# Fil rouge : présentation
# Module FIP INF 211

## 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 SGBD (PostgreSQL).
  - La gestion de la persistance sera confiée au framework de mapping relationnel/objet inclus dans la technologie EJB.
- 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 vue en première année (cycle en V, analyse des besoins, spécifications, etc.) seront utilisés mais sont supposés maîtrisés.

*Merci de communiquer toute correction ou remarque sur les documents du fil rouge à philippe.tanguy@telecom-bretagne.eu*

## Déroulement du projet

Le fil rouge se déroule sur cinq séances (quatre étapes principales) :

1. Séance 1 (13 septembre : 1 h 30) : ***conception des données de l'application***.
   Mise en place + compréhension du sujet.
   Conception du schéma logique et du script SQL de création des tables de la base de données.
2. Séance 2 (20 septembre : 3 h) : ***gestion de la persistance de l'application***.
   Génération des beans entités et développement des composants DAO
3. Séance 3 (27 septembre : 1 h 30) et séance 4 (7 octobre : 3 h) : ***gestion des services métier et de la couche présentation***.
   Développement des composants EJB qui assurent les services métier.
   Le développement de la couche interface utilisateur se fera conjointement.
4. Séance 5 (18 octobre : 3 h) : ***gestion de la syndication des informations***.
   Développement du composant qui assure la génération du flux RSS.
5. Séance 6 (20 octobre : 1 h 30) : ***recette de l'application***.

Il y a donc 13 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, partie 3.3.2.

## Cahier des charges : gestion de cabinet de recrutement

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

Une démonstration possible est disponible à l'URL : http://srv-labs-001.enst-bretagne.fr/CabinetRecrutement_WEB/ si certains dysfonctionnements apparaissent ou si l'ergonomie de l'interface vous parait ambigüe, merci de le signaler.

## 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 3.3.1 *objectifs de délais*.

Note : les livrables serviront à l'évaluation du fil rouge comptant pour 30 % de la note finale du module INF 211.

# *FIP INF 211 - Le fil rouge par rapport aux autres enseignements*

| BD | ARCHI | XML | FIL ROUGE |
|---|---|---|---|
| DATA 1 TP : *Découverte d'un SGBD relationnel*<br>DATA 2 C : *Données structurées et bases de données*<br>DATA 3 PC : *Conception d'une base de données relationnelle*<br>DATA 4 C : *Langages de requêtes relationnels*<br>DATA 5 PC : *Requêtes SQL*<br>DATA 6 TP : *Interfaces programmatiques sur un SGBD relationnel*<br>DATA 7 PC : *Intégration du paradigme relationnel et orienté objet* | | | |
| | | | **FR 1 : *Mise en place de l'environnement et conception des données de l'application* (13/09 1h30)** |
| DATA 8 TP : *Introduction à Hibernate (JPA ?)* | ARCHI 1 C : *Evolution des architectures techniques des SI* | | |
| | | | ***FR 2 : Gestion de la persistance de l'application (20/09 -> 3h00)*** |
| | ARCHI 2 C : *Technologie J2EE*<br>ARCHI 3 TP : *mise en place d'une architecture 3-tier* | | |
| | | | **FR 3 : *Gestion des services métier et de la couche présentation* (27/09 -> 1h30)** |
| DATA 9 C : *Transactions*<br>*DATA 10 TP : Transactions et mises à jour* | ARCHI 4 TP : *Mise en œuvre d'une architecture 3-tier (suite)* | DOC 1 C : *Notions fondamentales de XML*<br>DOC 2 C : *Validation de documents via les schémas XML*<br>DOC 3 TP : *Modélisation de données sous forme de schéma XML* | |
| | | | **FR 4 : *Fin services métier et gestion de la couche présentation* (07/10 -> 3h00)** |
| DATA 11 C : *Normalisation* | | DOC 4 C : *Requêtes XML à travers XPath et XQuery*<br>DOC 5 PC : *Exercices XPath/XQuery*<br>DOC 6 TP : *Exercices XPath/XQuery* | |
| | | | **FR 5 : *Gestion de la syndication des informations* (18/10 -> 3h00)** |
| DATA 12 PC : *Exercices de normalisation* | | | |
| | | | **FR 6 : *Recette* (20/10 -> 1h30)** |
| | | | **Total : 13h30** |

# Cahier des charges
# application « cabinet de recrutement »
# Module FIP INF 211

## Sommaire



## 1. Introduction

### 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. Les objectifs du produit

## 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.Contexte économique du produit

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

# 3. Exigences sur le produit

## 3.1.Capacités Fonctionnelles

### 3.1.1. Description des fonctionnalités

Les différentes fonctionnalités sont organisées par groupes (fonctionnalités liées aux entreprises, aux offres d'emploi, etc.) et, pour chacune d'entre-elles, sont 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 : fonctionnalité optionnelle.

Une présentation synthétique de ces niveaux d'importance est faire dans la partie 4.1.

*Fonctionnalités liées au 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 :<br>• le nom ;<br>• le descriptif de l'entreprise ;<br>• l'adresse postale. Pour simplifier, on considère qu'une entreprise n'occupe qu'un seul emplacement géographique. |
| **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.<br>Comme il n'y a pas d'exigences de sécurité, la gestion de l'authentification avec un mot de passe n'est pas nécessaire.<br>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 :<br>• l'identifiant (de la forme 'ENT_ + numéro de l'entreprise') ;<br>• le nom ;<br>• le descriptif de l'entreprise ;<br>• l'adresse postale. |
| **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 entreprise, niveau de qualification requis et date de dépôt et, par offre, la liste des candidatures potentielles avec un lien vers l'affichage complet.<br>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.<br>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'acquittement 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).<br>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 :<br>• Un candidat destinataire (sélectionné automatiquement à partir de l'offre d'emploi).<br>• Le corps du message.<br>• Une date de dépôt du message renseignée automatiquement. |
| **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 |

| **Nom** | **Référencer une nouvelle offre d'emploi** |
|---|---|
| **Description** | Chaque entreprise peut référencer une nouvelle offre d'emploi qu'elle publie sur le système. |
| **Utilisateur autorisé à accéder à la fonctionnalité** | L'entreprise |
| **Entrées** | Les données à saisir pour caractériser une offre d'emploi sont :<br>• le titre de l'offre ;<br>• le descriptif de la mission ;<br>• le profil recherché par l'entreprise ;<br>• Un ou plusieurs secteurs d'activité ainsi qu'un niveau de qualification exigé : voir fonctionnalités d'indexation. |
| **Sorties** | 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. |
| **Importance** | Vitale |

| **Nom** | **Lister des offres d'emploi** |
|---|---|
| **Description** | 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*. |
| **Utilisateur autorisé à accéder à la fonctionnalité** | Accès à tout utilisateur. |
| **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 | Afficher les informations d'une offre d'emploi |
|---|---|
| **Description** | Affichage des informations spécifiques à une offre d'emploi |
| **Utilisateur autorisé à accéder à la fonctionnalité** | L'accès est possible à tout utilisateur. Des fonctionnalités supplémentaires sont affichées si un utilisateur (entreprise ou candidat) est connecté. |
| **Entrées** | - |
| **Sorties** | Affichage de l'ensemble des données liées à l'offre d'emploi :<br>• le numéro de l'offre ;<br>• le titre ;<br>• la description de l'entreprise propriétaire de l'offre ;<br>• le descriptif de la mission ;<br>• le profil recherché ;<br>• le lieu de la mission (pour simplifier c'est l'adresse de l'entreprise) ;<br>• le niveau de qualification exigé ;<br>• la liste des secteurs d'activité ;<br>• la date de dépôt.<br>Optionnel (les fonctionnalités d'envoi de messages ont une importance mineure) :<br>• 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.<br>• 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). |
| **Importance** | Vitale |

| Nom | Mettre à jour les informations d'une offre d'emploi |
|---|---|
| **Description** | Une fois connectée, l'entreprise a la possibilité de mettre à jour les informations de ses offres d'emploi. |
| **Utilisateur autorisé à accéder à la fonctionnalité** | Seule l'entreprise propriétaire peut modifier les informations de l'offre d'emploi. |
| **Entrées** | Présentation des données de l'offre (numéro, titre, descriptif, profil recherché, niveau de qualification et secteurs d'activité) qui peuvent être modifiées.<br>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 offre d'emploi |
|---|---|
| **Description** | Une offre d'emploi résolue s'opère dans le système par sa suppression, l'offre n'est ainsi plus disponible.<br>La suppression d'une offre implique la suppression en cascade des éventuels messages envoyés et reçus dépendants de celle-ci. |
| **Utilisateur autorisé à accéder à la fonctionnalité** | Seule l'entreprise peut décider de la suppression d'une offre d'emploi dont elle est propriétaire. |
| **Entrées** | - |
| **Sorties** | Message d'acquittement indiquant que l'entreprise a bien été effacée. |
| **Importance** | Mineure |

| **Nom** | **Référencer une nouvelle candidature** |
|---|---|
| **Description** | Le référencement d'une candidature dans le système lui permet par la suite de se connecter. |
| **Utilisateur autorisé à accéder à la fonctionnalité** | Pour des fins de simplification, tout utilisateur du système peut référencer une nouvelle candidature. |
| **Entrées** | Un candidat se caractérise par :<br>• un numéro de candidature ;<br>• un nom ;<br>• un prénom ;<br>• une date de naissance.<br>• une adresse postale.<br>• une adresse électronique.<br>• une date de dépôt.<br>• le CV (potentiellement un document au format PDF mais, pour simplifier, un contenu en texte simple suffira) ;<br>• un ou plusieurs secteurs d'activité ainsi qu'un niveau de qualification exigé : voir fonctionnalités d'indexation. |
| **Sorties** | 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. |
| **Importance** | Vitale |

| **Nom** | **Lister les candidatures référencées** |
|---|---|
| **Description** | Affichage de la liste des candidatures. |
| **Utilisateur autorisé à accéder à la fonctionnalité** | Accès à tout utilisateur. |
| **Entrées** | - |
| **Sorties** | 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. |
| **Importance** | Vitale |

| Nom | Afficher les informations d'une candidature |
|---|---|
| **Description** | Affichage des informations spécifiques à une offre d'emploi |
| **Utilisateur autorisé à accéder à la fonctionnalité** | L'accès est possible à tout utilisateur. |
| **Entrées** | - |
| **Sorties** | Affichage de l'ensemble des données liées à la candidature :<br>• l'identifiant (de la forme 'CAND_ + numéro de l'entreprise') ;<br>• le nom ;<br>• le prénom ;<br>• la date de naissance ;<br>• l'adresse postale ;<br>• l'adresse email ;<br>• le CV ;<br>• le niveau de qualification possédé ;<br>• la liste des secteurs d'activité ;<br>• la date de dépôt. |
| **Importance** | 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.<br>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. |
| **Contraintes** | |
| **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).<br>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 connecté. |
| **Utilisateur autorisé à accéder à la fonctionnalité** | Le candidat |
| **Entrées** | Un message se caractérise par :<br>• une entreprise (sélectionnée automatiquement) ;<br>• une offre d'emploi sur laquelle le candidat veut postuler (sélectionnée automatiquement) ;<br>• le corps du message ;<br>• une date de dépôt du message renseignée automatiquement. |
| **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** | 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 :<br>• Le secteur d'activité.<br>Valeur à choisir parmi : *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*.<br>Note : pour des questions pratiques, cette liste peut être réduite.<br>• Le niveau de qualification.<br>Valeur à choisir parmi : CAP/BEP, Bac, Bac+3, Bac+5, Doctorat.<br><br>Chaque offre et chaque candidature est indexée par un ou plusieurs secteurs d'activité et un unique niveau de qualification.<br>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. |
| **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** | L'identifiant de l'utilisateur qui a été déduit lors de son référencement :<br>• Dans le cas d'une entreprise, l'identifiant est de la forme 'ENT_ + numéro de l'entreprise'.<br>• Dans le cas d'une candidature, l'identifiant est de la forme 'CAND_ + numéro de l'entreprise'. |
| **Sorties** | L'interface de l'application est adaptée au profil :<br>• Le profil (entreprise ou candidature) et le nom de l'utilisateur apparaissent (nom de l'entreprise ou nom/prénom du candidat).<br>• Le menu de l'application s'adapte au profil pour donner accès aux fonctionnalités.<br>• Les fonctionnalités accessibles aux utilisateurs non connectés restent visibles. |
| **Importance** | Vitale |

| **Nom** | **Se déconnecter du système d'information** |
|---|---|
| **Description** | Lorsque l'utilisateur connecté met fin à sa session, l'application se retrouve dans l'état initial. |
| **Utilisateur autorisé à accéder à la fonctionnalité** | Profil entreprise ou profil candidature. |
| **Entrées** | |
| **Sorties** | L'interface de l'application se retrouve dans l'état initial :<br>• Affichage du mode non connecté.<br>• Seules les fonctionnalités accessibles aux utilisateurs non connectés sont visibles. |
| **Importance** | Vitale |

### *Fonctionnalités de syndication de contenu*

| **Nom** | **Exporter automatiquement la liste actualisée des offres d'emploi au sein d'un flux RSS** |
|---|---|
| **Description** | 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. |
| **Utilisateur autorisé à accéder à la fonctionnalité** | Tout utilisateur qui s'abonne au flux. |
| **Entrées** | URL du flux. |
| **Sorties** | Le flux au format RSS qui est affiché au sein de l'agrégateur. |
| **Importance** | Importante |

| Nom | Exporter automatiquement la liste actualisée des candidatures au sein d'un flux RSS |
|---|---|
| **Description** | 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).<br>Les informations disponibles correspondent à la liste de toutes les candidatures. |
| **Utilisateur autorisé à accéder à la fonctionnalité** | Tout utilisateur qui s'abonne au flux. |
| **Entrées** | URL du flux. |
| **Sorties** | Le flux au format RSS qui est affiché au sein de l'agrégateur. |
| **Importance** | Mineure |

## 3.2.Exigences non fonctionnelles

### 3.2.1. 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 :
  - Se connecter au système d'information
- Fonctionnalités accessibles en mode non connecté et à tous les profils (principalement les fonctionnalités d'administration du système) :
  - Référencer une nouvelle entreprise
  - Lister les entreprises référencées
  - Afficher les informations d'une entreprise
  - Lister des offres d'emploi
  - Afficher les informations d'une offre d'emploi
  - Lister les candidatures référencées
  - Afficher les informations d'une candidature
- Fonctionnalités accessibles uniquement aux utilisateurs connectés avec un profil *entreprise* :
  - Mettre à jour les informations d'une entreprise
  - Supprimer une entreprise
  - Référencer une nouvelle offre d'emploi (+ indexation de l'offre)
  - Lister les offres d'emploi spécifiques à une entreprise
  - Mettre à jour les informations d'une offre d'emploi
  - Supprimer une offre d'emploi
  - Envoyer un message à un candidat pour une offre d'emploi donnée
  - Lister les messages reçus des candidats pour une offre d'emploi de l'entreprise
  - Lister les messages envoyés à des candidats pour une offre d'emploi de l'entreprise
  - Se déconnecter du système d'information
- Fonctionnalités accessibles uniquement aux utilisateurs connectés avec un profil *candidature* :
  - Référencer une nouvelle candidature (+ indexation de la candidature)
  - Lister les offres d'emploi qui correspondent à une candidature donnée
  - Mettre à jour les informations d'une candidature
  - Supprimer une candidature
  - Envoyer un message à une entreprise pour une offre d'emploi donnée
  - Lister les messages reçus des entreprises pour une offre d'emploi
  - Lister les messages envoyés à des entreprises pour une offre d'emploi
  - Se déconnecter du système d'information

- Fonctionnalités non liées à un profil (fonctionnalités automatiques) :
    - Exporter automatiquement la liste actualisée des offres d'emploi au sein d'un flux RSS
    - Exporter automatiquement la liste actualisée des offres candidature au sein d'un flux RSS

### 3.2.2. Portabilité

L'application à développer utilisera la technologie Java EE. À ce titre, l'application pourra être déployée sur tout serveur d'applications Java EE compatible. Le serveur choisit pour le développement et les tests est le serveur d'applications open source d'Oracle : GlassFish version 3.1. 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.

## 3.3.Exigences concernant le développement du produit

### 3.3.1. Objectifs de délais

La date de livraison pour le produit final est prévue pour le 24 octobre 2011, soit la dernière semaine du module INF 211. Avant cette livraison finale, différents jalons sont prévus avec dépôt de documents sur Moodle :

- Avant la séance 2 (dépôt avant le 19 septembre, minuit)
    - 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.
    - Le script SQL de création des tables.
- Avant la séance 3 (dépôt avant le 26 septembre, minuit)
    - L'implémentation Java des :
        - beans entités (classes Java qui stockent les données de l'application) ;
        - classes DAO (EJB Java qui assurent les fonctionnalités CRUD).
- Avant la dernière séance, séance 5 (dépôt avant le 19 octobre, minuit)
    - L'implémentation Java des composants EJB codant les services métier.
- Une semaine après la dernière séance (dépôt avant le 27 octobre, minuit)
    - Documentation technique de l'architecture 3-tier mise en œuvre pour l'application.

### 3.3.2. Objectifs de coûts

Le coût horaire est estimé à 60 heures pour le binôme soit 30 heures par personne. Cette durée comprend les 10h30 de séances encadrées.

