envoyer un message à une entreprise s'il estime que son profil est susceptible de correspondre à l'offre (voir fonctionnalités d'indexation). L'accès à la fonctionnalité d'envoi de message est faite à partir de la fonctionnalité d'affichage des informations de l'offre d'emploi si l'utilisateur est connecté.
Utilisateur autorisé à accéder à la fonctionnalité	Le candidat
Entrées	Un message se caractérise par : <ul style="list-style-type: none"> <li>• une entreprise (sélectionnée automatiquement) ;</li> <li>• une offre d'emploi sur laquelle le candidat veut postuler (sélectionnée automatiquement) ;</li> <li>• le corps du message ;</li> <li>• une date de dépôt du message renseignée automatiquement.</li> </ul>
Sorties	Une fois le message envoyé, le système affiche les différents éléments du message : numéro du message, la candidature concernée, l'offre d'emploi et l'entreprise destinataire, la date d'envoi et le corps du message.
Importance	Mineure

Nom :	Lister les messages reçus des entreprises pour une offre d'emploi
Description	Historique des messages reçus (proposition de rendez-vous, etc.) classés par date.
Utilisateur autorisé à accéder à la fonctionnalité	Le candidat.
Entrées	-
Sorties	Pour chaque message, la liste fait apparaître : le numéro du message, le candidat expéditeur, l'offre d'emploi et l'entreprise expéditrice, la date d'envoi et le corps du message.
Importance	Mineure

Nom :	Lister les messages envoyés à des entreprises pour une offre d'emploi
Description	Historique des messages envoyés (demande d'entretien, etc.) classés par date.
Utilisateur autorisé à accéder à la fonctionnalité	Le candidat.
Entrées	-
Sorties	Pour chaque message, la liste fait apparaître : le numéro du message, l'offre d'emploi concernée avec le nom de l'entreprise, la date d'envoi et le corps du message.
Importance	Mineure

### Fonctionnalités d'indexation du contenu (offres d'emploi et candidatures)

Nom :	Indexer une offre d'emploi ou une candidature
Description	<p>Lors du référencement d'une nouvelle offre d'emploi ou d'une nouvelle candidature, celles-ci sont indexées suivant les mêmes contenus afin de pouvoir les comparer pour les faire éventuellement correspondre. Les données de cette indexation portent sur deux types de valeurs :</p> <ul style="list-style-type: none"> <li>Le secteur d'activité. Valeur à choisir parmi : <i>Achats/Logistique, Assistanat/Secrétariat, Agriculture, Agroalimentaire, Assurance, Audit/Conseil/Expertises, BTP/Immobilier, Commercial, Communication/Art/Média/Mode, Comptabilité, Direction Générale/Executive, Distribution/Commerce, Electronique/Microélectronique, Environnement, Finance/Banque, Formation/Enseignement, Hôtellerie/Restauration/Tourisme, Industrie/Ingénierie/Production, Informatique, Juridique/Fiscal/Droit, Marketing, Public/Parapublic, Ressources Humaines, Santé/Social/Biologie/Humanitaire, Télécom/Réseaux.</i> Note : pour des questions pratiques, cette liste peut être réduite.</li> <li>Le niveau de qualification. Valeur à choisir parmi : <i>CAP/BEP, Bac, Bac+3, Bac+5, Doctorat.</i></li> </ul> <p>Chaque offre et chaque candidature est indexée par un ou plusieurs secteurs d'activité et un unique niveau de qualification.</p> <p>La potentielle correspondance entre une offre d'emploi et une candidature est déduite s'ils partagent au moins un secteur d'activité commun (intersection non nulle) avec un niveau de qualification identique.</p>
Utilisateur autorisé à accéder à la fonctionnalité	-
Entrées	-
Sorties	-
Importance	Vitale

### Fonctionnalités d'identification

Nom :	Se connecter au système d'information
Description	Afin d'accéder aux fonctionnalités spécifiques les utilisateurs peuvent se connecter au système.
Utilisateur autorisé à accéder à la fonctionnalité	Profil entreprise ou profil candidature.
Entrées	<p>L'identifiant de l'utilisateur qui a été déduit lors de son référencement :</p> <ul style="list-style-type: none"> <li>Dans le cas d'une entreprise, l'identifiant est de la forme 'ENT_ + numéro de l'entreprise'.</li> <li>Dans le cas d'une candidature, l'identifiant est de la forme 'CAND_ + numéro de la candidature'.</li> </ul>
Sorties	<p>L'interface de l'application est adaptée au profil :</p> <ul style="list-style-type: none"> <li>Le profil (entreprise ou candidature) et le nom de l'utilisateur apparaissent (nom de l'entreprise ou nom/prénom du candidat).</li> <li>Le menu de l'application s'adapte au profil pour donner accès aux fonctionnalités.</li> <li>Les fonctionnalités accessibles aux utilisateurs non connectés restent visibles.</li> </ul>
Importance	Vitale

<b>Nom :</b>	<b>Se déconnecter du système d'information</b>
<b>Description</b>	Lorsque l'utilisateur connecté met fin à sa session, l'application se retrouve dans l'état initial.
<b>Utilisateur autorisé à accéder à la fonctionnalité</b>	Profil entreprise ou profil candidature.
<b>Entrées</b>	-
<b>Sorties</b>	L'interface de l'application se retrouve dans l'état initial : <ul style="list-style-type: none"> <li>• Affichage du mode non connecté.</li> <li>• Seules les fonctionnalités accessibles aux utilisateurs non connectés sont visibles.</li> </ul>
<b>Importance</b>	Vitale

### Fonctionnalités de syndication de contenu

<b>Nom :</b>	<b>Exporter automatiquement la liste actualisée des offres d'emploi au sein d'un flux RSS</b>
<b>Description</b>	Pour une utilisation des informations (en lecture) en dehors du système, celui-ci publie la liste des offres d'emploi au sein d'un flux RSS (format XML) rendu disponible par l'application Web à destination d'un logiciel externe (agrégateur). Les informations disponibles correspondent à la liste de toutes les offres d'emploi.
<b>Utilisateur autorisé à accéder à la fonctionnalité</b>	Tout utilisateur qui s'abonne au flux.
<b>Entrées</b>	URL du flux.
<b>Sorties</b>	Le flux au format RSS qui est affiché au sein de l'agrégateur.
<b>Importance</b>	Importante

<b>Nom :</b>	<b>Exporter automatiquement la liste actualisée des candidatures au sein d'un flux RSS</b>
<b>Description</b>	Pour une utilisation des informations (en lecture) en dehors du système, celui-ci publie la liste des candidatures au sein d'un flux RSS (format XML) rendu disponible par l'application Web à destination d'un logiciel externe (agrégateur). Les informations disponibles correspondent à la liste de toutes les candidatures.
<b>Utilisateur autorisé à accéder à la fonctionnalité</b>	Tout utilisateur qui s'abonne au flux.
<b>Entrées</b>	URL du flux.
<b>Sorties</b>	Le flux au format RSS qui est affiché au sein de l'agrégateur.
<b>Importance</b>	Mineure

## 2.3.2 Exigences non fonctionnelles

### Sécurité

Les exigences au niveau de la sécurité sont faibles : les utilisateurs des deux profils existants se connectent sans mot de passe, l'objectif de cette application est d'abord pédagogique.

Cependant, dans un objectif réaliste, l'accès à certaines fonctionnalités est filtré :

- Fonctionnalités accessibles uniquement en mode non connecté :
  - o Se connecter au système d'information
- Fonctionnalités accessibles en mode non connecté et à tous les profils (principalement les fonctionnalités d'administration du système) :
  - o Référencer une nouvelle entreprise
  - o Lister les entreprises référencées
  - o Afficher les informations d'une entreprise
  - o Lister des offres d'emploi
  - o Afficher les informations d'une offre d'emploi
  - o Lister les candidatures référencées
  - o Afficher les informations d'une candidature
- Fonctionnalités accessibles uniquement aux utilisateurs connectés avec un profil *entreprise* :
  - o Mettre à jour les informations d'une entreprise
  - o Supprimer une entreprise
  - o Référencer une nouvelle offre d'emploi (+ indexation de l'offre)
  - o Lister les offres d'emploi spécifiques à une entreprise
  - o Mettre à jour les informations d'une offre d'emploi

- o Supprimer une offre d'emploi
- o Envoyer un message à un candidat pour une offre d'emploi donnée
- o Lister les messages reçus des candidats pour une offre d'emploi de l'entreprise
- o Lister les messages envoyés à des candidats pour une offre d'emploi de l'entreprise
- o Se déconnecter du système d'information
- Fonctionnalités accessibles uniquement aux utilisateurs connectés avec un profil *candidature* :
  - o Référencer une nouvelle candidature (+ indexation de la candidature)
  - o Lister les offres d'emploi qui correspondent à une candidature donnée
  - o Mettre à jour les informations d'une candidature
  - o Supprimer une candidature
  - o Envoyer un message à une entreprise pour une offre d'emploi donnée
  - o Lister les messages reçus des entreprises pour une offre d'emploi
  - o Lister les messages envoyés à des entreprises pour une offre d'emploi
  - o Se déconnecter du système d'information
- Fonctionnalités non liées à un profil (fonctionnalités automatiques) :
  - o Exporter automatiquement la liste actualisée des offres d'emploi au sein d'un flux RSS
  - o Exporter automatiquement la liste actualisée des offres candidature au sein d'un flux RSS

### Portabilité

L'application à développer utilisera la technologie Java EE. À ce titre, l'application pourra être déployée sur tout serveur d'applications compatible Java EE. Le serveur choisi pour le développement et les tests est le serveur d'applications open source d'Oracle : *GlassFish version 4*. Il est disponible pour les environnements Unix (*GNU-Linux*, *MacOS*, etc.) et *Windows*.

L'interface de l'application étant une interface Web, elle devra pouvoir être exécutée sur les principaux navigateurs : *Mozilla Firefox*, *Google Chrome*, *Internet Explorer*.

## 2.3.3 Exigences concernant le développement du produit

### Objectifs de délais

La date de livraison pour le produit final est prévue une semaine après la dernière séance du fil rouge. Avant cette livraison finale, différents jalons sont prévus avec dépôt de documents sur Moodle :

- Avant le début de la partie 2 (avant la séance 2) :
  - o Le schéma logique (destiné à l'implantation au sein d'un SGBD relationnel) dérivé à partir du diagramme de classes décrivant les données de l'application.
  - o Le script SQL de création des tables.
- Avant le début de la partie 3 (avant la séance 4) :
  - o L'implémentation Java des :
    - beans entités (classes Java qui stockent les données de l'application) ;
    - classes DAO (EJB Java qui assurent les fonctionnalités CRUD).
- Une semaine après la dernière séance (recette de l'application durant la séance 7) :
  - o L'implémentation Java des composants EJB codant les services métier.
  - o Documentation technique de l'architecture 3-tier mise en œuvre pour l'application.

### Objectifs de coûts

Le coût horaire est estimé à 60 heures pour le binôme soit 30 heures par personne. Cette durée comprend les 19h30 de séances encadrées.

## Exigences de réalisation

L'application utilisera obligatoirement les éléments suivants :

- SGBD PostgreSQL 9.3
- Langage Java 1.7
- Serveur d'applications Oracle GlassFish 4
- L'environnement de développement sera Eclipse IDE pour développement Java EE version Kepler

## 2.4 Synthèse des exigences

### 2.4.1 Hiérarchisation des capacités fonctionnelles

Les différents niveaux d'importance des exigences fonctionnelles organisent les développements autour de quatre incréments :

1. Gestion basique des offres d'emploi et des candidatures : création et consultation. Fonctionnalités vitales donc obligatoires.
2. Gestion complète des offres d'emploi et des candidatures : en plus de la création et de la consultation, suppression et mise à jour. Fonctionnalités mineures donc optionnelles.
3. Gestion des demandes d'entretien de la part des entreprises et de la part des candidats. Fonctionnalités mineures donc optionnelles.
4. Gestion du flux RSS des offres d'emploi. Fonctionnalité importante donc obligatoire.

#### Fonctionnalités liées au entreprises

Fonctionnalité	Importance
Référencer une nouvelle entreprise	Vitale
Lister les entreprises référencées	Vitale
Afficher les informations d'une entreprise	Vitale
Lister les offres d'emploi spécifiques à une entreprise	Vitale
<i>Mettre à jour les informations d'une entreprise</i>	<i>Mineure</i>
<i>Supprimer une entreprise</i>	<i>Mineure</i>
<i>Envoyer un message à un candidat pour une offre d'emploi donnée</i>	<i>Mineure</i>
<i>Lister les messages reçus des candidats pour une offre d'emploi de l'entreprise</i>	<i>Mineure</i>
<i>Lister les messages envoyés à des candidats pour une offre d'emploi de l'entreprise</i>	<i>Mineure</i>

#### Fonctionnalités liées aux offres d'emploi

Fonctionnalité	Importance
Référencer une nouvelle offre d'emploi	Vitale
Lister des offres d'emploi	Vitale
Afficher les informations d'une offre d'emploi	Vitale
<i>Mettre à jour les informations d'une offre d'emploi</i>	<i>Mineure</i>
<i>Supprimer une offre d'emploi</i>	<i>Mineure</i>

#### Fonctionnalités liées aux candidatures

Fonctionnalité	Importance
Référencer une nouvelle candidature	Vitale
Lister les candidatures référencées	Vitale
Afficher les informations d'une candidature	Vitale
Lister les offres d'emploi qui correspondent à une candidature donnée	Vitale
<i>Mettre à jour les informations d'une candidature</i>	<i>Mineure</i>
<i>Supprimer une candidature</i>	<i>Mineure</i>
<i>Envoyer un message à une entreprise pour une offre d'emploi donnée</i>	<i>Mineure</i>
<i>Lister les messages reçus des entreprises pour une offre d'emploi</i>	<i>Mineure</i>
<i>Lister les messages envoyés à des entreprises pour une offre d'emploi</i>	<i>Mineure</i>



**Fonctionnalités d'indexation du contenu (offres d'emploi et candidatures)**

Fonctionnalité	Importance
Indexer une offre d'emploi ou une candidature	Vitale

**Fonctionnalités d'identification**

Fonctionnalité	Importance
Se connecter au système d'information	Vitale
Se déconnecter du système d'information	Vitale

**Fonctionnalités de syndication de contenu**

Fonctionnalité	Importance
Exporter automatiquement la liste actualisée des offres d'emploi au sein d'un flux RSS	Importante
<i>Exporter automatiquement la liste actualisée des offres candidature au sein d'un flux RSS</i>	<i>Mineure</i>





## 3 — Fiche tâches/temps

Associée à chaque livrable, la fiche tâches/temps doit être complétée au fur et à mesure du temps passé sur le projet. La fiche doit tenir compte du temps passé durant les séances encadrées ainsi que du temps passé hors séances.

Le recueil de ces données répond à plusieurs objectifs :

- Pour les enseignants, il permettra d'évaluer la durée moyenne du temps de travail nécessaire à la réalisation de ce projet afin de l'adapter ultérieurement si nécessaire.
- Pour les élèves, il permettra d'évaluer votre propre productivité et d'avoir une idée de la répartition du temps passé dans chaque étape du projet.
- Point important : ces données ne seront pas utilisées pour l'évaluation du fil rouge.

### 3.1 Contenu

La fiche prend la forme d'un tableau : différentes informations sont données pour chacune de vos interventions.

- Le numéro de la séance de travail.
- La date de la séance.
- L'étape du travail : base de données, couche données, couche services, etc.
- Les objectifs de la séance : description du travail à réaliser.
- Le temps passé (en heures) : il s'agit du temps de travail cumulé du binôme (par exemple 6 h pour une séance de TP : 2 X 3 h). Il n'est pas fait de distinction entre les membres du binôme, ce n'est pas un outil de contrôle.
- La dernière colonne précise si le temps passé s'est fait en séance encadrée ou sur le temps de travail personnel.
- La dernière ligne récapitule le temps total passé sur le livrable.

**3.1.1 Exemple de fiche tâches/temps :****Nom\_1 / Nom\_2**

Numéro	Date	Étape	Objectif(s)	Temps passé	Séance ou temps perso
1	12/09/2015	Conception des données	Réalisation du schéma logique	1 h	Séance
2	12/09/2015	Conception des données	Réalisation du script de création de la base de données	1 h	Séance
3	12/09/2015	Conception des données	Réalisation du script de création de la base de données (suite)	2 h	Temps perso
...	...	...	...	...	...
...	...	...	...	...	...
...	...	...	...	...	...
...	...	...	...	...	...
<b>Total</b>				<b>3 268 h</b>	

## 4 — Partie 1 : *conception des données de l'application*

### 4.1 Déroulement

Cette première séance du fil rouge se divise en deux étapes principales :

1. La première étape vise à comprendre le sujet, à mettre en place l'environnement de travail et avoir une première vision de l'architecture de l'application. Celle-ci sera vue plus en détail au cours de l'enseignement *architectures applicatives*.
2. La seconde étape est centrée sur les données manipulées par l'application *cabinet de recrutement*, données déduites du cahier des charges. Le schéma conceptuel (diagramme de classes) des données de l'application est fourni, il faudra le dériver en un schéma logique et générer ensuite le script de création de la base de données.

### 4.2 Introduction

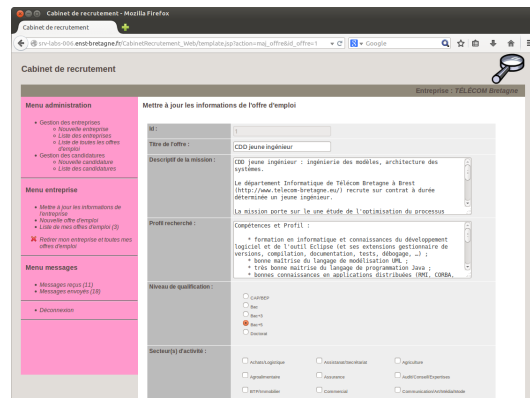
Cet point donne quelques informations sur le projet à développer dans le fil rouge :

- compréhension du sujet ;
- mise en place de l'environnement de travail ;
- présentation sommaire de l'architecture.

#### 4.2.1 L'application

Rappel : comme énoncé dans la présentation du fil rouge, l'objectif de ce travail est essentiellement technologique. Le cahier des charges fourni présente l'intégralité des fonctionnalités à développer. En cas d'ambiguïté dans la description d'une fonctionnalité, une implémentation de l'application Web est disponible. Il est possible d'y réaliser des tests : référencement d'une nouvelle entreprise, création d'une offre d'emploi, dépôt d'un CV, etc. La page 16 du cahier des charges liste les fonctionnalités par catégories et par ordre d'importance : vitale et importante (*fonctionnalités obligatoires*) et mineure (*fonctionnalités optionnelles*).

Une ébauche de l'application est déployée dans votre environnement de travail, elle implémente deux des fonctionnalités à développer : affichage de la liste des entreprises référencées et affichage des informations d'une entreprise donnée qui pourront servir d'exemple pour vos développements futurs.



[http://srv-labs-006.enst-bretagne.fr/CabinetRecrutement\\_Web/](http://srv-labs-006.enst-bretagne.fr/CabinetRecrutement_Web/)



[http://localhost:8080/CabinetRecrutement\\_WEB/liste\\_entreprises.jsp](http://localhost:8080/CabinetRecrutement_WEB/liste_entreprises.jsp)

#### 4.2.2 L'environnement de travail – les outils

L'ensemble des outils nécessaires à la réalisation du fil rouge se trouve au sein d'une machine virtuelle *VMware* dans laquelle un système *Linux Ubuntu 14* est installé : *Ubuntu14\_BD\_PROG*. Cette machine virtuelle est la même que celle utilisée dans les TP.

Deux documents additionnels sont disponibles :

- L'annexe du document de TD/TP : *Description et utilisation de la machine virtuelle VMWare*. Vous y trouverez de plus amples informations sur la machine virtuelle et les outils disponibles. Important : vous trouverez aussi dans ce document la procédure vous permettant de récupérer la machine virtuelle des salles de TP afin de l'exécuter sur votre ordinateur personnel (qu'il soit sous MacOS, Linux ou Windows).
- Si vous avez besoin de dupliquer l'environnement de développement (*JDK*, *PostgreSQL*, *Glassfish*, etc.), vous trouverez sur Moodle un document décrivant leur installation sur un système Linux Ubuntu. Document : *creationVM.pdf*.

#### 4.2.3 Mise en route de l'ébauche de l'application

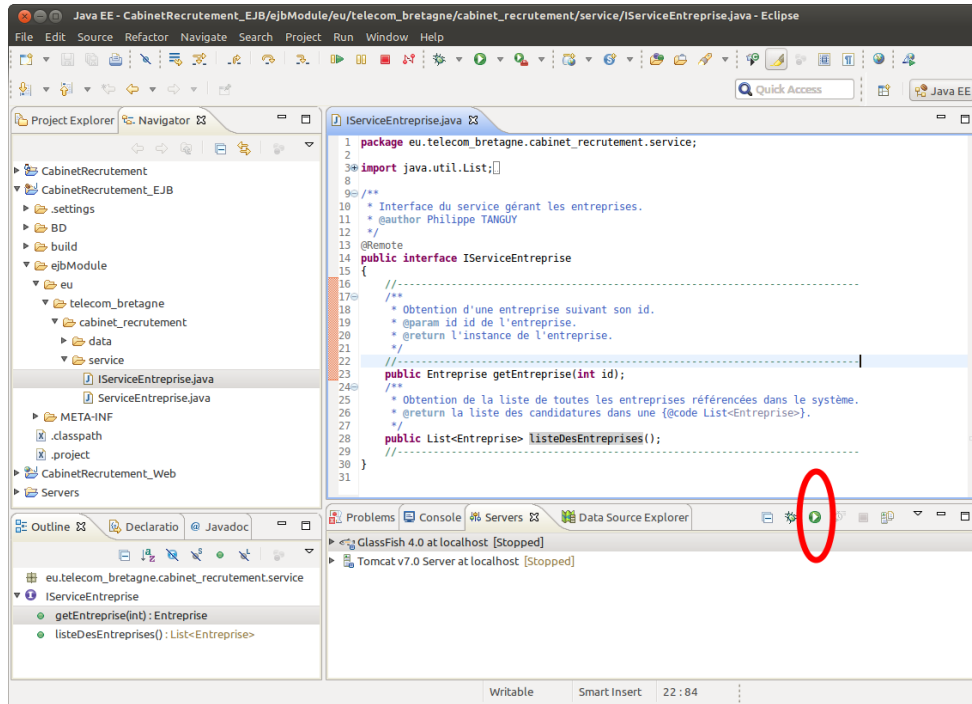
Lancez Eclipse. Le code de l'application est réparti sur trois projets qu'il faut ouvrir. Pour importer un projet dans Eclipse : menu *File > Import...*, choisir *Existing Projects into Workspace*, bouton *Next >*. Choisir alors le répertoire contenant les trois projets : */home/user/FIP INF 211 - Fil rouge*. Les trois projets apparaissent dans la fenêtre, ils se nomment :

- *CabinetRecrutement*
- *CabinetRecrutement\_EJB*
- *CabinetRecrutement\_WEB*

L'interface d'Eclipse est divisée en plusieurs parties :

- Sur la gauche apparaît la liste des projets dans l'onglet *Navigator* ou l'onglet *Project Explorer*.
- À droite, partie supérieure : la partie éditeur de code. Chaque fichier en cours d'édition s'ouvre au sein d'un onglet différent.
- À droite, partie inférieure : différents onglets parmi lesquels se situe l'onglet *Servers*, voir copie d'écran ci-dessous. C'est à partir de celui-ci que sera piloté le serveur *GlassFish* où est hébergée l'application. Sélectionnez le référencement du serveur (*GlassFish 4.0 at localhost*) :

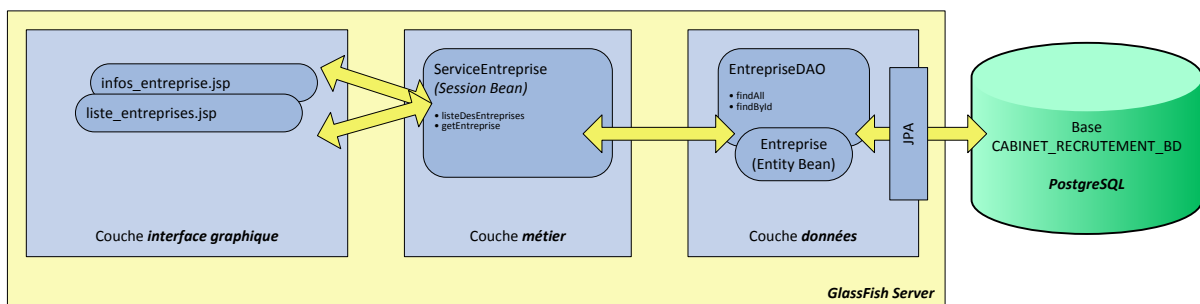
1. Ajoutez le projet au serveur : clic droit > *Add and Remove...*, l'application *CabinetRecrutement* apparaît dans la partie gauche, sélectionnez-la et cliquez sur le bouton *Add* >. L'application est maintenant référencée dans le serveur.
2. Le lancement du serveur se fait à l'aide du bouton vert (cerclé en rouge sur la copie d'écran). Au bout de quelques secondes, le serveur est prêt : son statut passe de *[Stopped]* à *[Started]*. Il est alors possible de lancer l'application dans un navigateur Web à l'URL :  
[http://localhost:8080/CabinetRecrutement\\_WEB/](http://localhost:8080/CabinetRecrutement_WEB/)  
 Note : il peut arriver, lors d'une mauvaise manipulation, que cet onglet soit fermé. Pour le réafficher : menu *Window* > *Show View* > *Servers*.



IDE Eclipse

#### 4.2.4 Architecture de l'application

Cette ébauche de l'application met déjà en œuvre l'architecture complète que vous verrez plus en détail au cours de l'enseignement architectures applicatives. Cette architecture est présentée ici succinctement pour vous permettre de vous retrouver dans le code source fourni qui est réparti dans les différents sous-projets.

Architecture simplifiée de l'application *cabinet de recrutement*

L'application est codée au sein de deux parties :

- La base de données hébergée dans un SGBDR PostgreSQL : la base *CABINET\_RECRUTEMENT\_BD*. Cette base est, pour le moment, simplissime. Elle ne comporte qu'une seule table qui stocke les informations des entreprises, la table *entreprise*. Le script de création de la base est disponible au sein du projet *CabinetRecrutement\_EJB*, répertoire *BD*.

L'outil pgAdmin ou Aqua Data Studio 4.7 (ADS 4.7) permet la connexion à la base, il est alors possible de l'interroger :

- o Utilisateur : *cabinet\_recrutement\_role*
- o Mot de passe : *abc*

**La séance 1 du fil rouge est centrée sur la base de données.**

- L'application qui utilise la base de données : elle est hébergée au sein du serveur GlassFish et est écrite suivant la technologie Java EE.

Cette application respecte une architecture répartie sur trois couches :

1. La couche **données** :

Quand la couche métier a besoin de manipuler des données (lecture, écriture, effacement ou modification) elle passe par cette couche qui sert d'intermédiaire entre celle-ci et la base de données. Elle comporte plusieurs éléments :

- (a) Les classes qui permettent la manipulation de nos données au sein de l'application Java. Pour le moment, seule la classe qui représente les données des entreprises est présente : la classe *Entreprise*. Chaque fois que l'application récupère un tuple (une ligne) de la table *entreprise*, les données sont représentées sous la forme d'une instance de cette classe. Dans l'architecture EJB, ces classes sont appelées des *Entity Beans*, *EJB Entities*, *classes entités*, etc.
- (b) Les classes qui assurent les fonctionnalités minimales autour des classes entités (création, récupération, mise à jour, effacement) sont nommées *DAO (Data Access Object)*. Les méthodes de ces classes sont appelées par la couche supérieure. Ce type de classe n'est représenté ici que par une seule classe : la classe *EntrepriseDAO*.
- (c) La correspondance entre les classes (et leurs attributs) et les tables (et leurs colonnes) est assurée par la technologie JPA (*Java Persistence API*). À voir dans la classe *Entreprise*, les annotations qui permettent de gérer cette correspondance. La technologie JPA fera l'objet ultérieurement d'un TP.

**Les séances 2 et 3 sont centrées sur le développement de la couche données.**

2. La couche **métier**

Au sein de cette couche se trouve toute la gestion (minimale...) de notre application : liste des entreprises et affichage des informations détaillées d'une entreprise particulière. Cette gestion métier de l'application est confiée à un (ou plusieurs) *Session Bean(s)*. Pour le moment, une seule classe : *ServiceEntreprise*. Cette classe (et les futures classes à venir) offre ses services (d'où son nom) à la couche supérieure (*interface graphique*) et se sert de la couche inférieure (données) sans jamais accéder directement à la base de données.

3. La couche **interface graphique** (appelée aussi couche **IHM**)

Cette couche est responsable de la génération des interfaces Web de l'application. Elle est réalisée à l'aide de la technologie *JSP (Java Server Pages)* que l'on peut comparer à la technologie PHP. Comme en PHP, les pages JSP mêlent au sein du même fichier à la fois du code HTML pour la représentation des pages au sein du navigateur Web et du code (ici du code Java : tests conditionnels, boucles, appels vers les Session Beans, etc.). Trois JSP existent dans l'état actuel du projet : *index.jsp*, *liste\_entreprises.jsp* et *infos\_entreprise.jsp*.