### 3.3.3. Exigences de réalisation

L'application utilisera obligatoirement les éléments suivants :

- SGBD PostgreSQL 8.4
- Langage Java 1.6
- Serveur d'applications Oracle GlassFish 3.1
- L'environnement de développement sera Eclipse IDE pour développement Java EE version 3.7 (Helios)

# 4. Synthèse des exigences

## 4.1.Hiérarchisation des exigences 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 |
| *Mettre à jour les informations d'une entreprise* | *Mineure* |
| *Supprimer une entreprise* | *Mineure* |
| *Envoyer un message à un candidat pour une offre d'emploi donnée* | *Mineure* |
| *Lister les messages reçus des candidats pour une offre d'emploi de l'entreprise* | *Mineure* |
| *Lister les messages envoyés à des candidats pour une offre d'emploi de l'entreprise* | *Mineure* |

### *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 |
| *Mettre à jour les informations d'une offre d'emploi* | *Mineure* |
| *Supprimer une offre d'emploi* | *Mineure* |

### *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 |
| *Mettre à jour les informations d'une candidature* | *Mineure* |
| *Supprimer une candidature* | *Mineure* |
| *Envoyer un message à une entreprise pour une offre d'emploi donnée* | *Mineure* |
| *Lister les messages reçus des entreprises pour une offre d'emploi* | *Mineure* |
| *Lister les messages envoyés à des entreprises pour une offre d'emploi* | *Mineure* |

### *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 |
| *Exporter automatiquement la liste actualisée des offres candidature au sein d'un flux RSS* | *Mineure* |


# Description de l'environnement de travail

## UV1 – INF301

## Année 2011-2012

M.T. Segarra, P. Tanguy

## Objectifs

L'objectif de ce document est de vous présenter brièvement l'environnement de travail des différentes séances de TP. Ces séances requièrent l'utilisation de plusieurs outils :

- un serveur de bases de données PostGresql et son outil d'administration *PgAdmin*
- l'IDE Eclipse
- un conteneur Web : Tomcat

# 1 Utilisation de la virtualisation

## 1.1 La virtualisation

Afin de vous faciliter le travail d'installation et de configuration nécessaires à l'utilisation des différents outils, vous utiliserez un outil de virtualisation. La virtualisation consiste à faire fonctionner sur un seul ordinateur plusieurs systèmes d'exploitation comme s'ils fonctionnaient sur des ordinateurs distincts.

Plusieurs types de virtualisation existent, mais tous fonctionnent selon un même principe :

- Un système d'exploitation principal (appelé système hôte) est installé sur l'ordinateur et sert de système d'accueil à d'autres systèmes d'exploitation.
- Dans le système d'exploitation hôte, un logiciel de virtualisation (appelé hyperviseur) est installé. Celui-ci crée des environnements clos, isolés, avec des ressources bien précises : ces environnements clos sont appelées des machines virtuelles.
- D'autres systèmes d'exploitation (appelés systèmes invités) peuvent alors être installés dans des machines virtuelles. Leur instance est totalement isolée du système hôte et des autres systèmes invités.

La virtualisation est donc une méthode faisant fonctionner un ou des systèmes d'exploitation invités dans des machines virtuelles, au-dessus d'un système d'exploitation hôte. Pour plus d'informations sur la virtualisation vous pouvez consulter la page wikipédia [1] qui lui est consacrée.

## 1.2 L'outil VMware Player

Dans le cadre de la réalisation des travaux pratiques, vous utiliserez l'outil de virtualisation *VMware* accessible à partir de toutes les machines Linux et Windows de Télécom Bretagne via *VMware Player*. *VMware* est gratuit et propriétaire. Son principe de fonctionnement est celui présenté dans la figure 1 : un environnement virtuel complet simule littéralement un nouvel ordinateur complet, avec du « faux matériel ». À quelques rares exceptions, le système d'exploitation invité ne communique qu'avec ce faux matériel simulé, rendant étanche l'environnement virtualisé.

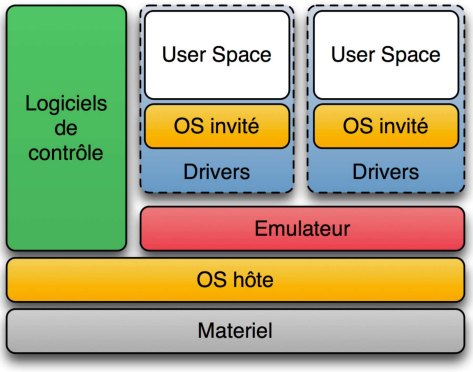



FIGURE 1 – Virtualisation dans *VMware*

# 2 La machine virtuelle VMware pour les travaux pratiques

## 2.1 Les logiciels disponibles

Nous avons préparé une machine virtuelle *VMware* avec un système d'exploitation Ubuntu 11.4 et contenant tous les outils logiciels nécessaires à la réalisation des travaux pratiques :

**Eclipse 3.7 aussi appelé Indigo.** Il se lance à partir du bureau Ubuntu avec l'icône nommée `indigo`. La version installée est adaptée à la modélisation UML via l'outil eUML2 et au développement d'applications Web : gestion de projets Web *dynamiques*, pilotage du conteneur Web, coloration syntaxique pour les fichiers Web...

**Un serveur de bases de données PostgreSQL.** Il s'agit d'un SGBDR *open source*. La version installée est la version 8.4. Outre sa gratuité, l'un de ses avantages est sa proximité avec le SGBDR Oracle concernant la syntaxe du langage SQL. PostgreSQL est livré avec un programme d'administration qui permet d'écrire et de tester des requêtes SQL : *pgAdmin3*. L'icône `pgadmin`

1. `http://fr.wikipedia.org/wiki/Virtualisation`

présent sur le bureau de la machine virtuelle permet de lancer cet outil. À noter une excellente documentation disponible en ligne à l'URL `http://docs.postgresqlfr.org/8.4/` ;

**Un conteneur Web Tomcat v7.** Tomcat est un conteneur Web *open source*. C'est au sein de ce conteneur que sont déployées les *servlets* developées dans le cadre des travaux pratiques.

## 2.2 Caractérisques de la machine virtuelle

Les caractéristiques de la machine virtuelle dont vous devez tenir compte lors de votre travail sont :

- les mises à jour d'Ubuntu sont désactivées. Le système ne pourra donc être mis à jour ;
- la machine virtuelle a accès au réseau. Dans le cas de son utilisation à partir des machines de Télécom Bretagne, cet accès se fait sur le réseau SALSA. Ceci veut dire que tous les 30 min / 1 h, la machine virtuelle se déconnecte du réseau et vous devez vous reconnecter. Attention donc aux téléchargements en cours ou à la connexion éventuelle avec un SGBDR existant en dehors de la machine virtuelle (serveur Oracle de Télécom Bretagne par exemple). Assurez vous que vous avez le temps de connexion nécessaire ;
- pour partager des fichiers ou répertoires entre la machine hôte et la machine virtuelle vous avez deux options :
  - faire du *« drag-and-drop »* entre la machine hôte et la machine virtuelle ;
  - utiliser la notion de *répertoire partagé*. En effet, il est possible de rendre accessibles des répertoires de la machine hôte. Pour ceci, le menu *VM -> Settings -> Options*, *Sharing Folders* puis *Always enabled* permet de configurer le(s) répertoire(s) qu'on souhaite rendre accessible(s) de la machine virtuelle. Une fois accessibles, ces répertoires peuvent être accedés dans la machine virtuelle dans `/mnt/hgfs`.
    Notre machine virtuelle a déjà été configurée pour avoir accès à deux répertoires existant dans toutes les machines Ubuntu de Télécom Bretagne : `/home/`*`votre_login`* qui correspond au répertoire `/mnt/hgfs/Mes Documents` de la machine virtuelle et `/users/local`, qui correspond au répertoire `/mnt/hgfs/Echange` de la machine virtuelle. De plus, pour le premier, une icône `Mes Documents` est aussi disponible sur le bureau de la machine virtuelle.

## 2.3 Principes d'utilisation de la machine virtuelle à Télécom Bretagne

Le principe d'utilisation de la machine virtuelle sur les machines de Télécom Bretagne est divisé en trois étapes :

1. Lancement de l'exécution de la machine virtuelle ;
2. Sauvegarde du travail effectué en séance ;
3. Récupération du travail effectué pendant la séance précédente.

**Lancement de l'exécution de la machine virtuelle.** Pour lancer la machine virtuelle depuis une machine de Télécom Bretagne, vous devez utiliser l'icône `Lancement Machines Virtuelles` disponible sur le bureau de la machine hôte lorsque vous vous connectez sur votre compte. Choisissez la machine virtuelle *Ubuntu_BD_PROG*.

**Sauvegarde du travail effectué sur une machine virtuelle.** Le travail effectué sur la machine virtuelle *Ubuntu_BD_PROG* est sauvegardé dans le répertoire `/users/local/votre_login/Ubuntu_BD_PROG`. Il s'agit d'un répertoire local à la machine hôte. Pour pouvoir utiliser ce travail sur une autre machine de Télécom Bretagne vous devez copier ce répertoire, soit sur votre compte CAMPUS, soit sur une clé USB ou un autre dispositif externe.

**Récupération du travail effectué sur une machine virtuelle.** Pour reprendre le travail sur la machine virtuelle, il suffit de copier le répertoire `Ubuntu_BD_PROG` (copié lors de la séance précédente) dans le répertoire `/users/local/votre_login`. Si ce répertoire n'existe pas, vous devez en créer un. Ensuite, il suffit d'utiliser l'icône *Lancement Machines Virtuelles* disponible sur le bureau de la machine hôte et choisir la machine virtuelle *Ubuntu_BD_PROG*.

## 2.4 Récupération de la machine virtuelle pour travailler sur la machine personnelle

Pour pouvoir utiliser cette machine virtuelle sur vos machines personnelles, il faut d'abord installer *VMware Player*, le logiciel *VMware* qui exécute des machines virtuelles. Il doit s'agir au minimum de la version 3 (dans les salles de TP de Télécom Bretagne il s'agit de la version 3.1.4). Ensuite, suivre la procédure suivante :

- Récupérer les répertoires `/MachinesVirtuelles/Ubuntu_BD_PROG` (on l'appelera par la suite, répertoire *modèle*) et `/users/local/votre_login/Ubuntu_BD_PROG` (on l'appelera par la suite répertoire *delta*), sur le disque de la machine personnelle. Attention, le répertoire *modèle* peut prendre plusieurs gigas ;
- dans le répertoire *delta* sur votre machine personnelle, ouvrez avec un éditeur le fichier avec extension *.vmdk* qui n'a pas de *-sXX* (il s'agit juste d'un fichier texte). Changer la valeur de la variable *parentFileNameHint* pour le faire pointer sur l'emplacement du fichier *.vmdk* du répertoire *modèle* sur votre machine ;
- ensuite à l'utilisation, il faut **toujours** effectuer le lancement de la machine virtuelle à partir du fichier *.vmx* du répertoire *delta*. Sinon, vos modifications ne seront pas prises en compte lors du prochain démarrage de la machine virtuelle.

Pour réutiliser en salle de TP de Télécom Bretagne le travail effectué sur le poste personnel :

- Recopier le répertoire *delta* de votre machine personnelle dans `/users/local/votre_login` de la machine de Télécom Bretagne ;
- modifier le fichier *.vmdk* qui n'a pas de *-sXX* de ce *delta* pour remettre la variable *parentFileNameHint* à sa valeur initiale.

# Fil rouge : instructions enseignants
# Module FIP INF 211

## Présentation globale du fil rouge

Rappeler que l'objectif du FR est un objectif principalement techno. Le travail sera organisé en binômes.
Comme le sujet est nouveau et qu'on ne sait pas la quantité de travail personnel que le projet exigera, il leur est demandé de joindre une un fiche tâche/temps. Un exemple de avec des explications pour la remplir est disponible sur Moodle.

Ils disposent de trois documents imprimés qui sont aussi disponibles sur Moodle. Les deux premiers concernent toutes les séances du fil rouge, ils ont été distribués en début de module.

- Doc (court) de présentation du fil rouge.
  - Rappeler que le fil rouge couvre les trois principaux domaines du module : BD, archi technique et XML avec un focus plus important sur la partie archi. D'ailleurs (cela leur a déjà été dit), la partie archi du FR peut être vu comme un « gros » TP pour cet enseignement. En ce sens, les documents des séances sont relativement détaillés et guident le travail des élèves.
  - Le déroulement du projet a été modifié depuis l'impression des documents pour les élèves, la nouvelle version du document est dispo sur Moodle. La séance consacrée à la *gestion des services métier et de la couche présentation* se déroule maintenant sur deux séances. Cette séance supplémentaire se déroulera le 7/10. 4h30 au lieu de 10h30 pour cette partie soit un total de 13h30 encadrées au lieu des 10h30 prévues initialement. Le travail demandé reste le même.
- Le cahier des charges.
  Le sujet est le développement d'une application Web permettant la gestion d'un cabinet de recrutement.
  La partie principale de ce document est la description complète des fonctionnalités à implémenter. En complément du CdC, une démonstration de l'outil est disponible à l'URL :
  http://srv-labs-001.enst-bretagne.fr/CabinetRecrutement_WEB/template.jsp
  Peut-être une rapide démonstration ?
  Leur dire que chaque fonctionnalité est étiquetée avec un niveau d'importance : vitale et importante (obligatoire), mineure (optionnelle). On peut avoir une note correcte avec uniquement les fonctionnalités obligatoires mais l'implémentation des fonctionnalités optionnelles est un plus.

De façon plus annexe (documents non imprimés, disponibles uniquement via Moodle) :

- La fiche tâche/temps.
- Le document de Mayte sur l'utilisation de la machine virtuelle : *description de l'environnement de travail*.

## Séance 1 : *conception des données de l'application*

Durée, 1h30 : pour les élèves, pas mal de choses à lire et à faire.

- Le document de la séance 1.
  Pas mal de blabla pour expliquer le contexte technique du fil rouge :
  - La machine virtuelle à utiliser : un document écrit par Mayte en explique la manipulation.
  - Au sein de la machine virtuelle, les outils qui seront utilisés : *Eclipse JEE*, *serveur d'applications GlassFish*, *SGBD PostgreSQL*, *pgAdmin* et/ou *Aqua Data Studio*.
  - Un rapide topo sur l'architecture de l'application. Note : cette séance se déroule en amont de la partie ARCHI, ils verront ça plus en détail plus tard mais ils ont ici quelques infos.
  - Un aussi rapide topo sur l'organisation des projets dans Eclipse.

  Le travail technique à réaliser est décrit à partir de la page 7 :
  - Dérivation du schéma conceptuel donné en un schéma logique (ils l'ont déjà fait en PC). Il n'y a pas d'héritage et globalement (il me semble…) pas d'ambigüité non plus.
  - Génération du script de création de la base de données pour PostgreSQL à partir du schéma logique.
  - Quelques éléments syntaxiques pour PostgreSQL sont fournis.

  Le doc se termine par les livrables à fournir avant la séance suivante.

## Séance 2 : *gestion de la persistance des données (JPA)*

Durée, 3h00 : un peu plus de travail de travail technique que dans la séance 1.

Quatre parties dans le document (les deux dernières décrivent le travail à réaliser) :

1. Déroulement.
2. Infos préliminaires.
3. Génération des classes entités.
4. Création des DAO.

Si vous voulez être bavard, il est possible de situer le travail de la séance dans le schéma d'archi présenté dans le document de la séance 1, page 5. Cela situe le travail même si Mayte n'a pas encore abordé les archis n-tier avec eux. Il n'est bien évidement pas nécessaire d'entrer dans les détails.

Le domaine couvert par cette partie est similaire à celui du TP JPA. Attention l'approche est ici inverse de celle du TP : *bottom-up* (les classes sont générées à partir de la base) à la place de *top-down* (le contraire donc…). Leur préciser :

- Une courte explication a été faite dans le doc de la séance 1.
- Le travail est plus « facile » que le TP JPA : pour une bonne compréhension ils ont néanmoins besoin d'analyser la structure des classes ainsi que le mapping.

Partie *informations préliminaires*.

Dans cette partie, la présentation de la config d'*Eclipse -> PostgreSQL* et *Eclipse -> GlassFish* est donnée à titre indicatif. En revanche, pour la config de *GlassFish -> PostgreSQL*, il est important que ce soit compris (j'espère que le petit schéma aidera). L'application, une fois déployée, se sert de cette configuration pour se connecter à la base.

Partie *génération des classes entités*.

RAS, s'ils suivent correctement le déroulement (et que les tables existent dans leur base de données…), ils se retrouvent avec les classes entités et le mapping tout fait ! Si l'informatique était tout le temps comme ça, ils prendraient tous l'option info en 3A !!!

Leur préciser tout de même que le paramétrage choisi est un paramétrage par défaut qui posera ultérieurement des problèmes qu'ils ont commencé à aborder dans le TP JPA : les erreurs liées aux associations LAZY. Celles-ci ne doivent pas apparaître dans la séance. Au cas où, les faire régénérer temporairement les classes entités en choisissant dans

l'écran 3 la valeur *EAGER* dans le paramétrage *Association fetch*. La gestion de ces problèmes sera vu au cours des séances 3 et 4.

Partie *création des DAO*.

Le premier travail consiste à terminer le DAO existant : *EntrepriseDAO*. Les méthodes CRUD à écrire sont expliquées et ils disposent d'infos sur les méthodes de la classe *EntityManager* à utiliser. Ils disposent aussi d'une courte explication du langage de requêtes JPQL (*Java Persistence Query Language*).

Note : certains poseront peut-être la question de savoir pourquoi il n'y a pas ici de code de gestion des transactions comme dans le TP JPA (*tx.begin()*, *tx.commit()*, etc.). Par défaut (c'est le cas ici), lors de chaque appel d'une des méthodes du DAO, une transaction est créée et le code nécessaire encadre alors cet appel. Le *begin* et le *commit* existent mais sont gérés de manière transparente par le conteneur d'EJB. L'annotation suivante ajoutée à chaque méthode *@TransactionAttribute(TransactionAttributeType.REQUIRED)* matérialise le comportement par défaut. Elle n'est pas obligatoire et alourdit la lecture du code.

La suite du travail consiste à développer les autres DAO sur le modèle d'*EntrepriseDAO*. Certains de ceux-ci ont besoin de méthodes particulières que j'ai listées dans le document.

Partie *procédure de contrôle du bon fonctionnement des DAO*.

Pour faire plaisir à Mayte, ☺, on ne parlera pas ici de tests (dans le sens JUnit) mais de bon fonctionnement…

Contrairement au TP JPA, il n'est pas possible de faire une classe exécutable (avec un *main*) qui instancierait un DAO pour en appeler ensuite les méthodes. Un DAO connait son *EntityManager* grâce au mécanisme d'injection de dépendances géré au sein du conteneur d'EJB : c'est le rôle de l'annotation *@PersitenceContext* au-dessus de la déclaration de l'EntityManager (*EntityManager entityManager;*). Il en résulte que dans notre cas, pour appeler une méthode d'un DAO, ceci n'est possible que dans l'application déployée. C'est pourquoi les tests (!!!) se font au sein de la servlet *ControlesDAOServlet*. J'espère que les explications du document accompagnées d'un exemple est suffisant. Je peux passer dans les salles dans le cas contraire.