**Les séances 4 et 5 sont centrées sur le développement des couches métier et IHM.**

L'organisation en package du code Java de l'application est calquée sur cette architecture, on trouve les trois packages qui correspondent aux trois couches :

- `eu.telecom_bretagne.cabinet_recrutement.data` : classes de la couche *données*.  
Ce package se divise lui-même en 2 sous-packages :
  - o `dao` : classes DAO
  - o `model` : classe entité.
- `eu.telecom_bretagne.cabinet_recrutement.service` : classes de la couche *métier*.
- `eu.telecom_bretagne.cabinet_recrutement.front` : classes nécessaires à la couche *IHM*.  
Ce package contient un sous-package (package `utils`) contenant des classes utilitaires nécessaires aux pages JSP de la couche *IHM*.



Cette organisation du code n'est pas obligatoire (il vous est possible de choisir une autre organisation...), elle permet cependant de s'y retrouver plus facilement dans le développement de l'application. Ce code est réparti dans l'ensemble des projets Eclipse.

#### 4.2.5 Organisation des projets de développement dans Eclipse

Les développements de l'application sont organisés autour des quatre projets ouverts précédemment qui recoupent en partie (en partie seulement, ce serait trop simple...) l'architecture présentée ci-dessus. Les types de projets nécessaires :

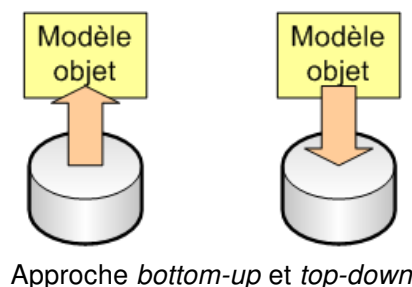
- Un projet de type *Enterprise Application Project* (**CabinetRecrutement**) : C'est un projet « conteneur » qui ne contient pas directement le code de l'application mais qui référence les trois autres sous-projets. Il produit, pour le déploiement de l'application un fichier *Enterprise Archive* ou *EAR* : ce type de projet s'appelle aussi *EAR Project*.
- Le codage de la logique métier de l'application se fait au sein d'un projet de type *EJB Project* (**CabinetRecrutement\_EJB**).
  - Le projet *CabinetRecrutement\_EJB* contient le code de la couche *données* et de la couche *métier* (répertoire *ejbModule*). Il contient aussi le script de création de la base de données au sein du répertoire *BD* (fichier *Création base cabinet recrutement version élèves.sql*) à la racine du projet.
- Le projet client qui utilisera la logique métier développée dans les projets précédents. Le choix opéré ici est de développer un client Web utilisant la technologie JSP qui nécessite un projet de type *Dynamic Web Project* (**CabinetRecrutement\_Web**). Les classes utilitaires sont présentes dans ce projet (répertoire *src* à la racine du projet).

### 4.3 Génération de la base de données

Les données persistantes de l'application connaissent des états différents :

- Elles sont stockées dans la base de données sous la forme de valeurs au sein de colonnes qui font partie d'une table.
- Quand elles sont manipulées par l'application, elles sont alors présentes sous la forme de valeurs que peuvent prendre les attributs de certaines classes (*les classes entités*).

Cette double structuration – le modèle objet du langage Java et le modèle relationnel du SGBD – coexiste mais doit rester complètement cohérent car ils sont interdépendants. Deux approches sont possibles :



- L'approche **top-down** (du haut vers le bas) : Le modèle objet existe (ou est imposé) et l'on souhaite générer le modèle relationnel. Dans ce cas, le schéma de la base est réalisé à partir d'un ensemble de classes Java.
- L'approche **bottom-up** (du bas vers le haut) : Le modèle de stockage existe (ou est imposé) et l'on souhaite générer le modèle objet. L'approche est ici inverse de la précédente. Le code des classes et la correspondance avec la base (appelée aussi *mapping objet/relationnel*) est généré après examen de la base de données. Ce travail fera l'objet de la deuxième séance du fil rouge.

Dans les deux cas, des outils permettent de générer automatiquement le double du modèle. Dans le fil rouge l'approche choisie est l'approche *bottom-up*, l'approche *top-down* sera vue durant le TP JPA.

### 4.3.1 Schéma conceptuel (*diagramme de classes*) des données de l'application

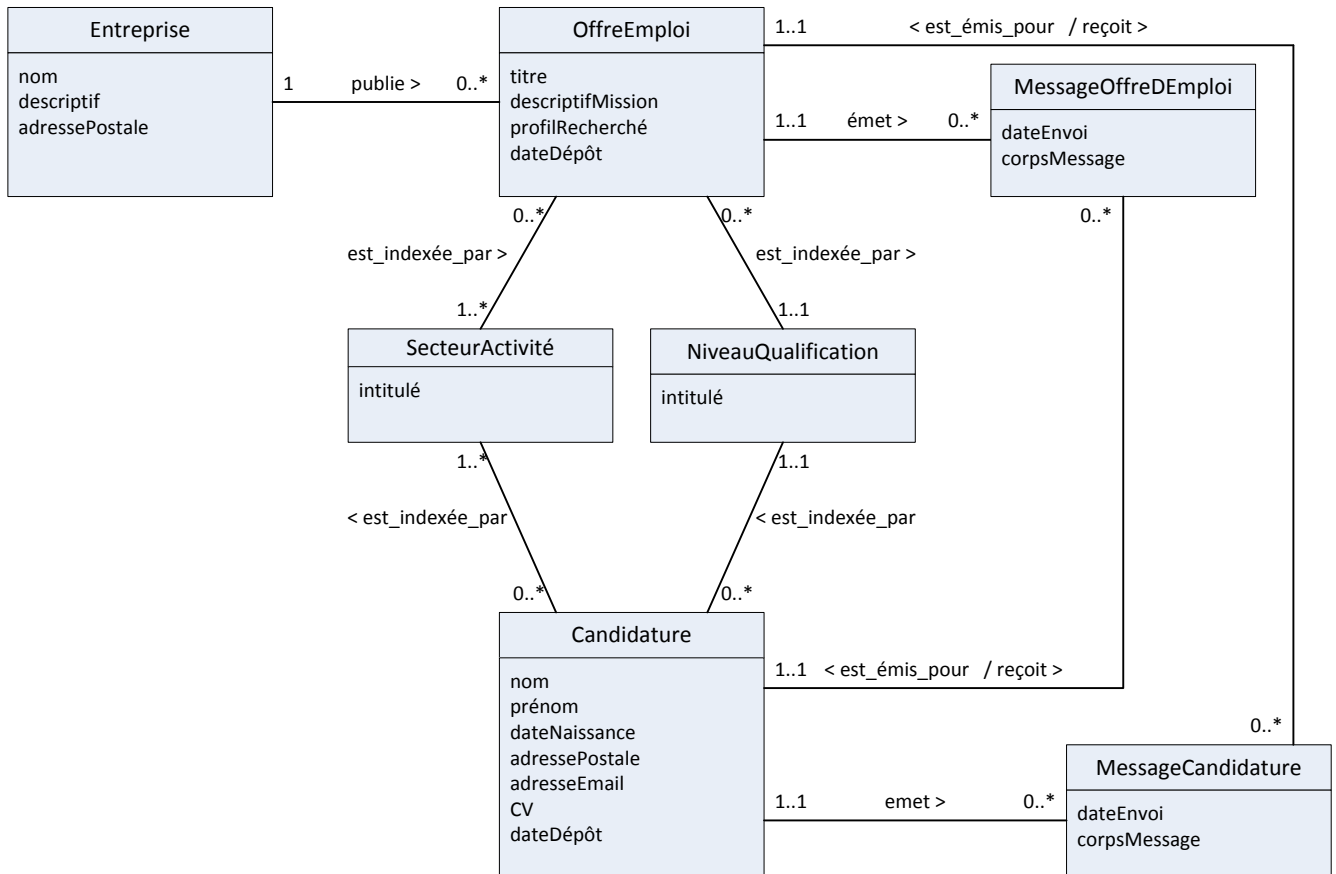


Schéma conceptuel des données de l'application

Les données de ce schéma conceptuel sont issues du cahier des charges. En cas de besoin, s'y référer pour une meilleure compréhension.

Note : les classes *MessageOffreDEmploi* et *MessageCandidature* sont liées à des fonctionnalités optionnelles, il est possible de ne pas les prendre en compte à ce stade du travail.

### 4.3.2 Travail à réaliser

Le processus à réaliser ici est similaire au travail réalisé au cours du TD *modélisation conceptuelle et dérivation en un schéma logique*. Il se divise en deux étapes :

1. Dérivation du schéma conceptuel en un schéma logique.
2. Génération du script de création de la base de données à partir du schéma logique.

Note : les valeurs des secteurs d'activité et des niveaux de qualification ne changent pas. Prévoir l'insertion de ces valeurs directement dans le script SQL.

#### Aide pour la création du schéma logique

- NOTER LES NOM ET PRÉNOM DES DEUX MEMBRES DU BINÔME SUR LE SCHÉMA.
- Il n'est pas nécessaire d'utiliser un quelconque outil informatique à ce stade, un schéma réalisé avec un crayon et une feuille de papier est amplement suffisant à condition évidemment qu'il soit propre et lisible...
- À partir des entités du diagramme de classes, identifiez les relations (tables).
- Identifiez les clés primaires de ces relations, les clés référentielles et les éventuelles contraintes (non nullité, etc.).
- Prendre en compte les associations suivant les cardinalités définies au niveau conceptuel (cas des associations 1-N, N-M, etc.). Cette prise en compte se fait à l'aide du mécanisme *clé référentielle* -> *clé primaire*.

- Identifier les éventuelles contraintes du diagramme de classes, impossibles à prendre en compte à ce niveau. Ces contraintes sont cependant importantes, envisager une manière possible de les prendre en compte.

#### Aide pour la création du script

- NOTER LES NOM ET PRÉNOM DES DEUX MEMBRES DU BINÔME DANS LE SCRIPT.
- La syntaxe de PostgreSQL est proche de la syntaxe d'Oracle. Il est possible de s'inspirer des scripts Oracle existants sur Moodle pour réaliser celui-ci. Les principaux éléments syntaxiques qui diffèrent sont donnés plus bas. Voir aussi plus haut l'URL vers la documentation en ligne de PostgreSQL.
- Une ébauche du script existe, il se situe dans le projet *CabinetRecrutement\_EJB* au sein du répertoire *BD*. Quelques conseils :
  1. Il est préférable d'utiliser le script existant et de le compléter.
  2. Il est aussi préférable d'utiliser un seul fichier pour toutes les tables plutôt que de les répartir en plusieurs fichiers.
  3. L'utilisation d'Eclipse vous apportera plus de confort, il assure la coloration syntaxique du code SQL, ce que ne fera pas un éditeur de texte basique.
- Conservez la structure du script :
  1. Destruction des objets : **drop table ...**  
 Au cours de vos différents tests, vous jouerez plusieurs fois le script. Les tables existantes mais modifiées doivent être détruites avant d'être recrées. Cette première partie du script est là pour faire le « ménage » !
  2. Création des tables : **create table ...**  
 La création de toutes les tables se situe dans cette partie. Attention, la présence d'une clé référentielle d'une table A vers une table B implique que la table B préexiste à la table A. Cet ordre est à inverser dans la destruction (partie précédente).
  3. Insertion de données de test : **insert into ...**
- L'exécution du script se fait à l'aide de l'outil *pdAdmin3* ou *Aqua Data Studio*. Une fois connecté (voir paramètres du compte), ouvrir une fenêtre de requêtes (icône SQL) et copier/coller le script à partir d'Eclipse. D'autres manières de faire sont possibles mais celle-ci est la plus simple.
- ATTENTION : contrairement au script qui vous est donné, n'utilisez pas les double-quotes (caractère ") pour le nom des tables. Supprimez celles qui existent pour la table *Entreprise*.

#### Quelques éléments syntaxiques pour PostgreSQL

- Comme Oracle, PostgreSQL ne différencie pas les caractères minuscules et majuscules dans les mots clés du langage (create, insert, into, etc.), dans les noms des tables et dans les noms des colonnes. Par contre il fait une différence dans les valeurs textuelles :
  - o `select * from entreprise` est équivalent à `SELECT * FROM ENTREPRISE`
  - o `select * from candidature where nom='toto'` ne donnera pas le même résultat que `select * from candidature where nom='Toto'`
- Le séparateur des chaînes de caractères est le simple quote : '.
- La création d'une table est identique à Oracle. Exemple :

```
create table nom_de_la_table (
    colonne_1    type ,
    colonne_2    type ,
    ...
);
```

Attention, dans l'ébauche de script SQL fourni, le nom de la table entreprise est encadré par des double-quotes. Supprimez-les et ne les utilisez pas pour la suite de votre script.

- La création d'une clé primaire avec une séquence est légèrement différente : dans une syntaxe abrégée (suffisante ici) il suffit d'utiliser le type *serial* et d'ajouter la contrainte *primary key* (voir table *entreprise* dans le script fourni). À l'exécution du script, PostgreSQL générera automatiquement une séquence qui sera nommée de la façon suivante : `nomTable_nomColonnePK_seq`. Par exemple, pour

la table *entreprise*, la séquence se nomme *entreprise\_id\_seq*. Voir aussi dans le script, la partie insertion de données pour l'utilisation des séquences.

- La création d'une clé référentielle est similaire à Oracle. Dans une syntaxe abrégée (la contrainte n'est alors pas nommée), il suffit d'indiquer pour la colonne qui porte la contrainte :

```
nom_colonne    type_colonne    [not null]    references    nom_table
```

La partie entre crochets est optionnelle.

Exemple :

```
ref_entreprise    integer    not null    references    entreprise
```

- Une contrainte de non nullité se fait en ajoutant `not null` après le type de la colonne. Exemple :

```
nom    varchar(50)    not null
```

- Types des données utiles :
  - o `serial` : type entier (*integer*) qui permet la génération automatique d'une séquence
  - o `integer` : type entier sur 4 octets (largement suffisant pour la dimension de notre système d'information...)
  - o `varchar(n)` : chaîne de n caractères max
  - o `text` : chaîne de caractères non bornée en taille. Typiquement le CV d'une candidature est de ce type, ce qui n'est pas le cas du nom et du prénom par exemple.
  - o `date` : stockage d'une date (sans le stockage de l'heure, inutile ici). Pour le stockage de l'heure uniquement, il faut utiliser le type `time` et pour le stockage de la date avec l'heure, il faut utiliser le type `timestamp`.  
Pour insérer une date dans une colonne de ce type, il faut insérer une chaîne de caractères (entourée de simples quotes) au format '2011-09-02' soit le *02 septembre 2015*.

## 4.4 Livraison

Le schéma logique (un scan de la version manuscrite est suffisant) ainsi que le script de création de la base de données seront à déposer sur Moodle avant la séance 2 du fil rouge. Faire une archive contenant les deux parties du livrable (au format zip, 7z, tar, bz2, etc.) qui porte **IMPÉRATIVEMENT LES NOMS DES DEUX MEMBRES DU BINÔME**.



## 5 — Partie 2 : *persistance des données (JPA)*

### 5.1 Déroutement

À ce stade du fil rouge, le modèle relationnel pour le stockage des données de l'application au sein du SGBD PostgreSQL est prêt : du schéma logique a été déduit le script SQL de création des tables, ce script a été exécuté et quelques données de test sont présentes dans la base.

Le développement de l'application cabinet de recrutement qui manipulera les données stockées dans cette base peut commencer. La première tâche à réaliser concerne la gestion de la persistance des données par l'application. Celle-ci est développée en langage Java, langage objet, dont le modèle est entièrement différent du modèle relationnel utilisé par le SGBD PostgreSQL. Cette gestion de la persistance doit donc assurer la correspondance entre ces deux modèles dont le nom le plus courant, en bon français, est le **mapping objet/relationnel**.

L'application est déployée dans un serveur GlassFish qui implémente la spécification EJB 3. Cette spécification comprend une partie liée à la gestion de la persistance : JPA (*Java Persistence API*). Cette API sera au centre du travail de la séance constitué de l'essentiel de la couche basse de l'application : **la couche données**. Il peut être séparé en deux parties principales :

- La création des classes qui « encapsuleront » les données : les **Entity Beans** (appelées aussi *EJB Entités* ou *classes entités*).
- La création des classes qui assureront les fonctionnalités minimales autour des classes entités : création, récupération, mise à jour, effacement appelées aussi fonctionnalités CRUD (*Create, Retrieve* ou *Read, Update* et *Delete*). Ces classes, sous-parties de la couche données, sont appelées conventionnellement (ce n'est pas obligatoire...) des **DAO** (*Data Access Object*).

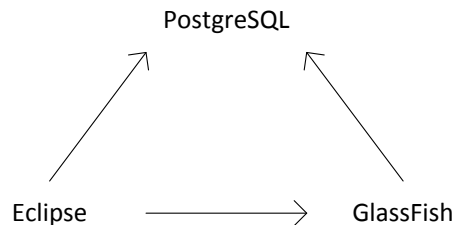
#### Pour aller plus loin...

- Tutorial sur le pattern DAO :  
<http://cyrille-herby.developpez.com/tutoriels/java/mapper-sa-base-donnees-avec-pattern-dao/>
- Résumé de la technologie JPA à l'URL :  
<http://igm.univ-mlv.fr/~dr/XPOSE2007/acollign ORM-JPA/jpa-tech2.html>
- Support de cours sur JPA par un enseignant de l'université de Nice :
  - o Partie 1 : <http://deptinfo.unice.fr/~grin/mescours/minfo/modpersobj/supports/jpa2-1-6.pdf>
  - o Partie 2 : <http://deptinfo.unice.fr/~grin/mescours/minfo/modpersobj/supports/jpa2-2-6.pdf>

Bien évidemment, d'autres ressources sur le sujet sont disponibles sur le Web ou en librairie.

## 5.2 Informations préliminaires

Ces quelques informations permettent de comprendre comment les différents outils qui sont liés entre eux fonctionnent. Ces informations ne sont pas fondamentales pour la réalisation du travail mais elles fournissent une aide à la compréhension des mécanismes qui sont mis en œuvre dans le projet. Elles peuvent être aussi utiles si vous voulez dupliquer l'environnement de travail du fil rouge ou si vous voulez mettre en place un environnement similaire.



### 5.2.1 Configuration d'Eclipse pour l'accès à PostgreSQL

L'IDE Eclipse installé dans la machine virtuelle est configuré à la fois pour un accès au serveur GlassFish (pilotage du serveur via l'onglet *Servers*) et pour une connexion à la base *CABINET\_RECRUTEMENT\_BD*. Cette deuxième configuration est nécessaire aux outils Eclipse qui seront utilisés dans la séance. Le paramétrage de cette configuration est assuré par plusieurs choses :

1. Le driver JDBC pour PostgreSQL est référencé dans Eclipse. Voir : menu *Window > Preferences*, dans l'arborescence à gauche ouvrir *Data Management > Connectivity > Driver Definitions*. Sélectionnez *PostgreSQL JDBC Driver* puis le bouton *Edit...* pour accéder au paramétrage. Bien évidemment, ne rien modifier dans ce paramétrage pour que la connexion fonctionne...  
*Note* : un driver JDBC est un module logiciel qui assure les fonctionnalités de connexion à un SGBD. Il existe des drivers JDBC spécifiques pour Oracle, PostgreSQL, MySQL, etc.
2. Une connexion à la base *CABINET\_RECRUTEMENT\_BD* est paramétrée (elle utilise la définition du driver JDBC ci-dessus). Il est possible, via ce référencement, de manipuler les données dans la base. Pour y accéder il faut ouvrir l'onglet *Data Source Explorer*. Dans cet onglet, les données de la base sont accessibles à partir de l'arborescence *Database Connections > PostgreSQL (localhost) - CABINET\_RECRUTEMENT\_BD*. L'accès à la base de cette manière n'est pas le plus ergonomique qui soit mais ce référencement est indispensable pour d'autres outils Eclipse (outils de retro-conception pour la génération des classes entités, voir plus loin).

### 5.2.2 Configuration d'Eclipse pour l'accès à GlassFish

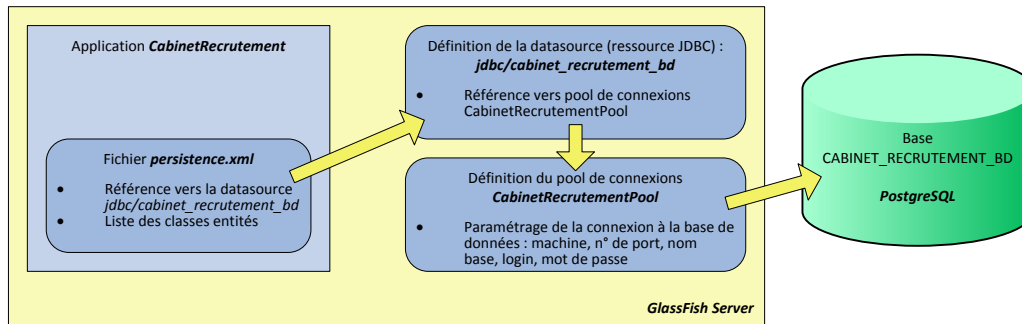
Le paramétrage de l'accès au serveur GlassFish, outre son pilotage (lancement/arrêt), permet un déploiement automatique de l'application dès qu'une modification est apportée. Lors de la modification d'un des fichiers source, Eclipse le compile automatiquement. GlassFish détecte la modification de la(des) classe(s) compilée(s) et re-déploie l'application qui peut alors être testée. Pour configurer ce paramétrage :

1. Création d'un *Runtime Environment* : menu *Window > Preferences*, ouvrir *Server > Runtime Environments*. L'entrée *GlassFish 4.0* permet de configurer l'accès dans le système de fichiers vers le serveur en lui-même.
2. La référence vers le serveur est ensuite ajoutée à l'onglet *Servers*. Cette référence se sert du *Runtime Environment* créé au-dessus.

### 5.2.3 Configuration de GlassFish pour l'accès à PostgreSQL

Eclipse nécessite une connexion à la base pour le bon fonctionnement de certains outils mais l'application, une fois déployée, en a aussi besoin. L'application étant déployée dans GlassFish, une partie de cette

configuration y est paramétrée, l'autre partie se faisant au sein de l'application elle-même. Cette manière de faire permet à l'application d'abstraire les paramètres de connexion (ils sont totalement indépendants de l'application), le paramétrage devient ainsi une tâche d'administration système.



Configuration de l'application pour la connexion à la base de données

La configuration de GlassFish pour la connexion à la base de données se fait grâce à la console d'administration Web accessible à l'URL <http://localhost:4848/common/index.jsf>. Elle se fait à deux niveaux :

1. Définition d'un pool de connexions JDBC.

Note : pour que GlassFish se connecte à une base PostgreSQL il lui faut, comme Eclipse, avoir accès à un driver JDBC. Ce driver, un fichier JAR nommé *postgresql-9.3-1101.jdbc41.jar*, est déposé dans le répertoire */opt/glassfish4/glassfish/domains/domain1/lib/ext*.

Dans l'arborescence de la partie gauche de l'interface, ouvrir *Ressources > JDBC > Pools de connexions JDBC*. À cet endroit se retrouvent paramétrés plusieurs pools (ceux des TP architecture) dont celui de l'application du fil rouge : *CabinetRecrutementPool*. C'est ici que sont notés les paramètres de connexion à la base (onglet *Autres propriétés*) : nom de la machine, n° de port, nom de la base, login et mot de passe de l'utilisateur de la base de données.

2. Définition d'une datasource (appelée ici ressource JDBC).

GlassFish gère l'accès à un certain nombre de ressources (dont les datasources) grâce à un système d'annuaire ; les applications hébergées peuvent ainsi y accéder. La ressource JDBC n'est qu'une entrée d'annuaire qui pointe vers le pool de connexions ci-dessus. Pour accéder à ce paramétrage, ouvrir *Ressources > JDBC > Ressources JDBC*, vous y trouverez la datasource de l'application : *jdbc/cabinet\_recrutement\_bd*.

Pour se connecter à la base de données, l'application n'a plus qu'à appeler la « référence » de la connexion (la ressource JDBC) disponible au sein de l'annuaire disponible dans GlassFish. Cette référence est notée dans un fichier (au format XML) disponible dans l'application : le fichier *persistance.xml* situé dans le projet *CabinetRecrutement\_EJB*, répertoire *ejbModule/META-INF*. Ce fichier comprend plusieurs choses : la liste des classes entités à prendre en compte pour le mapping objet/relationnel et le nom de la datasource dans l'élément *<jta-data-source>*.

### 5.3 Génération des classes entités

Le propos de cette partie des développements est identique à ce qui a été vu dans le TP *JPA pour le mapping objet-relationnel*. En revanche, l'approche utilisée sera une approche contraire : *bottom-up* à la place de *top-down*. Quelques explications ont été données page 25 du document de la partie 1 (*conception des données de l'application*).

Dans l'approche *bottom-up*, la base de données préexiste au modèle objet. Deux solutions sont alors possibles : écrire manuellement la représentation objet des données (les classes) ainsi que la correspondance (le mapping) avec les structures de la base (les tables et les colonnes) ou utiliser un outil qui fera automatiquement ce travail. C'est ce second choix qu'il vous est demandé de réaliser.

Cette version d'Eclipse (Eclipse pour *Java Enterprise Edition*) dispose d'un outil pour faire cette retro-conception. Pour y accéder :

- Sélectionnez le projet *CabinetRecrutement\_EJB* dans l'onglet *Project Explorer* ou *Navigator*, clic droit > menu *JPA Tools > Generate Entities from Tables...* Note, l'outil pour l'approche inverse (*top-down*) existe aussi : *Generate Tables from Entites*.  
Un assistant composé de plusieurs pages apparaît (en cas de doute, il est possible de revenir en arrière).
- Page 1 : **Select Tables**
  - o Sélectionnez la connexion : *PostgreSQL (localhost) – CABINET\_RECRUTEMENT\_BD*.
  - o Activez la connexion (bouton *Connect*).
  - o Sélectionnez le schéma *public*. Les tables que vous avez créées doivent apparaître, sélectionnez-les.  
Vérifiez que la case *Update class list in persistence.xml* est bien cochée.
  - o Bouton *Next >*.
- Page 2 : **Table Associations**  
La liste des associations entre les tables apparaît, il est possible de sélectionner les associations une à une et d'accéder au paramétrage qui peut être adapté (on ne le fera pas ici) :
  - o L'attribut de jointure.
  - o La navigabilité entre les classes qu'il est possible de (dé)sélectionner.  
Essayez de comprendre ce qu'implique cette notion de navigabilité.
  - o Le nom des attributs qui représente les associations.
  - o Bouton *Next >*.
- Page 3 : **Customize Default Entity Generation**  
Différents réglages liés à la génération des classes entités sont opérés ici.
  - o Méthode de génération des valeurs de clés primaires, *Key generator*. PostgreSQL utilise des séquences, valeur qu'il faut choisir dans la liste déroulante.
  - o Sequence name : saisir le pattern *\$table\_\$pk\_seq* qui correspond au nom des séquences générées automatiquement par PostgreSQL.
  - o *Entity access* : *Field*
  - o *Associations fetch* : *Default*
  - o *Collection properties type* : conservez *java.util.Set*.
  - o Package : choisir le package *eu.telecom\_bretagne.cabinet\_recrutement.data.model*.  
La classe *Entreprise* est déjà présente dans ce package.
- Page 4 : **Customize Individual Entities**
  - o Page récapitulative par classe entité des choix de la page précédente. Rien à modifier ici.
- Bouton *Finish*.

Vérifiez que les classes soient générées dans le package indiqué et que les références de celles-ci existent bien dans le fichier *persistence.xml*.

Examinez les classes entités et le mapping. Au besoin, l'assistant de création peut être rejoué si on veut modifier le paramétrage : celui-ci est mémorisé, il suffit alors d'indiquer les modifications. De même, il peut être intéressant de comparer le modèle généré avec celui présent dans le diagramme de classes UML présenté dans le document de la partie 1 (voir page 26).

### Petite modification à apporter aux classes générées

Il est nécessaire de compléter l'annotation *@SequenceGenerator* de l'attribut correspondant à la clé primaire de chacune des différentes entités en ajoutant le paramètre *allocationSize=1* qui fixe l'incrément de la séquence. Par défaut, la valeur de cet incrément est de 50, ce qui provoque une erreur à l'exécution.



Par exemple, pour la classe *Entreprise*, il faut remplacer...

```
@Id
@SequenceGenerator(name="ENTREPRISE_ID_GENERATOR", sequenceName="ENTREPRISE_ID_SEQ")
@GeneratedValue(strategy=GenerationType.SEQUENCE, generator="ENTREPRISE_ID_GENERATOR")
private Integer id;
```

...par...

```
@Id
@SequenceGenerator(name="ENTREPRISE_ID_GENERATOR", sequenceName="ENTREPRISE_ID_SEQ", allocationSize=1)
@GeneratedValue(strategy=GenerationType.SEQUENCE, generator="ENTREPRISE_ID_GENERATOR")
private Integer id;
```

## 5.4 Création des DAO

Une fois les classes entités créées, afin de pouvoir les manipuler ensuite au sein de l'application, il est nécessaire de créer les classes utilitaires qui vont assurer le lien entre la couche métier (celle qui assure les fonctionnalités de l'application) et nos toutes nouvelles entités

Ces classes assureront les fonctionnalités CRUD pour les classes entités : il y aura donc un DAO pour la classe *Entreprise* (*EntrepriseDAO* dont l'ébauche existe déjà), un DAO pour la classe *Candidature* (*CandidatureDAO*), etc. Les DAO sont une partie de la couche données, ils seront donc localisés dans le package `eu.telecom_bretagne.cabinet_recrutement.data.dao`. Par convention ils portent le nom de la classe entité suffixé par DAO.