Pour les livrables, leur dire de ne pas oublier la fiche tâche/temps.

## Séances 3 et 4 : *gestion des services métier et de la couche présentation*

Durée S3, 1h30, durée S4, 3h00.

Comme convenu avec les élèves, la séance 3 peut être vu comme la fin du travail commencé à la séance 2 (classes entité JPA / composants DAO) ou comme la séance du début du travail de cette étape. À priori, je pense que la majorité des élèves choisira de terminer la séance 2.

Sur Moodle, ils disposent pour cette étape de plusieurs choses :

1. Le document de séance : trois parties, voir plus bas.
2. Un document annexe présentant les composants applicatifs du corrigé. Le découpage en composants qui y est proposé n'est donné qu'à titre indicatif. Ils peuvent, ou non, le suivre. S'ils décident de choisir autre chose voir oralement avec eux leur justification et estimer s'ils ne font pas fausse route.
3. Un exemple de gestion de session en JSP pris du corrigé. Voir le doc. de séance.
4. Deux liens indiqués dans le doc. de séance vers des supports sur la technologie HTML et sur la technologie JSP.

Le document de séance est organisé en trois parties qui ne se dérouleront pas forcément de manière temporelle. Je leur recommande plutôt de faire plusieurs itérations. D'autre part, ce document est plutôt un document qui fournit de l'information plutôt qu'un document donnant des instructions.

Partie *poursuite du développement du composant ServiceEntreprise*

Comme pour les DAO, une ébauche de cette couche existe déjà avec la présence du composant EJB *ServiceEntreprise*. La première étape consiste à le compléter. L'essentiel du texte décrit l'état actuel du composant. La bonne compréhension de ce qui est déjà fait devrait permettre la poursuite du travail.

Partie *ajout de nouveaux composants*

L'essentiel de cette partie consiste à décrire l'utilisation de l'assistant de création d'un composant EJB (bean session). C'est queque chose qu'ils ont déjà fait avec les composants DAO. La seule différence ici est que les composants de la couche métier doivent avoir une interface distante.

L'autre sous-partie parle du référencement des nouveaux composants au sein de la classe *ServicesLocator*.

Partie *la couche présentation : l'interface graphique de l'application*

Dans cette partie sont données des informations sur la technologie JSP (un lien vers une présentation rapide de la techno est donné en début de document). Pour introduire la techno, je commente la page JSP *liste_entreprises.jsp* qui est représentative de l'essentiel des pages JSP qu'ils auront à fournir, au moins pour celles qui réalisent de l'affichage de données obtenues auprès de la couche métier. RAS sur ce contenu à priori.

Les points durs sur la partie IHM seront sans doute :

1. La gestion des formulaires Web. Si ce point vous pose un problème, je peux passer dans les salles.
2. La gestion des sessions HTTP. J'ai déposé sur Moodle un exemple d'une gestion de sessions tiré du corrigé. Là encore, si besoin, je peux passer dans les salles pour des explications.

Problème non référencé dans le document : exceptions sur les associations marquées à LAZY

Un problème qui sera certainement troublant pour eux, que je n'ai pas documenté et sur lequel il tomberont à coup sûr…

Dans la génération des classes entité, le choix qu'il leur était demandé de faire sur un paramètre appelé *Fetch type* était *default*. Ce choix se traduit par du *EAGER* sur les association de cardinalité 1 (pas de problème pour y accéder) mais *LAZY* sur les cardinalité multiples (erreurs à prévoir…).

L'erreur qui remonte avec *EclipseLink* (rappel, le provider JPA du projet fil rouge n'est pas *Hibernate* et l'erreur est donc différente) est de la forme :

```
Exception Description: An attempt was made to traverse a relationship using indirection that had a null Session.
This often occurs when an entity with an uninstantiated LAZY relationship is serialized and that lazy relationship
is traversed after serialization.  To avoid this issue, instantiate the LAZY relationship prior to serialization.
```

Pour gérer ça plusieurs solutions :

1. Régénérer les classes entité en choisissant *EAGER* pour le paramétrage du Fetch type. Dans ce cas, toutes les associations du mapping sont positionnées à *EAGER*. Plus aucun problème de *LAZY* mais leur faire pointer les difficultés potentielles : gestion de la mémoire (effet spaghettis quand on ramène une seule instance), perte de performance avec une multitude de requêtes SQL sur la base pour pouvoir tout ramener, etc. Note : même si ce n'est pas une bonne solution, c'est celle que j'ai choisi pour le corrigé (!). Sur de petites quantités de données, c'est gérable…
2. Gérer, par association, le choix EAGER. Note, Eclipse fournit un outil qui permet ce paramétrage (qui ne fait que modifier le mapping de l'association). Accès : menu *Window* > *Show View*. Choisir *JPA* puis *JPA Details*. Gestion du mapping du type presse-bouton, pas mal.
3. Ajouter un *join fetch* dans la requête JPQL qui ramènera les instances.
4. Ne pas utiliser les interfaces distantes. En effet quand une association est à LAZY si le consommateur du service et le service en lui-même sont dans le même conteneur, le contenu LAZY est renseigné dynamiquement et l'erreur LAZY disparait. Il faut dans ce cas dans le *ServicesLocator* récupérer les composant EJB par la classe et non par l'interface distante. Ce qui est possible ici et ce qui serait d'ailleurs le choix à préconiser.

Suivant le niveau de compréhension des élèves (+ faible vers - faible), on peut aller du choix 1 (voire 4) au choix 2 (voire 3).

## Séance 5 : *gestion de la syndication des informations*

Durée 3h.

La durée de la séance est largement suffisante pour effectuer le travail demandé. Deux flux RSS sont demandés dans le cahier des charges : le flux permettant l'accès à la liste des offres d'emploi (initialement obligatoire, devenu optionnel) et celui donnant accès à la liste des candidatures (optionnel). Le second flux se gère exactement de la même manière que le premier sans difficulté technique additionnelle.

Le travail de cette séance est complètement satellite du travail principal du projet. L'objectif est double :

1. Montrer que les fonctionnalités métier développées précédemment sont « factorisables » et peuvent donc être utilisées dans un autre contexte. La liste des offres d'emploi sert dans l'interface Web de l'application, c'est la même fonctionnalité qui est utilisée par le gestionnaire de flux RSS. Ce gestionnaire aurait pu être déployé dans une application complètement indépendante. Pour des questions pratiques ce n'est toutefois pas le cas.
2. Le deuxième objectif est lui plus technologique : montrer la relative facilité de gestion de contenu XML à l'aide d'outils de plus haut niveau que les outils dont ils ont entendu parler en cours (parseurs XML : SAX, DOM, etc.).

Une petite réflexion pourrait être faite avec les groupes les plus avancés qui est liée à l'objectif 2 : de manière plus générale, il pourrait être intéressant de leur faire sentir que malgré la relative complexité de l'application qu'ils développent, le code qu'ils doivent produire est relativement concis.

Par exemple :

- Dans l'application ils ont développé un certain nombre de composants EJB qui peuvent être accédés à distance (RMI et/ou WebService), dans un contexte transactionnel, etc. Hors complexité algorithmique interne, quasiment inexistante ici, la gestion logistique de l'application est restreinte : quelques annotations tout au plus. Le reste est du code Java, simple, tout ce qui a de plus classique.
- La gestion d'un flux XML valide par rapport à un schéma n'est pas forcément une chose aisée à développer si on passe par des outils de plus bas niveau. Ici, après une génération automatique de classes, construire un flux RSS relativement complexe dans sa structure ne demande qu'une trentaine de lignes de code !

L'idée est de montrer qu'on est capable de faire des choses compliquées (*très*) facilement à condition d'avoir une maîtrise des environnements techniques, des frameworks, des API, etc. Plus que les avantages syntaxiques d'un langage par rapport à un autre (Python est plus concis que Java…), le plus important reste « l'écosystème » qui l'entoure. La complexité de l'application est autant, sinon plus, une complexité liée à l'environnement technique (serveur JEE) qu'au code de l'application.

<u>Déroulement de la séance</u>

Le travail est organisé en deux parties avec un support verbeux (un peu trop ?) avec lequel ils devraient être relativement autonomes :

1. Comprendre comment est structuré un flux RSS. Pour ça ils ont accès au schéma XML et à sa représentation graphique dans le document de séance. Ils peuvent aussi examiner le code source d'un des flux de l'application exemple disponible. Pas de complexité importante dans cette partie, il suffit de comprendre l'imbrication des éléments RSS.
2. Développer leur gestionnaire de flux RSS en se servant de la technologie JAXB. Le travail se résume à compléter deux méthodes (une par flux) dans une classe dont l'ébauche est fournie. Je vous fournis dans un document à part le code de cette classe qui fera office de correction.

## Séance 6 : *recette de l'application*

Durée 1h30.

La fiche recette de l'application du fil rouge est construite à partir du cahier des charges et vise à vérifier si les fonctionnalités ont été implémentées ou non.

Remarques :

- Les fonctionnalités sont listées avec leur niveau d'importance. Rappel : vitale et importante = obligatoire, mineure = optionnelle.
- Une fonctionnalité peut être réalisée ou non, si elle l'est, la fiche demande si la prise en compte de certaines contraintes est réalisée.
- Il y a certainement des doublons : par exemple, la vérification de la fonctionnalité de connexion existe et en plus on demande de vérifier qu'une fonctionnalité est accessible à un compte particulier. On fait ce qu'on peut...
- Je n'ai pas mis dans la fiche la fonctionnalité d'indexation des offres et des candidatures, c'est en fait une sous-partie du référencement d'une offre ou d'une candidature. La correspondance offre(s)/candidature(s) fait l'objet d'une fonctionnalité présente dans la fiche.

Déroulement proposé

La fiche est donnée en début de séance, ils ont 10 minutes pour la remplir. Ensuite, à raison de 6 binômes par groupe et 10 minutes par binôme, il reste 10 minutes de rab... un peu court !

Modalités de correction

Ce que j'aimerais prendre en compte :

- la réalisation des fonctionnalités obligatoires.
- Un bonus pour l'investissement dans le travail (curiosité, intérêt, etc.)
- Le respect, dans l'implémentation de l'application, de l'architecture proposée.
  On peut leur demander oralement une explication sur la structure de leur code se basant sur une fonctionnalité particulière.

Exemples de notation

- Toutes les fonctionnalités obligatoires (et uniquement celles-ci) + investissement ultra minimal + respect très moyen de l'archi → 10.
- Uniquement les fonctionnalités obligatoires + bon investissement + quelques accrocs (traitements faits dans les JSP par exemple) → 12, 13, 14.
- Toutes les fonctionnalités + très bon investissement + respect de l'architecture → + de 16.

Mon échelle part ici de la note 10, bien évidemment on peut avoir des cas particuliers avec des notes inférieures. Par ailleurs, je pense que le projet est un excellent moyen de les faire progresser et mettre des notes un peu « serrées » présente un risque de démotivation pour les années à venir qui irait à l'encontre de ce qu'on essaie de faire... on peut en discuter.

# Fil rouge : fiche tâche/temps
# Module FIP INF 211

Associée à chaque livrable, la fiche tâche/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.

## *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.

## *Exemple de fiche tâche/temps :*

**Nom 1 / Nom 2**

| **Numéro** | **Date** | **Étape** | **Objectifs** | **Temps passé** | **Séance / temps perso.** |
|---|---|---|---|---|---|
| 1 | 13/09/2011 | Conception des données | Réalisation du schéma logique. | 1 h | Séance |
| 2 | 13/09/2011 | Conception des données | Réalisation du script de création de la base de données | 1 h | Séance |
| … | … | … | … | … | … |
| Total | | | | 3 268 h | |

# Fil rouge : séance 1
# Module FIP INF 211

## Conception des données de l'application

### Déroulement

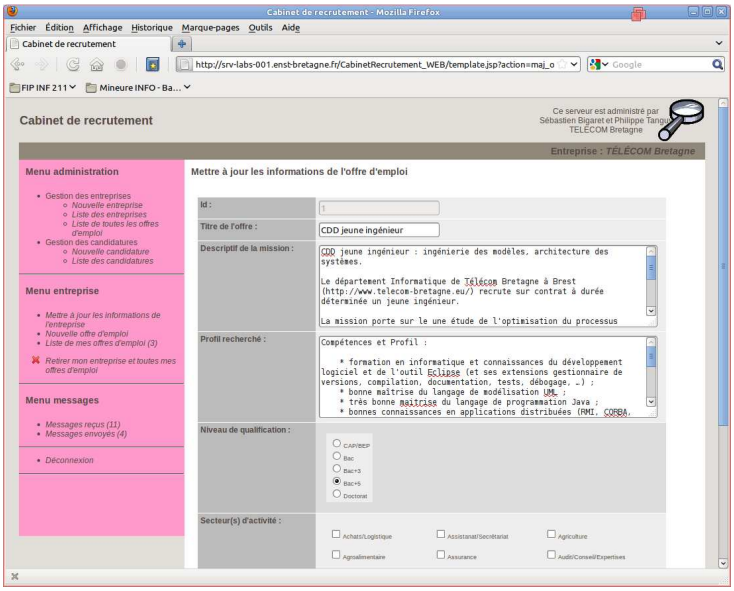
Cette première séance du fil rouge se divise en deux étapes principales :

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.

# Introduction : compréhension du sujet, mise en place de l'environnement de travail et présentation sommaire de l'architecture.

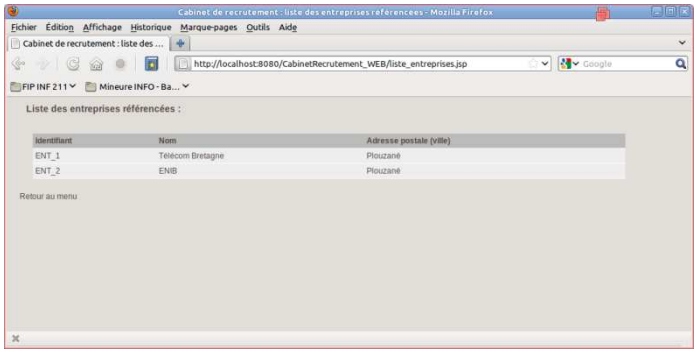
## *L'application*

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'ambigüité 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 15 du cahier des charges liste les fonctionnalités par catégories et par ordre d'importance : vitale et importante (fonctionnalités obligatoires) et mineure (fonctionnalités optionnelles).




*http://srv-labs-001.enst-bretagne.fr/CabinetRecrutement_WEB/*

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://localhost:8080/CabinetRecrutement_WEB/liste_entreprises.jsp*

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 11.4* est installé. Pour lancer la machine virtuelle, double-cliquez sur le fichier *Ubuntu.vmx* situé dans le répertoire *Ubuntu_BD_PROG*. Une fois le système Linux lancé, connectez-vous avec l'utilisateur ayant pour login ***user*** et pour mot de passe ***password***. Cette machine virtuelle sert à la fois au fil rouge ainsi qu'aux TP de l'enseignement *architectures applicatives*.

Voir le document *description de l'environnement de travail* pour de plus amples informations sur la machine virtuelle utilisée.

Les outils à votre disposition :

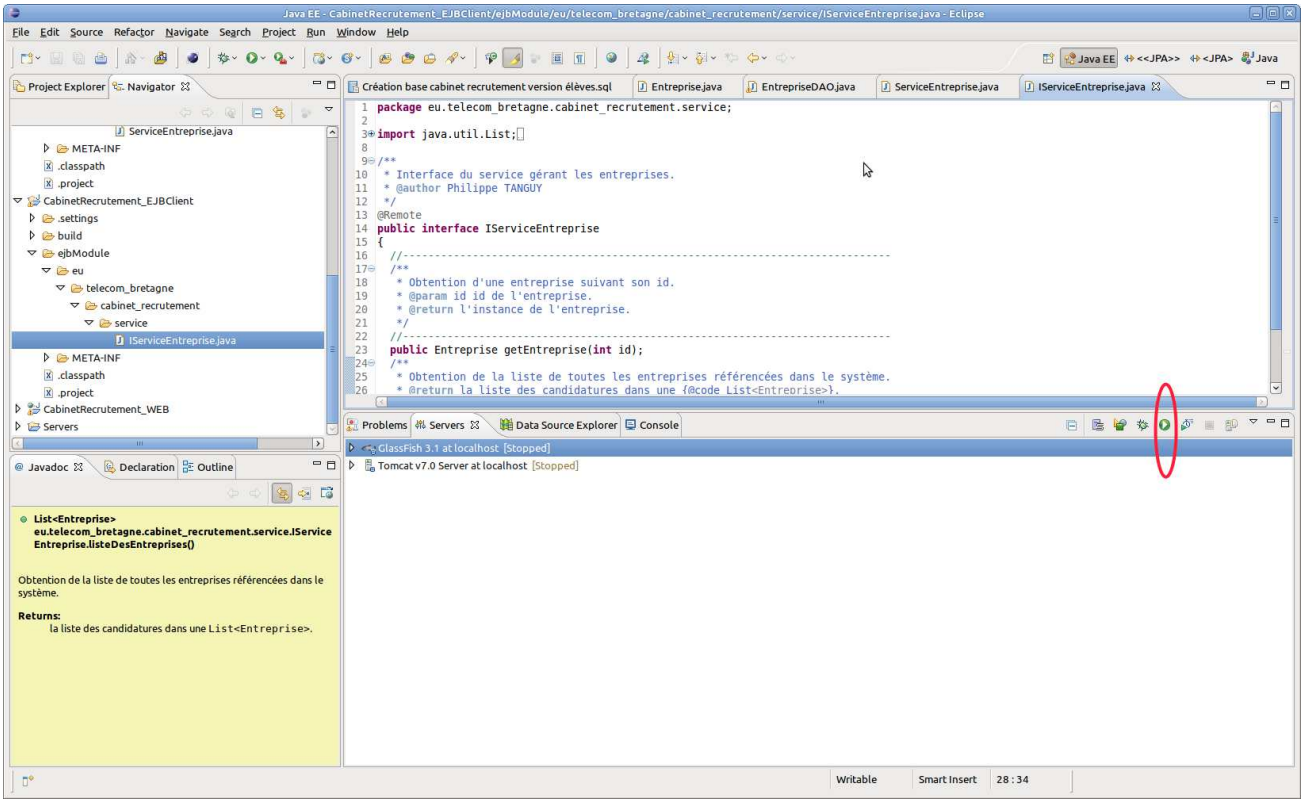
| | |
|---|---|
| ***EDI Eclipse*** | Package Eclipse utilisé : *Eclipse Java EE IDE for Web developers - Version Indigo*<br><br>Eclipse est l'un des environnements de développement intégré (IDE) les plus populaires actuellement. C'est un outil professionnel largement utilisé. Il se lance à partir du bureau Ubuntu avec l'icône nommée *Eclipse - Indigo*.<br>Cette version est adaptée au développement d'applications de type Java EE (*Enterprise Edition*). Elle comprend un ensemble d'outils additionnels adaptés à cette utilisation : gestion des projets JEE, gestion de connexions à des SGBD, pilotage de serveurs d'applications (GlassFish dans notre cas), coloration syntaxique pour les fichiers Web (HTML, CSS, etc.), les scripts SQL, etc. |
| ***Serveur d'applications GlassFish*** | GlassFish Server Open Source Edition 3.1.1<br><br>GlassFish est un serveur d'applications compatible Java EE 6 qui existe en deux produits : l'un open source (*GlassFish Server Open Source Edition* utilisée ici) et l'autre commercial (*Oracle GlassFish Server*) non utilisé ici. C'est au sein de ce serveur qu'est déployée notre application. Il offre différentes fonctionnalités : serveur Web, infrastructure d'exécution de composants logiciels, etc.<br><br>Dans notre cas, le pilotage du serveur (démarrage/arrêt) se fera par l'intermédiaire d'Eclipse. Une fois le serveur lancé, une interface d'administration Web est disponible à l'URL : http://localhost:4848/common/index.jsf |
| ***SGBD PostgreSQL*** | Le SGBD PostgreSQL est un SGBD open source. La version installée est la version 8.4. Outre sa gratuité, l'un des ses avantages est sa proximité avec le SGBD Oracle concernant la syntaxe du langage SQL.<br><br>PostgreSQL est livré avec un programme d'administration qui permet d'écrire et de tester des requêtes SQL : *pgAdmin3*. L'icône *PgAdmin* présente sur le bureau permet de lancer cet outil.<br><br>Un autre outil est installé pour une utilisation similaire : *Aqua Data Studio*, icône *ADS 4.7*.<br><br>À noter, une excellente documentation disponible en ligne à l'URL : http://docs.postgresqlfr.org/8.4/ |