### 5.4.1 Implémentation complète du DAO de la classe entité *Entreprise* : *EntrepriseDAO*

L'ébauche de cette classe est déjà fonctionnelle mais incomplète, une fois complétée, elle pourra servir de modèle pour les autres DAO. Cette classe est un composant EJB, l'annotation `@Stateless` au-dessus de la déclaration de la classe l'indique. Elle dispose d'un attribut, de type *EntityManager*, qui sert dans chacune des méthodes et dont le but est d'assurer le lien avec la base de données. À noter, cet attribut n'est pas instancié dans la classe (il n'existe pas de `new EntityManager(...)`). À l'exécution, la référence vers cet objet sera assurée par le conteneur d'EJB à l'aide du mécanisme appelé *injection de dépendance*.

Cette classe ne possède pour le moment que deux méthodes (fonctionnalité R de l'acronyme CRUD) :

- `public Entreprise findById(Integer id)`  
La méthode retourne une instance de la classe *Entreprise* pour une valeur d'id donnée.
- `public List<Entreprise> findAll()`  
La méthode retourne la liste (au sein d'une instance de `java.util.List`) de toutes les entreprises référencées dans la base.

Afin de compléter cette classe, il sera nécessaire d'ajouter les méthodes suivantes (les noms donnés sont optionnels, il est possible de les modifier) :

- `public Entreprise persist(Entreprise entreprise)`  
Fonctionnalité C de l'acronyme CRUD : cette méthode stockera dans la base de données les informations d'une entreprise (présent sous la forme d'une instance d'*Entreprise*).
- `public Entreprise update(Entreprise entreprise)`  
Fonctionnalité U de l'acronyme CRUD : cette méthode rend persistant les mises à jour opérées sur une instance d'*Entreprise* passée en paramètre.
- `public void remove(Entreprise entreprise)`  
Fonctionnalité D de l'acronyme CRUD : cette méthode supprime les données d'une entreprise passée en paramètre.

Les méthodes de mise à jour et de suppression sont liées à des fonctionnalités marquées comme mineures (donc optionnelles) dans le cahier des charges. Cependant pour une meilleure compréhension des mécanismes il est intéressant d'implémenter complètement la classe.

### Méthodes principales de la classe EntityManager

Comme indiqué plus haut, la liaison avec la base de données est assurée par l'intermédiaire de l'EntityManager. Les principales méthodes de cette classe (c'est en fait une interface) sont :

- **void persist**(java.lang.Object entity)  
Rend persistant l'instance de la classe entité passée en paramètre.
- **<instance d'entité> find**(classe de l'entité, valeur de la clé primaire)  
Retourne l'instance de l'entité dont on connaît la clé primaire : peut être une chaîne de caractères, un entier, etc. Classe de l'entité : par exemple, pour la classe *Entreprise*, *Entreprise.class*.
- **Query createQuery**(java.lang.String qlString)  
Permet d'exprimer une requête exprimée en JPQL (attention, ce n'est pas du SQL !). Sur l'objet *Query* retourné, il est possible d'obtenir le résultat de la requête par l'appel de la méthode *getResultList()*.
- **Entity merge**(java.lang.Object entity)  
Met à jour dans la base les modifications apportées à une instance de classe entité. L'instance mise à jour est renvoyée par la méthode.
- **void remove**(java.lang.Object entity)  
Supprime de la base une instance de classe entité.

### Éléments de syntaxe du langage de requête JPQL

Le langage JPQL (*Java Persistence Query Language*) est une des possibilités permettant d'exprimer des requêtes sur une base de données dans la spécification JPA. Sa syntaxe la rend proche du langage SQL (*select*, *update*, *delete*, etc.), ce qui facilite son apprentissage mais qui peut rendre aussi sa lecture ambiguë...

Exemple 1 : requête JPQL de la méthode *findAll()* :

```
select entreprise from Entreprise entreprise order by entreprise.id
```

Le langage JPQL travaille avec le modèle objet et non le modèle relationnel. D'un point de vue pratique, cela veut dire que dans la requête ci-dessus, derrière la clause *from* le terme *Entreprise* désigne la classe entité et NON LA TABLE *entreprise*. De même, derrière *order by*, *entreprise.id* désigne l'attribut *id* de la classe *Entreprise* et non la colonne *id* de la table *entreprise*.

Exemple 2 : expression d'une jointure. Dans le cas d'un employé appartenant à un département (présence d'un attribut *departement* de type *Departement* dans la classe *Employe*), la requête récupérant la liste des employés appartenant à un département donné (trié par nom d'employé descendant) s'exprime de la manière suivante :

```
select emp from Employe emp join emp.departement dept
where emp.nom="nom_du_dept_recherché" order by emp.nom desc
```

#### 5.4.2 Création des autres DAO

Le DAO *EntrepriseDAO* servira de modèle pour le développement des autres DAO. Certains d'entre-eux nécessiteront d'être adaptés par rapport au DAO existant :

- **CandidatureDAO**
  - o Ajout d'une méthode pour l'obtention de la liste des candidatures qui correspondent à un secteur d'activité et un niveau de qualification donnés.
- **OffreEmploiDAO**
  - o Ajout d'une méthode pour l'obtention de la liste des offres d'emploi pour une entreprise donnée.
  - o Ajout d'une méthode pour l'obtention de la liste des offres d'emploi qui correspond à un secteur d'activité et un niveau de qualification donnés.
- **SecteurActiviteDAO**
  - o La liste des secteurs d'activité n'évolue pas, elle reste fixe. Il n'est donc pas nécessaire d'en supprimer, la méthode *remove(...)* est donc inutile.

- o Suivant les choix de navigabilité faits lors de la création des classes entités, une instance de la classe *SecteurActivite* peut référencer un ensemble d'offres d'emploi et un ensemble de candidatures. Si c'est le cas (c'est le choix par défaut), la méthode *update(...)* est alors nécessaire.

- **NiveauQualificationDAO**

- o Mêmes remarques que pour le DAO *SecteurActiviteDAO*.

Un DAO est une classe de type bean session (un EJB). Pour créer ce type de classes, Eclipse fournit un assistant :

- Sélectionnez le projet *CabinetRecrutement\_EJB* dans l'onglet *Project Explorer* ou *Navigator*, clic droit > menu *New... > Session Bean (EJB 3.x)...*
- Page 1 : **Create EJB 3.x Session Bean**
  - o Saisir le nom de package de la classe :  
`eu.telecom_bretagne.cabinet_recrutement.data.dao.`
  - o Saisir le nom de la classe.
  - o Laisser les choix par défaut pour *State type (Stateless)* et *Create business interface*.
- Page 2, rien à modifier. Bouton *Finish*.

L'ébauche de la classe est créée mais la seule méthode existante est le constructeur par défaut. Les annotations *@Stateless* et *@LocalBean* indiquent qu'il s'agit bien d'un EJB.

## 5.5 Procédure de contrôle du bon fonctionnement des DAO

Afin de contrôler que les DAO fonctionnent correctement, il n'est pas possible de créer une classe (avec une méthode main) qui instancierait un DAO sur lequel il serait possible de tester les appels de méthodes. En effet pour fonctionner, les DAO nécessitent d'être déployés sur GlassFish afin de bénéficier de toute la « mécanique » offerte par l'environnement d'exécution (connexion à la base, mécanisme d'injection de dépendance, etc.). Les tests doivent donc se faire sur le serveur.

Cette manipulation, un peu plus complexe qu'une simple classe exécutable, se fait au sein du projet *CabinetRecrutement\_WEB*. Une classe particulière qui génère du contenu Web (une classe de type *Servlet*) est accessible à l'URL : `http://localhost:8080/CabinetRecrutement_WEB/ControlesDAO` (le lien est présent dans les marque-pages de Firefox). Cette page affiche le résultat du test. Le code est accessible dans la classe *ControlesDAOServlet* (package `eu.telecom_bretagne.cabinet_recrutement.front.controlesDAO`).

La référence vers le DAO à tester est faite via la classe *ServiceLocator* (package `eu.telecom_bretagne.cabinet_recrutement.front.utils`). Une fois cette référence récupérée, les méthodes du DAO peuvent être appelées et les résultats sont affichés grâce aux appels `out.println(contenu_à_afficher)`.

Un exemple de contrôle pour les deux méthodes fournies dans la classe *EntrepriseDAO* est donné, l'adaptation pour les autres DAO devrait se faire facilement.

## 5.6 Livraison

Le code des classes développées, classes entités et classes des DAO, sera à déposer sur Moodle avant la première séance de la partie 3. Faire une archive contenant uniquement les fichiers sources (au format *zip*, *7z*, *tar*, *bz2*, etc.). Ce fichier devra porter impérativement les noms des deux membres du binôme.



## 6 — Annexes au document de la partie 2

### 6.1 Gestion correcte des deux extrémités d'une association avec JPA

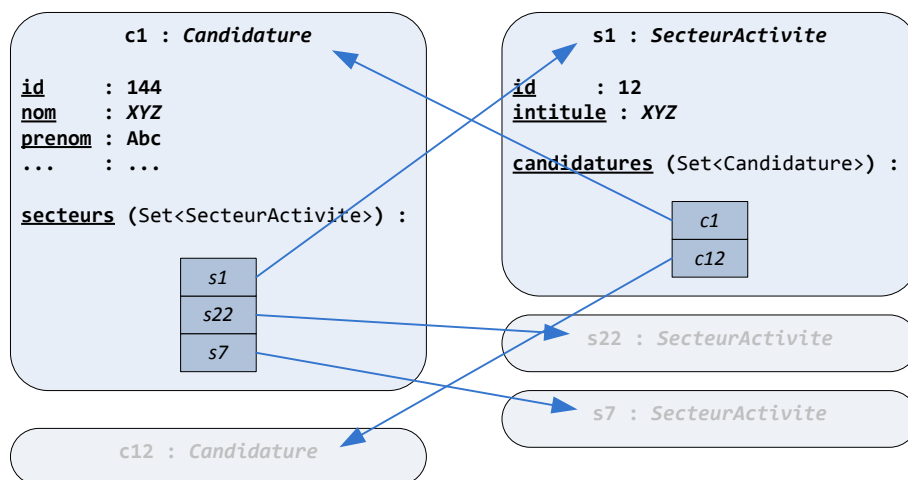
Dans le transparent 22 du support de cours additionnel *Mapping objet-relationnel avec JPA* il est indiqué que le développeur est responsable de la gestion correcte des deux extrémités d'une association. Ne pas prendre en compte cette obligation implique que l'on rencontrera nécessairement à un moment ou un autre des incohérences dans les données de l'application.

L'exemple donné dans ce support est celui d'un employé qui change de département. Pour que cette problématique soit plus proche de celle du fil rouge, continuons avec l'exemple – complètement similaire avec ce qui est indiqué dans le support – d'une candidature à laquelle sont associées plusieurs secteurs d'activités.

Lors de la création des beans entités, le choix par défaut est de créer des associations bidirectionnelles :

- L'instance de *Candidature* connaît (elle référence) l'ensemble des instances de *SecteurActivite* qui lui sont associées.
- Inversement, une de ces instances de *SecteurActivite* connaît l'ensemble des instances de *Candidature* associées.

De cette manière on construit un graphe (une instance en référence une autre, etc.) qui peut être résumé dans le schéma suivant :



Pour la création d'une nouvelle candidature, le « mini » algorithme est le suivant :

1. Instanciation :

```
Candidature nouvelleCandidature = new Candidature();
```

2. Renseignement des différents attributs de la nouvelle candidature par l'appel des méthodes *setXXX(...)* :

```
nouvelleCandidature.setNom("XYZ");
nouvelleCandidature.setPrenom("Abc");
etc.
```

3. Mise à jour des secteurs d'activité. Pour chacun d'eux (une boucle est nécessaire) :

(a) Récupération de l'instance auprès du DAO correspondant :

```
Secteur s = secteurActiviteDAO.findById(12);
```

(b) Ajout de celle-ci à la structure de données *secteurs* de l'instance de *Candidature* :

```
secteurs.add(s); -> secteur s'obtient en faisant
nouvelleCandidature.getSecteurActivites()
```

(c) Gestion du côté opposé de l'association : ajout de la candidature nouvellement créée à la structure de données référençant les candidatures dans l'instance du secteur d'activité :

```
s.getCandidatures().add(nouvelleCandidature);
```

*Dans un contexte où la persistance dans une base de données n'est pas assurée, on reste dans le modèle objet et on est alors sûr d'avoir géré correctement la bidirectionnalité de notre association. Dans notre application, la persistance est assurée via JPA et il va falloir, en plus, gérer le stockage dans la base.*

(d) Toujours dans la boucle gérant le secteur d'activité, comme celui-ci a été modifié, il faut répercuter la modification dans la base :

```
secteurActiviteDAO.update(s);
```

*Si cette mise à jour n'est pas faite, lors de l'examen de la base, vous vous apercevez que la candidature a été créée correctement mais l'association avec les secteurs d'activité n'est pas faite : la table d'association n'est pas remplie !*

4. Les informations sur la nouvelle candidature sont prêtes, il ne reste plus qu'à la rendre persistante sur la base de données :

```
candidatureDAO.persist(c1);
```

Attention, si l'exemple de la création d'une candidature a été déroulée, la même démarche est à appliquer dans les tous les cas de figure qui impliquent les deux côtés d'une association. Par exemple :

- Suppression d'un ou plusieurs secteurs d'activité lors de la mise à jour d'une candidature => les secteurs d'activité qui la référençaient doivent être mis à jour.
- Création et mise à jour d'une offre d'emploi.
- Etc.

## 6.2 Complément d'information sur les états d'un bean entité – Incidence sur la suppression

Un bean entité peut connaître 4 états différents dans son cycle de vie :

- **Nouveau (new)** : l'instance du bean existe en mémoire mais n'a pas encore été rendue persistante dans la base. Lors de la création d'une nouvelle instance c'est l'état qui existe avant l'appel à la méthode *entityManager.persist()*.
- **Géré (managed)** : état rencontré après l'appel à la méthode *persist()* par exemple. Le bean est associé avec les données dans la BD, les changements seront répercutés (modification, suppression).
- **Détaché (detached)** : le bean n'est plus associé au contexte de persistance. Les modifications ne seront pas prises en compte dans la base : l'instance possède une vie propre par rapport à la base.

- **Supprimé (removed)** : le bean est associé à la BD, au contexte, et est programmé pour être supprimé (les données seront supprimées aussi).

Les deux états principaux à prendre en compte sont l'état **géré** et **détaché**.