Lancez Eclipse (icône *Eclipse - Indigo*). Le code de l'application est réparti sur quatre projets qu'il faut ouvrir séparément. Pour importer un projet dans Eclipse : menu *File* > *Import…*, choisir *Existing Projects into Workspace*, bouton *Next >*. Choisir alors le répertoire principal du projet. Les quatre projets se trouvent dans le répertoire : */home/user/workspace*. Ils se nomment :

- *CabinetRecrutement_EAR*
- *CabinetRecrutement_EJB*
- *CabinetRecrutement_EJBClient*
- *CabinetRecrutement_WEB*

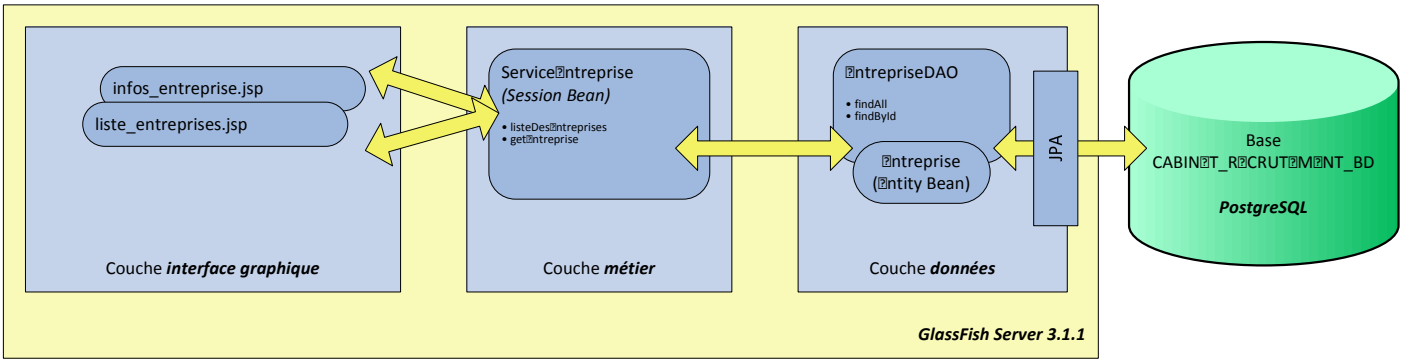
L'interface d'Eclipse est divisée en plusieurs parties :

- Sur la gauche apparaît la liste des projets dans l'onglet *Navigator* ou *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 3.1 at localhost*) :
    1. Ajoutez le projet au serveur : clic droit > *Add and Remove…*, l'application *CabinetRecrutement_EAR* 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, Republish]*. Il est alors possible de lancer l'application dans un navigateur Web à l'URL : 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*

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 SGBD 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 pgAdmin3 permet la connexion à la base, il est alors possible de l'interroger :
  - Utilisateur : *cabinet_recrutement_role*
  - Mot de passe : *abc*

  ***La séance 1 du fil rouge est centrée sur la base de données.***
- L'application au dessus de la base de données qui est hébergée au sein du serveur GlassFish, écrite suivant la technologie Java EE.

Cette application respecte une architecture répartie sur trois niveaux (*3-tier*) :

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.

   ***La séance 2 est centrée sur le développement de cette couche.***
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 3 et 4 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 :
  - *dao* : classes DAO
  - *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.

## *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 :

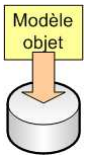
- Un projet de type *Enterprise Application Project* (***CabinetRecrutement_EAR***) :
  C'est un projet « conteneur » qui ne contient pas directement le code de l'application mais qui référence les trois autres sous-projets. Il produit, pour le déploiement de l'application un fichier *Enterprise Archive* ou *EAR* : ce type de projet s'appelle aussi *EAR Project*.
- Le codage de la logique métier de l'application se fait au sein de deux projets. Le premier est un projet type *EJB Project* (***CabinetRecrutement_EJB***) et le second est un projet automatiquement ajouté au premier lors de la création de celui-ci : projet de type *EJB Client Project* (***CabinetRecrutement_EJBClient***).
  - 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 *CabinetRecrutement_EJBClient* contient une partie du code de la couche métier : les interfaces (au sens Java) des classes Session Beans.
- 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).

## Génération de la base de données

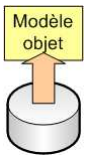
Les données persistantes de l'application connaissent des états différents :

- 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 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.

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

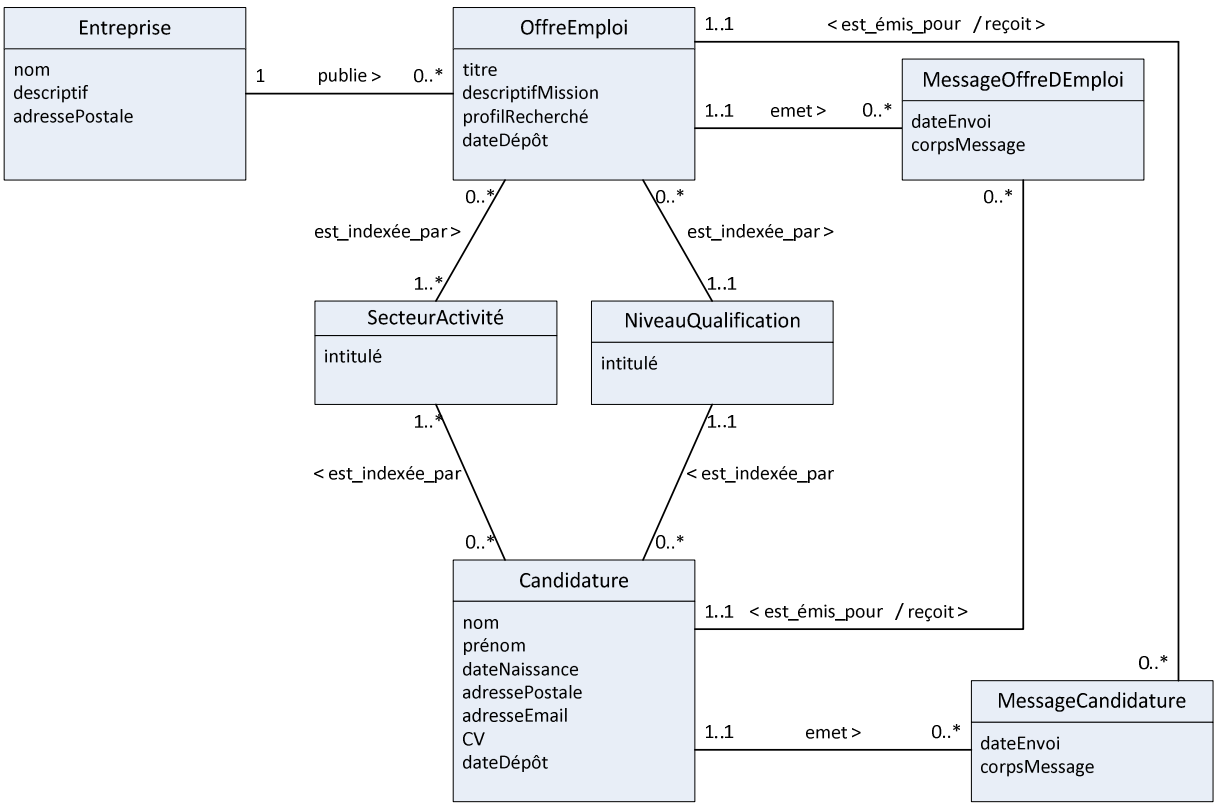



*Schéma conceptuel des données de l'application*

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.

## *Travail à réaliser*

Le processus à réaliser ici est similaire au travail réalisé au cours de la PC *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.
- 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 tous les objets de la base (tables, etc.) 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 rejouerez plusieurs fois le script. Les tables existantes mais modifiées doivent être détruites avant d'être recréées. 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, meilleur choix, *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.

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 :
  - *select * from entreprise* est équivalent à *SELECT * FROM ENTREPRISE*
  - *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_table" (
        colonne_1 type ,
        colonne_2 type ,
        …
) ;
```

- 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*). 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"*

  Les doubles quotes sont optionnelles.
- 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 :
  - *serial* : type entier (*integer*) qui permet la génération automatique d'une séquence
  - *integer* : type entier sur 4 octets (largement suffisant pour la dimension de notre système d'information…
  - *varchar(n)* : chaîne de *n* caractères max
  - *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.
  - *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 2011.

## Livrables

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 (dépôt avant le 19 septembre). 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 (je me répète…).

Les modalités pour le dépôt du fichier vous seront communiquées ultérieurement.

# Fil rouge : corrigé séance 1
# Module FIP INF 211

## Conception des données de l'application

### Génération de la base de données

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

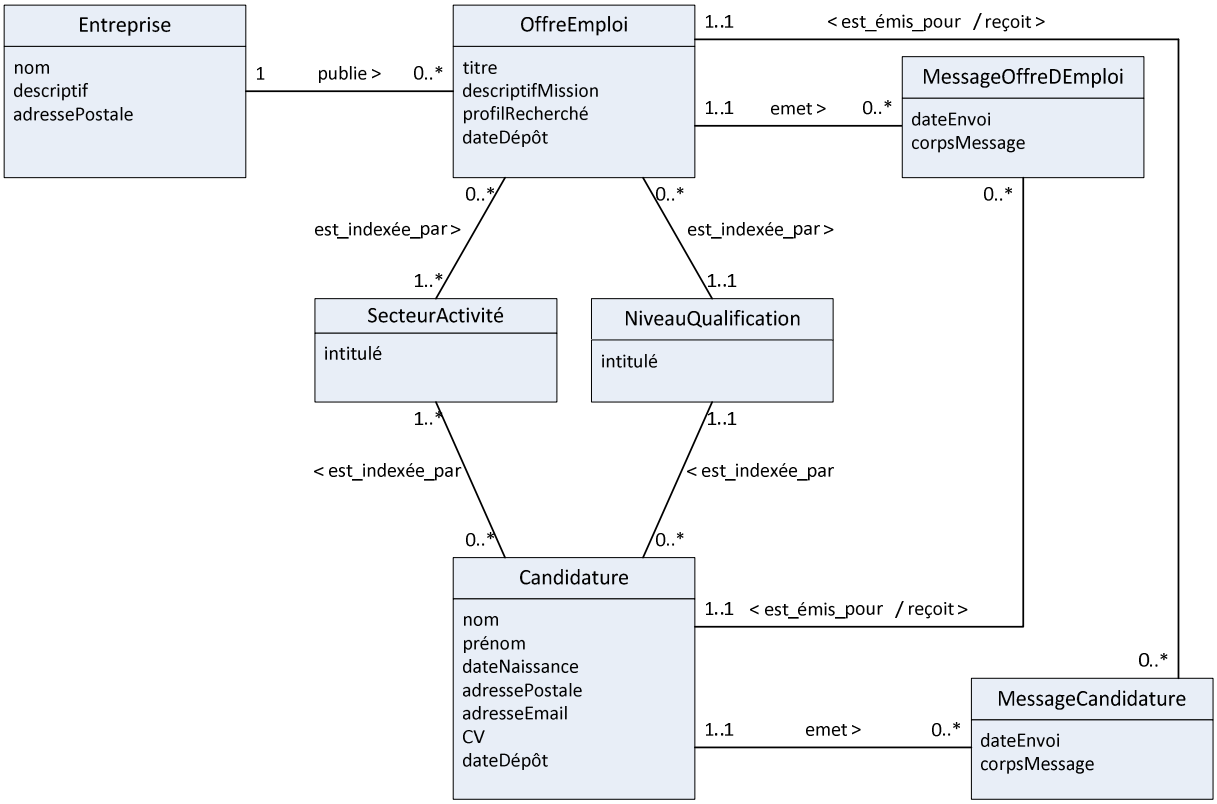



*Schéma conceptuel des données de l'application*

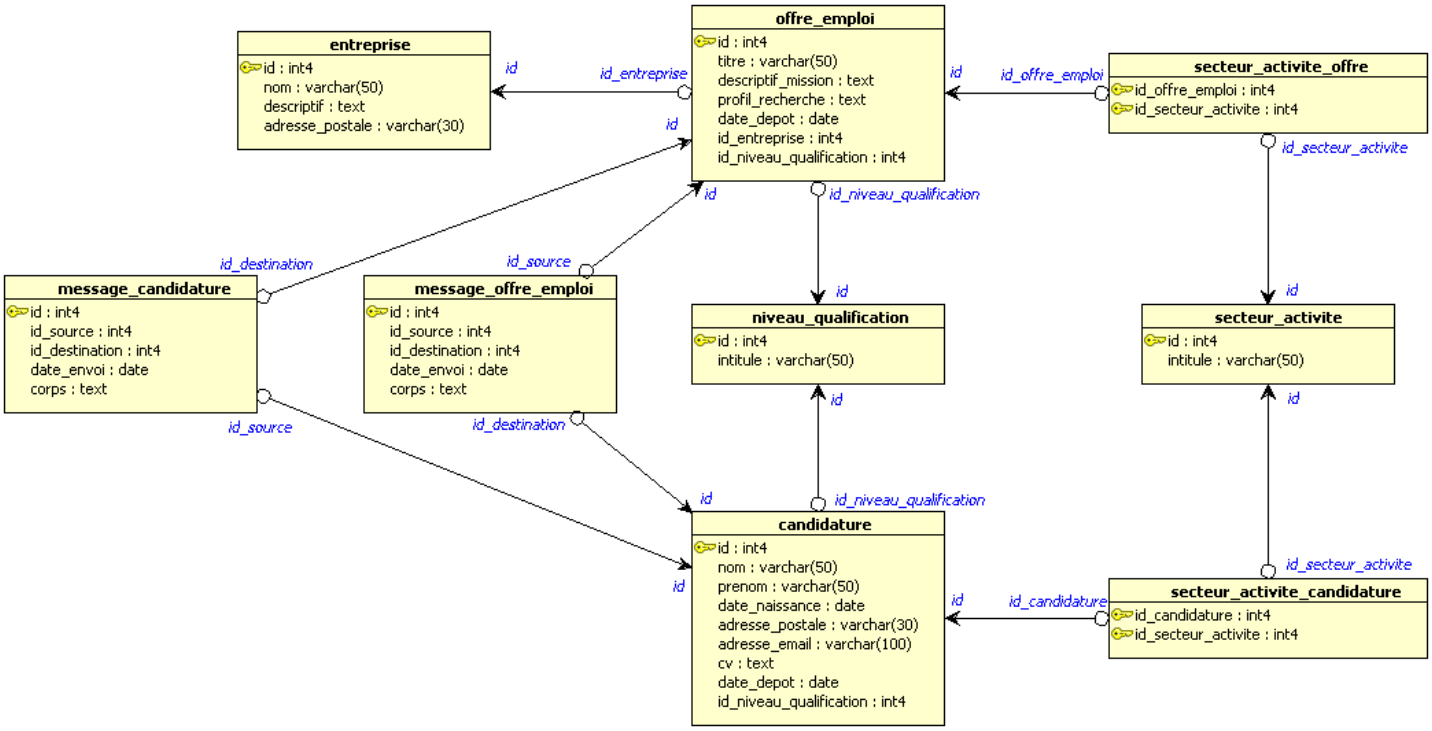
Les données de ce schéma conceptuel sont issues du cahier des charges. En cas de besoin, s'y référer pour une meilleure compréhension.

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.

## *Travail à réaliser*

Le processus à réaliser ici est similaire au travail réalisé au cours de la PC *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.

```
-- Titre :             Création base cabinet recrutement V2.sql
-- Version :           0.1
-- Date création :     28 juin 2011
-- Date modification : 16 juillet 2011
-- Auteur :            Philippe Tanguy
-- Description :       Script de création de la base de données pour le SI "gestion de cabinet de
--                     recrutement"
--                     Note : script pour PostgreSQL 8.4
--                     Version 2 : cette version prend en compte la gestion des demandes
--                     d'entretien entre les entreprises et les candidats (messages) et
--                     l'obsolescence des candidatures et des offres d'emploi quand celles-ci ne
--                     sont plus d'actualité.

-- +-----------------------------------------------------------------------------------------+
-- | Suppression des tables                                                                  |
-- +-----------------------------------------------------------------------------------------+

drop table if exists "message_offre_emploi";
drop table if exists "message_candidature";
drop table if exists "secteur_activite_offre";
drop table if exists "secteur_activite_candidature";
drop table if exists "offre_emploi";
drop table if exists "entreprise";
drop table if exists "candidature";
drop table if exists "secteur_activite";
drop table if exists "niveau_qualification";
```

```
-- +-----------------------------------------------------------------------------------------------+
-- | Création des tables                                                                           |
-- +-----------------------------------------------------------------------------------------------+

create table "secteur_activite"
(
  id       serial primary key,
  intitule varchar(50) not null
);

create table "niveau_qualification"
(
  id       serial primary key,
  intitule varchar(50) not null
);

create table "entreprise"
(
  id              serial primary key,
  nom             varchar(50) not null,
  descriptif      text,
  adresse_postale varchar(30) -- Pour simplifier, adresse_postale = ville.
);

create table "offre_emploi"
(
  id                      serial primary key,
  titre                   varchar(50) not null,
  descriptif_mission      text,
  profil_recherche        text,
  date_depot              date,
  id_entreprise           integer not null references "entreprise",
  id_niveau_qualification integer not null references "niveau_qualification"
);

create table "secteur_activite_offre"
(
  id_offre_emploi     integer not null references "offre_emploi",
  id_secteur_activite integer not null references "secteur_activite",
  primary key (id_offre_emploi,id_secteur_activite)
);

create table "candidature"
(
  id serial primary key,
  nom varchar(50) not null,
  prenom varchar(50),
  date_naissance date,
  adresse_postale varchar(30), -- Pour simplifier, adresse_postale = ville.
  adresse_email   varchar(100),
  cv              text,
  date_depot      date,
  id_niveau_qualification integer not null references "niveau_qualification"
);

create table "secteur_activite_candidature"
(
  id_candidature      integer not null references "candidature",
  id_secteur_activite integer not null references "secteur_activite",
  primary key (id_candidature,id_secteur_activite)
);

create table "message_offre_emploi"
(
  id             serial primary key,
  id_source      integer not null references "offre_emploi",
  id_destination integer not null references "candidature",
  date_envoi     date,
  corps          text not null
);

create table "message_candidature"
(
  id             serial primary key,
  id_source      integer not null references "candidature",
  id_destination integer not null references "offre_emploi",
  date_envoi     date,
  corps          text not null
);
```

```
-- +-----------------------------------------------------------------------------------------------+
-- | Insertion de quelques données de pour les tests                                               |
-- +-----------------------------------------------------------------------------------------------+

-- Insertion des secteurs d'activité

insert into "secteur_activite" values (nextval('secteur_activite_id_seq'),'Achats/Logistique');                 --  1
insert into "secteur_activite" values (nextval('secteur_activite_id_seq'),'Assistanat/Secrétariat');            --  2
insert into "secteur_activite" values (nextval('secteur_activite_id_seq'),'Agriculture');                       --  3
insert into "secteur_activite" values (nextval('secteur_activite_id_seq'),'Agroalimentaire');                   --  4
insert into "secteur_activite" values (nextval('secteur_activite_id_seq'),'Assurance');                         --  5
insert into "secteur_activite" values (nextval('secteur_activite_id_seq'),'Audit/Conseil/Expertises');          --  6
insert into "secteur_activite" values (nextval('secteur_activite_id_seq'),'BTP/Immobilier');                    --  7
insert into "secteur_activite" values (nextval('secteur_activite_id_seq'),'Commercial');                        --  8
insert into "secteur_activite" values (nextval('secteur_activite_id_seq'),'Communication/Art/Média/Mode');      --  9
insert into "secteur_activite" values (nextval('secteur_activite_id_seq'),'Comptabilité');                      -- 10
insert into "secteur_activite" values (nextval('secteur_activite_id_seq'),'Direction Générale/Executive');      -- 11
insert into "secteur_activite" values (nextval('secteur_activite_id_seq'),'Distribution/Commerce');             -- 12
insert into "secteur_activite" values (nextval('secteur_activite_id_seq'),'Electronique/Microélectronique');    -- 13
insert into "secteur_activite" values (nextval('secteur_activite_id_seq'),'Environnement');                     -- 14
insert into "secteur_activite" values (nextval('secteur_activite_id_seq'),'Finance/Banque');                    -- 15
insert into "secteur_activite" values (nextval('secteur_activite_id_seq'),'Formation/Enseignement');            -- 16
insert into "secteur_activite" values (nextval('secteur_activite_id_seq'),'Hôtellerie/Restauration/Tourisme');  -- 17
insert into "secteur_activite" values (nextval('secteur_activite_id_seq'),'Industrie/Ingénierie/Production');   -- 18
insert into "secteur_activite" values (nextval('secteur_activite_id_seq'),'Informatique');                      -- 19
insert into "secteur_activite" values (nextval('secteur_activite_id_seq'),'Juridique/Fiscal/Droit');            -- 20
insert into "secteur_activite" values (nextval('secteur_activite_id_seq'),'Marketing');                         -- 21
insert into "secteur_activite" values (nextval('secteur_activite_id_seq'),'Public/Parapublic');                 -- 22
insert into "secteur_activite" values (nextval('secteur_activite_id_seq'),'Ressources Humaines');               -- 23
insert into "secteur_activite" values (nextval('secteur_activite_id_seq'),'Santé/Social/Biologie/Humanitaire'); -- 24
insert into "secteur_activite" values (nextval('secteur_activite_id_seq'),'Télécom/Réseaux');                   -- 25

-- Insertion des niveaux de qualification

insert into "niveau_qualification" values (nextval('niveau_qualification_id_seq'),'CAP/BEP');   --  1
insert into "niveau_qualification" values (nextval('niveau_qualification_id_seq'),'Bac');       --  2
insert into "niveau_qualification" values (nextval('niveau_qualification_id_seq'),'Bac+3');     --  3
insert into "niveau_qualification" values (nextval('niveau_qualification_id_seq'),'Bac+5');     --  4
insert into "niveau_qualification" values (nextval('niveau_qualification_id_seq'),'Doctorat');  --  5
```

# Fil rouge : séance 2
# Module FIP INF 211

## Gestion de la persistance des données (JPA)

### Déroulement

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, etc.).
- La création de 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*).

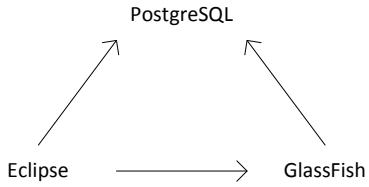
#### *Pour aller plus loin…*

- 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 :
  - Partie 1 : http://deptinfo.unice.fr/~grin/mescours/minfo/modpersobj/supports/jpa2-1-6.pdf
  - 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.

# 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 mis en œuvre. 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.




## *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 outil 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).

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

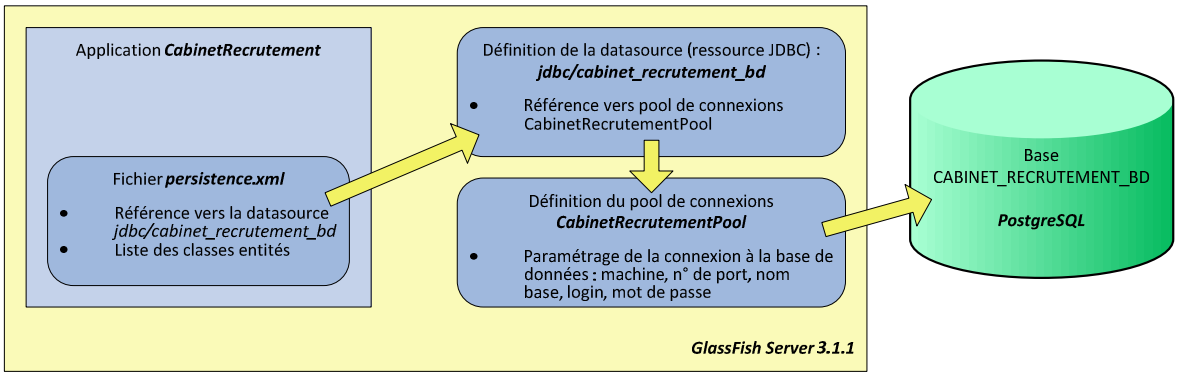
Le paramétrage de l'accès au serveur GlasshFish, outre son pilotage (lancement/arrêt), permet un déploiement automatique de l'application dès qu'une modification est apportée. Lors de la modification d'un des fichiers source, Eclipse le compile automatiquement. GlassFish détecte la modification de la(des) 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 3.1 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.

## *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:8080/CabinetRecrutement_WEB/ 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-9.0-801.jdbc4.jar*, est déposé dans le répertoire */opt/glassfish-3.1/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 connexions 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 *persistence.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>*.

## 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 7 du document de la séance 1.

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 (mapping) avec les structures de la base (les tables…) 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***
  - Sélectionnez la connexion : *PostgreSQL (localhost) – CABINET_RECRUTEMENT_BD*.
  - Activez la connexion (bouton *Connect*).
  - 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.
  - 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) :
  - L'attribut de jointure.
  - La navigabilité entre les classes qu'il est possible de (dé)sélectionner. Essayez de comprendre ce qu'implique cette notion de navigabilité.
  - Le nom des attributs qui représente les associations.
  - Bouton *Next >*.
- Page 3 : ***Customize Default Entity Generation***
  Différents réglages liés à la génération des classes entités sont opérés ici.
  - Méthode de génération des valeurs de clés primaires, *Key generator*. PostgreSQL utilise des séquences, valeur qu'il faut choisir dans la liste déroulante.
  - *Sequence name* : saisir le pattern $table_$pk_seq qui correspond au nom des séquences générées automatiquement par PostgreSQL.
  - *Entity access* : *Field*
  - *Associations fetch* : *Default*
  - *Collection properties type* : conservez *java.util.Set*.
  - *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***
  - 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 séance 1.

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;
```

## 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.

### *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 assuré 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* donné.
- *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 connait 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()*.
- *void* ***merge****(java.lang.Object entity)*
  Met à jour dans la base les modifications apportée à une instance de classe entité.
- *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 ambigüe…

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*

## *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*
  - 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*
  - Ajout d'une méthode pour l'obtention de la liste des offres d'emploi pour une entreprise donnée.
  - 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*
  - 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.
  - 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*
    - 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***
    - Saisir le nom de package de la classe : *eu.telecom_bretagne.cabinet_recrutement.data.dao*.
    - Saisir le nom de la classe.
    - 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.

## 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 (un 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.

## Livrables

Le code des classes développées, classes entités et classes des DAO, sera à déposer sur Moodle avant la séance 3 du fil rouge (dépôt avant le 26 septembre). 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 des binômes.

# Fil rouge : séance 2 – Requêtes JPQL
# Module FIP INF 211

## Gestion de la persistance des données (JPA)

### 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;
}
```

# Fil rouge : corrigé séance 2
# Module FIP INF 211

## Gestion de la persistance des données (JPA)

### Génération des classes entités.

Le nom des classes générées et le nom des attributs dépendent du nom des tables et des colonnes.

À titre d'information, le code de la classe *Entreprise* un peu arrangé pour en facilité la lecture :

```
package eu.telecom_bretagne.cabinet_recrutement.data.model;

import java.io.Serializable;
import javax.persistence.*;
import java.util.Set;

/**
 * The persistent class for the entreprise database table.
*/
@Entity
public class Entreprise implements Serializable
{
  //---------------------------------------------------------------------------
  private static final long serialVersionUID = 1L;

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

  private String nom;

  private String descriptif;

  @Column(name = "adresse_postale")
  private String adressePostale;

  // bi-directional many-to-one association to OffreEmploi
  @OneToMany(mappedBy = "entreprise")
  private Set<OffreEmploi> offresEmploi;
  //---------------------------------------------------------------------------
  public Entreprise() {}
  //---------------------------------------------------------------------------
  public Integer          getId()             { return this.id;             }
  public String           getNom()            { return this.nom;            }
  public String           getDescriptif()     { return this.descriptif;     }
  public String           getAdressePostale() { return this.adressePostale; }
  public Set<OffreEmploi> getOffresEmploi()   { return this.offresEmploi;   }
  //---------------------------------------------------------------------------
  public void setId(Integer id)                                 { this.id             = id;             }
  public void setNom(String nom)                                { this.nom            = nom;            }
  public void setDescriptif(String descriptif)                  { this.descriptif     = descriptif;     }
  public void setAdressePostale(String adressePostale)          { this.adressePostale = adressePostale; }
  public void setOffresEmploi(Set<OffreEmploi> offresEmploi) { this.offresEmploi   = offresEmploi;   }
  //---------------------------------------------------------------------------
}
```

# Création des DAO

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

```java
package eu.telecom_bretagne.cabinet_recrutement.data.dao;

import java.util.List;

import javax.ejb.LocalBean;
import javax.ejb.Stateless;
import javax.persistence.EntityManager;
import javax.persistence.PersistenceContext;
import javax.persistence.Query;

import eu.telecom_bretagne.cabinet_recrutement.data.model.Entreprise;

/**
 * Session Bean implementation class EntrepriseDAO
 * @author Philippe TANGUY
 */
@Stateless
@LocalBean
public class EntrepriseDAO
{
  //-------------------------------------------------------------------------
  @PersistenceContext
  EntityManager entityManager;
  //-------------------------------------------------------------------------
  /**
   * Default constructor.
   */
  public EntrepriseDAO()
  {
    // TODO Auto-generated constructor stub
  }
  //-------------------------------------------------------------------------
  public Entreprise persist(Entreprise entreprise)
  {
    entityManager.persist(entreprise);
    return entreprise;
  }
  //-------------------------------------------------------------------------
  public Entreprise findById(Integer id)
  {
    return entityManager.find(Entreprise.class, id);
  }
  //-------------------------------------------------------------------------
  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;
  }
  //-------------------------------------------------------------------------
  public Entreprise update(Entreprise entreprise)
  {
    entityManager.merge(entreprise);
    return findById(entreprise.getId());
  }
  //-------------------------------------------------------------------------
  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'EntityManager.

    // L'entité était déjà attachée ou a été rattachée, on peut donc la supprimer...
    entityManager.remove(entreprise);
  }
  //-------------------------------------------------------------------------
}
```

Les requêtes JPQL dans les différents DAO :

À noter, les requêtes JPQL sont ici des requêtes paramétrées.

**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;
}
```