Dès qu'une modification est apportée à l'instance d'un bean, celui-ci se trouve dans un état **détaché**, les modifications n'existent alors que dans la mémoire de l'application. Afin de synchroniser les données modifiées avec la base, il est alors nécessaire de faire un appel à la méthode **merge()** de l'*Entity Manager*. C'est d'ailleurs ce que vous faites dans les méthodes *update(...)* de vos différents DAO : l'état du bean redevient **géré** et les données préalablement modifiées sont répercutées dans la base.

Lors d'une suppression, l'erreur suivante peut apparaître : *Entity must be managed to call remove* et la suppression ne s'opère pas. En effet, un bean entité **NE PEUT PAS ÊTRE SUPPRIMÉ S'IL EST DANS UN ÉTAT DÉTACHÉ**. La solution consiste donc à le rattacher si nécessaire (méthode *contains(...)*) avant de la supprimer.

Si dessous, le code de la suppression d'une entreprise dans le DAO *EntrepriseDAO* :

```
public void remove(Entreprise entreprise)
{
    if(!entityManager.contains(entreprise)) // Si l'entité n'est pas dans un état
                                              // "géré" (managed), il est impossible
                                              // de la supprimer directement, erreur
    {
        // "Entity must be managed to call
        // remove"
        entreprise = entityManager.merge(entreprise); // Il faut la "rattacher" au
                                                         // contexte de persistance
                                                         // par l'appel de la méthode
                                                         // merge de l'Entity Manager.
    }
    // L'entité était déjà attachée ou a été rattachée, on peut donc la supprimer...
    entityManager.remove(entreprise);
}
```

### 6.3 Requêtes JPQL dans les différents DAO

Voici la liste de l'ensemble des requêtes JPQL de l'application exemple disponible en ligne. Ces requêtes seront bien évidemment à adapter à la structure de vos classes entités (le nom des classes et des attributs peuvent différer).

Remarques sur la syntaxe :

- Malgré sa ressemblance à SQL, les requêtes JPQL agissent sur le modèle objet des classes entités et non sur les tables de la base de données, le résultat est donc une liste d'instances.
- Corollaire de la remarque précédente, quand une jointure est faite dans la requête, l'attribut de jointure est un attribut de la classe.
- Il existe d'autres formes de jointures (*left join*, *inner join*, etc.), celle qui est présentée ici est la plus simple. Elle utilise le mot clé *join*. Dans l'exemple en ligne, seules deux requêtes utilisent des jointures. Il s'agit en fait de quasiment la même : liste des candidatures (ou offres d'emploi) pour un secteur d'activité et un niveau de qualification.
- Pour passer des paramètres à une requête, il est possible de les insérer en utilisant la concaténation de chaînes de caractères. La manière présentée ici insère dans la requête un (plusieurs) paramètre(s) dont le nom est préfixé par le caractère ':' (exemple -> :idSA). La mise à jour de la valeur dans la requête se fait par l'appel de la méthode *setParameter* sur l'objet *Query*. Attention, dans *setParameter*, on n'utilise plus le préfixe ':'.

### CandidatureDAO

```
public List<Candidature> findBySecteurActiviteAndNiveauQualification(int idSecteurActivite, int idNiveauQualification)
{
    Query query = entityManager.createQuery("select c from Candidature c join c.secteursActivite secteur " +
                                           "where secteur.id = :idSA and c.niveauQualification.id = :idNQ " +
                                           "order by c.id desc");

    query.setParameter("idSA", idSecteurActivite);
    query.setParameter("idNQ", idNiveauQualification);
    List<Candidature> l = query.getResultList();
    return l;
}

public List<Candidature> findAll()
{
    Query query = entityManager.createQuery("select candidature from Candidature candidature " +
                                           "order by candidature.id desc");

    List l = query.getResultList();
    return (List<Candidature>)l;
}
```

### OffreEmploiDAO

```
public List<OffreEmploi> findByEntreprise(int idEntreprise)
{
    Query query = entityManager.createQuery("select offreEmploi from OffreEmploi offreEmploi " +
                                           "where offreEmploi.entreprise.id = :idE " +
                                           "order by offreEmploi.id desc");

    query.setParameter("idE", idEntreprise);
    List<OffreEmploi> l = query.getResultList();
    return l;
}

public List<OffreEmploi> findBySecteurActiviteAndNiveauQualification(int idSecteurActivite, int idNiveauQualification)
{
    Query query = entityManager.createQuery("select oe from OffreEmploi oe join oe.secteursActivite secteurs " +
                                           "where secteurs.id = :idSA and oe.niveauQualification.id = :idNQ " +
                                           "order by oe.id desc");

    query.setParameter("idSA", idSecteurActivite);
    query.setParameter("idNQ", idNiveauQualification);
    List<OffreEmploi> l = query.getResultList();
    return l;
}

public List<OffreEmploi> findAll()
{
    Query query = entityManager.createQuery("select offreEmploi from OffreEmploi offreEmploi " +
                                           "order by offreEmploi.id desc");

    List l = query.getResultList();
    return (List<OffreEmploi>)l;
}
```

### EntrepriseDAO

```
public List<Entreprise> findAll()
{
    Query query = entityManager.createQuery("select entreprise from Entreprise entreprise " +
                                           "order by entreprise.id");

    List l = query.getResultList();
    return (List<Entreprise>)l;
}
```

### NiveauQualificationDAO

```
public List<NiveauQualification> findAll()
{
    Query query = entityManager.createQuery("select niveauQualification from NiveauQualification niveauQualification " +
                                           "order by niveauQualification.id");

    List l = query.getResultList();
    return (List<NiveauQualification>)l;
}
```



**SecteurActiviteDAO**

```
public List<SecteurActivite> findAll()
{
    Query query = entityManager.createQuery("select secteurActivite from SecteurActivite secteurActivite " +
                                           "order by secteurActivite.id");

    List l = query.getResultList();
    return (List<SecteurActivite>)l;
}
```





## 7 — Partie 3 : *services métier et couche présentation*

### 7.1 Déroulement

Les couches les plus basses de l'application *cabinet de recrutement* sont maintenant en place. Elles sont composées de la base de données ainsi que de la couche données qui y accède. Les fonctionnalités métier accompagnées de la couche qui gèrent l'interface graphique de l'application seront le centre de ces deux séances. Le principe est ici de développer conjointement les éléments de ces deux couches autour d'une fonctionnalité particulière puis de réaliser sur le même modèle le travail pour les autres fonctionnalités.

Chacune de ces deux couches héberge des composants particuliers :

- La couche service qui assure les fonctionnalités métier est constituée d'un ensemble de **beans session** (comme le sont d'ailleurs aussi les composants DAO).
- La couche interface graphique est constituée de **pages JSP** dont le but est la génération dynamique de contenu Web. Cette couche pourrait aussi être constituée de Servlets Java (comme le TP architectures) dont le but est identique mais avec une approche différente.

Une ébauche de ces deux couches existe déjà qu'il faudra compléter dans un premier temps. La suite du travail consistera à compléter l'ensemble des fonctionnalités telles que définies au cahier des charges.

*Un document annexe est présent sur Moodle : annexes séances 3 et 4. Il présente la liste des composants développés dans l'application exemple ([http://srv-labs-006.enst-bretagne.fr/CabinetRecrutement\\_WEB/](http://srv-labs-006.enst-bretagne.fr/CabinetRecrutement_WEB/)). Les informations qui y sont présentées ne sont fournies qu'à titre indicatif mais elles peuvent constituer une aide à la réalisation de cette étape.*

#### Pour aller plus loin...

- Sur Moodle dans la partie consacrée au module FIP INF 211, partie 4 (*architecture des applications d'un SI*) sont présentées un ensemble de références bibliographiques traitant des technologies JEE : <http://formations.telecom-bretagne.eu/fad/course/view.php?id=23428>
- Memento HTML : <http://nephi.unice.fr/CoursHTML/index.php>
- Cours rapide ( 30 transparents) sur la technologie JSP : <http://www.lifl.fr/~seinturi/middleware/jsp.pdf>. Ce cours est assez ancien (2002) mais les principes de base de la technologie sont inchangés.

## 7.2 Poursuite du développement du composant *ServiceEntreprise*

La version actuelle de ce composant ne réalise que deux fonctionnalités basiques (qui en fait ne font qu'encapsuler deux fonctionnalités CRUD du DAO *EntrepriseDAO*) :

- ***getEntreprise(int id)*** : obtention d'une entreprise par son id.  
Cette méthode réalise la fonction *afficher les informations d'une entreprise* décrite dans le cahier des charges page 8.
- ***listeDesEntreprises()*** : obtention de la liste de toutes les entreprises référencées dans la base.  
Cette méthode réalise la fonction *lister les entreprises référencées* décrite dans le cahier des charges page 8.

### 7.2.1 Examen de l'état actuel de la classe

Le code de la classe se situe dans le projet *CabinetRecrutement\_EJB*, au sein du package `eu.telecom_bretagne.cabinet_recrutement.service`. La convention de nommage utilisée dans le fil rouge montre bien que cette classe fait partie de la couche service.

La classe *ServiceEntreprise* est un *bean session*. Comme dans le mapping objet/relationnel en JPA, ceci est paramétré à l'aide d'annotations Java. Cette classe en possède trois :

- ***@Stateless***  
C'est l'annotation principale qui définit la classe comme étant un bean session. C'est un bean sans état, c'est-à-dire qu'il ne mémorise aucune donnée entre deux requêtes.
- ***@LocalBean***  
Cette annotation précise que le bean peut être invoqué directement par sa classe et non uniquement par ces, éventuelles, interfaces (interface locale ou interface distante dite aussi *remote*).  
Note : cette manière de faire est celle qui a été utilisée pour tester le bon fonctionnement des DAO. Ceux-ci ne possédant pas d'interfaces, ils étaient récupérés dans l'interface graphique (via la classe *ServicesLocator*) directement par le nom de la classe. Comme les beans session de la couche service seront récupérés ici à travers leur interface *remote*, cette annotation est ici optionnelle. Elle a été ajoutée automatiquement par l'assistant d'Eclipse de création des beans session et n'a pas été supprimée ensuite.

Le DAO *EntrepriseDAO* est utilisé dans les deux méthodes existantes. Pour le moment, une seule dépendance est nécessaire, ce ne sera plus le cas quand les fonctionnalités seront étendues (voir document annexe). Chaque dépendance est représentée dans la classe de la manière suivante :

- `@EJB private EntrepriseDAO entrepriseDAO;`  
L'annotation *@EJB* précise que la référence vers le DAO sera renseignée par le conteneur au moment du déploiement de l'application.  
D'autre part, le choix opéré ici est un choix plus simple que dans le TP architectures : comme chaque DAO est annoté par *@LocalBean*, il n'est pas nécessaire de typer la référence du DAO (*entrepriseDAO*) par son interface (qui n'existe pas...) mais directement par la classe. Sans l'annotation *@LocalBean*, cela aurait été impossible.

L'interface et la classe concrète partagent le même package :

`eu.telecom_bretagne.cabinet_recrutement.service`. Le nom de l'interface est : *IServiceEntreprise*.

### 7.2.2 Ajout de nouvelles fonctionnalités

L'ajout de nouvelles fonctionnalités dans le composant *ServiceEntreprise* se traduit matériellement par l'ajout de nouvelles méthodes dans le bean. Comme le bean est accédé via son interface distante, il est nécessaire d'ajouter d'abord la signature de la méthode au sein de l'interface. Une erreur apparaît alors dans la classe concrète : il faut y ajouter l'implémentation de la nouvelle méthode.

## 7.3 Ajout de nouveaux composants

### 7.3.1 Création d'un bean session

Ajouter de nouveaux composants revient à ajouter de nouveaux beans session. Eclipse dispose d'un assistant qui permet d'automatiser cette tâche. Pour y accéder :

- Sélectionnez le projet *CabinetRecrutement\_EJB* dans l'onglet *Project Explorer* ou *Navigator*, clic droit > *New > Session Bean (EJB 3.x)* Un assistant composé de deux pages apparaît (en cas de doute, il est possible de revenir en arrière).
- **Page 1**  
Plusieurs éléments à renseigner :
  - o Le nom du package : `eu.telecom_bretagne.cabinet_recrutement.service`
  - o Le nom de la classe : la convention utilisée (à suivre, ou pas...) jusqu'à présent fait précéder le nom du bean par le préfixe *Service*.
  - o *State type* : laissez à *Stateless*.
  - o Partie *Create business interface* : cocher *Remote*, un nom pour l'interface est proposé du type *NomDuBeanRemote*. Toujours par convention, il est souvent préféré de préfixer une interface par le caractère 'I' qui rend compte du caractère fonctionnel (I pour interface). Comme il n'est pas prévu d'avoir une interface locale (les interfaces des beans sont toutes distantes), le suffixe *Remote* peut être oublié. Le nom de l'interface devient alors : *INomDuBean*.
  - o Laissez cochée la case *No-interface* qui permet l'ajout de l'annotation *@LocalBean*.
- **Page 2**  
Rien à modifier ici, bouton *Finish*.

Vérifiez que la classe et l'interface remote soient bien créées dans le bon package et les bons projets.

### 7.3.2 Référencement du nouveau bean dans la classe *ServicesLocator*

Une fois le bean créé, avant de le tester (au sein de la Servlet *ControlesDAOServlet*) ou de l'utiliser dans l'interface Web de l'application, il est nécessaire de le référencer dans la classe *ServicesLocator*. Le commentaire de la méthode *getRemoteInterface* donne des indications pour composer le nom JNDI nécessaire pour retrouver le bean. À noter, les noms JNDI des beans publiés dans le serveur GlassFish apparaissent dans le fichier de log du serveur.

## 7.4 La couche présentation : l'interface graphique de l'application

L'ébauche de l'interface graphique de l'application utilise la technologie JSP (*Java Server Pages*) que l'on peut comparer aux technologies PHP ou ASP (Microsoft). Dans toutes ces technologies le code source de ces pages mêle, d'un point de vue syntaxique, à la fois du code de présentation Web (*HTML* principalement mais aussi éventuellement du *code CSS* et/ou *Javascript* et/ou etc.) et du code Java notamment pour les appels aux beans session précédemment développés.

La durée du module ne permet pas d'assurer un cours sur les interfaces Web mais, au moins à un premier niveau, la technologie est relativement simple. Cette partie vise à expliquer l'existant déjà fourni à partir duquel il sera possible d'écrire les interfaces pour les nouvelles fonctionnalités.