**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;
}
```

# Fil rouge : séances 3 et 4
# Module FIP INF 211

## Gestion des services métier et de la couche présentation

### 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[1] 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-001.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.

[1] *Le contenu Web dynamique s'oppose au contenu Web statique : un page Web écrite en HTML a un contenu fixe qui n'évolue pas au contraire d'une page Web dynamique dont le contenu peut évoluer au cours du temps. C'est le cas ici : la liste des entreprises dépend du contenu de la base au moment de la consultation.*

## 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 4.
- ***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 charge page 3.

### *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.
- ***@WebService***
  Cette annotation est présente dans le code source mais aurait dû être supprimée. Elle n'est que pour tester la fonctionnalité. Elle permet de transformer le bean en un Web Service sans ajouter une seule ligne de code. Comme cette annotation est présente, il vous est possible de tester le Web Service grâce au testeur de GlassFish à l'URL : http://localhost:8080/ServiceEntrepriseService/ServiceEntreprise?Tester

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.

Par choix, il a été décidé que le bean *ServiceEntreprise* puisse être accédé de façon distante pour de futures évolutions (si vous voulez tester ☺…). Ceci n'est pas obligatoire dans notre configuration de déploiement, l'interface Web comme la couche service sont hébergées au sein du même GlassFish. Les méthodes du bean sont donc présentes au sein de l'interface (dans le sens langage Java) distante dont le code source se trouve dans le projet *CabinetRecrutement_EJBClient*.

L'interface et la classe concrète partagent le même package : *eu.telecom_bretagne.cabinet_recrutement.service*. Le nom de l'interface est : *IServiceEntreprise*.

### *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.

## Ajout de nouveaux composants

### *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 :
  - Le nom du package : *eu.telecom_bretagne.cabinet_recrutement.service*
  - Le nom de la classe : la convention utilisée (à suivre, ou pas…) jusqu'à présent fait précéder le nom du bean par *Service*.
  - *State type:* laissez à *Stateless*.
  - 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*.
  - 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.

### *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.

## 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/glassfish3.1.1/glassfish/domains/domain1/generated/jsp/CabinetRecrutement_EAR/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 ».

### *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* :
  - Le code de la Servlet permettant de tester les fonctionnalités des composants EJB : *ControlesDAOServlet*.
  - La classe utilitaire qui permet de retrouver les références des composants EJB de la couche métier : classe *ServicesLocator*.
  - Un 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 *scriplet*.
- Lignes 15 à 32 : début du codage HTML (basique…) de l'interface. Cette partie est le codage statique de la page. À noter :
  - 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.
  - 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*).
  - 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.

## *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 aborde cette technique (transparent 146 et 147).

## *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_d_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 :
  - *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*.
  - *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.
  - *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).

# Fil rouge : annexes séances 3 et 4
# Module FIP INF 211

## Gestion des services métier et de la couche présentation

Ce document présente les composants applicatifs de la couche service de l'application *cabinet de recrutement*. Le contenu de ce document n'est donné qu'à titre indicatif, d'autres choix de conception tout aussi valables auraient pu être fait. Ce qui est présenté ici correspond à l'exemple de l'application qui est déployé à l'URL : http://srv-labs-001.enst-bretagne.fr/CabinetRecrutement_WEB/.

Outre les composants et leurs fonctionnalités associées, sont aussi présentés :

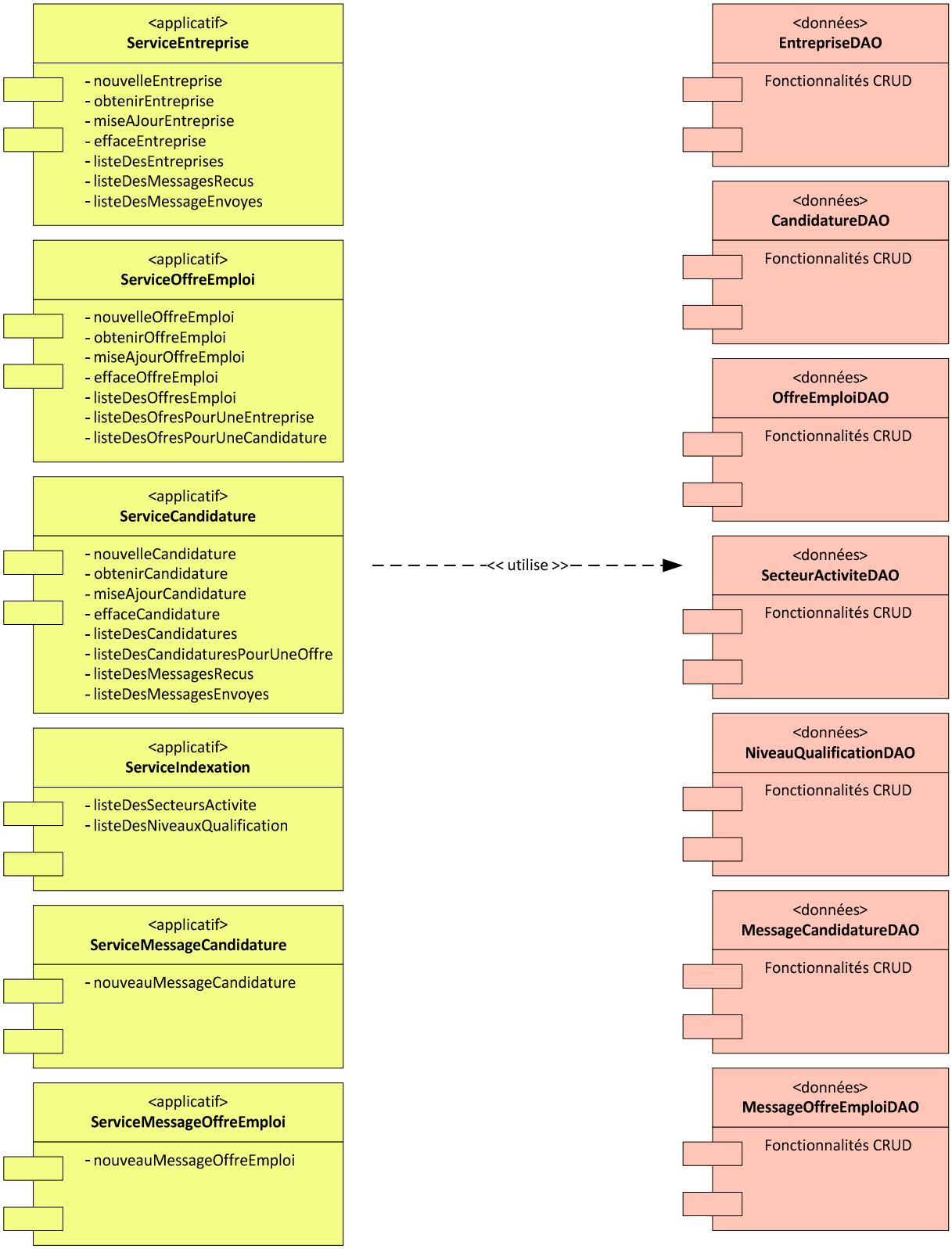
- Un exemple de diagramme de séquences permettant de définir les relations de dépendances entre les composants applicatifs et les composants DAO.
- La liste des dépendances de chacun des composants applicatifs avec les composants DAO.

### *Pour aller plus loin…*

- Site Web en français présentant de manière synthétique UML (*Unified Modelling Language*) : http://uml.free.fr/

## Composants applicatifs

Attention, le diagramme présenté ici est un diagramme de *composants* et non un diagramme de *classes*. Même si ce type de diagramme est déjà orienté réalisation, on se situe à un niveau plus abstrait. La structure interne de chacun de ces composants peut être d'un ensemble de classes qui interagissent entre elles. Dans l'application du fil rouge, cette structure interne est simple : chacun des composants présentés ici n'est constituée que d'une seule classe et chacune des fonctionnalités correspond donc à une méthode de la classe réalisant le composant.