L'application du TP architectures n'utilise pas de pages JSP mais des Servlets Java. Si vous le préférez, il est aussi tout à fait possible d'utiliser cette approche. Dans le cas de la technologie JSP, ce qui est réellement exécuté côté serveur est une Servlet. En effet le serveur génère à partir du code de la JSP le code source de la Servlet équivalente (c'est une sorte de compilation), cette Servlet est ensuite compilée (génération d'un fichier *.class*) puis exécutée. Ce processus est exécuté lors de la première invocation de la page JSP. Il est peut être utile de consulter ce code source, la pile d'erreurs (*stacktrace*) peut y faire référence. Le code Java des Servlets générées se trouve dans le répertoire `/opt/glassfish4/glassfish/domains/domain1/generated/jsp/CabinetRecrutement/`  
`CabinetRecrutement_WEB_war/org/apache/jsp`

Rappel : le fil rouge n'est pas un concours de design Web, l'essentiel est de pouvoir avoir accès aux fonctionnalités de l'application même si votre interface est un peu « rustique ».

### 7.4.1 Examen de l'existant

Le codage de l'interface graphique de l'application est l'objet du projet *CabinetRecrutement\_WEB*. On y trouve deux parties principales :

- Le contenu Web en lui-même à l'intérieur du répertoire *WebContent* : pages JSP, pages HTML mais aussi fichiers CSS, fichiers images, etc. Tout ce qui est déposé ici est accessible via le serveur à l'URL `http://localhost:8080/CabinetRecrutement_WEB/`. C'est le répertoire de base du contenu Web. À noter : tout fichier possédant l'extension *.jsp* est considéré comme une page JSP et sera donc compilé lors de sa première invocation, compilation qui peut éventuellement échouer.
- Le code Java d'un certain nombre de classes utilisables par les pages JSP (répertoire *src*). Ces classes appartiennent au sous package `eu.telecom_bretagne.cabinet_recrutement.front` :
  - o Le code de la Servlet permettant de tester les fonctionnalités des composants EJB : *ControlesDAOServlet*.
  - o La classe utilitaire qui permet de retrouver les références des composants EJB de la couche métier : classe *ServicesLocator*.
  - o Une classe utilitaire (*Utils*) qui fournit des fonctionnalités pour la manipulation des dates (format `java.util.Date` ↔ `String` : méthodes *string2Date* et *date2String*) et pour la gestion des retours chariot en HTML (elle remplace les `'\n'` en leur équivalent HTML : `'<BR/>'`).

Trois JSP existent à ce stade du projet :

- ***index.jsp*** : page d'affichage du menu.
- ***liste\_entreprises.jsp*** : affiche la liste de toutes les entreprises référencées dans la base.
- ***infos\_entreprise.jsp*** : affiche le détail des informations d'une entreprise donnée.

#### Focus sur le JSP *liste\_entreprise.jsp*

L'examen de ce code source permettra de comprendre le principe de base du codage d'une page JSP qui pourra être reproduit.

- Ligne 1 : on précise que cette JSP est écrite en Java, et que l'encodage des caractères correspond aux langues de l'Europe de l'ouest (*ISO-8859-1*). Cette ligne est à copier/coller lors de la création d'une nouvelle JSP.
- Lignes 3 à 6 : une page JSP contient du code Java. Certaines parties de ce code nécessitent une visibilité et, comme toutes les classes (ce JSP en deviendra une), il est nécessaire de les importer. Toutes les importations sont séparées par une virgule.
- Lignes 8 à 13 : on trouve du code Java. L'écriture de code Java dans une JSP est encadrée par `<% ... %>` (en PHP l'équivalent est `<?php ... ?>`). C'est du code java « classique » : le JSP récupère la référence vers le composant EJB *ServiceEntreprise* (à l'aide de *ServicesLocator*) puis s'en sert pour obtenir la liste des entreprises par l'appel de la méthode *listeDesEntreprises()*. Cette liste est référencée dans la variable *serviceEntreprise* qui a une visibilité sur l'ensemble du JSP. Le code Java encadré par `<% ... code Java ... %>` est appelé un scriptlet.
- Lignes 15 à 32 : début du codage HTML (basique...) de l'interface. Cette partie est le codage statique de la page. À noter :
  - o Ligne 20, le lien vers le fichier CSS qui code la mise en forme de l'interface (couleur de fond, polices de caractères utilisées, mise en forme des tableaux, etc.). Ce fichier CSS vous est fourni, il vous est possible de l'utiliser et éventuellement de le modifier pour avoir une charte graphique cohérente sur l'ensemble de l'application.
  - o Ligne 27 : le tableau affichant la liste des entreprises fait référence à une mise en forme particulière définie dans le fichier CSS (tableau de type *affichage*).
  - o Ligne 27 : le codage HTML d'un tableau débute par `<table>` et se termine par `</table>`. Une ligne du tableau est encadrée par `<tr>` et `</tr>` (pour la ligne d'entêtes des colonnes : `<th> ... </th>`). Une cellule (d'une ligne d'un tableau) est encadrée par `<td>` et `</td>`.

- Ligne 33 : début du code Java qui affiche dynamiquement le contenu de la liste des entreprises. Cette liste, récupérée ligne 12, est parcourue à l'aide d'une boucle. Pour chaque entreprise de la liste, une ligne du tableau HTML est créée (`<tr> ...contenu... </tr>`). L'affichage de chaque information atomique (nom, adresse, etc.) de l'entreprise est réalisé dans une cellule de tableau à l'aide d'une déclaration JSP de la forme `<%= ...contenu à afficher... %>`. Voir, par exemple, l'affichage de l'adresse de l'entreprise ligne 40.
- Lignes 45 à 51 : fin du codage HTML statique de la page.

### 7.4.2 Gestion des formulaires

La saisie des données via l'interface se réalise à l'aide de formulaires HTML. Le support référencé en début de ce document (*cours sur la technologie JSP*) aborde cette technique (transparents 146 et 147).

### 7.4.3 Divers

Le transparent 145 présente un certain nombre d'objets dits *objets implicites*. Ces objets sont des instances pré-déclarées qu'il n'est pas nécessaire d'instancier (elles l'ont déjà été par le conteneur Web). Parmi ces objets implicites, vous aurez éventuellement besoin de :

- Objet **out** : permet d'écrire dans la page. Par exemple : `<% out.println(entreprise.getId()); %>` (ce qui est complètement équivalent à `<%=entreprise.getId()%>`).
- Objet **request** : il représente la requête envoyée au serveur. Son utilisation principale est la récupération des paramètres (envoyés par formulaire ou par URL). Par exemple, la liste des entreprises (JSP *liste\_entreprises.jsp* ligne 39) génère, par entreprise, un lien HTML pour l'affichage des informations détaillées :  
`http://localhost:8080/CabinetRecrutement_WEB/infos_entreprise.jsp?id=1`  
 Dans l'URL, le paramètre est situé après le caractère '?' et se nomme id.  
 La récupération de la valeur (ici 1) se fait dans le JSP *infos\_entreprise.jsp* à la ligne 11 grâce à la méthode `getParameter("nom_du_parametre")` de l'objet request.
- Objet **session** : cet objet permet la gestion des sessions pour un utilisateur. Grâce à cet objet, il est possible de mémoriser des informations, pour un utilisateur donné, entre deux requêtes (deux pages JSP, etc.). Les données stockées peuvent être de toute nature suivant la nature de l'application Web : caddy virtuel pour un site de vente en ligne, bon de commande, etc. Dans notre cas, il sera possible de stocker dans la session l'utilisateur qui se connecte (soit une entreprise soit un candidat). Trois méthodes principales :
  - o `session.setAttribute("nom_de_l_objet_dans_la_session", valeur)`  
 Dépôt de la donnée. La valeur est stockée sous la forme d'une instance d'Object.
  - o `Object contenu = session.getAttribute("nom_de_l_objet_dans_la_session")`  
 L'objet possède le type le plus générique (*Object*). Il faut le transtyper (caster) pour pouvoir l'utiliser. Dans notre cas, stockage de l'utilisateur connecté, si l'objet vaut *null*, c'est qu'il n'y a aucun utilisateur connecté. Il est alors facile de réaliser des tests.
  - o `session.invalidate()`  
 Réinitialisation de la session qui pourra être utilisée dans notre cas pour la déconnection.

Note : un exemple de gestion des sessions dans le corrigé vous est donné sur Moodle : la page JSP gérant les connexion au système (page *connexion.jsp*), la deconnexion (page *deconnexion.jsp*) et un exemple d'utilisation (page affichant l'utilisateur connecté : *bandeau.jsp*).

La page JSP *connexion.jsp* constitue aussi un exemple de gestion de formulaire (formulaire simple de saisie de l'identifiant).





## 8 — Partie 4 : *syndication des informations*

### 8.1 Déroulement

À ce stade du projet, les fonctionnalités principales sont implémentées : la manipulation des informations de l'application (consultation/création d'entreprises, d'offres d'emploi, de candidatures) peut être faite via l'interface Web. En plus de l'accès via cette interface, le cahier des charges (page 14) prévoit une exportation automatique de certaines des informations (*syndication*) qui pourront être exploitées en dehors de l'application : la liste des offres d'emploi et la liste des candidatures. Les fonctionnalités métier nécessaires à cette exportation existent déjà dans certains des composants développés, il suffira donc de les utiliser.

L'exportation de ces informations se fera en utilisant l'un des formats les plus courants disponible actuellement sur le Web : le *format RSS*. Si le sigle est resté inchangé, sa signification a évolué au cours des différentes versions : dans la version actuelle, 2.0, la signification est *Really Simple Syndication*. Le format RSS est une structuration simple des informations à communiquer qui se base sur XML et est largement utilisé. D'autres formats concurrents existent, notamment le format *atom*, mais il ne seront pas abordés ici.

L'objectif de cette séance concerne la manipulation en Java de ce format XML particulier. Elle ne se fera pas en utilisant classiquement un parseur (*SAX*, *DOM*, *JDOM*, etc.) mais avec une approche différente qui utilise le parallèle qui peut être fait entre la structuration des données dans un format XML, exprimée à l'aide d'un schéma, et la structuration en langage Java exprimée à l'aide de classes. La technologie qui supporte cette approche est une technologie livrée en standard avec le JDK : elle se base sur un outil (commande *xjc* pour la génération de classes Java à partir d'un schéma XML) et une API (*JAXB pour Java API for XML Binding*).

#### Pour aller plus loin...

- Exemples de flux RSS :
  - Liste des flux RSS du journal *Le Monde* :  
<http://www.lemonde.fr/web/rss/0,48-0,1-0,0.html>
  - Liste des flux RSS *Radio-France* :  
<http://www.radiofrance.fr/boite-a-outils/rss/>
- Informations sur le format RSS (Wikipedia) : <http://fr.wikipedia.org/wiki/RSS>
- Spécifications du format RSS 2.0 : <http://cyber.law.harvard.edu/rss/rss.html>, traduction française : <http://www.scriptol.fr/rss/RSS-2.0.html>
- Présentation de la technologie : <http://www.jmdoudoux.fr/java/dej/chap-jaxb.htm#jaxb-2>
- Tutorial officiel (Oracle) de la technologie JAXB :  
[http://download.oracle.com/docs/cd/E17802\\_01/webservices/webservices/docs/2.0/tutorial/doc/](http://download.oracle.com/docs/cd/E17802_01/webservices/webservices/docs/2.0/tutorial/doc/)

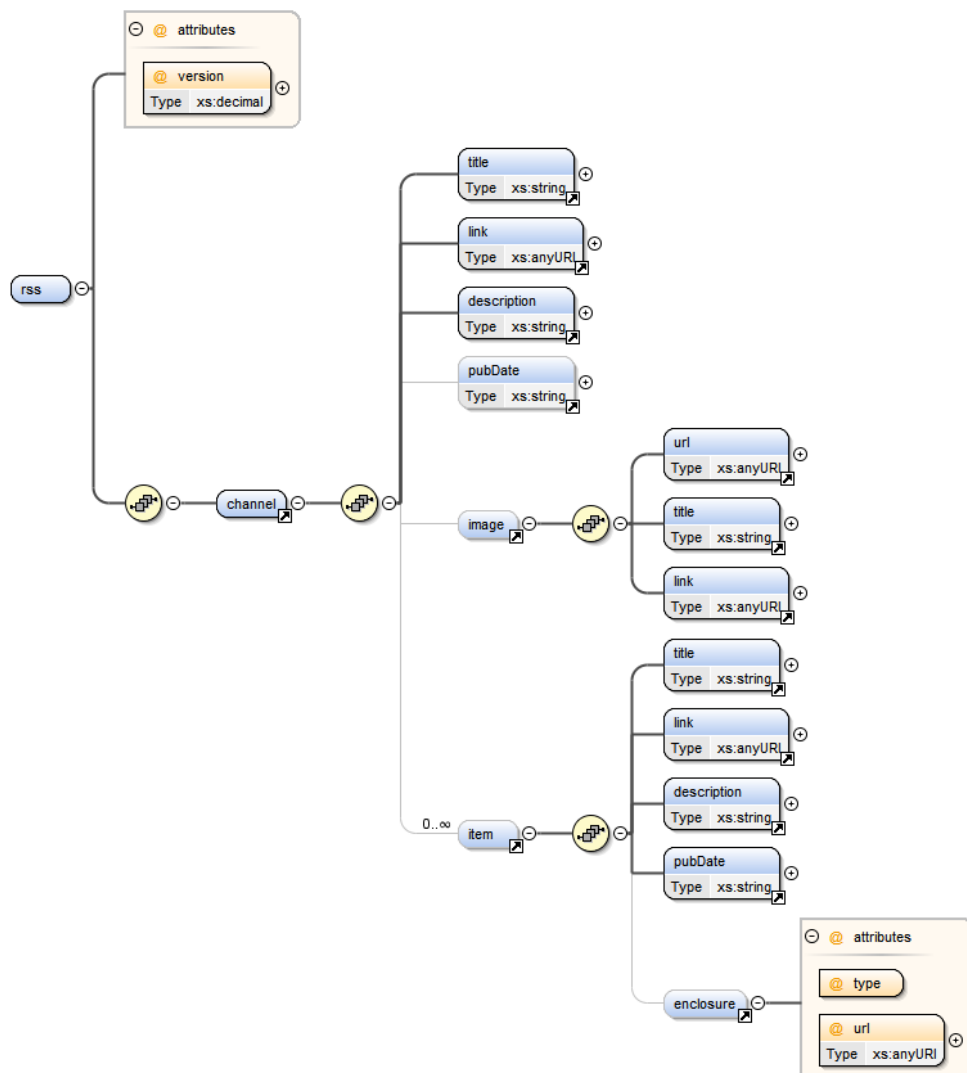
## 8.2 Les technologies mises en œuvre

### 8.2.1 Le format RSS

Le format RSS est donc un moyen de consulter rapidement des informations diverses (actualités, réseaux sociaux, etc.) à l'aide de différents outils qui seront les consommateurs de ces flux bruts pour en assurer la visualisation. Différents types d'outils en permettent la lecture :

- Les lecteurs de flux : *Google Reader*, *Feedshow*, *Thunderbird*, *Firefox*, *Internet Explorer*, etc. Certains d'entre eux sont disponibles sous la formes d'outils accessibles en ligne, d'autres sont des outils à installer.
- Les agrégateurs RSS combinent les informations en provenance de plusieurs flux pour les afficher sous la forme de composants graphiques au sein d'une page Web. Parmi les plus connus : *iGoogle* et *Netvibes*.
- Les informations publiées en RSS peuvent aussi être intégrées directement au sein de certains sites Web.

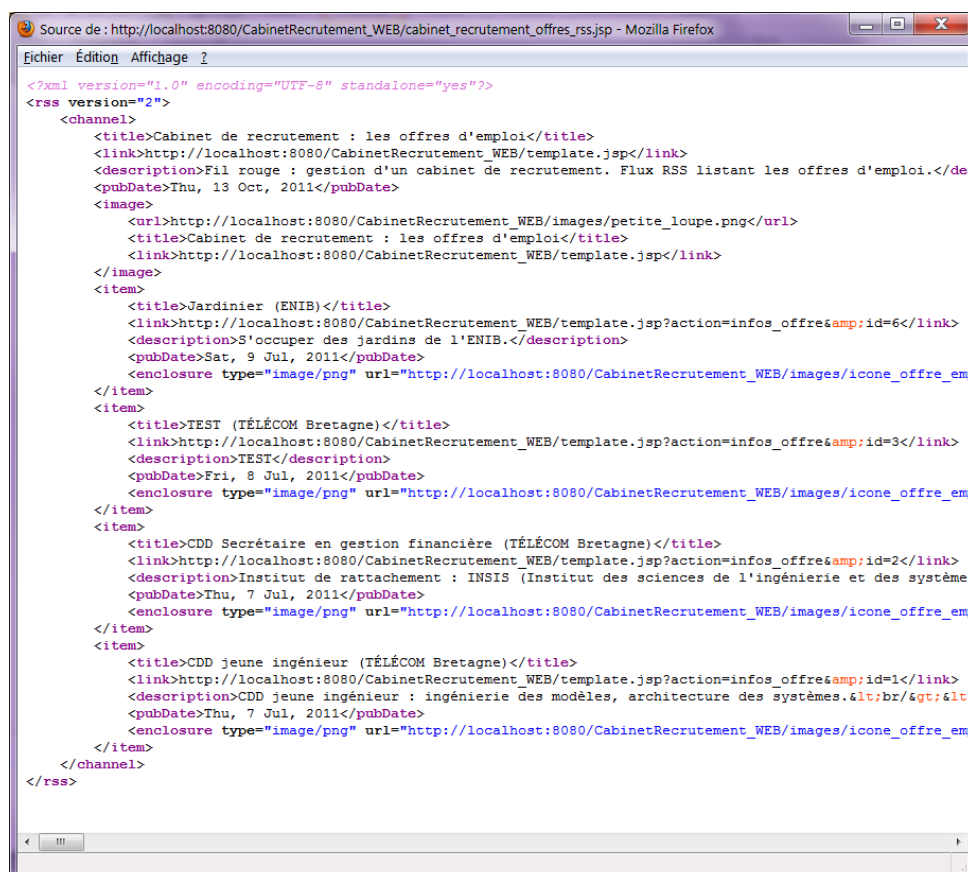
Le format RSS n'est ni standardisé ni normalisé, il existe donc des variations possibles dans le format. Pour la version 2.0, seule une spécification existe (voir le site Web indiqué dans la partie *Pour aller plus loin...*). Pour utiliser JAXB, il est nécessaire de partir d'un schéma XML représentant le format afin de générer les classes. Celui utilisé dans l'application a été généré à partir d'un exemple de flux RSS avec l'outil *Oxygen* : il a été simplifié mais reste totalement compatible avec les lecteurs et agrégateurs RSS. La représentation graphique de ce schéma XML (disponible sur Moodle) montre la structure d'un flux RSS. Des commentaires provenant de la spécification RSS explique la structure du format RSS sont disponibles au sein du fichier.



Représentation graphique du schéma XML décrivant le format RSS



Visualisation du flux dans l'agrégateur Web Netvibes



Exemple de sérialisation XML du flux RSS des offres d'emploi

La racine du document XML est l'élément `rss`, il contient un unique élément : `channel`, qui représente la réelle racine du flux. Cet élément contient trois sous-éléments obligatoires :

- `title` : le titre du flux. Ici, *Cabinet de recrutement : les offres d'emploi*.
- `link` : le lien vers la racine du site. Dans notre cas, la page d'accueil de l'application cabinet de

recrutement.

- *description* : une courte description du flux. Cette description n'est pas toujours utilisée dans les visualisateurs de flux mais elle est néanmoins obligatoire dans la spécification.

Les autres sous-éléments sont optionnels :

- *pubDate* : date de la dernière mise à jour du flux.
- *image* : image représentant le flux. Même présente, elle n'est pas toujours utilisée (*Netvibes* ne s'en sert pas mais *Internet Explorer* la visualise). Si elle est présente, les trois sous-éléments qui la constituent sont obligatoires : *url*, *title* et *link*.
- Une suite d'éléments *item*. Cette suite peut être vide (aucun élément *item* : dans ce cas, le flux n'est pas justifié !). Chaque *item* représente une information. Dans l'application, l'information élémentaire est une offre d'emploi. L'information est, elle-même, structurée dans plusieurs sous-éléments, tous obligatoires sauf l'élément *enclosure* :
  - o *title* : le titre de l'offre.
  - o *link* : le lien qui permettra d'afficher les informations de l'offre (un JSP, *infos\_offre.jsp* par exemple renseigné avec l'id de l'offre à afficher).
  - o *description* : le description de la mission.
  - o *pubDate* : la date de dépôt de l'offre.
  - o *enclosure* : élément optionnel, permet de lier un objet média à l'item. Dans l'exemple, on l'associe à une image (attribut *type* égal à *image/png*) et on indique dans l'attribut *url* le lien pour l'afficher. C'est cette image qui est affichée par *Netvibes* pour chacune des offres.

### 8.2.2 Java API for XML Binding (JAXB)

Les données constitutives du flux sont structurées dans les éléments XML définis dans le schéma. Il aurait été possible de le générer en utilisant classiquement un parseur XML qui aurait récupéré les données auprès d'un des composants métier de l'application pour construire à la volée le flux XML. L'approche utilisée dans cette séance est différente, elle pourrait être comparée (grossièrement...) à la technologie du mapping objet/relationnel où on fait une équivalence entre les tables/colonnes de la base de données sous-jacente et les objets/attributs qui sont manipulés dans l'application. Dans le cas de la technologie JAXB, il est possible de faire une équivalence entre les éléments XML définis dans le schéma et les classes du monde Java.

L'API JAXB est une API standard disponible par défaut dans le JDK. Les classes qui la composent font partie du package `javax.xml.bind`.

Les fonctionnalités de cette API sont doubles :

1. Elle permet de générer un flux XML (ou un fichier, etc.) à partir d'instances de classe : fonctionnalité de **sérialisation** appelée en anglais *marshalling*. Elle produit du XML mais n'en consomme pas. Cette fonctionnalité est celle qui sera utilisée dans l'application.
2. Elle permet aussi la fonctionnalité inverse, créer des instances de classes à partir des données en provenance d'un flux XML : fonctionnalité de **désérialisation** appelée en anglais *unmarshalling*. Cette fonctionnalité ne sera pas utilisée dans l'application.

Via cette API, la représentation des données dans l'application (en Java donc) se fera sous la forme d'instances de classe. La première étape consiste donc à générer ces classes.

Associé à l'API JAXB, un outil est disponible en ligne de commandes : *xjc*. Cet outil est un compilateur qui, à partir d'un schéma XML, génère le code source des classes correspondant aux éléments XML du schéma.

## 8.3 Travail à réaliser

### 8.3.1 Génération des classes

Dans notre cas, à partir du schéma représentant le format RSS seront générées les classes suivantes : *Rss*, *Channel*, *Item*, *Image* et *Enclosure*.

Pour générer les classes :

1. Récupérer sur Moodle l'archive ZIP *RSS.zip*. Un fois décompactée, le schéma XML se trouve dans le fichier *RSS.xsd*.
2. L'utilisation de cet outil se fait au sein d'un terminal. L'option `-p` sera utilisée pour fixer le nom du package des classes générées :

```
xjc -p eu.telecom_bretagne.cabinet_recrutement.front.rss RSS.xsd
```

Le choix du package est laissé à votre appréciation. Le choix fait ici place la gestion du flux RSS dans l'interface graphique (package `eu.telecom_bretagne.cabinet_recrutement.front`) où un sous-package qui lui est dédié est créé : package *rss*.

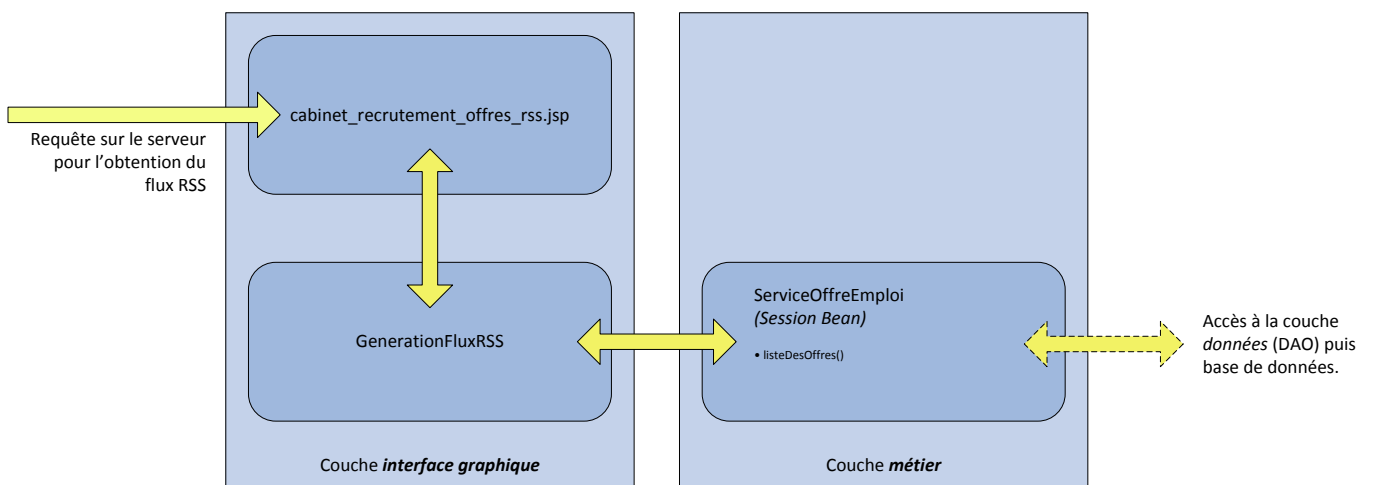
L'outil *xjc* génère alors les fichiers suivants :

- o *eu/telecom\_bretagne/cabinet\_recrutement/front/rss/Channel.java*
- o *eu/telecom\_bretagne/cabinet\_recrutement/front/rss/Enclosure.java*
- o *eu/telecom\_bretagne/cabinet\_recrutement/front/rss/Image.java*
- o *eu/telecom\_bretagne/cabinet\_recrutement/front/rss/Item.java*
- o *eu/telecom\_bretagne/cabinet\_recrutement/front/rss/Rss.java*
- o *eu/telecom\_bretagne/cabinet\_recrutement/front/rss/ObjectFactory.java*

Outre les cinq classes structurant les données dans le flux RSS, une sixième classe est générée : *ObjectFactory*. Cette classe a un rôle utilitaire : elle contient un ensemble de méthodes (*create\**) permettant de créer les instances : *createRss()*, *createItem()*, etc.

3. Intégrez ces classes dans le répertoire *src* du projet *CabinetRecrutement\_WEB*.
4. Examinez le contenu de ces classes pour en comprendre la structure. Cette compréhension est nécessaire pour la suite du travail.

### 8.3.2 Génération du flux RSS



Principe de la génération du flux RSS

Le flux RSS s'obtient en envoyant une requête (HTTP) sur un JSP déployé dans l'application Web : *cabinet\_recrutement\_offres\_rss.jsp*. Ce fichier est disponible dans l'archive *RSS.zip*. La seule action du JSP est d'appeler la méthode *offresEmploi(...)* de la classe *GenerationFluxRSS* dont l'ébauche est aussi disponible dans l'archive. Cette classe récupère les données par l'appel du composant EJB *ServiceOffreEmploi* (dont le nom peut différer dans votre projet). La logique métier existe et est déjà utilisée dans l'application. La fonctionnalité de génération du flux RSS ne fait que de l'utiliser dans un contexte différent.

Le chaînage des différents appels est déjà réalisé, une fois les deux fichiers intégrés dans l'application, le travail se déroulera au sein de la classe *GenerationFluxRSS*.

1. Préparation de l'environnement de travail :
  - o Recopiez le JSP *cabinet\_recrutement\_offres\_rss.jsp* dans le répertoire *WebContent* du projet *CabinetRecrutement\_WEB*.  
À noter : il n'y a pas de passages à la ligne dans ce JSP : **NE PAS EN AJOUTER !** Cela perturberait la génération du flux.
  - o Recopiez la classe *GenerationFluxRSS* dans le même package que les classes générées pas *xjc* : `eu.telecom_bretagne.cabinet_recrutement.front.rss`.
2. Dans la méthode *offresEmploi(...)*, adaptez éventuellement la récupération du composant EJB permettant de récupérer la liste des offres d'emploi.

À ce stade, un flux RSS minimal ne contenant aucune information peut être récupéré. Ce flux n'est pas conforme au schéma mais est correctement détecté par le navigateur Web comme étant un flux RSS. L'accès au flux se fait via l'URL : `http://localhost:8080/CabinetRecrutement_WEB/cabinet_recrutement_offres_rss.jsp`.

Le source XML de son contenu est le suivant :

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<rss version="2">
  <channel/>
</rss>
```

3. Complétez cette méthode : voir les commentaires de la classe pour la réalisation de ce travail.  
Prendre en compte dans un premier temps les éléments obligatoires du format RSS et les éléments optionnels dans un second temps.  
Remarques :
  - o Pour ces images, il est possible d'utiliser celles présentes dans le répertoire *WebContent/images* du projet *CabinetRecrutement\_WEB*.
  - o Pour la gestion des dates, la classe *Utils* (package `eu.telecom_bretagne.cabinet_recrutement.front.utils`) offre une méthode gérant le format de date exigé dans la spécification RSS : méthode *date2StringRSS(Date uneDate)*.
4. Une seconde méthode, *candidatures*, est présente dans la classe. Elle permet de récupérer un flux RSS contenant la liste des candidatures. La visualisation de ce flux nécessite un JSP spécifique qui n'est pas présent et qu'il faut donc créer.



Technopôle Brest-Iroise  
CS 83818  
29238 Brest Cedex 3  
France  
+33 (0)2 29 00 11 11  
[www.telecom-bretagne.eu](http://www.telecom-bretagne.eu)