*Ensemble des composants de l'application cabinet de recrutement*

Remarque : les fonctionnalités de ces composants sont globalement simples. Beaucoup de celles-ci ne sont que des encapsulations des fonctionnalités CRUD assurées par les composants DAO.

## Diagramme de séquences

À titre d'exemple est présenté ici le diagramme de séquence d'une des fonctionnalités les plus évoluées de l'application : *récupération de la liste des candidatures correspondant à une offre d'emploi* du composant applicatif *ServiceCandidature*. À partir de ce diagramme, il est aisé de définir les dépendances (stéréotypées *« utilise »*) qui seront présentées plus loin dans le document.

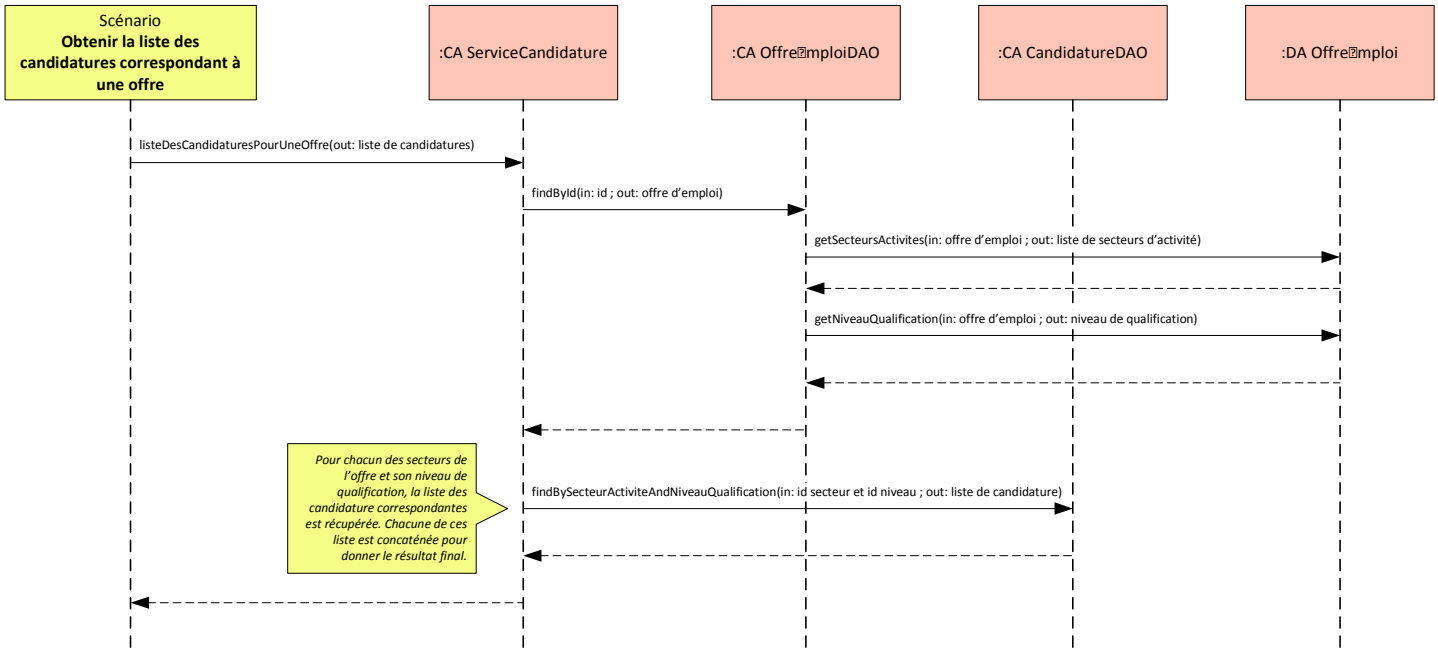
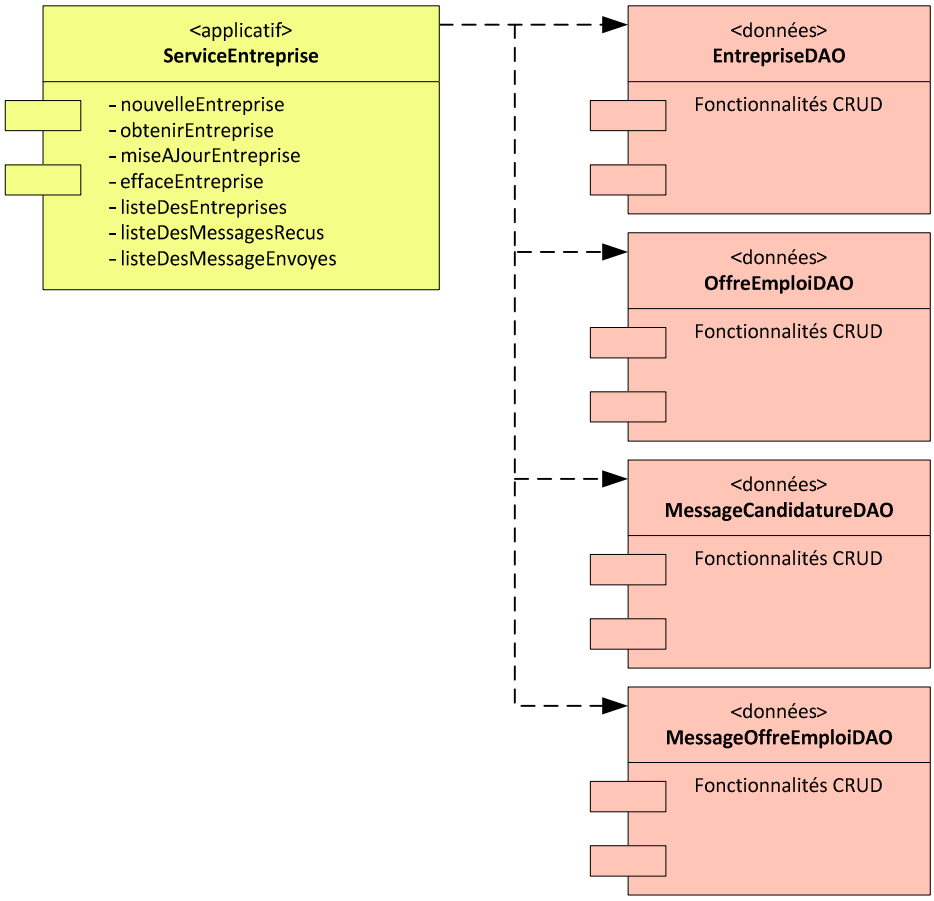


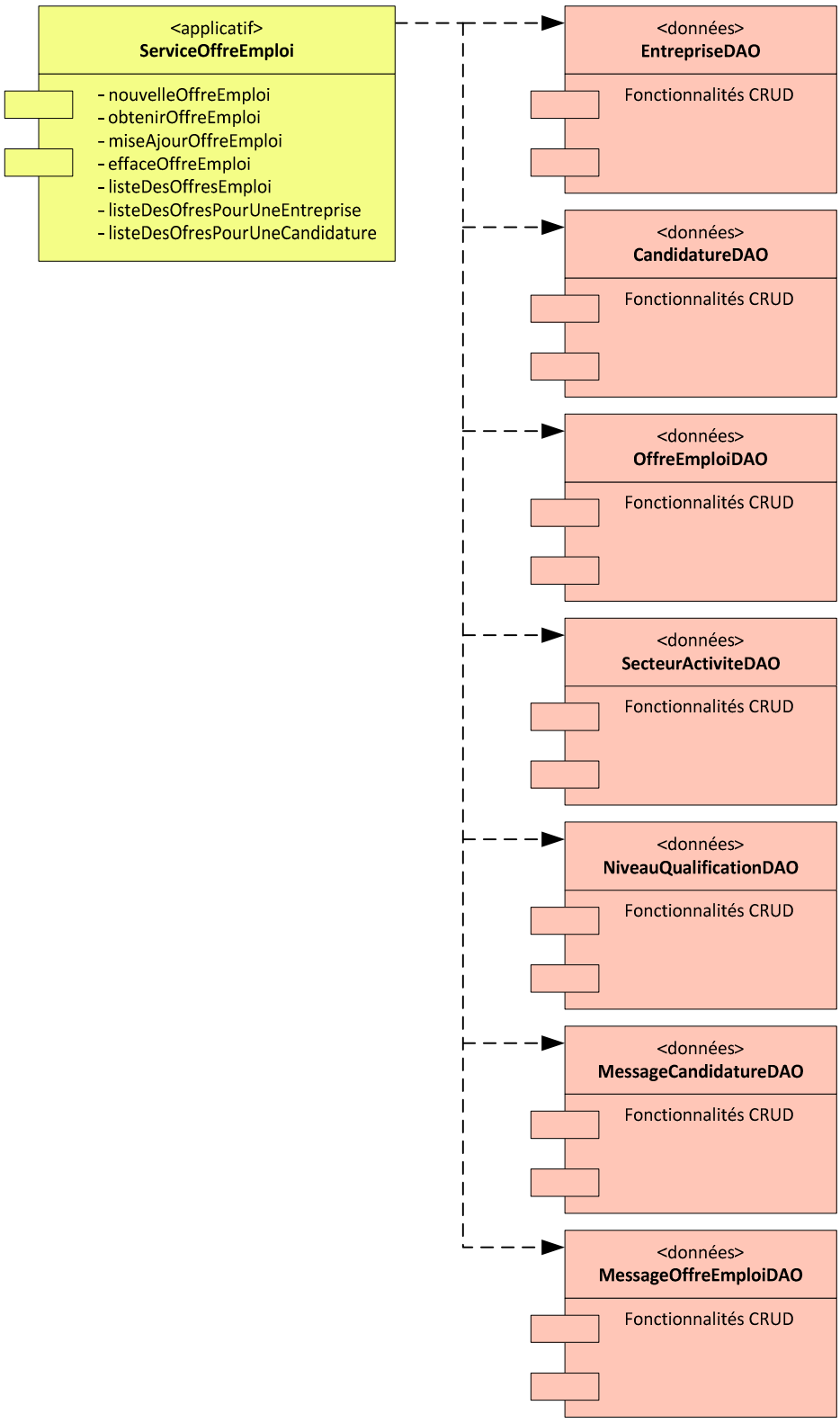

*Diagramme de séquences applicatives du scénario d'obtention de la liste de candidatures correspondant à une offre*

## Relation de dépendances avec les composants DAO

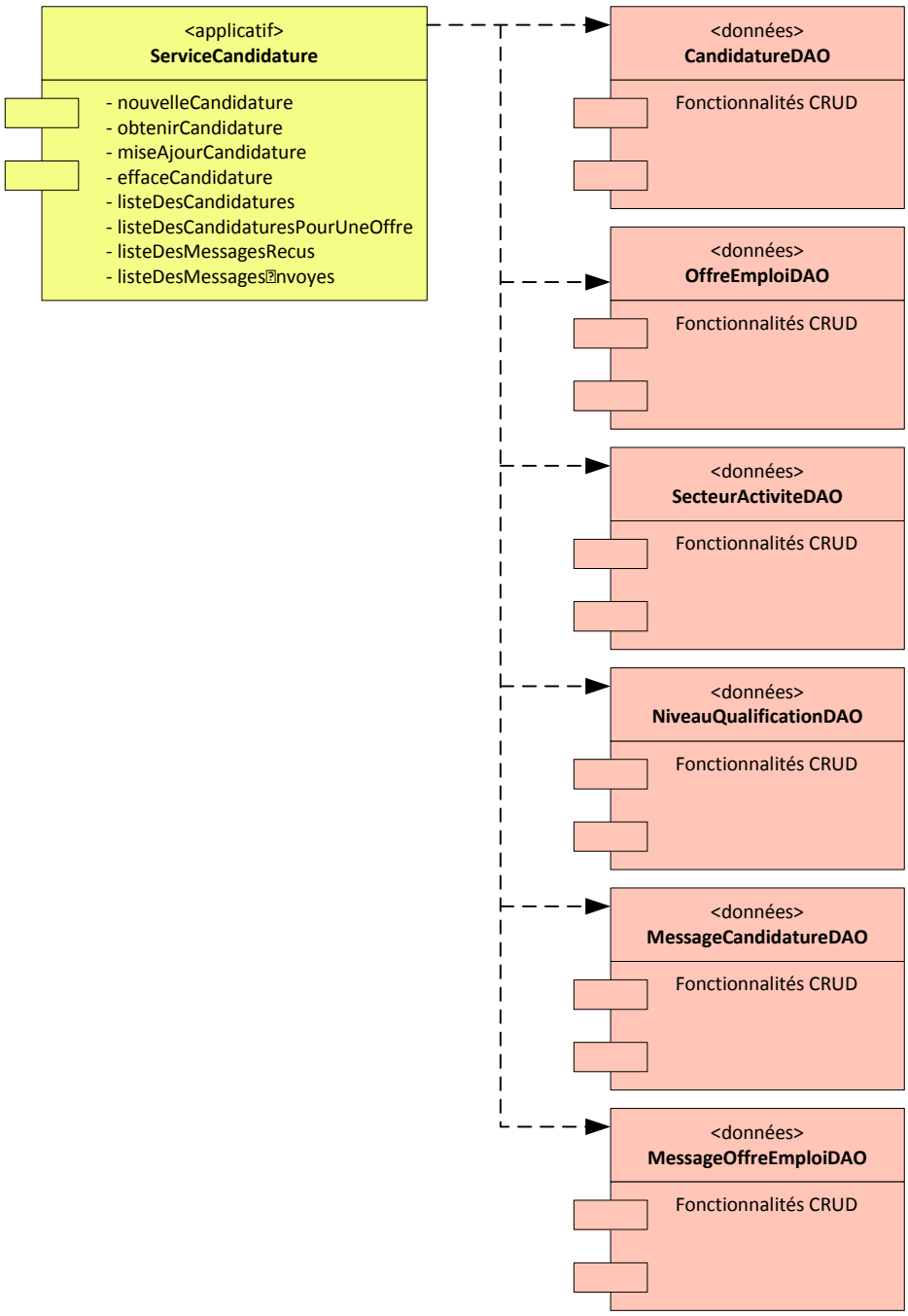
Tous les liens de dépendances présentés dans les diagrammes suivant sont des liens stéréotypés *« utilise »*. Afin de ne pas trop les surcharger, seules les flèches sont représentées ici.



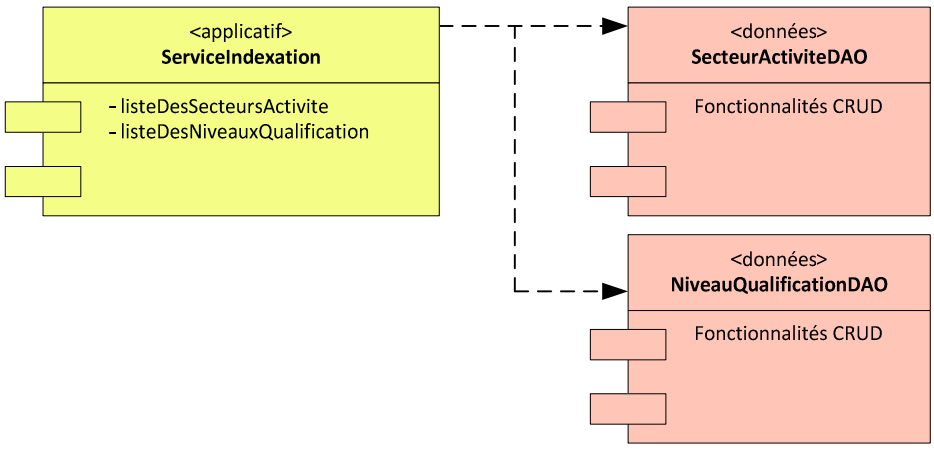

*Dépendances du composant ServiceEntreprise*



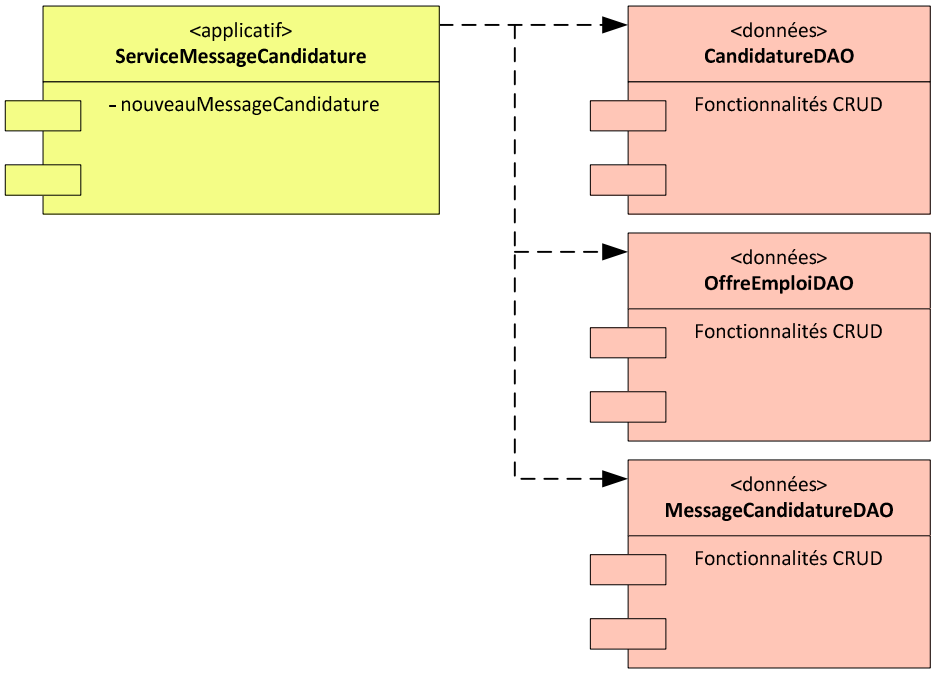

*Dépendances du composant ServiceOffreEmploi*



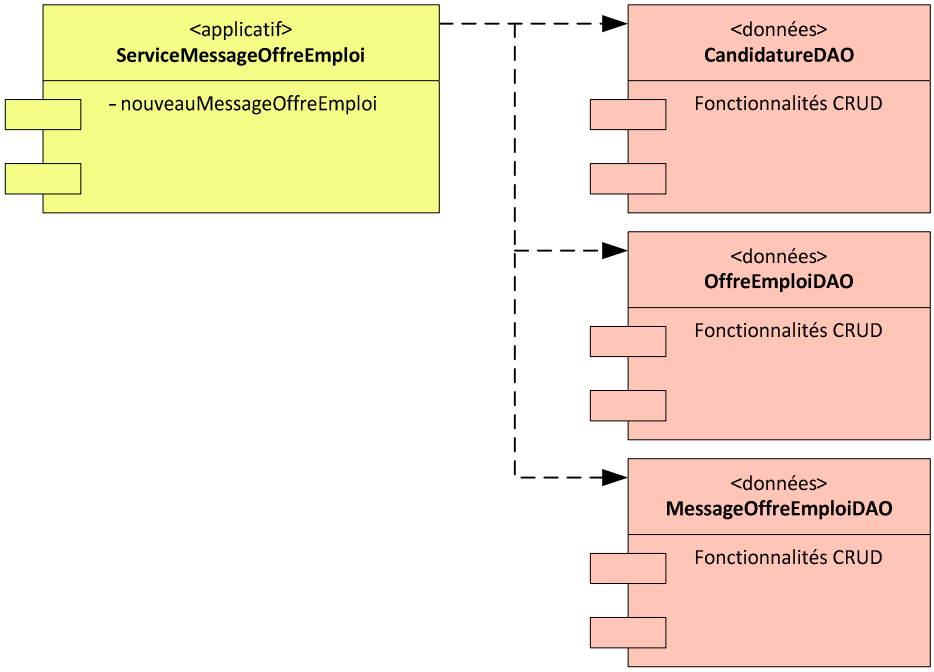

*Dépendances du composant ServiceCandidature*




*Dépendances du composant ServiceIndexation*




*Dépendances du composant ServiceMessageCandidature*




*Dépendances du composant ServiceMessageOffreEmploi*

# Fil rouge : séance 5
# Module FIP INF 211

## Gestion de la syndication des informations

## 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 (pages 12 et 13) 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. Un autre format concurrent existe, le format *atom*, mais il ne sera pas abordé 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/

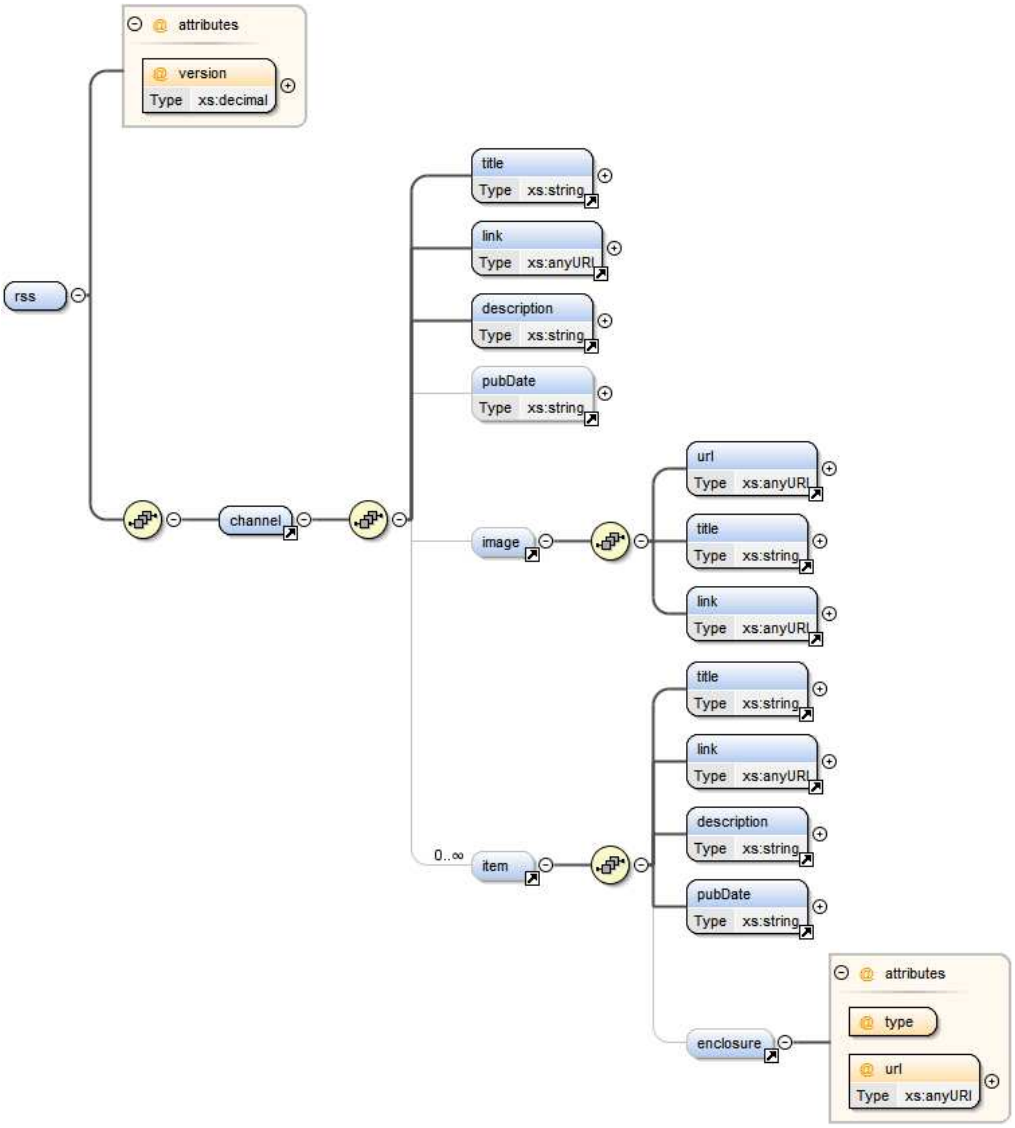
# Les technologies mises en œuvre

## *Le format RSS*

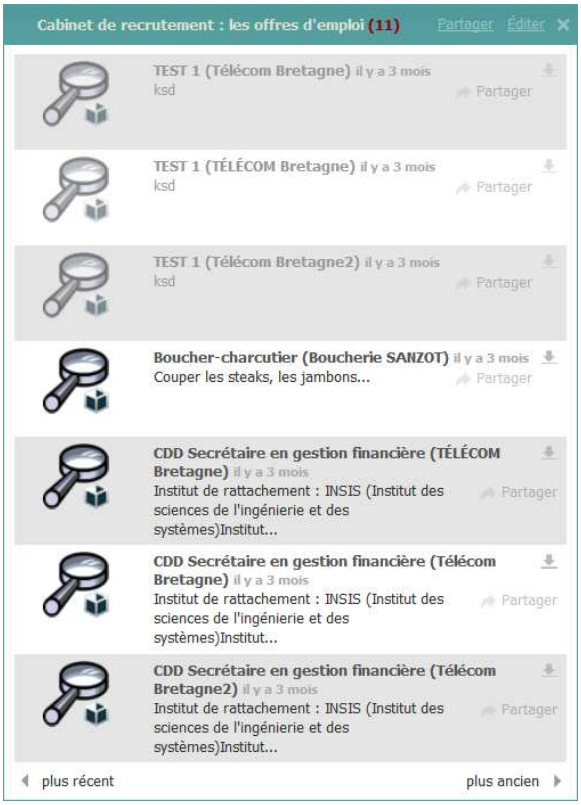
Le format RSS est donc un moyen de consulter rapidement des informations diverses (actualités, réseaux sociaux, etc.) à l'aide de différents outils qui seront les consommateurs de ces flux bruts pour en assurer la visualisation. Différents types d'outils en permettent la lecture :

- 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.

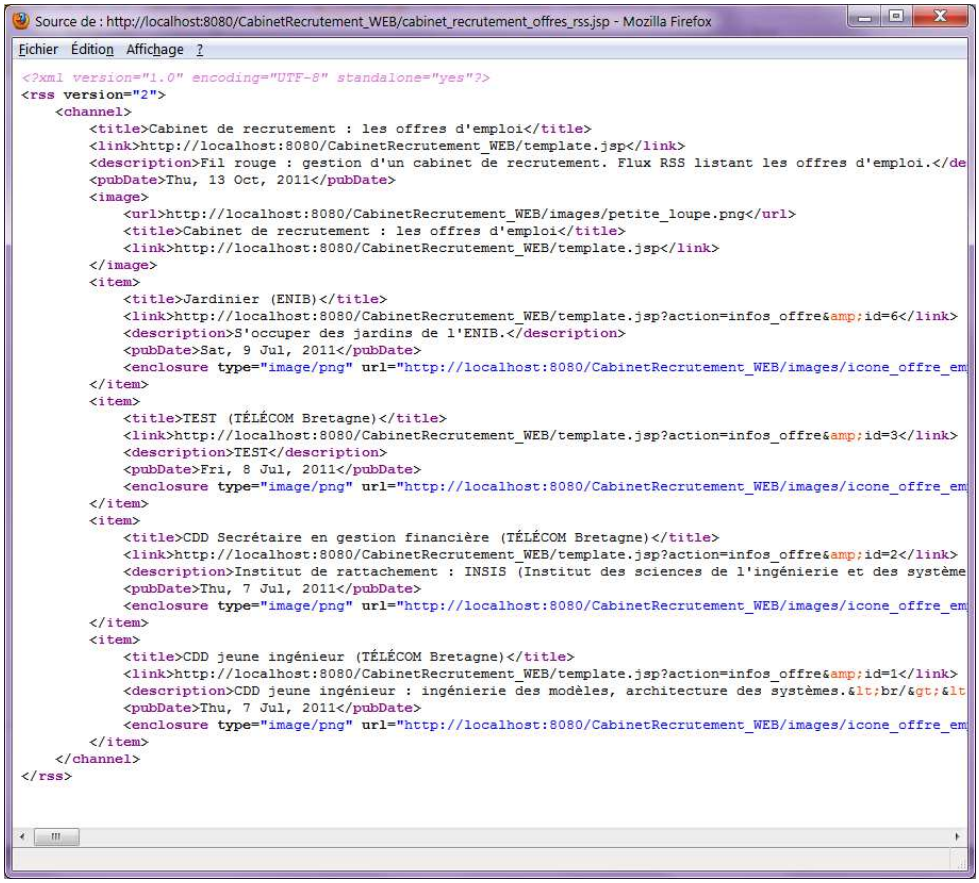
Le format RSS n'est pas standardisé ni normalisé, il existe donc des variations possibles dans le format. Pour la version 2.0, seule une spécification existe (voir le site Web indiqué dans la partie *Pour aller plus loin…*). Pour utiliser JAXB, il est nécessaire de partir d'un schéma XML représentant 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* :
  - *title* : le titre de l'offre.
  - *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).
  - *description* : le description de la mission.
  - *pubDate* : la date de dépôt de l'offre.
  - *enclosure* : élément optionnel, permet de lier un objet média à l'item. Dans l'exemple, on l'associe à une image (attribut *type* égal *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.

## *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*.
   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 : elle produit du XML mais n'en consomme pas.

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 élément XML du schéma.

# Travail à réaliser

## *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 :

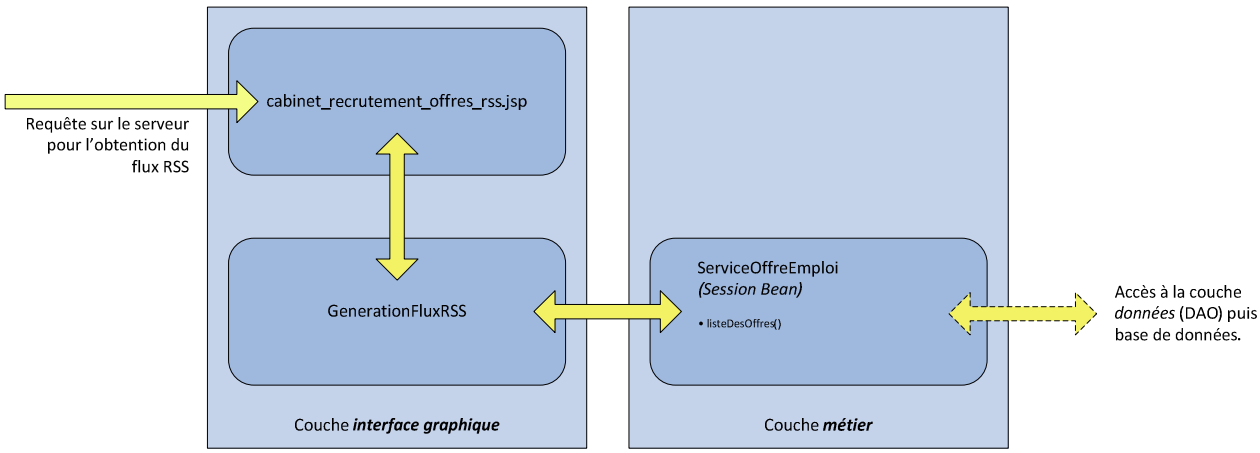
1. Récupérer sur Moodle l'archive ZIP *RSS.zip*. Un fois décompactée, le schéma XML se trouve dans le fichier *RSS.xsd*.
2. L'utilisation de cet outil se fait au sein d'un terminal. L'option *–p* sera utilisée pour fixer le nom du package des classes générées :

   *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*.
   xjc génère alors les fichiers suivants :
   - *eu/telecom_bretagne/cabinet_recrutement/front/rss/**Channel**.java*
   - *eu/telecom_bretagne/cabinet_recrutement/front/rss/**Enclosure**.java*
   - *eu/telecom_bretagne/cabinet_recrutement/front/rss/**Image**.java*
   - *eu/telecom_bretagne/cabinet_recrutement/front/rss/**Item**.java*
   - *eu/telecom_bretagne/cabinet_recrutement/front/rss/**Rss**.java*
   - *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.

## *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*.

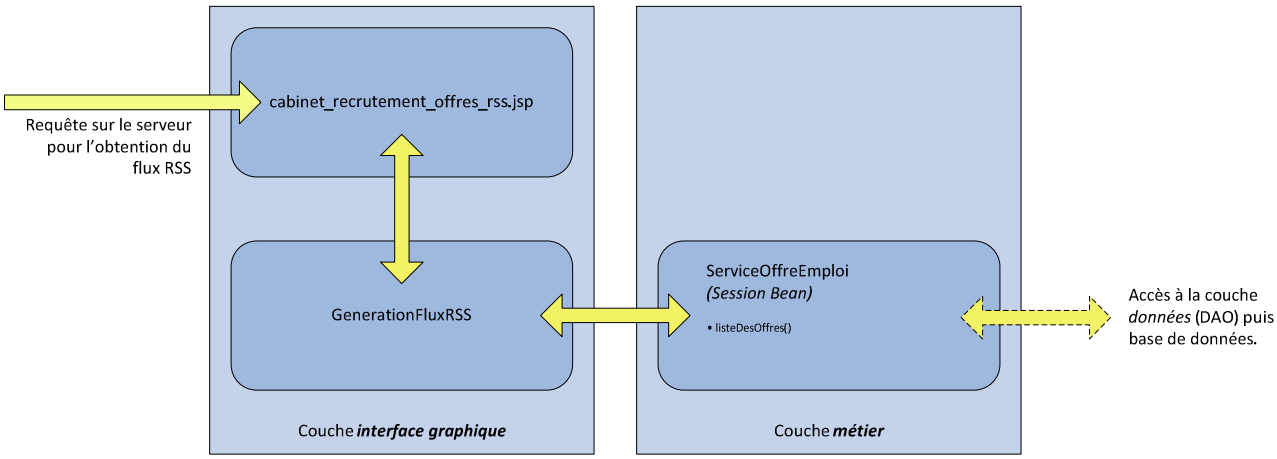
1. Préparation de l'environnement de travail :
   - Recopiez le JSP *cabinet_recrutement_offres_rss.jsp* dans le répertoire *WebContent* du projet *CabinetRecrutement_WEB*.
     À noter : il n'y a pas de passages à la ligne dans ce JSP : NE PAS EN AJOUTER ! Cela perturberait la génération du flux.
   - Recopiez la classe *GenerationFluxRSS* dans le même package que les classes générées pas *xjc* : package *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 :
   - Pour ces images, il est possible d'utiliser celles présentes dans le répertoire *WebContent/images* du projet *CabinetRecrutement_WEB*.
   - 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 qu'il faut donc créer.

# Fil rouge : corrigé séance 5
# Module FIP INF 211

## Gestion de la syndication des informations




## Les JSP

### *cabinet_recrutement_offres_rss.jsp*

```
<%@page import="eu.telecom_bretagne.cabinet_recrutement.front.rss.GenerationFluxRSS"%><%@ page language="java"
contentType="text/xml; charset=UTF-8" pageEncoding="ISO-8859-1"%><%GenerationFluxRSS.offresEmploi(out,
getServletContext().getInitParameter("URL_BASE"));%>
```

Pour une lecture facilitée tout en sachant que les passages à la ligne empêchent le bon fonctionnement :

```
<%@page import="eu.telecom_bretagne.cabinet_recrutement.front.rss.GenerationFluxRSS"%>
<%@ page language="java" contentType="text/xml; charset=UTF-8" pageEncoding="ISO-8859-1"%>
<%GenerationFluxRSS.offresEmploi(out, getServletContext().getInitParameter("URL_BASE"));%>
```

### *cabinet_recrutement_candidatures_rss.jsp*

```
<%@page import="eu.telecom_bretagne.cabinet_recrutement.front.rss.GenerationFluxRSS"%><%@ page language="java"
contentType="text/xml; charset=UTF-8" pageEncoding="ISO-8859-1"%><%GenerationFluxRSS.candidatures(out,
getServletContext().getInitParameter("URL_BASE"));%>
```

## La classe GenerationFluxRSS

```
package eu.telecom_bretagne.cabinet_recrutement.front.rss;

import java.io.Writer;
import java.math.BigDecimal;
import java.util.Date;
import java.util.List;

import javax.servlet.jsp.JspWriter;
import javax.xml.bind.JAXBContext;
import javax.xml.bind.JAXBException;
import javax.xml.bind.Marshaller;

import eu.telecom_bretagne.cabinet_recrutement.data.model.Candidature;
import eu.telecom_bretagne.cabinet_recrutement.data.model.OffreEmploi;
import eu.telecom_bretagne.cabinet_recrutement.front.utils.ServicesLocator;
import eu.telecom_bretagne.cabinet_recrutement.front.utils.ServicesLocatorException;
import eu.telecom_bretagne.cabinet_recrutement.front.utils.Utils;
import eu.telecom_bretagne.cabinet_recrutement.service.IServiceCandidature;
import eu.telecom_bretagne.cabinet_recrutement.service.IServiceOffreEmploi;

/**
 * Classe permettant la gestion des flux RSS publiant la liste des offres
 * d'emploi et la liste des candidatures. La classe contient deux méthodes
 * statiques utilisables au sein d'un JSP :
 * <ul>
 *     <li>{@code GenerationFluxRSS.offresEmploi (Writer writer, String urlBase)}</li>
 *     <li>{@code GenerationFluxRSS.candidatures(Writer writer, String urlBase)}</li>
 * </ul>
 * @author Philippe TANGUY
 */
public class GenerationFluxRSS
{
  //---------------------------------------------------------------------------
  /**
   * Construction du flux RSS de la liste des offres d'emploi. Celles-ci sont obtenues
   * par l'appel de la méthode {@code listeDesOffres()}, voir : {@link IServiceOffreEmploi}.
   * @param writer  l'instance du {@link Writer} sur lequel sera écrit le flux RSS.
   *                La méthode étant appelée au sein d'un JSP, celui-ci est l'instance
   *                de l'objet prédéfini {@code out}, instance de {@link JspWriter}.
   * @param urlBase l'URL de base (une chaîne de caractères) permettant la récupération
   *                des éléments du flux.
   * @throws JAXBException
   * @throws ServicesLocatorException
   */
  public static void offresEmploi(Writer writer, String urlBase) throws JAXBException, ServicesLocatorException
  {
    // Récupération du service de gestion des offres d'emploi à l'aide de
    // la classe ServiceLocator.
    // A éventuellement adapter à votre projet.
    IServiceOffreEmploi serviceOffreEmploi =
        (IServiceOffreEmploi)ServicesLocator.getInstance().getRemoteInterface("ServiceOffreEmploi");

    // Récupération des offres d'emploi.
    List<OffreEmploi> offres = serviceOffreEmploi.listeDesOffres();

    // Création du "contexte" JAXB. Celui-ci est paramétré avec le nom du package
    // contenant les classes générées par l'outil xjc.
    JAXBContext jc = JAXBContext.newInstance("eu.telecom_bretagne.cabinet_recrutement.front.rss");

    // Le "marshaller" est la classe permettra de gérer la sérialisation :
    // instances --> flux XML.
    Marshaller marshaller = jc.createMarshaller();
    marshaller.setProperty(Marshaller.JAXB_FORMATTED_OUTPUT, true); // Pour que le flux généré soit joli tout plein...

    // Instance de l'objet ObjectFactory qui permettra de créer les instances qui
    // permettront au marshaller de générer le flux XML.
    ObjectFactory fabrique = new ObjectFactory();

    // Création de l'objet racine (élément <rss>)
    Rss rss = fabrique.createRss();
    // Mise à jour du numéro de version RSS
    rss.setVersion(new BigDecimal(2));
```

```
  // Création du "channel" (élément <channel>)
  Channel channel = fabrique.createChannel();
  channel.setTitle("Cabinet de recrutement : les offres d'emploi");
  channel.setLink(urlBase + "/CabinetRecrutement_WEB/template.jsp");
  channel.setDescription("Fil rouge : gestion d'un cabinet de recrutement. Flux RSS listant les offres d'emploi.");
  // Normalement la date est celle du dépôt de la dernière offre. Ici, pour simplifier,
  // c'est la date en cours.
  channel.setPubDate(Utils.date2StringRSS(new Date()));

  // Création de l'image (optionnel dans la spécif RSS), mise à jour des données
  // de celle-ci et affectation de l'image au channel.
  Image logo = fabrique.createImage();
  logo.setTitle("Cabinet de recrutement : les offres d'emploi");
  logo.setUrl(urlBase + "/CabinetRecrutement_WEB/images/petite_loupe.png");
  logo.setLink(urlBase + "/CabinetRecrutement_WEB/template.jsp");
  channel.setImage(logo);

  // On parcours la liste des offres d'emploi. Pour chacune de celles-ci,
  // un item est créé, renseigné et ajouté au channel.
  for(OffreEmploi offre : offres)
  {
    Item item = fabrique.createItem();
    item.setTitle(offre.getTitre() + " (" + offre.getEntreprise().getNom() + ")");
    item.setLink(urlBase + "/CabinetRecrutement_WEB/template.jsp?action=infos_offre&id=" + offre.getId());
    item.setDescription(Utils.text2HTML(offre.getDescriptifMission()));
    Enclosure enclosure = fabrique.createEnclosure();
    enclosure.setUrl(urlBase + "/CabinetRecrutement_WEB/images/icone_offre_emploi.png");
    enclosure.setType("image/png");
    item.setEnclosure(enclosure);
    item.setPubDate(Utils.date2StringRSS(offre.getDateDepot()));

    // Une petite subtilité : on pourrait penser pouvoir ajouter directement
    // l'item au channel mais comme il y a potentiellement plusieurs items, il
    // y a une List<Item> qui les référencie. Cette liste est accessible via la
    // méthode getItem() qui renvoie la liste, sur cette liste est ajouté le nouvel
    // item.
    channel.getItem().add(item);
  }

  // Inclusion du channel dans le rss.
  rss.setChannel(channel);

  // ----------------------------------------
  // A compléter...
  //
  // Principe, pour chaque offre :
  //   - créer un élement item
  //   - renseigner les infos
  //   - inclusion de l'item dans le channel
  // ----------------------------------------

  // A ce stade l'objet rss est complet (les données ont toutes été incluses), on
  // procède à la sérialisation. La méthode prend en paramètres l'objet à sérialiser
  // (rss), le flux sera écrit sur le writer.
  // Le writer (instance de l'interface Writer) est en fait l'objet out provenant
  // du JSP. ce qui sera écrit sur ce flux sera renvoyé par le serveur Web sur le
  // navigateur (ou l'outil affichant le flux RSS).
  marshaller.marshal(rss, writer);
}
```

```
  //---------------------------------------------------------------------------
  /**
   * Construction du flux RSS de la liste des candidature. Celles-ci sont obtenues par
   * l'appel de la méthode {@code listeDesCandidatures()}, voir : {@link IServiceCandidature}.
   * @param writer  l'instance du {@link Writer} sur lequel sera écrit le flux RSS.
   *                La méthode étant appelée au sein d'un JSP, celui-ci est l'instance
   *                de l'objet prédéfini {@code out}, instance de {@link JspWriter}.
   * @param urlBase l'URL de base (une chaîne de caractères) permettant la récupération
   *                des éléments du flux.
   * @throws JAXBException
   * @throws ServicesLocatorException
   */
  public static void candidatures(Writer writer, String urlBase) throws JAXBException, ServicesLocatorException
  {
    IServiceCandidature serviceCandidature =
        (IServiceCandidature) ServicesLocator.getInstance().getRemoteInterface("ServiceCandidature");
    List<Candidature> candidatures = serviceCandidature.listeDesCandidatures();

    JAXBContext jc = JAXBContext.newInstance("eu.telecom_bretagne.cabinet_recrutement.front.rss");
    Marshaller marshaller = jc.createMarshaller();
    marshaller.setProperty(Marshaller.JAXB_FORMATTED_OUTPUT, true);
    ObjectFactory fabrique = new ObjectFactory();

    Rss rss = fabrique.createRss();
    rss.setVersion(new BigDecimal(2));

    Channel channel = fabrique.createChannel();
    channel.setTitle("Cabinet de recrutement : les candidatures");
    channel.setLink(urlBase + "/CabinetRecrutement_WEB/template.jsp");
    channel.setDescription("Fil rouge : gestion d'un cabinet de recrutement. Flux RSS listant les candidatures.");
    channel.setPubDate(Utils.date2StringRSS(new Date()));
    Image logo = fabrique.createImage();
    logo.setTitle("Cabinet de recrutement : les candidatures");
    logo.setUrl(urlBase + "/CabinetRecrutement_WEB/images/petite_loupe.png");
    logo.setLink(urlBase + "/CabinetRecrutement_WEB/template.jsp");
    channel.setImage(logo);

    for(Candidature candidature : candidatures)
    {
      Item item = fabrique.createItem();
      item.setTitle(candidature.getNom() + " " + candidature.getPrenom());
      item.setLink(urlBase + "/CabinetRecrutement_WEB/template.jsp?action=infos_candidature&id=" + candidature.getId());
      item.setDescription(Utils.text2HTML(candidature.getAdresseEmail() + "\n" + candidature.getCv()));
      Enclosure enclosure = fabrique.createEnclosure();
      enclosure.setUrl(urlBase + "/CabinetRecrutement_WEB/images/icone_candidature.png");
      enclosure.setType("image/png");
      item.setEnclosure(enclosure);
      item.setPubDate(Utils.date2StringRSS(candidature.getDateDepot()));

      channel.getItem().add(item);
    }

    rss.setChannel(channel);
    marshaller.marshal(rss, writer);
  }
  //---------------------------------------------------------------------------
}
```

# Fil rouge : recette de l'application
# Module FIP INF 211

Binôme : ……

……

| | Fonctionnalité | Réalisée (O/N) | Contrainte(s) prise(s) en compte | Réalisée (O/N) | Commentaires |
|---|---|---|---|---|---|
| **Gestion des entreprises** | Référencer une nouvelle entreprise.<br><br>*[vitale]* | | Prise en compte de : identifiant, nom, descriptif, adresse postale. | | |
| | Lister les entreprises référencées.<br><br>*[vitale]* | | | | |
| | Afficher les informations d'une entreprise.<br><br>*[vitale]* | | Affichage de : identifiant, nom, descriptif, adresse postale. | | |
| | Mettre à jour les informations d'une entreprise.<br><br>*[mineure]* | | Fonctionnalité accessible uniquement à l'entreprise. | | |
| | Supprimer une entreprise.<br><br>*[mineure]* | | Fonctionnalité accessible uniquement à l'entreprise. | | |
| | Envoyer un message à un candidat pour une offre d'emploi donnée.<br>*[mineure]* | | Fonctionnalité accessible uniquement à l'entreprise. | | |
| | Lister les messages reçus des candidats.<br><br>*[mineure]* | | Fonctionnalité accessible uniquement à l'entreprise. | | |
| | Lister les messages envoyés aux candidats.<br><br>*[mineure]* | | Fonctionnalité accessible uniquement à l'entreprise. | | |

| | | | | | |
|---|---|---|---|---|---|
| **Gestion des offres d'emploi** | Référencer une offre d'emploi.<br><br>*[vitale]* | | Fonctionnalité accessible uniquement à l'entreprise. | | |
| | | | Prise en compte de : titre de l'offre, descriptif mission, profil recherché, date dépôt, secteur(s) activité et niveau de qualification. | | |
| | Lister les offres d'emploi.<br><br>*[vitale]* | | | | |
| | Afficher les informations d'une offre d'emploi .<br><br>*[vitale]* | | Affichage des informations pour tous les utilisateurs. | | |
| | | | Fonctionnalité optionnelle d'envoi de message : à candidat si entreprise connectée ou entreprise si candidat connecté. | | |
| | Mettre à jour les informations d'une offre d'emploi.<br><br>*[mineure]* | | Fonctionnalité accessible uniquement à l'entreprise. | | |
| | Supprimer une offre d'emploi.<br><br>*[mineure]* | | Fonctionnalité accessible uniquement à l'entreprise. | | |
| **Gestion des candidatures** | Référencer une nouvelle candidature.<br><br>*[vitale]* | | Prise en compte de : nom, prénom, date naissance, adresse, email, date dépôt, CV, secteur(s) activité et niveau de qualification. | | |
| | Lister les candidatures référencées.<br><br>*[vitale]* | | | | |
| | Afficher les informations d'une candidature.<br><br>*[vitale]* | | Fonctionnalité accessible uniquement au candidat. | | |
| | Lister les offres d'emploi qui correspondent à une candidature.<br>*[vitale]* | | Fonctionnalité accessible uniquement au candidat. | | |
| | Mettre à jour les informations d'une candidature.<br><br>*[mineure]* | | Fonctionnalité accessible uniquement au candidat. | | |
| | Supprimer une candidature. | | Fonctionnalité accessible uniquement au candidat. | | |

| | | | | | |
|---|---|---|---|---|---|
| | *[mineure]* | | | | |
| **Gestion des candidatures (suite)** | Envoyer un message à une entreprise pour une offre d'emploi donnée.<br>*[mineure]* | | Fonctionnalité accessible uniquement au candidat. | | |
| | Lister les messages reçus des entreprises.<br>*[mineure]* | | Fonctionnalité accessible uniquement au candidat. | | |
| | Lister les messages envoyés aux entreprises.<br>*[mineure]* | | Fonctionnalité accessible uniquement au candidat. | | |
| **Gestion de l'identification** | Se connecter au système d'information.<br>*[vitale]* | | | | |
| | Se déconnecter du système d'information.<br>*[vitale]* | | | | |
| **Gestion de la syndication du contenu** | Exporter la liste des offres d'emploi au sein d'un flux RSS.<br>*[mineure]* | | | | |
| | Exporter la liste des candidatures au sein d'un flux RSS.<br>*[mineure]* | | | | |

| **Autre(s) commentaire(s)** |
|---|
| |